



Universitat Politècnica de Catalunya

**Contribución a la Protección de Agentes Móviles frente a
Hosts Maliciosos. Detección de Ataques y Políticas de
Castigo**

Tesis Doctoral

Óscar Esparza Martín

Departamento de Ingeniería Telemática

Universitat Politècnica de Catalunya

Director

Dr. Miguel Soriano Ibáñez

This document has been produced using L^AT_EX 2_ε.

Barcelona, Mayo 2004

Resumen

Un agente móvil es una entidad software compuesta por código, datos y estado que tiene capacidad para migrar de host en host realizando ciertas acciones en nombre de una entidad. La flexibilidad que ofrecen los agentes móviles los hace especialmente atractivos para servicios que requieran del tratamiento de datos dispersos en múltiples ubicaciones. Asimismo, su uso permite mejorar algunos parámetros críticos como el rendimiento y la latencia de la red. Desgraciadamente, existen serios problemas relativos a la seguridad que impiden el uso masivo de sistemas de agentes móviles. En esta Tesis se estudian las propuestas existentes de protección del agente frente a los ataques del host que lo ejecuta. Asimismo, se introducen tres nuevas aportaciones que pretenden contribuir a solucionar el problema de los hosts maliciosos. Las propuestas de prevención de ataques o son demasiado costosas o no dan una protección adecuada. Es por ello que nos centraremos en las propuestas de detección pues han sido menos estudiadas y son más fáciles de implementar. Como primera contribución de esta Tesis, se introduce un Protocolo de Detección de Sospechosos para mejorar y complementar algunos aspectos de la propuesta de detección más conocida, la de las trazas criptográficas [Vig98]. Dado que el mecanismo conjunto anterior es todavía demasiado costoso, como segunda contribución se presenta un mecanismo propio de detección más ligero. Dicho mecanismo está basado en el empotrado de marcas de agua en el agente, esto es, Watermarking de Agentes Móviles. Los mecanismos de detección son poco útiles si no hay posibilidad de castigar a los hosts maliciosos. Por ello, como tercera contribución se presenta una nueva entidad sancionadora, la Autoridad de Revocación de Hosts.

Agradecimientos

Puedes llegar a cualquier parte, siempre que andes lo suficiente.

Lewis Carroll

Quiero dedicar especialmente este trabajo a mi mujer, sin ella forzándome a trabajar todos los días no sé cuando hubiera podido acabar la Tesis. Gracias Silvia, sin ti esto no hubiera sido posible, y perdón por todos los escaqueos y excusas que te he puesto estos últimos meses, creo que más que tú no ha sufrido nadie con la Tesis. Espero tener más tiempo a partir de ahora para compartir contigo. También quiero agradecer al “Papa” la ayuda en el camino y sus buenos consejos. Al final no será necesario eso de que “aquí se rompen piernas”. A Jose, por la tregua en la batalla por la pelota que ahora se ha convertido en batalla con el agua. A todo ISG: Jordi, Josep, Marcos, Juan, Marcel, Esteve y ahora Xisca, por esos momentos en el bar estando en familia. A todos los miembros de Sertel por sus muchos vicios y pocas virtudes (¿o era al revés?), seguid siendo como sois, y a ver cuando nos vamos de copas de nuevo. A mi familia y a la de Silvia, porque no cambien nunca. A todos mis amigos, que son muchos y no los nombro por no olvidarme a ninguno. En fin, a todos aquellos que en mayor o menor medida han contribuido directa o indirectamente, aguantándome estos meses de mucha presión, gracias a todos.

Lista de Acrónimos

AC Attribute Certificate

AEC Agent Execution Certificate

AGLET AGent appLET

ASDK Aglet Software Development Kit

CA Certificate Authority

Cervantes CERTificate VALidatioN TEST-bed

COD Code On Demand

CRL Certificate Revocation List

DES Digital Encryption Standard

ETE Execution Time Estimator

HoRA Host Revocation Authority

HRL Host Revocation List

HRP-T Host Revocation Protocol for Traces

HRP-TP Host Revocation Protocol for Traces with Privacy

HRP-M Host Revocation Protocol for MAW
HRP-MP Host Revocation Protocol for MAW with Privacy
IC Identity Certificate
JCA Java Cryptography Architecture
JCE Java Cryptography Extension
JDK Java Development Kit
JVM Java Virtual Machine
LAN Local Area Network
MA Mobile Agent
MAW Mobile Agent Watermarking
NAT Network Address Translation
NTP Network Time Protocol
OCSP Online Certificate Status Protocol
OHSP Online Host Status Protocol
OWHF One Way Hash Function
PKI Public Key Infrastructure
PMI Privilege Management Infrastructure
PRP-T Provisional Revocation Protocol for Traces
PRP-TP Provisional Revocation Protocol for Traces with Privacy
REV Remote Evaluation

RPC Remote Procedure Call

RSA Rivest Shamir Adleman

SDP Suspicious Detection Protocol

SHA-1 Secure Hash Function v.1

TST Traces Storage Timestamp

TTE Transmission Time Estimator

TTP Trusted Third Party

Índice general

1. Introducción	1
1.1. Acerca de la Tesis	1
1.2. Motivación y Objetivos	2
1.3. Organización de la Tesis	3
1.4. Introducción a los Sistemas Distribuidos	4
1.5. Paradigmas de Diseño en Sistemas Distribuidos	6
1.5.1. Traspaso de Mensajes	6
1.5.2. Invocación de Procedimientos Remotos (RPC)	8
1.5.3. Código Móvil	9
1.6. Agentes Móviles	12
1.6.1. Aplicaciones de los Agentes Móviles	14
1.6.2. Ventajas de los Agentes Móviles	17
1.6.3. Inconvenientes de los Agentes Móviles	19
1.7. Terminología	22
1.8. Contribuciones de esta Tesis	24
1.8.1. Revistas y Publicaciones JCR	26
1.8.2. Congresos Internacionales	27
1.8.3. Congresos Nacionales	27
2. Estado del Arte	29
2.1. Introducción al Problema de los Hosts Maliciosos	29

2.1.1.	Ataques	30
2.1.2.	Requisitos de Seguridad	33
2.2.	Propuestas de Prevención de Ataques	35
2.2.1.	Modelos Basados en Confianza	35
2.2.2.	Hardware Específico de Ejecución	36
2.2.3.	Agentes Cooperativos	36
2.2.4.	Entropía de Agentes Móviles	37
2.2.5.	Generación de Claves Dependientes del Entorno	38
2.2.6.	Ofuscación del Código	38
2.2.7.	Computación de Funciones Cifradas	39
2.3.	Propuestas de Detección de Ataques	40
2.3.1.	Replicación y Voto	41
2.3.2.	Trazas Criptográficas	41
2.3.3.	Estados de Referencia	42
2.4.	Conclusiones del Capítulo	43
3.	Protocolo de Detección de Sospechosos (SDP)	45
3.1.	Mejoras sobre Propuestas Existentes	45
3.2.	Funcionamiento de SDP	46
3.2.1.	Fase de Configuración Inicial	47
3.2.2.	Fase de Envío del Agente Móvil	51
3.2.3.	Fase de Validación de Resultados	54
3.3.	Integración con Otras Propuestas	56
3.3.1.	Trazas Criptográficas	56
3.3.2.	Ofuscación	57
3.4.	Mejoras	57
3.5.	Ataques al Protocolo	59
3.5.1.	Ataques de un Único Host	59
3.5.2.	Ataques de Confabulación	60
3.6.	Inconvenientes	61
3.7.	Implementación y Evaluación del Rendimiento de SDP	63

Índice general

3.7.1. Herramientas Software y Especificaciones Hardware	63
3.7.2. Resultados de las Pruebas	65
3.8. Conclusiones del Capítulo	72
4. Watermarking de Agentes Móviles (MAW)	75
4.1. Nuevo Esquema de Detección de Ataques	75
4.2. Watermarking de Software	76
4.3. Watermarking de Agentes Móviles (MAW)	78
4.3.1. Empotrado de la Marca	80
4.3.2. Transferencia de la Marca	81
4.3.3. Verificación de la Marca y Extracción de Resultados	85
4.3.4. Ejemplo de Funcionamiento del MAW	87
4.3.5. Ventajas	91
4.3.6. Inconvenientes	94
4.3.7. Ataques	95
4.4. Conclusiones del Capítulo	96
5. Autoridad de Revocación de Hosts (HoRA)	97
5.1. Políticas de Castigo en Sistemas de Agentes Móviles	97
5.2. Castigo Basado en Revocación de Hosts	100
5.3. Consulta del Estado	102
5.3.1. Política de Consulta Off-line	102
5.3.2. Política de Consulta On-line	105
5.4. Revocación de Hosts	107
5.5. Protocolos de Revocación para las Trazas Criptográficas	108
5.5.1. Protocolos sin Privacidad	109
5.5.2. Protocolos con Privacidad	121
5.5.3. Ataques a los Protocolos	133
5.5.4. Inconvenientes	136
5.6. Protocolos de Revocación para MAW	136
5.6.1. Protocolo sin Privacidad	137

Índice general

5.6.2. Protocolo con Privacidad	142
5.6.3. Ataques a los Protocolos	149
5.7. Inconvenientes de la Revocación de Hosts	150
5.8. Conclusiones del Capítulo	152
6. Conclusiones y Líneas Futuras	153
Bibliografía	161

Índice de figuras

1.1. Paradigmas de Diseño en Aplicaciones Distribuidas	7
1.2. Traspaso de Mensajes	8
1.3. Invocación de Procedimientos Remotos	9
1.4. Evaluación Remota	11
1.5. Código Bajo Demanda	12
1.6. Agente Móviles	14
2.1. Ejemplo de un Agente Móvil de Búsqueda	31
2.2. Ejecución Honesta del Agente de Búsqueda	32
3.1. Fase de Configuración Inicial	49
3.2. Cronograma de la Fase de Configuración Inicial	50
3.3. Fase de Envío del Agente	53
3.4. Cronograma de la Fase de Envío del Agente	54
3.5. Tiempo de Ejecución en los Hosts	66
3.6. Tiempo de Ejecución en los Hosts para Windows y Linux	67
3.7. Tiempo de Ejecución en el Host Origen	68
3.8. Tiempo Total Empleado por el Agente	68
3.9. CPU en el Host	70
3.10. CPU en el Host para Windows y Linux	70
3.11. CPU en el Host Origen	71
3.12. Tamaño del Agente	72

Índice de figuras

4.1. Empotrado de la Marca	82
4.2. Transferencia de la Marca	84
4.3. Migración del agente con MAW	85
4.4. Verificación de la Marca y Extracción de Resultados	86
4.5. Agente de Búsqueda con MAW	88
4.6. Ejecución del Agente de Búsqueda con MAW	91
4.7. Reglas de Integridad que Deben Cumplir los Contenedores	92
5.1. Formato de la HRL	104
5.2. Consulta On-line con OHSP	106
5.3. Envío del Agente con el Mecanismo de las Trazas	113
5.4. Envío de las Trazas	113
5.5. HRP-T	118
5.6. PRP-T	122
5.7. Envío del Agente con el Mecanismo de las Trazas y Privacidad	126
5.8. Envío de las Trazas Cuando se Requiere Privacidad	126
5.9. HRP-TP	130
5.10. PRP-TP	134
5.11. Envío del Agente con MAW	139
5.12. HRP-M	142
5.13. Envío del Agente con MAW y Privacidad	146
5.14. HRP-MP	148

Capítulo 1

Introducción

1.1. Acerca de la Tesis

Esta Tesis doctoral ha sido desarrollada en el Information Security Group (ISG)¹ del departamento de Ingeniería Telemática (ENTEL)² de la Universidad Politécnica de Cataluña (UPC)³. Los desarrollos a los que esta Tesis ha contribuido se enmarcan dentro de los siguientes proyectos de investigación:

- ACIMUT: Acceso a Internet Seguro Mediante UMTS (CICYT TIC2000-1120-C03-03).
- DISQET: Distribución de Información Segura con Calidad de Servicio (QoS) para Entornos Telemáticos (CICYT TIC2002-00249).
- UBISEC: *Ubiquitous Networks with a Secure Provision of Services, Access, and Content Delivery* (IST-FP6 506926).

¹ISG home: <http://isg.upc.es>

²ENTEL home: <http://www-entel.upc.es>

³UPC home: <http://www.upc.es>

1.2. Motivación y Objetivos

La sociedad actual está inmersa en una verdadera revolución causada por la incorporación de las tecnologías de la información a la vida diaria de las personas. Para muchos sectores de la población, hoy en día resulta imprescindible acceder a los grandes volúmenes de información que se encuentran dispersos por todo el Mundo. Incluso desde el punto de vista económico, la toma de decisiones en entornos corporativos o industriales depende en gran medida de la calidad de la información que se consulta, y de que se acceda a ella en el menor tiempo posible y en el lugar adecuado. En definitiva, las nuevas tecnologías han hecho posible el acceso en tiempo real a información de cualquier tipo generada por personas diseminadas geográficamente a lo largo de todo el Mundo.

La Internet actual podría considerarse hoy en día la mayor fuente de información existente. Sin embargo, no cumple en muchos aspectos las condiciones necesarias para ofrecer a los usuarios los servicios que demandan de forma adecuada. Millones de personas acceden a la Red y comparten sus contenidos de forma totalmente desordenada. Con los mecanismos actuales, la búsqueda de información suele resultar lenta y tediosa, sobretodo por la dificultad que implica seleccionar la información que buscamos del resto. Acceder a dicha información en un tiempo adecuado puede resultar muy difícil en determinadas condiciones de la red, ya que no se garantiza ninguna calidad de servicio. Incluso se encuentran limitaciones a la hora de acceder a la Red desde cualquier ubicación, ya que la tecnología de acceso mediante terminales móviles esta justo despegando en estos momentos.

La aparición de una nueva generación de servicios está supeditada a la existencia de nuevos mecanismos de acceso a la información que faciliten la tarea a los usuarios. Éstos mecanismos deben ser capaces de seleccionar las informaciones válidas de aquellas que no lo son, y hacerlas llegar al usuario en un tiempo prudencial. Asimismo, se debe dar la posibilidad de acceso a los usuarios desde cualquier tipo de terminal y tecnología. Muy particularmente, se debe hacer énfasis en los terminales de tipo inalámbrico que no están sujetos a las limitaciones que suponen una ubicación fija.

Desde nuestro punto de vista, esta nueva generación de mecanismos y servicios

Capítulo 1. Introducción

pueden basarse en el paradigma de los agentes móviles. Un agente móvil es una entidad software compuesta básicamente por código, datos y estado que tiene capacidad para migrar de host en host realizando ciertas acciones en nombre de un usuario. La flexibilidad que ofrece el uso de agentes móviles los hace especialmente atractivos para todos aquellos servicios que requieran del tratamiento automatizado de un gran volumen de datos dispersos en múltiples ubicaciones. Como por naturaleza están diseñados para migrar de una máquina a otra, son muy útiles para entornos heterogéneos donde existen múltiples plataformas hardware o software, como por ejemplo Internet. El uso de agentes móviles también permite mejorar algunos aspectos como el rendimiento de la red, e incluso es posible distribuir la carga computacional a otras máquinas menos cargadas. Este hecho unido a la posibilidad de realizar una ejecución off-line puede aprovecharse para dar soporte a terminales móviles, que como es bien sabido tienen serios problemas de conectividad, ancho de banda disponible, CPU o batería.

Desgraciadamente, existen serios problemas relativos a la seguridad que impiden el uso masivo de sistemas de agentes móviles. En particular, la protección del agente frente a los ataques del host que lo ejecuta se considera el problema más difícil de resolver con diferencia de todos los relativos a seguridad en sistemas de agentes móviles. A éste se le conoce como el problema de los hosts maliciosos, y los desarrollos realizados en esta Tesis pretenden contribuir a la resolución del mismo.

Como objetivo de esta Tesis pretendemos realizar el estudio de nuevos mecanismos de protección de agentes. Dichos mecanismos deben ser eficaces a la hora de evitar los ataques de hosts maliciosos, o al menos deben intentar minimizar los efectos que producen. Asimismo, no deben representar un gran incremento en los parámetros más críticos del sistema, como son el ancho de banda de la red, la CPU o la capacidad de almacenamiento de los terminales.

1.3. Organización de la Tesis

La organización de la Tesis se detalla a continuación:

1.4. Introducción a los Sistemas Distribuidos

- En el Capítulo 1, “Introducción”, introducimos el paradigma de los agentes móviles dentro de los sistemas distribuidos, así como los principales inconvenientes que presenta relativos a la seguridad.
- En el Capítulo 2, “Estado del Arte”, se describe el problema de los hosts maliciosos, así como las principales propuestas de protección de agentes publicadas hasta el momento.
- En el Capítulo 3, “Protocolo de Detección de Sospechosos (SDP)”, se introducen ciertas mejoras sobre la propuesta de las trazas criptográficas de Vigna [Vig98] mediante la adición de un Protocolo de Detección de Sospechosos.
- En el Capítulo 4, “Watermarking de Agentes Móviles (MAW)”, se introduce un nuevo método de detección de ataques basado en el empotrado de marcas de agua en el agente móvil mediante el uso de técnicas de watermarking de software.
- En el Capítulo 5, “Autoridad de Revocación de Hosts (HoRA)”, se incorpora una nueva entidad al sistema de agentes móviles con capacidad para castigar los hosts maliciosos, la Autoridad de Revocación de Hosts.
- En el Capítulo 6, “Conclusiones y Líneas Futuras”, se encuentran las conclusiones y líneas futuras de esta Tesis.

1.4. Introducción a los Sistemas Distribuidos

En los inicios de la computación moderna, aquellas instituciones o compañías que querían acceder a las nuevas tecnologías se veían limitadas a la única existente, los supercomputadores. Estos dispositivos eran caros, grandes y altamente sofisticados. De hecho, disponer de uno era prácticamente un lujo difícilmente alcanzable, no sólo por el coste de adquisición del dispositivo, sino porque el personal que debía utilizarlos debía tener un alto grado de preparación. El modo de funcionamiento era sencillo, los usuarios lanzaban procesos desde terminales sin inteligencia, y éstos

Capítulo 1. Introducción

se ejecutaban en el supercomputador. Por lo que respecta a las interacciones entre supercomputadores, la mayoría de ellos se veían obligados a trabajar de forma independiente sin llegar a compartir ningún tipo de recurso o información. Esto, en gran medida, se debía al escaso número de supercomputadores y a la disgregación geográfica de los mismos, que hacía casi imposible su interconexión mediante las redes existentes.

Varios factores propiciaron un cambio de dicha situación inicial. En primer lugar, la aparición de nuevos procesadores mucho más económicos permitió la fabricación de computadoras de bajo coste, que penetraron ampliamente en el mercado. Por otra parte, la aparición de las Redes de Área Local (LAN) hicieron posible la interconexión de estas computadoras, dando lugar a sistemas compuestos por un número grande de computadoras conectadas mediante redes de alta velocidad. En contraposición con los sistemas centralizados iniciales, a éstos se los conoce como sistemas distribuidos.

Podemos definir un sistema distribuido como un conjunto de computadoras independientes que, desde el punto de vista del usuario, parecen un sistema coherente y único [TvS02]. El principal objetivo de un sistema distribuido es permitir a los usuarios un acceso fácil y transparente a los diversos recursos de la red, sin importar la ubicación de los mismos. Con ello además se pretende que dichos usuarios compartan la información y los recursos costosos.

La mayoría de las aplicaciones existentes en los sistemas distribuidos actuales se basan en la que se conoce como arquitectura cliente-servidor. Los clientes son procesos que demandan un servicio mediante el envío de peticiones. Por su parte, los servidores son los procesos que ofrecen dicho servicio y para ello permanecen latentes esperando la llegada de las peticiones para poder responderlas.

En el resto del Capítulo se estudian las principales formas de comunicar procesos, que es uno de los factores prioritarios a la hora de diseñar aplicaciones en un sistema distribuido.

1.5. Paradigmas de Diseño en Sistemas Distribuidos

Cuando se diseña una aplicación para un sistema distribuido, una de las principales decisiones a tomar es cómo se realizan las comunicaciones entre los procesos. De dicha decisión muy probablemente dependerá el rendimiento del sistema y muchos aspectos relativos a la flexibilidad de la aplicación.

Catalogaremos los mecanismos para comunicar procesos en dos grandes subgrupos:

- Mecanismos que intercambian datos: en la transacción sólo se intercambian datos y la lógica de la aplicación permanece estática en la misma plataforma de ejecución. Dentro de esta tipología encontraremos los mecanismos de traspaso de mensajes y la Invocación de Procedimientos Remotos (*Remote Procedure Call* o RPC).
- Mecanismos que intercambian código: las entidades no se limitan a intercambiar datos, sino que la lógica de la aplicación puede migrar para cambiar de plataforma de ejecución. A este paradigma de funcionamiento también se le conoce como código móvil, y los mecanismos que se pueden englobar dentro de esta tipología son el código bajo demanda (*Code On Demand* o COD), la evaluación remota (*Remote Evaluation* o REV) y los agentes móviles (*Mobile Agent* o MA).

1.5.1. Traspaso de Mensajes

La forma más extendida de comunicar procesos se basa en el traspaso o intercambio de mensajes. Cuando un proceso origen tiene que mandar una información a un proceso destino lo único que debe hacer es crear un mensaje estructurado con un determinado patrón. Dicho mensaje se hará llegar al proceso destino usando los mecanismos convencionales. Por su parte, el proceso destino interpretará dicho mensaje usando el mismo patrón y extraerá la información del mensaje. La respuesta, en caso que fuera necesaria, se realizará del mismo modo. Este tipo de mecanismo es

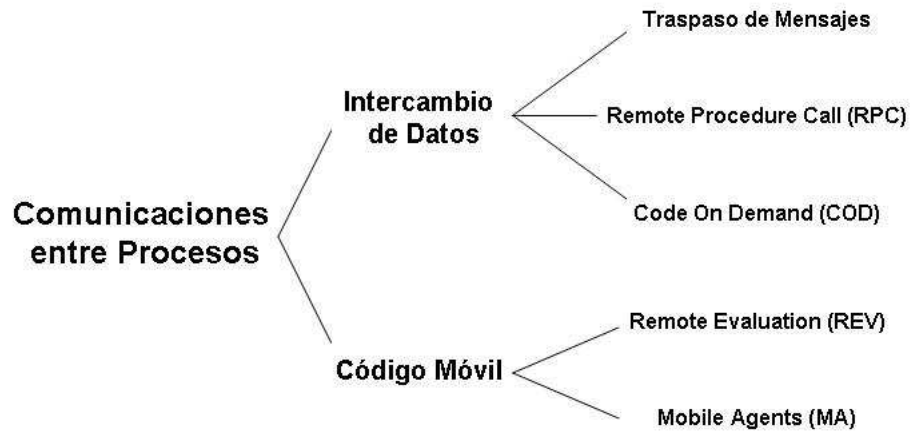


Figura 1.1: Paradigmas de Diseño en Aplicaciones Distribuidas

por naturaleza asíncrono, de manera que el proceso origen puede continuar su ejecución normalmente. En la Figura 1.2 se muestra el intercambio de mensajes entre dos procesos.

El traspaso de mensajes, pese a ser uno de los mecanismos más fáciles de implementar, plantea una serie de problemas que deben ser resueltos. Por lo que respecta a la forma de realizar la comunicación, el sistema debe estar preparado ante una posible corrupción o pérdida de mensajes, con la consiguiente retransmisión de los mismos. En cuanto a la información que se intercambia, un uso abusivo de mensajes o la transferencia de un volumen de datos muy grande puede hacer desaconsejable el uso de traspaso de mensajes debido al considerable aumento del ancho de banda requerido.

1.5. Paradigmas de Diseño en Sistemas Distribuidos

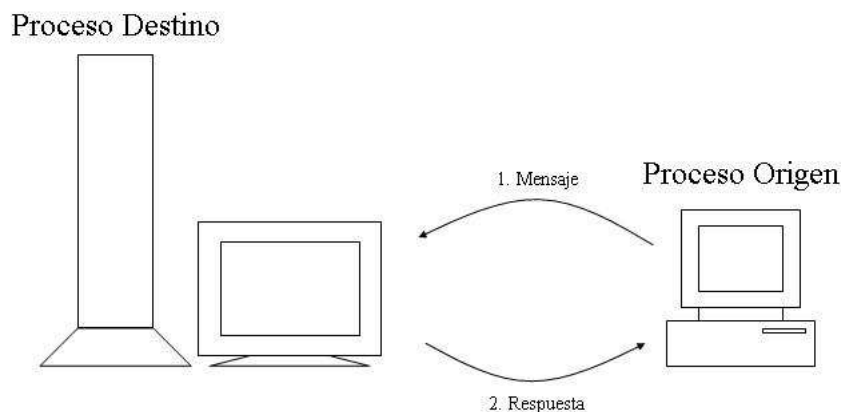


Figura 1.2: Traspaso de Mensajes

1.5.2. Invocación de Procedimientos Remotos (RPC)

Para intercambiar datos entre procesos, como alternativa al uso de traspaso de mensajes aparece la Invocación de Procedimientos Remotos (RPC), esto es, la invocación de procedimientos que se encuentran en otras máquinas. A modo de ejemplo, el proceso A abre un canal de comunicaciones con la máquina remota para invocar al proceso B . Después de la invocación, el proceso A queda en espera y se inicia la ejecución del proceso B en la máquina remota. Los parámetros de la ejecución se pasan en la invocación inicial. Una vez finalizado el proceso B , los resultados (si los hay) se envían al origen. Este tipo de mecanismo es por naturaleza síncrono, ya que el proceso origen queda en espera hasta que finaliza la ejecución del proceso destino. Con el uso de RPC se pretende procesar la información lo más cerca posible de la fuente. En el servidor se encuentran tanto los datos que queremos procesar como la lógica de servicio. Traspasando la ejecución al servidor podemos reducir el ancho de banda necesario ya que no es necesario transportar todos los datos al cliente para

Capítulo 1. Introducción

procesarlos localmente, sino que a éste sólo le llegan los resultados procesados. En la figura 1.3 se muestra un ejemplo de intercambio de datos usando RPC.

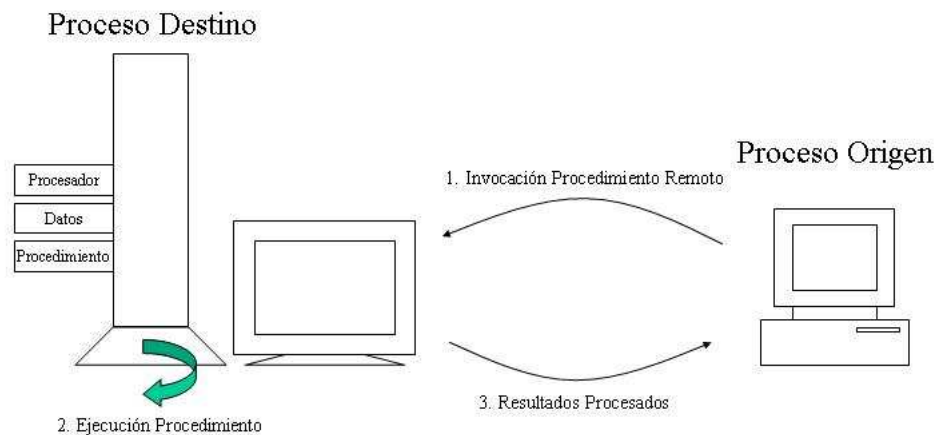


Figura 1.3: Invocación de Procedimientos Remotos

Como es lógico, el uso de RPC no está exento de problemas. Por lo que respecta a su funcionamiento, si tenemos en cuenta que el procedimiento se ejecuta en una máquina distinta, los parámetros que se intercambian los procesos deben ser totalmente compatibles. También se deben prever las acciones a tomar en caso que uno de los dos procesos implicados quedase bloqueado durante la ejecución. En cuanto a la visión del usuario que invoca el procedimiento, la comunicación se realiza a través de interfaces previamente definidos, y por tanto estáticos.

1.5.3. Código Móvil

En los dos anteriores paradigmas de diseño los procesos sólo intercambiaban datos. Sin embargo, existe la posibilidad de transferir código para aumentar la flexibilidad del sistema, incluso cuando está en ejecución (en tal caso hablaríamos de

1.5. Paradigmas de Diseño en Sistemas Distribuidos

movilidad de procesos). A este paradigma de diseño se le conoce como código móvil.

Existen varias razones por las cuales puede ser útil mover el código en lugar de los datos. La mayoría de dichas razones suelen ir ligadas al rendimiento de las máquinas o de la red. Por ejemplo, puede resultar útil cambiar la ubicación de una ejecución para ubicarla en máquinas que estén menos cargadas (en términos de CPU u otros parámetros de rendimiento). Asimismo, el uso de la movilidad del código también puede reducir el ancho de banda necesario, ya que, al igual que sucedía en RPC, la idea es realizar la ejecución de los datos lo más cerca posible de la fuente. De todos modos, las razones por las cuales la movilidad del código es útil no están ni mucho menos limitadas al rendimiento de la red. También es posible conseguir ventajas que aumenten la flexibilidad o prestaciones del sistema. Por ejemplo, la movilidad de código puede facilitar al cliente la configuración de los servicios que demanda, debido a que no tiene que ceñirse a una serie de interfaces predefinidos y estáticos, ya que envía la lógica de servicio que quiere que se ejecute.

A continuación resumimos las tres principales formas de código móvil que pueden encontrarse.

Evaluación Remota (REV)

Los clientes pueden utilizar la Evaluación Remota (*Remote Evaluation* o REV) para mejorar las capacidades del servidor. Para ello los clientes envían la lógica de servicio deseada, esto es, el código que quiere que el servidor ejecute (*code pushing*) para más tarde demandar su ejecución. Una vez realizada la petición de servicio, el servidor ejecuta el código y responde con los resultados. Este modelo de funcionamiento en realidad no es más que una mejora de RPC donde el cliente puede realizar previamente una personalización del procedimiento que va a demandar. En la Figura 1.4 se muestra un ejemplo de evaluación remota.

Código Bajo Demanda (COD)

En el paradigma del código bajo demanda (*Code On Demand* o COD) el cliente demanda la descarga de un determinado código mediante el envío de una petición al

Capítulo 1. Introducción

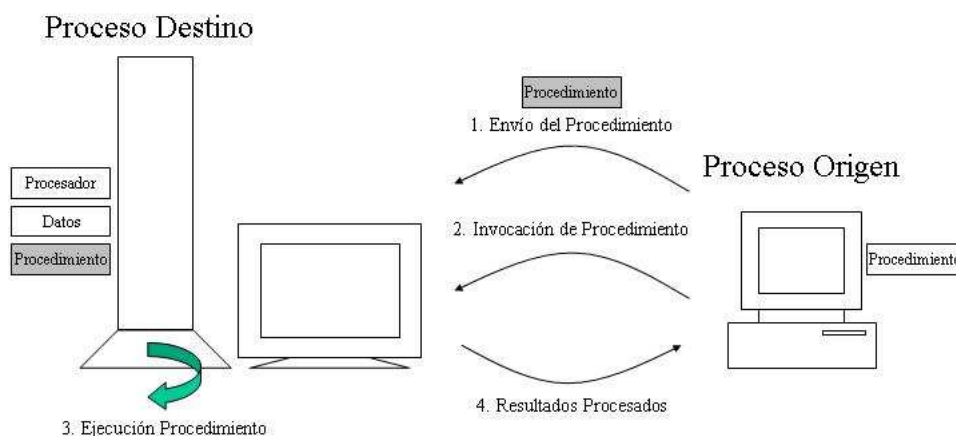


Figura 1.4: Evaluación Remota

servidor. El servidor responde a la petición con el envío del código solicitado (*code pulling*), de manera que el cliente puede ejecutarlo localmente cuando sea necesario. El servidor actúa como repositorio de software para que los clientes incrementen su capacidad de proceso. En la Figura 1.5 se muestra un ejemplo de código bajo demanda.

Agente Móvil (MA)

En sistemas distribuidos, el término agente móvil se utiliza para designar a un componente software que es capaz de migrar de una máquina a otra ejecutando su código para realizar ciertas acciones en nombre de un usuario. Existe la tendencia a confundir o fusionar dicho concepto con el de agente inteligente que proviene de los entornos de Inteligencia Artificial. Un agente inteligente es una entidad software que ejecuta un conjunto de operaciones con cierto grado de independencia o autonomía, y con ello representa los deseos y objetivos del usuario, pudiendo interactuar con el

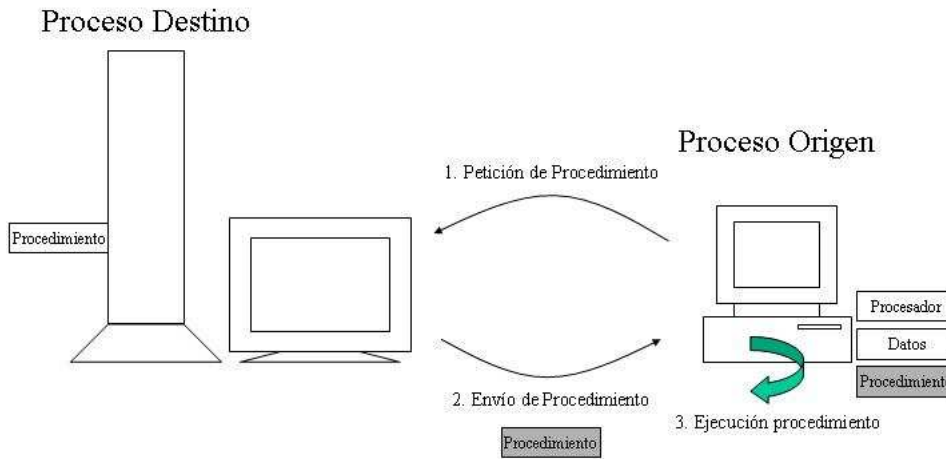


Figura 1.5: Código Bajo Demanda

entorno o incluso colaborar con otros agentes o usuarios, posiblemente remotos. Según dicha definición, se asume que el agente dispone de inteligencia e incluso capacidad para reaccionar con el entorno y recopilar conocimientos a partir de las relaciones que mantenga con otros elementos o de las acciones en las que tome parte. Por tanto, por lo que respecta a los agentes inteligentes la movilidad es una propiedad opcional [FPV98].

En el desarrollo de esta Tesis realizaremos el estudio de los problemas de seguridad de los agentes móviles en sistemas distribuidos, ya que dichos problemas se dan por la movilidad del código, y no por la ausencia o presencia de inteligencia.

1.6. Agentes Móviles

Partiendo de la definición de los sistemas distribuidos, consideramos que un agente móvil (o simplemente agente a partir de ahora) es una entidad software con capacidad para migrar de host en host ejecutando el código que transporta para realizar

Capítulo 1. Introducción

ciertas acciones en nombre de un usuario. Los agentes están compuestos básicamente por código, datos, itinerario y estado. En el código se encuentran programadas las acciones que debe realizar el agente, pudiéndose personalizar la ejecución con los datos. El itinerario del agente dictamina qué hosts ejecutarán el agente y en qué orden. Dicho itinerario puede estar predeterminado por el usuario, o bien decidirse durante la ejecución del agente. Por su parte, el estado indica en qué punto se encuentra la ejecución. Atendiendo a cómo se comporta el agente respecto al estado, se pueden definir dos tipos básicos de movilidad de código [FPV98]:

- Movilidad débil: el agente migra de máquina en máquina, pero no se envía ninguna información sobre el estado de la ejecución en las anteriores máquinas. Por tanto, el código del agente siempre se ejecuta desde el mismo punto o estado inicial.
- Movilidad fuerte: el agente migra de máquina en máquina, pero en este caso además del código se envía el estado de la ejecución. El proceso mientras migra va cambiando de estado, de manera que la ejecución en una determinada máquina puede continuar en el punto donde se quedó la máquina anterior.

La movilidad débil es más fácil de implementar debido a que solamente se requiere que el código sea ejecutable en todas las máquinas, y esto es factible si se utilizan lenguajes de tipo portable, como por ejemplo Java. En cambio, la movilidad fuerte, pese a ser más flexible, presenta mayores dificultades de implementación. Esto se debe a que el estado del agente puede plantear problemas a la hora de migrar a otra máquina, ya que puede depender o hacer referencia a recursos que se estaban utilizando localmente, como por ejemplo ficheros o puertos de una comunicación, y que pueden no estar disponibles en la máquina a la cual va a migrar el agente.

En la Figura 1.6 se puede apreciar un ejemplo de funcionamiento de un agente móvil. En el ejemplo, un usuario desea realizar ciertas acciones en otras máquinas de la red, y para ello crea un agente cuyo código tendrá programadas dichas acciones. El usuario se encuentra ubicado en el que llamaremos host origen, esto es, la máquina que envía el agente en primera instancia. El agente debe llevar consigo todo lo que

necesita para poder ejecutarse en otras máquinas. Esto implica variables, información del usuario, clases que utiliza en su ejecución, en definitiva, todo lo necesario para que el agente sea autónomo una vez fuera del host origen. Para ello se realiza una operación llamada serialización, que permite al agente convertir todos esos datos en un vector de bytes que puedan ser enviados por la red. Cuando el agente llega a un host, éste debe deserializarse antes de ejecutarse. Durante la ejecución el agente puede comunicarse con otros agentes o entidades externas, probablemente mediante traspaso de mensajes [FGS96b], e incluso clonarse y migrar a otras máquinas. Finalmente, si hubiera resultados de la ejecución, el agente puede enviarlos en un mensaje o migrar él mismo al host origen para finalizar la ejecución.

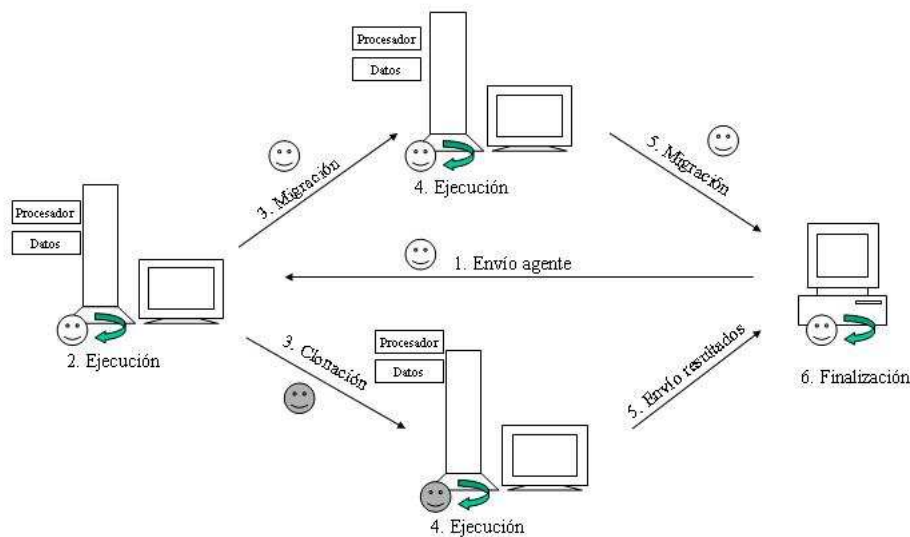


Figura 1.6: Agente Móviles

1.6.1. Aplicaciones de los Agentes Móviles

La flexibilidad, autonomía y bajo coste de los agentes móviles los hacen especialmente atractivos para muchos servicios que requieren un alto grado de automa-

Capítulo 1. Introducción

tización. Sin embargo, debe quedar claro que ninguna de las aplicaciones que vamos a enumerar son exclusivamente realizables mediante el uso de agentes móviles. Dichas aplicaciones o bien están en funcionamiento en la actualidad o bien podrían haberse realizado con otro tipo de esquemas como traspaso de mensajes o RPC. Sin embargo, el uso de agentes puede facilitar o mejorar alguno o varios aspectos del servicio.

Antes de utilizar agentes en una determinada aplicación, debe estudiarse de qué forma se va a realizar la migración para evaluar la que más nos interesa. De dicha decisión pueden depender aspectos muy relevantes como el rendimiento de la red o la facilidad de uso del usuario. En particular, podemos diferenciar dos tipos de agentes atendiendo al número de saltos que realizan [Ord96]:

- *Single-hop agent*: el agente sólo realiza una migración hasta el host destino, de manera que vuelve directamente al host origen una vez ha realizado las tareas que tenía asignadas. Este tipo de agentes son especialmente útiles para realizar tareas que requieren computar un gran volumen de datos o que precisan mucho tiempo de ejecución. De esta manera, para reducir el tiempo total de cómputo es más rentable enviar de forma paralela un agente que forzarlo a migrar de forma secuencial de host en host.
- *Multiple-hop agent*: el agente migra de host en host ejecutándose en cada una de ellos. Por tanto, realiza varios saltos antes de volver de nuevo al host origen. Si el sistema de agentes tiene capacidad para movilidad débil, este tipo de agentes son especialmente útiles para realizar tareas simples y repetitivas en múltiples máquinas. Si el sistema de agentes tiene capacidad para movilidad fuerte, este tipo de agentes pueden resolver problemas más complejos utilizando el estado de la ejecución.

Teniendo en cuenta esta tipificación, nos encontramos una serie de servicios en los cuales parece natural el empleo de agentes móviles [ST97, FPV98, JK99]:

- **Diseminación de información**: es posible utilizar las propiedades de movilidad y autonomía de un agente para transportar información y hacerla llegar a un conjunto de máquinas determinadas.

1.6. Agentes Móviles

- Búsquedas exhaustivas por la red: el agente puede realizar búsquedas por un conjunto de servidores siguiendo unos determinados criterios programados por el usuario. Las fuentes de la información pueden definirse de forma estática por el usuario, o bien determinarse dinámicamente durante la ejecución del agente.
- Control de equipos remotos: un usuario puede enviar un agente a un dispositivo remoto para que tome el control del mismo o lo configure. Otras tareas relevantes que puede realizar el agente son la monitorización del estado del dispositivo, la notificación de eventos, la toma de estadísticas o el control de alarmas. De hecho, una de las principales líneas de investigación en sistemas de agentes móviles tiene como objetivo la gestión de redes.
- Redes activas: muy en consonancia con el anterior punto, los agentes móviles son la solución natural para las comunicaciones en redes activas, esto es, redes en las cuales los nodos son programables mediante el envío de programas.
- Procesado paralelo y distribuido: los agentes móviles también son útiles para repartir el procesado de problemas complejos entre distintas máquinas. Los agentes móviles tienen la capacidad de migrar e incluso clonarse, de manera de dicha computación puede realizarse tanto de forma distribuida como paralela.
- *E-commerce*: los agentes realizan acciones en nombre de un usuario, con lo cual pueden ser utilizados para realizar de forma remota una transacción económica, como por ejemplo la compra o venta de un artículo, la negociación de un precio, e incluso la firma de un contrato.

Obviamente, puede ser desaconsejable la ejecución o el envío de agentes desde terminales que tengan algún tipo de limitación. No todos los usuarios finales tendrán ni la capacidad computacional ni los conocimientos para realizar el envío o la ejecución de un agente. En tal caso la solución pasa por delegar dicha tarea a un servidor especializado. En la actualidad es descabellado pensar que un usuario ubicado en una PDA o en un teléfono móvil tenga los conocimientos o las herramientas necesarias para programar un agente y enviarlo a la red. En cambio, es posible incorporar a la

Capítulo 1. Introducción

red repositorios desde donde los terminales pueden descargarse agentes preprogramados. Por otra parte, también es dudoso que el usuario dictamine de forma directa el itinerario del agente. Por norma general, dicha decisión vendrá determinada por una consulta a un servicio de directorio o páginas amarillas [CGH⁺97].

1.6.2. Ventajas de los Agentes Móviles

Existen múltiples razones por las cuales puede resultar útil enviar un código a una máquina remota. La mayoría de ellas suelen ir ligadas al rendimiento de las máquinas o de la red, sin embargo, otras tienen mucho que ver con el incremento de la flexibilidad de las aplicaciones. Estas son algunas de las ventajas que puede representar el uso de agentes móviles [CHK95, CGH⁺97, FPV98]:

- Distribución de la carga de procesos: si una máquina debe realizar tareas computacionalmente complicadas o que representen un tiempo de ejecución grande, puede resultar útil migrar parte de dichos procesos a máquinas que estén menos cargadas en términos de CPU mediante el envío de un agente.
- Autonomía: los agentes móviles pueden programarse para tomar decisiones en nombre de un usuario, con lo cual puede llevar a cabo cualquier trámite en nombre del mismo de forma autónoma, esto es, sin intervención del usuario emisor.
- Reducción de la latencia: los agentes móviles actúan de forma autónoma sin necesidad de interactuar con el usuario que los creó. Es por ello que el agente puede reducir la latencia si da respuestas en tiempo real a tareas que hubieran requerido de la intervención del usuario.
- Ahorro de ancho de banda: enviar la lógica del servicio lo más cerca de la fuente de la información por lo general supone un ahorro de ancho de banda. Como el agente se ejecuta directamente donde se encuentran los datos, sólo es necesario enviar los resultados. Se ahorra ancho de banda siempre y cuando el tamaño de la información intercambiada entre las máquinas (el agente y los resultados)

sea menor que la información que se debería haber enviado para procesarla en el cliente.

- Soporte para terminales móviles: los terminales móviles (teléfonos móviles, PDAs, ordenadores portátiles) por lo general se consideran computacionalmente poco potentes si los comparamos con dispositivos fijos equivalentes, ya que suelen disponer de capacidades de CPU, memoria, almacenamiento o batería inferiores. Asimismo, la naturaleza de los terminales móviles hace que dependan de enlaces inalámbricos con anchos de banda limitados y alta latencia. Es por ello que permanecen desconectados la mayoría del tiempo, bien sea porque el coste de la conexión es económicamente alto, bien sea por falta de cobertura de la red, o bien sea para ahorrar energía. Con el uso de agentes móviles en este tipo de terminales se podrían descargar las tareas computacionalmente costosas en otra máquina más potente. Por otra parte, el uso de agentes puede liberar preciados recursos de la red, ya que el terminal sólo envía el agente para realizar las tareas, permaneciendo desconectado hasta que quiera recuperar los resultados.
- Personalización de servicios: por lo general la mayoría de los servicios cuentan con interfaces estáticos. Es por ello que cada vez que un usuario quiere agregar nuevas funcionalidades se deben realizar cambios sobre los interfaces del servidor. En cambio, mediante el envío de un agente es posible personalizar la lógica de servicio sin tener que depender de unos interfaces estáticos.
- Adaptación a entornos heterogéneos: por naturaleza los agentes móviles deben adaptarse a entornos diversos pues deben migrar de máquina en máquina. Esa es la razón por la cual la mayoría de sistemas de agentes móviles usan lenguajes interpretados como Java. Este tipo de lenguajes, pese a penalizar el rendimiento del sistema, no sólo permiten una interacción transparente con la plataforma sino que además ofrecen la posibilidad de controlar los recursos a los que acceden los agentes móviles [CHK95].

Capítulo 1. Introducción

- Tolerancia a fallos: es posible programar los agentes móviles para tomar decisiones si hay un fallo en alguna de las plataformas o en la red, pudiendo buscar caminos alternativos o finalizar su ejecución volviendo al origen. En caso de perder un agente, es posible recuperar el hilo de la ejecución si se conserva el estado en que permanecía, enviándose de nuevo desde el punto en que se dejó. Incluso es posible mandar en paralelo el mismo agente a varias máquinas para redundar la ejecución del mismo.

En realidad, ninguna de las anteriores ventajas son exclusivas de los agentes móviles. Sin embargo, lo que hace especialmente interesante el uso de agentes móviles es que es el único paradigma de diseño que las presenta todas.

1.6.3. Inconvenientes de los Agentes Móviles

Los agentes móviles constituyen una herramienta de gran interés para la mayoría de aplicaciones que requieren un alto grado de automatización. Lamentablemente, siempre que se realiza el envío de un código ejecutable a una máquina remota aparecen una serie de inconvenientes que deben tenerse en cuenta. El primer inconveniente que surge al transferir un código o proceso en ejecución a una máquina remota es garantizar que la máquina remota, su sistema operativo o cualquier otro recurso implicado en la ejecución es compatible con dicho código. Afortunadamente, los problemas relativos a conflictos software y hardware se resuelven mediante el uso de lenguajes interpretados como Java.

Sin embargo, los problemas derivados de la seguridad representan un serio inconveniente que impide un uso masivo de los sistemas de agentes móviles [JK99, Jan00]. Es por ello que en el desarrollo de esta Tesis nos centraremos en este tipo de problemas, que no son de tan fácil solución como el anterior. Más particularmente, centraremos nuestros esfuerzos en el que muchos autores han considerado el problema más difícil de resolver respecto a la seguridad de sistemas de agentes móviles, el problema de los hosts maliciosos [FGS96b].

Seguridad en Sistemas de Agentes Móviles

Como hemos comentado, el uso masivo de agentes móviles se ha visto limitado por serios problemas relativos a la seguridad. Afortunadamente, no todas las aplicaciones que pueden beneficiarse del uso de agentes móviles requieren seguridad y por tanto pueden ser utilizados libremente. Sin embargo, otras aplicaciones sí requieren altos niveles de seguridad, como por ejemplo servicios relativos a e-commerce. Para dichas aplicaciones se deben buscar los mecanismos adecuados para proteger todas las entidades que intervengan en las transacciones.

Los sistemas de agentes tienen que encarar los problemas de seguridad habituales de cualquier sistema distribuido, y además deben solucionar otros problemas nuevos relacionados con la movilidad del código. Para el análisis de las amenazas consideraremos que hay dos entidades principales: la plataforma de ejecución (o host) y el agente móvil. Atendiendo a esta tipificación, estos son los principales problemas que nos encontramos en cuanto a seguridad del sistema de agentes móviles [Che98]:

Protección del Host Los administradores de sistemas quieren proteger sus máquinas de posibles ataques que pretendan aprovecharse de las debilidades de la plataforma de ejecución. No estudiaremos los problemas debidos a ataques perpetrados por elementos externos (como por ejemplo otros hosts), ya que este tipo de ataques pueden aparecer en cualquier tipo de plataforma independientemente de si utiliza agentes móviles o no. Los ataques que centrarán nuestro interés son los realizados por los agentes que llegan al host para ejecutarse. Si tenemos en cuenta que al host pueden llegar agentes provenientes de orígenes muy diversos y que las relaciones de confianza pueden no existir, nadie nos asegura que dicho agente tenga buenas intenciones. Los principales ataques que puede realizar un agente son [JK99]:

- Suplantación: un agente malicioso se hace pasar por otro agente para usurpar sus recursos o simplemente dañar la reputación del usuario que envió el agente suplantado o al mismo host.
- Negación de Servicio: el agente consume todos los recursos disponibles del

Capítulo 1. Introducción

sistema para impedir realizar tareas de otro tipo o incluso forzar a la plataforma a quedar fuera de servicio. Este tipo de ataque puede ser intencionado o deberse a errores en la programación del agente.

- Acceso no autorizado: un agente que ha sido autenticado correctamente y accede a la plataforma intenta acceder a recursos para los cuales no tiene permisos. Un ejemplo podría ser la lectura o modificación de información que se encuentre en espacios de memoria reservados o discos duros.

A pesar de tener mecanismos de autenticación que verifiquen el origen del agente, no es posible asegurar que el agente no oculte un virus o "Caballo de Troya". La mayoría de ataques contra la plataforma son evitables o detectables con técnicas comunes de seguridad como son la limitación del entorno de ejecución de los agentes (acceso a memoria y operaciones restringidas, límites de consumo de recursos) con mecanismos del tipo *sandboxing* como los utilizados en Java, así como un control de acceso adecuado (comprobación de credenciales y establecimiento de privilegios), que puede venir determinado por el nivel de confianza del agente [KSB02]. Otro tipo de propuestas se basan en la incorporación de pruebas que determinan si el código es malicioso o no. En ese sentido destacamos las siguientes dos propuestas. En [FGS96a] se introduce la noción de evaluación del estado (*State Appraisal*). El agente además del código transporta una función de evaluación del estado que verifica si el estado del agente es peligroso o no para la plataforma. Esta misma función de evaluación será la encargada de realizar la petición de permisos a la plataforma a la hora de realizar el control de acceso. La plataforma dará acceso a determinados recursos en función del estado en que se encuentre el agente y su política de seguridad. El problema que implica esta solución es evaluar qué estados de un agente son peligrosos y cuales no. En [NL98], los autores introducen la idea de *Proof-Carrying Codes*, esto es, unas pruebas que demuestran que el código no es malicioso y que el creador del código anexa en el agente. Cuando el agente llega a un host ejecuta una función de comprobación que genera un predicado de seguridad que, comparándolo con las pruebas enviadas por el agente, determina que las pruebas corresponden al código.

1.7. Terminología

Protección de las Comunicaciones Un agente mientras migra de host a host puede recibir ataques de cualquier entidad externa. Estos ataques pueden solucionarse con técnicas criptográficas bien conocidas como son el cifrado de la información y la firma digital. Sin embargo, este tipo de medidas no impiden los ataques, si bien facilitan la posible búsqueda de responsabilidades. En mayor o menor medida, hoy en día el uso de dichas técnicas criptográficas están supeditadas a la existencia de una Infraestructura de Clave Pública (*Public Key Infrastructure* o PKI). De hecho, supondremos en todo momento que todos los hosts del sistema de agentes móviles disponen de un Certificado de Identidad (*Identity Certificate* o IC) que ligue la identidad del host con su clave pública. Gracias a dichos certificados es posible verificar las firmas digitales y por tanto evitar suplantaciones o ataques de repudio.

Protección del Agente Los ataques que puede recibir un agente en una plataforma de ejecución o bien provienen de la misma plataforma, o bien de otro agente. La protección de un agente de los ataques de otros agentes maliciosos tiene solución con la separación de dominios de ejecución para evitar la interacción directa de los mismos. Sin embargo no existe una solución conocida que dé protección global al agente frente a los ataques realizados por la plataforma. Este tipo de ataques son conocidos como el problema de los hosts maliciosos, y son con diferencia los más complicados de resolver respecto a seguridad en sistemas de agentes móviles [FGS96b, Rot99]. En el Capítulo 2 se detallan los principales ataques que un host malicioso puede realizar sobre un agente móvil, así como las principales propuestas de protección del agente publicadas hasta el momento.

1.7. Terminología

Para facilitar la comprensión del documento, identificaremos algunos términos usuales que aparecerán repetidas veces en el transcurso de la Tesis. En primer lugar, un agente móvil (o simplemente agente) es una entidad software con capacidad para migrar autónomamente de host en host ejecutando unas ciertas tareas en nombre de

Capítulo 1. Introducción

un usuario. Otras capacidades opcionales que pueda tener el agente móvil, como la capacidad de reacción con el entorno o el aprendizaje dependerán de la inteligencia que el usuario haya incorporado en el código del agente.

Debemos distinguir también entre los hosts o plataformas de ejecución en las que puede encontrarse el agente. Entendemos por host origen a la plataforma que envía el agente en primera instancia, y por lo general, asumimos que es la máquina del usuario en nombre del cual se realizan las tareas. Dicho host origen deberá tener todas las capacidades software y hardware necesarias para crear, enviar y recibir un agente móvil. Al resto de máquinas con capacidad para ejecutar al agente las denotaremos indistintamente como hosts o plataformas de ejecución. Dichos hosts deberán disponer de las capacidades software y hardware necesarias para recibir, ejecutar y enviar los agentes móviles, así como para permitir que el agente se comuniqué con otras entidades.

Incorporamos aquí también la notación que se utilizará para identificar el traspaso de mensajes o agentes en los distintos protocolos de la Tesis, así como la notación relativa a las técnicas criptográficas:

- Denotamos como $Agent_{x \rightarrow y}()$ al agente móvil que va desde el host x hacia el host y .
- Denotamos como $Message_{x \rightarrow y}()$ al mensaje que va desde el host x hacia el host y .
- Denotamos como $H(D)$ al valor de la función de hash (*One Way Hash Function* o OWHF) del documento D .
- Denotamos como $sign_{\alpha}[D]$ al documento D firmado por la entidad α .
- Denotamos como $S_{K_s}[D]$ al documento D cifrado mediante algoritmo simétrico y K_s como clave de sesión.
- Denotamos como $S_{K_s}^{-1}[D]$ al documento D descifrado mediante algoritmo simétrico y K_s como clave de sesión.

1.8. Contribuciones de esta Tesis

- Denotamos como $E_{K_{priv}}[D]$ al documento D cifrado mediante algoritmo de clave pública y K_{priv} como clave privada.
- Denotamos como $D_{K_{pub}}[D]$ al documento D descifrado mediante algoritmo de clave pública y K_{pub} como clave pública.

1.8. Contribuciones de esta Tesis

En el transcurso de esta Tesis revisaremos las principales propuestas de protección de agentes publicadas hasta el momento, teniendo como objetivo tratar de introducir alguna mejora en alguna de ellas. Desde nuestro punto de vista, los esquemas de prevención de ataques o bien no dan la protección adecuada, o bien son económica o computacionalmente imposibles de implementar. Por esa razón intentaremos proteger a los agentes basándonos en esquemas de detección de ataques. Obviamente, los mecanismos de detección de ataques son poco útiles si una vez detectado el ataque no puede imponerse un castigo al host malicioso. Como no es posible imponer sanciones si no existen pruebas del comportamiento malicioso del host, los esquemas de detección que utilicemos además deberán aportar pruebas del carácter malicioso del host.

Como primera contribución de esta Tesis se propone realizar mejoras sobre una de las propuesta de detección de ataques existentes, la propuesta de las trazas criptográficas de Vigna [Vig98]. Desde nuestro punto de vista, ésta es la propuesta de detección más conocida y la que menos dificultades presenta a la hora de implementarse. Con el uso de las trazas no sólo se pueden detectar alteraciones de la ejecución del código, sino que además se prueba el carácter malicioso del host. Desgraciadamente, la propuesta tiene algunos inconvenientes que limitan su aplicación. En primer lugar, la verificación de las trazas se realiza sólo en caso de sospecha de los hosts, pero en ningún punto se detalla cómo un host se convierte en sospechoso. En segundo lugar, los hosts deben almacenar las trazas durante un periodo de tiempo indeterminado ya que el host origen puede solicitarlas para verificar la ejecución. Nuestra contribución se basa en la resolución de ambos inconvenientes mediante el

Capítulo 1. Introducción

uso de un Protocolo de Detección de Sospechosos (*Suspicious Detection Protocol* o SDP). El protocolo se basa en limitar el tiempo que dispone un host para ejecutar un agente. A la vuelta del agente, el host origen verifica si hay algún tipo de incoherencia temporal en la ejecución o transmisión del agente. Si finalmente se detecta algún tipo de incoherencia temporal en los tiempos de ejecución o transmisión de algún host, éste se convierte en sospechoso y se le pueden solicitar las trazas de ejecución. El mecanismo es sencillo y soluciona los inconvenientes que presentaba la propuesta de las trazas criptográficas, ya que disponemos de un mecanismo para determinar qué hosts son sospechosos justo cuando la ejecución del agente finaliza, con lo cual las trazas se pueden solicitar directamente y no se deben almacenar durante un periodo de tiempo indeterminado. SDP ha sido implementado y probado para verificar su aplicabilidad en sistemas de agentes móviles con resultados positivos.

Como segunda contribución de esta Tesis se presenta un mecanismo propio de detección de ataques, ya que el de las trazas criptográficas de Vigna consideramos que es todavía demasiado costoso. La nueva propuesta está basada en el empotrado de marcas digitales dentro del agente mediante el uso de técnicas de watermarking de software. El host origen modifica el código del agente para empotrarle una marca digital, de manera que durante la ejecución la marca se traspassa a los resultados. Obviamente, la marca debe aparentar que forma parte de los resultados reales, esto es, un observador debe tenerlo difícil para distinguir qué partes de los resultados son marca y cuales no. A la vuelta del agente, el host origen verifica si la marca esperada se encuentra en los resultados. El host puede considerarse honesto si la marca permanece inalterada, mientras que si la marca ha sido modificada el host es malicioso. La propuesta del watermarking de agentes móviles mejora varios aspectos relevantes si la comparamos con la de las trazas. En primer lugar, las pruebas que certifican el comportamiento de un host se transportan con el agente, con lo cual ya no es necesario actuar en base a sospechas y es posible verificar la integridad de la ejecución de todos los hosts. En segundo lugar, el proceso de verificación es más sencillo y no requiere de la reejecución del agente. Finalmente, el nuevo mecanismo evita la necesidad del almacenamiento de las pruebas en los hosts.

1.8. Contribuciones de esta Tesis

Como última contribución de esta Tesis se presenta una nueva entidad en el sistema de agentes móviles con capacidad para castigar hosts maliciosos. Cualquier mecanismo de detección de ataques es del todo ineficaz si no se utiliza conjuntamente con algún mecanismo de castigo. Para ello introduciremos una Autoridad de Revocación de Hosts (*Host Revocation Authority* o HoRA). La HoRA debe considerarse como una Tercera Parte de Confianza (*Trusted Third Party* o TTP) en un sistema de agentes móviles, de la misma manera que se considera una entidad de confianza a la autoridad de certificación en la PKI. La HoRA controla qué hosts han sido revocados en el sistema de agentes móviles, esto es, los hosts que se ha probado que actuaron maliciosamente. Antes de enviar un agente, el host origen debe consultar el estado de los hosts del itinerario para verificar si están revocados o no. Los hosts revocados se eliminarán del itinerario, de manera que el castigo que se impone a los host malicioso es impedir que vuelvan a ejecutar ningún otro agente. Dicha plataforma está en fase de implementación y se están realizando pruebas de su uso conjunto con el mecanismo de las trazas criptográficas y SDP.

Finalmente, a continuación se indican las publicaciones conseguidas relacionadas con el desarrollo de esta Tesis:

1.8.1. Revistas y Publicaciones JCR

- M. Soriano, O. Esparza, M. Fernández, J. Forné, J. Muñoz and J. Pegueroles "*Secure Electronic Brokerage Mechanisms for Mobile Electronic Commerce*". Aceptado en *Electronic Commerce Research Journal*.
- O. Esparza, M. Soriano, J. Muñoz, and J. Forné, "*Implementation and Performance Evaluation of a Protocol for Detecting Suspicious Hosts*", en *Mobile Agents for Telecommunication Applications (MATA 2003)*, Volúmen 2881 de *Lecture Notes of Computer Science*, Springer-Verlag, 2003.
- O. Esparza, M. Soriano, J. Muñoz, and J. Forné, "*Protocols for Malicious Host Revocation*", en *International Conference on Information and Communications Security (ICICS 2003)*, Volúmen 2836 de *Lecture Notes of Computer Science*,

Capítulo 1. Introducción

Springer-Verlag, 2003.

- O. Esparza, M. Fernández, M. Soriano, J. Muñoz, and J. Forné, “*Mobile agent watermarking and fingerprinting: tracing malicious hosts*”, en Conference on Database and Expert Systems Applications (DEXA 2003), Volúmen 2736 de Lecture Notes of Computer Science, Springer-Verlag, 2003.
- O. Esparza, M. Soriano, J. Muñoz, and J. Forné, “*Host Revocation Authority: a Way of Protecting Mobile Agents from Malicious Hosts*”, en International Conference on Web Engineering (ICWE 2003), Volúmen 2722 de Lecture Notes of Computer Science, Springer-Verlag, 2003.

1.8.2. Congresos Internacionales

- O. Esparza, M. Fernandez, M. Soriano, “*Protecting mobile agents by using traceability techniques*”, en International Conference on Information Technology: Research and Education (ITRE 2003), IEEE Communications Society, 2003.
- O. Esparza, M. Soriano, J. Muñoz, and J. Forné, “*A protocol for detecting malicious hosts based on limiting the execution time of mobile agents*”, en The Eighth IEEE Symposium on Computers and Communications (ISCC'2003), IEEE Computer and Communications Society, 2003.
- O. Esparza, M. Soriano, J. Muñoz, and J. Forné, “*Limiting the execution time in a host: a way of protecting mobile agents*”, en IEEE Sarnoff Symposium Advances in Wired and Wireless Communications, 2003.

1.8.3. Congresos Nacionales

- O. Esparza, M. Soriano “*Políticas de revocación para la protección de agentes móviles*”, en IV Jornadas de Ingeniería Telemática, JITEL'03, 2003.

1.8. Contribuciones de esta Tesis

- O. Esparza, “*Un protocolo de detección de hosts sospechosos basado en limitar el tiempo de ejecución en los hosts*”, en Simposio de Comercio Electrónico, SCE’03, 2003.
- O. Esparza, M. Soriano, “*Análisis de la seguridad en agentes móviles*”, en XII Jornadas Telecom I+D, November 2002.

Capítulo 2

Estado del Arte

2.1. Introducción al Problema de los Hosts Maliciosos

Tal como se comentó en el Capítulo 1, el problema de los hosts maliciosos es considerado por muchos autores el más difícil de resolver respecto a la seguridad en sistemas de agentes móviles [FGS96b, JK99, Rot99]. Si tenemos en cuenta que los agentes pueden ejecutarse en hosts con muy diversos grados de confianza, sería ingenuo no esperar un comportamiento malicioso por su parte. Los hosts pueden intentar sacar provecho del agente modificando el código, los datos, el modo de ejecución, el estado, las comunicaciones, el itinerario o incluso los resultados, ya que tienen control total sobre la ejecución. Por la misma razón, es imposible evitar los ataques de negación de servicio, ya que el host puede ejecutar el código de forma incorrecta, finalizarlo antes de tiempo, o simplemente no permitir la migración hacia otro host.

Para usar los mecanismos de criptografía clásica es necesario tener un entorno de confianza donde realizar las funciones de cifrado o firma, y además almacenar la clave secreta sin posibilidad de escuchas o alteraciones. Por el contrario, para un agente móvil resulta imposible almacenar en claro una clave secreta pues ésta podría ser leída e incluso modificada por el host [Che96]. Esta es la razón por la cual hasta el momento no existe una solución conocida que dé protección global al agente frente

2.1. Introducción al Problema de los Hosts Maliciosos

a los ataques realizados por un host malicioso.

2.1.1. Ataques

Existen múltiples razones por las cuales un host puede iniciar un ataque contra el agente. Los hosts maliciosos pueden realizar ataques “razonables” para obtener un beneficio económico, una ejecución favorable o dañar la reputación de otra entidad, si bien pueden deberse a otras razones más “personales”, como por ejemplo la pura diversión, la voluntad de destrucción o ansias de demostrar un buen nivel de conocimientos sobre el tema. Que proliferen más un tipo de ataques que otros dependerá en gran medida del escenario donde se realiza la ejecución. Por ejemplo, no debe esperarse el mismo tipo de ataques del servidor de una multinacional, que de un PC anónimo de Internet. Del primero podrían esperarse actitudes maliciosas que buscasen un beneficio. En cambio, del segundo también pueden esperarse actitudes más personales, como por ejemplo atacar al agente por el orgullo de haber roto la seguridad del sistema, como sucede con los *hackers* en Internet. Es por ello que antes de implantar cualquier esquema de protección se debe realizar el estudio de quienes son los hosts que potencialmente pueden atacarnos y las razones por las cuales pueden hacerlo.

En la Figura 2.1 se incluye un ejemplo de agente de búsqueda de información para ilustrar algunos de los ataques que pueden realizarse. El agente del ejemplo visitará varios servidores para buscar el vuelo más barato entre Barcelona y París. Se establece un precio inicial de 300 y un itinerario por el que debe ejecutarse el agente. De esta manera, una vez finalizada la ejecución en el último host del itinerario, el agente nos indica el mejor precio en la variable *bestprize* y el servidor que lo ofrece en la variable *bestairline*. No se ha incluido protección criptográfica a ninguna parte del agente para no añadir complicación al ejemplo. En la Figura 2.2 se aprecia de forma gráfica cómo variaría el estado del agente en cada host si todos actuaran de forma honesta.

Teniendo en cuenta el ejemplo, éstos son los principales ataques que pueden realizarse contra el agente móvil [JK99]:

Capítulo 2. Estado del Arte

DATA BLOCK

```
Address home=myPC;
City origin= Barcelona;
City destination=Paris;
Address itinerary[]={Airfrance,Iberia,Spanair};
Integer index_itinerary=0;
Integer bestprize=300;
Address bestairline=void;
```

CODE BLOCK

```
1 public void cheapestFlightAgent () {
2   if (itinerary[index_itinerary].flightprice(origin,destination)
   <bestprize) {
3     bestprize=itinerary[index_itinerary].flightprice(origin,destination);
4     bestairline=itinerary[index_itinerary];
5   }
6   if (index_itinerary >= (itinerary.lenght - 1) go(home);
7   go(itinerary[++index_itinerary]);
8 }
```

Figura 2.1: Ejemplo de un Agente Móvil de Búsqueda

- Negación de servicio: un host realiza un ataque de negación de servicio cuando impide de algún modo la ejecución del agente. En el ejemplo de la Figura 2.1 cualquiera de los servidores pueden impedir la ejecución del agente simplemente eliminándolo. Este tipo de ataques son imposibles de evitar pues el host tiene control total sobre la ejecución. Solamente es posible detectarlos y castigar al host que los realiza. También se considerarán ataques de negación de servicio a los cambios aleatorios que pueda realizar un host malicioso en el código o los datos, ya que pueden llevar al agente a comportarse de forma inesperada.
- Repudio: un host realiza un ataque de repudio cuando niega una acción que sí tuvo lugar. En el ejemplo de la Figura 2.1, el servidor de Iberia podría negar que dio un precio de 250 para el vuelo. Es posible evitar este tipo de ataques e incluso buscar responsabilidades si la información intercambiada está

2.1. Introducción al Problema de los Hosts Maliciosos

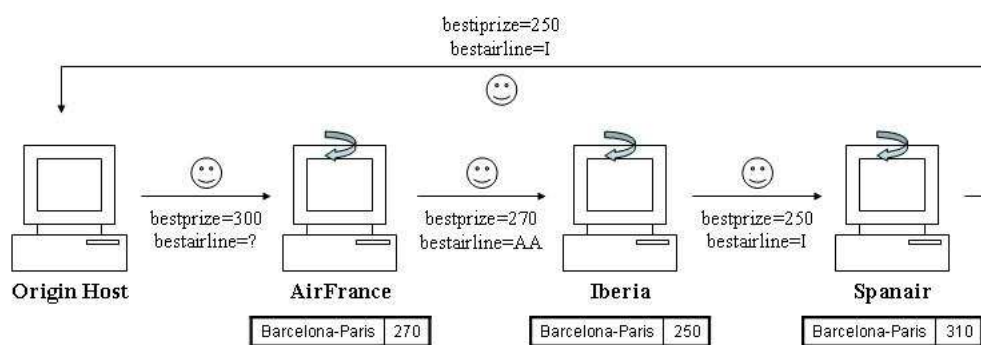


Figura 2.2: Ejecución Honesta del Agente de Búsqueda

correctamente firmada por la entidad emisora.

- Escuchas: no es posible asegurar la confidencialidad de algunas partes esenciales del agente, como son el código, los datos, las comunicaciones o los resultados, ya que los agentes los necesitan en claro para poder realizar la ejecución. Por tanto, los hosts pueden utilizar dicha información privilegiada para obtener una ejecución favorable. En el ejemplo, el último de los servidores (Spanair) puede consultar la variable *bestprize* y cambiar el precio en su base de datos a otro inferior (por ejemplo 249). Al finalizar la ejecución *bestairline* indica al servidor de Spanair y por tanto el producto es finalmente vendido a dicho servidor deshonesto. Es imposible evitar o detectar las escuchas de cualquier dato no cifrado que transporte el agente, de la misma manera que tampoco es posible evitar que el host guarde una copia de los datos cifrados para criptoanalizarlos más tarde.
- Manipulación: es posible asegurar la confidencialidad, integridad y autenticidad del código, datos o resultados que provienen de otros hosts usando técnicas

Capítulo 2. Estado del Arte

como el cifrado o la firma digital. Sin embargo, es difícil detectar o prevenir las manipulaciones realizadas en el agente durante la ejecución. El host puede manipular cualquiera de los contenidos del agente durante la ejecución. Por ejemplo, el segundo servidor (Iberia) podría poner el precio que quisiera en la variable *bestprize* (por ejemplo 300) y a él mismo en *bestairline*, para luego mandar el agente directamente al host origen. Desde nuestro punto de vista, este tipo de ataques son los más peligrosos, y es por ello que las contribuciones de esta Tesis están enfocadas a intentar resolverlos.

- Confabulación: en un sistema abierto como Internet las relaciones de confianza son limitadas. Por ello se podría suponer que es difícil que un grupo de hosts confabulen contra un agente ya que los confabuladores deben estar en el itinerario del agente, y además deben confiar entre ellos. En el ejemplo, los servidores de AirFrance y Spanair podrían confabular y mandarse entre ellos el agente, dejando fuera del itinerario al servidor de Iberia. Los ataques de este tipo son difíciles de detectar o prevenir, de hecho la mayoría de propuestas publicadas hasta el momento son vulnerables a confabulaciones. Sin embargo existen técnicas de protección del itinerario [MB03] que pueden ser utilizadas para limitar los efectos de las confabulaciones.

2.1.2. Requisitos de Seguridad

Para diseñar los mecanismos de protección adecuados debemos conocer cuales son los requisitos de seguridad que buscamos en un sistema de agentes móviles. Dado que la mayoría de técnicas de protección se basan en técnicas de cifrado y firma digital, asumiremos que disponemos de una PKI y que todos los hosts disponen de un certificado de identidad válido. A continuación se enumeran los principales requisitos de seguridad [CGH⁺97]:

- Confidencialidad: los contenidos del agente (código, datos, estado, itinerario y resultados) sólo deben ser leídos por las entidades autorizadas, protegiéndolos de posibles escuchas que puedan sufrir durante el trayecto del agente. Es posi-

2.1. Introducción al Problema de los Hosts Maliciosos

ble asegurar la confidencialidad del agente mientras transita de una máquina a otra con técnicas criptográficas comunes como el cifrado de la información. Sin embargo, no es tan sencillo encontrar mecanismos que aseguren la confidencialidad de la ejecución, esto es, que el host que ejecuta el agente no pueda extraer ningún tipo de información de los contenidos o tareas del agente.

- **Integridad:** cualquier manipulación que sufra el agente debe ser detectada. Ninguna entidad no autorizada debe poder alterar el agente o sus contenidos sin que se detecte dicha alteración. Es posible asegurar la integridad del agente mientras transita de una máquina a otra con técnicas criptográficas comunes como la firma digital. Sin embargo, no es tan sencillo asegurar la integridad de la ejecución, esto es, asegurar que los hosts no han alterado la correcta ejecución del agente.
- **Autenticación:** todas las entidades implicadas deben estar plenamente identificadas para evitar suplantaciones o ataques de repudio. Es posible asegurar la autenticidad de la información que transporta el agente si está convenientemente firmada por la entidad emisora.
- **Control del itinerario:** debe asegurarse la protección del itinerario del agente. Existen múltiples requisitos a cumplir en este ámbito, como que el itinerario que dictaminó el host origen no pueda alterarse por los hosts (integridad), que se pueda verificar que el agente proviene de un host del itinerario y no de otra entidad (verificación de fuente), o que los hosts dispongan del mínimo de información del resto del itinerario (confidencialidad), de hecho sólo deben conocer el host previo y el siguiente. Este área de investigación es también de difícil resolución y es por ello que todavía continúa abierto [DW99, BRSR99, MB01, MB03].

El resto del Capítulo describe brevemente las principales propuestas que han sido publicadas hasta el momento para dar protección al agente, haciendo especial incapié en aquellas que afectan a los ataques de manipulación. Estas propuestas pueden

catalogarse en las que pretenden prevenir los ataques antes que se produzcan, y las que intentan detectar los ataques después de la ejecución.

2.2. Propuestas de Prevención de Ataques

Las técnicas de detección de ataques no son demasiado útiles en aquellos servicios donde los beneficios obtenidos atacando al agente pueden ser mayores que la sanción que pueda imponerse. En dichos casos es aconsejable el uso de técnicas de prevención de ataques. Desafortunadamente no se ha publicado ninguna propuesta que consiga evitar satisfactoriamente los ataques. A continuación enumeramos las principales propuestas publicadas hasta el momento que intentan evitar los ataques de hosts maliciosos antes que se produzcan.

2.2.1. Modelos Basados en Confianza

Como primera aproximación al problema, se puede limitar la ejecución de los agentes sólo a aquellas máquinas que consideremos de confianza [Ord96], esto es, máquinas de las cuales no se espera ningún tipo de actividad anómala o maliciosa. Incluso se puede pensar en un cierto control social de los agentes y establecer unos varemos de "reputación" para los hosts.

La principal limitación de esta propuesta es el número de hosts que podemos considerar de confianza, más si tenemos en cuenta entornos como Internet donde existe un inmenso potencial de aplicaciones que utilicen agentes, pero sin embargo los hosts que podemos considerar de confianza son muy pocos. Asimismo, sin medidas de protección es difícil distinguir del resto a un host de confianza que se ha tornado malicioso.

Como mejora sustancial del esquema anterior, existe la posibilidad de utilizar modelos de delegación de confianza [KM01, Rob02]. Con este tipo de mecanismos, de las relaciones de confianza que existen entre algunos elementos se pueden inferir otras relaciones. Sin embargo, dichas propuestas no indican los mecanismos de protección a utilizar una vez se ha determinado el nivel de confianza de un host. Del mismo

2.2. Propuestas de Prevención de Ataques

modo, quedan en el aire aquellos casos en los que no se pueden inferir ningún tipo de relación de confianza.

2.2.2. Hardware Específico de Ejecución

Algunas propuestas consideran que la solución al problema de los hosts maliciosos pasa por un equipo o subsistema cerrado de tipo hardware donde los agentes se ejecutan de forma segura [Yee97, WSB99]. En ese supuesto, ni siquiera el dueño del dispositivo debe poder manipular la ejecución del agente sin dejar el dispositivo inutilizado. Incluso se ha planteado que las tarjetas inteligentes o *smartcards* son el medio natural para conseguir esta protección [Fün99]. En cierto modo, la idea que reflejan estas propuestas es la de delegar la ejecución del agente a una TTP, que en este caso es un equipo hardware anexo al host y del que no se espera ningún tipo de actitud maliciosa.

Sin embargo, una solución de este tipo obligaría a adquirir un equipo hardware a cada host con capacidad para ejecutar agentes. Además, se debe tener en cuenta la confianza que se puede depositar en el suministrador del hardware, ya que nadie asegura que éste no incluya una puerta trasera con la idea de controlar el dispositivo remotamente. Además, en el caso particular de las tarjetas inteligentes, este tipo de dispositivos *tamper-proof* pueden resultar útiles en algunos entornos, pero sin embargo son inadecuados para cualquier aplicación que requiera un mínimo de cómputo ya que su capacidad es limitada.

2.2.3. Agentes Cooperativos

En [Rot99], se presenta la idea de la protección mutua. En la opinión de Roth, en un entorno hostil como Internet donde las relaciones de confianza son limitadas podemos suponer que es difícil que varios hosts confabulen para actuar en contra de un agente, ya que difícilmente confiarán los unos en los otros. Por ello, el autor propone el uso de agentes cooperativos que comparten secretos y decisiones, teniendo en cuenta que la parte individual del secreto no da información sobre el total. El

almacenamiento de resultados confidenciales y la toma de decisiones se realizan en el agente cooperativo, que además viaja por una ruta disjunta.

El uso de esta técnica puede repercutir seriamente sobre el rendimiento del sistema. Donde antes sólo se enviaba un agente ahora se deben enviar varios agentes cooperativos. Dichos agentes cooperativos deben permanecer en la red hasta que el último finalice su tarea. También se debe realizar el estudio de las actuaciones a realizar en caso de pérdida o finalización abrupta de uno de los agentes cooperativos, con la pérdida de los resultados del resto de agentes que ello comporta. Por lo que respecta a las comunicaciones entre agentes, se ve notablemente incrementada ya que debe realizarse el traspaso de la datos entre los cooperativos. Además, el sistema debe asegurar que las comunicaciones entre agentes van a ser posibles en todo momento. Con todo, la posibilidad de confabulación no se elimina por completo, pese a que se dificulta al viajar los agentes por rutas disjuntas.

2.2.4. Entropía de Agentes Móviles

En [Ng02], el autor define el concepto de entropía de agentes móviles como la relación entre las intenciones que tiene un agente de realizar una tarea y la probabilidad de que dicho evento ocurra. Asumiendo que los hosts maliciosos atacarán con más frecuencia unos agentes que otros, la idea es hacer que las intenciones del agente no sean conocidas. Para ello se pueden utilizar dos mecanismos: (1) la diseminación de intenciones (*intention spreading*) que consiste en disminuir la entropía del agente mediante la adición de tareas que el usuario no ha demandado. De esta manera se dificulta la labor de un posible atacante al no conocer las intenciones reales del agente; y (2) la reducción de intenciones (*intention shrinking*) que consiste en dividir las tareas que debe realizar un agente en varios agentes cooperativos que las hagan por separado, teniendo éstos una entropía alta.

El primero de los mecanismos tiene el inconveniente del malgasto de recursos. Además, nada impide al host malicioso intentar sacar provecho de todas las tareas que realice el agente. En cuanto al segundo de los mecanismos, comparte los mismos inconvenientes que la propuesta anterior de los agentes cooperativos [Rot99].

2.2.5. Generación de Claves Dependientes del Entorno

En la propuesta de generación de claves de entorno (*Environmental Key Generation*) [RS98], el agente busca una clave en el entorno para descifrar su código. El agente permanece “despistado” (*clueless*) hasta que se dan las condiciones de entorno adecuadas y puede descifrar su código. Dicha situación se da cuando el host origen deposita la clave en un lugar predeterminado, por ejemplo una página web, un servidor de noticias o cualquier otro tipo de recurso, tanto local como remoto.

El esquema publicado es muy costoso ya que obliga a monitorizar el entorno en busca de la clave y a intentar descifrar el código continuamente. Además, con esta medida sólo evitamos que la plataforma pueda analizar el código antes de ejecutarlo, y una vez descifrado el código el agente está desprotegido y queda a merced del host que puede realizar cualquier tipo de ataque. Asimismo, los hosts pueden confabular con el servidor que deposita la clave para intentar analizar el código anticipadamente.

2.2.6. Ofuscación del Código

Una *blackbox* es un entorno software del cual sólo podemos tener conocimiento de las entradas y salidas, pero tanto el código como los datos internos no pueden ser ni leídos ni modificados. Desgraciadamente no ha sido descubierto hasta el momento ningún algoritmo o función que ofrezca una protección de este tipo. Sin embargo, sí es posible implementar una *blackbox* limitada en tiempo [Hoh98] que protege al agente durante un cierto tiempo, después del cual ya no se asegura ni la confidencialidad ni la integridad de la ejecución. La propuesta está basada en la ofuscación del código mediante algoritmos de enredado (*mess-up*) como método para dificultar la lectura y por tanto la comprensión del código. Para dificultar aún más el análisis se pueden tomar una serie de medidas adicionales, como que agentes iguales tengan códigos ofuscados distintos, que se incluyan en el código ofuscado todas las librerías que necesite el agente (incluidas las que realicen funciones criptográficas) y que se realicen ejecuciones parciales y dependientes del entorno.

La propuesta asegura la protección del agente durante un cierto tiempo, pero esti-

mar dicho tiempo de protección no es sencillo. Por una parte, el tiempo de protección depende del algoritmo de enredado. Sin embargo, no existe ningún mecanismo que evalúe la bondad de un algoritmo de enredado. Por otra parte, el tiempo de protección depende de la capacidad que tenga el host malicioso de analizar el código ofuscado, y especialmente de si se requiere intervención humana para el análisis o se realiza mediante un automatismo. Aun así, un host con suficiente tiempo puede llegar a analizar dicho código, entender su funcionamiento e intentar sacar algún beneficio leyendo o modificando su contenido. Además, el rendimiento del sistema se resiente ya que la longitud del código ofuscado es sustancialmente mayor que la del código normal.

2.2.7. Computación de Funciones Cifradas

El uso de programas cifrados [ST98] ha sido propuesto como la única forma de asegurar la confidencialidad e integridad de la ejecución de forma criptográficamente segura. Los hosts ejecutan directamente el código cifrado, con lo cual no pueden extraer ninguna información de él, y por tanto no es posible modificarlo a su favor. Para recuperar los resultados es necesaria una función de descifrado. A modo de ejemplo, supongamos que el host Alice posee una función privada $f()$ y desea evaluar dicha función en x , donde x es una entrada privada que conoce el host Bob. El host Alice debe ser el único en tener conocimiento del resultado $f(x)$, pero no debe obtener ninguna información de la entrada x , de la misma manera que Bob tampoco debe obtener información de la función $f()$. Para conseguirlo deben realizar las siguientes acciones:

- Alice envía la función cifrada $E(f)()$ a Bob.
- Bob ejecuta la función cifrada en x y devuelve el valor $E(f)(x)$ a Alice.
- Mediante una función de descifrado Alice puede obtener $f(x) = D(E(f)(x))$.

El esquema planteado en [ST98] sólo plantea la situación de un agente que va a un único host a realizar una tarea y vuelve (*single-hop agent*). Esta limitación se elimina

2.3. *Propuestas de Detección de Ataques*

en [CCKM00], pudiendo el agente atravesar múltiples hosts. En la propuesta inicial tampoco se contemplaba la toma de decisiones del agente en el host debido a valores intermedios. Esta limitación también se elimina en [ACCK01], donde se introduce el concepto de servicio genérico de computación segura. Este servicio permite al agente tomar decisiones intermedias mientras se ejecuta en un host remoto. Las operaciones criptográficas se ejecutarán en una TTP en lugar de en el host, garantizando que dicha TTP no podrá extraer ningún tipo de información de la ejecución. El servicio de computación segura se debe ubicar en un servidor común para todos los agentes, teniendo que estar éste siempre en funcionamiento para que el sistema de agentes funcione.

Esta propuesta, pese a ser la más segura conceptualmente, presenta un inconveniente muy difícil de salvar, la dificultad en encontrar funciones que puedan ejecutarse cifradas. En [DF02] se introduce un homomorfismo con capacidad para realizar remotamente funciones aritméticas de forma cifrada. Sin embargo, la propuesta sólo puede aplicarse de forma parcial dado que las funciones aritméticas no cubren todas las necesidades de un agente móvil.

2.3. **Propuestas de Detección de Ataques**

En esta categoría incluiremos aquellas propuestas que pretenden detectar los ataques de manipulación después de la ejecución del agente. El objetivo de este tipo de propuestas es disuadir a los hosts maliciosos de realizar ataques, ya que si son detectados podemos tratar de imponerles un castigo. Los esquemas de detección de ataques son más fáciles de implementar que los esquemas que tratan de prevenirlos. Sin embargo padecen una serie de inconvenientes que impiden su aplicación en todos los entornos. En primer lugar, es necesaria una TTP que arbitre entre todas las partes y que pueda imponer sanciones cuando sea necesario. Además, si el beneficio obtenido al realizar el ataque es superior al castigo que pueda imponerse, sin duda estaremos incentivando comportamientos maliciosos.

2.3.1. Replicación y Voto

En [MvRSS96] se introduce la idea de replicación y voto. Los hosts se agrupan en distintas etapas por las que circularán los agentes, teniendo en cuenta que los hosts que se encuentren en la misma etapa deben ser independientes entre ellos y tener los mismos datos y recursos. En cada etapa los hosts ejecutan en paralelo réplicas del mismo agente y envían otro conjunto de réplicas hacia la siguiente etapa. En algunas de estas etapas los hosts comparan los resultados, de manera que se toman como correctos los resultados obtenidos por la mayoría, mientras que los resultados que difieren de la mayoría pueden considerarse comprometidos. Esta propuesta ofrece un mecanismo tolerante a errores de ejecución de agentes, y además permite detectar actitudes maliciosas por parte de los hosts.

El primer inconveniente de la propuesta viene dado por la misma replicación, que constituye un malgasto de recursos. El segundo inconveniente es que estamos asumiendo que los resultados de todas las replicas deben ser los mismos si todos los hosts actúan de forma honesta, con lo cual todos los hosts de una misma etapa deben tener los mismos recursos y datos. Sin embargo, esto no concuerda con la propiedad de independencia de los hosts, esto es, que todos los hosts deben tener intereses distintos para atacar al agente. Si un servidor tiene la misma información muy probablemente estará gestionado por una misma entidad, y por tanto ambos servidores tendrán los mismos intereses para atacar al agente.

2.3.2. Trazas Criptográficas

En [Vig98], Vigna introduce el mecanismo de las trazas criptográficas. Mientras el agente se ejecuta toma trazas de las instrucciones que alteran el estado del agente debido a datos de entrada externos. Como las trazas pueden tener un tamaño considerable, no se envían directamente al host origen sino que permanecen almacenadas en el host durante un cierto tiempo. En lugar de las trazas se envía al host origen un hash de las mismas para evitar ataques de repudio. Si el host origen sospecha de un host, le demanda las trazas, y con los datos de entrada que contienen ejecuta el

2.3. Propuestas de Detección de Ataques

agente de nuevo. El host es honesto si la nueva ejecución concuerda con la ejecución original. Si por el contrario las ejecuciones no concuerdan, quiere decir que el host es malicioso. Esta propuesta no sólo detecta los ataques, sino que también aporta pruebas del comportamiento malicioso de los hosts.

La propuesta presenta serios inconvenientes que impiden su directa aplicación. En primer lugar, el tamaño de las trazas hace desaconsejable su envío al host origen para todos los hosts. Esto obliga al host origen a solicitar las trazas en caso de sospecha, pero no se presenta ningún mecanismo que indique cómo un host pasa a ser sospechoso. En segundo lugar, como el host origen puede solicitar las trazas después de la ejecución, cada host debe almacenarlas durante un periodo de tiempo indeterminado. De hecho, una de las contribuciones de esta Tesis está enfocada en introducir mejoras sobre la propuesta de las trazas criptográficas para eliminar estos inconvenientes.

2.3.3. Estados de Referencia

En [Hoh00] se introduce la noción de estados de referencia (*Reference States*), es decir, estados producidos por hosts de confianza o referencia. La idea es detectar modificaciones en la ejecución comparando un estado de referencia con el estado del agente después de la ejecución en un host. El autor puntualiza que, en mayor o menor medida, todos los mecanismos de detección de ataques publicados hasta el momento utilizan estados de referencia, ya que comparan el estado del agente con otro que se ha producido al ejecutarlo en un host de confianza. Adicionalmente, Hohl incorpora un nuevo método de detección de ataques similar al de las trazas criptográficas de Vigna [Vig98], con la salvedad que la verificación se realiza en el siguiente host de confianza en lugar de en el host origen.

La propuesta representa un considerable incremento de la carga de las hosts, ya que la verificación consiste en ejecutar el agente de nuevo en el siguiente host de confianza. Además, que el host sea de confianza no significa que pueda acceder a los datos de entrada de hosts anteriores (que podrían ser considerados confidenciales) para poder reejecutar el agente.

2.4. Conclusiones del Capítulo

En el presente Capítulo hemos introducido el problema de los hosts maliciosos como el más difícil de resolver respecto a la seguridad en sistemas de agentes móviles. Los hosts pueden realizar prácticamente cualquier ataque sobre el agente dado que tienen control total sobre el entorno de ejecución: el código, los datos, las comunicaciones, el itinerario, el modo de ejecución o los resultados. Un host malicioso puede negar el servicio al agente, realizar escuchas, alterar cualquier parte del agente, o incluso confabular con otras entidades para sacar algún provecho o simplemente dañar la reputación de otra entidad involucrada.

En este Capítulo también se ha incorporado un breve resumen de las propuestas más representativas que intentan proteger al agente de los ataques de hosts maliciosos. Sin embargo, desde nuestro punto de vista ninguna de dichas propuestas es adecuada para su uso en un sistema real. Por un lado, los mecanismos de prevención de ataques son o bien demasiado costosos, o bien imposibles de implementar en un sistema real. Por otro lado, las propuestas de detección de ataques actuales son todavía demasiado complicadas, pese a ser las más prometedoras.

Es por ello que los objetivos de esta Tesis se centran en la búsqueda de mecanismos de protección eficaces y computacionalmente factibles, para evitar en la medida de lo posible los ataques de hosts maliciosos, o al menos minimizar los efectos que producen. En particular, las contribuciones de la Tesis pretenden asegurar la integridad de la ejecución del agente, esto es, asegurar que la ejecución está libre de ataques de manipulación. Como primera aportación de la Tesis, en el Capítulo 3 se introducen algunas mejoras en la propuesta de detección de ataques más conocida, la de las trazas criptográficas de Vigna [Vig98]. Como segunda aportación de la Tesis, en el Capítulo 4 se detalla una propuesta propia de detección de ataques basada en el empotrado de marcas digitales dentro del agente, más ligera computacionalmente que la de las trazas criptográficas. Finalmente y como tercera aportación, en el Capítulo 5 se introducen los mecanismos para castigar a aquellos hosts que han sido detectados como maliciosos.

2.4. Conclusiones del Capítulo

Capítulo 3

Protocolo de Detección de Sospechosos (SDP)

3.1. Mejoras sobre Propuestas Existentes

Como primera contribución de esta Tesis se pretende introducir algunas mejoras en una de las propuestas de protección de agentes existentes. Desde nuestro punto de vista, las propuestas de prevención de ataques existentes son o bien demasiado costosas o bien demasiado difíciles de implementar. Es por ello que consideramos más prometedoras las propuestas de detección de ataques y en ellas centraremos nuestros esfuerzos. En particular, intentaremos introducir mejoras sobre la propuesta de detección de ataques más conocida y la que consideramos que es la más representativa de las publicadas hasta el momento, la de las trazas criptográficas de Vigna [Vig98].

Tal como se mencionó en el Capítulo 2, con el mecanismo de las trazas es posible detectar ataques de manipulación y además probar el carácter malicioso del host que los realiza. Sin embargo, la propuesta aún tiene serios inconvenientes que limitan su uso directo. Dado que el tamaño de las trazas depende del volumen de datos de entrada del agente, éstas pueden ser demasiado grandes. Como su envío directo al host origen representaría un elevado coste para la red, sólo serán demandadas por

3.2. Funcionamiento de SDP

el host origen si existen sospechas sobre el comportamiento del host. Es por ello que, como primer inconveniente de la propuesta, la verificación de la integridad de la ejecución no se realiza sobre todos los hosts, sino que se realiza sólo sobre aquellos que son sospechosos. Sin embargo, en la propuesta no se indica en ningún momento cómo un host pasa a ser sospechoso. Asimismo y como segundo inconveniente de la propuesta, cada host debe almacenar sus trazas durante un periodo de tiempo indeterminado, ya que el host origen puede solicitarlas. Este factor hace poco admisible la propuesta, ya que en ese caso los servidores deberían reservar una gran capacidad de almacenamiento sólo para guardar las trazas de ejecuciones pasadas.

Para solucionar ambos inconvenientes sería necesario algún tipo de mecanismo que detectara qué hosts han actuado de forma sospechosa, y que lo hiciera en un tiempo razonable para limitar el tiempo que los servidores deben almacenar las trazas. Por ello, como primera aportación de esta Tesis se introduce un nuevo Protocolo de Detección de Sospechosos (*Suspicious Detection Protocol* o SDP) que permite detectar actitudes sospechosas en los hosts justo cuando el agente llega al origen [ESMF03d, ESMF03a]. De esta manera, a los hosts sospechosos se les podrán demandar las trazas en un tiempo mínimo, limitándose el tiempo de almacenamiento de las mismas. En suma, es posible ofrecer un mecanismo de detección y prueba de ataques efectivo y utilizable con el uso conjunto de ambos mecanismos.

3.2. Funcionamiento de SDP

Partiendo de la premisa que un host malicioso necesita tiempo para analizar el agente y realizar las modificaciones pertinentes, es posible detectar comportamientos sospechosos simplemente controlando el tiempo que permanece el agente en un host. Esa será la idea central de SDP, controlar la coherencia temporal de la ejecución. También cabe destacar que solamente es posible detectar actitudes sospechosas si el host persigue un objetivo con el ataque. Las modificaciones aleatorias en el código o los datos pueden no ser detectadas por SDP, ya que realizarlas no requiere de un tiempo representativo. Este tipo de modificaciones no constituyen un ataque de

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

manipulación sino que en realidad son ataques de negación de servicio, ya que pueden conducir al agente a comportarse de forma inesperada.

SDP se divide en tres fases claramente diferenciadas: (1) una fase de configuración inicial, en la cual los hosts reciben todos los parámetros necesarios para el uso de SDP; (2) una fase de envío del agente móvil, en la cual el agente realiza las tareas asignadas y además toma ciertas medidas temporales; y finalmente (3) una fase de validación de resultados, en la cual se verifica la coherencia temporal de la ejecución para identificar a los hosts que pueden considerarse sospechosos. En la especificación del protocolo también se incluye la forma de dar protección criptográfica a los contenidos del agente.

3.2.1. Fase de Configuración Inicial

En esta fase, todos los hosts que deben ejecutar el agente reciben los parámetros necesarios para utilizar el protocolo. A continuación se especifican los pasos de la fase de configuración inicial:

1. Para iniciar la fase de configuración inicial el host origen debe realizar las siguientes tareas:
 - a) Genera una referencia temporal general que denotaremos como T_o .
 - b) Genera una clave de sesión K_s que será utilizada para cifrar el agente mientras transita de host a host. Para ello es posible utilizar cualquier algoritmo de cifrado simétrico. En particular, en la implementación de nuestro sistema están disponibles a tal efecto DES y Triple-DES.
 - c) Decide una tolerancia temporal σ para dar a los hosts un cierto margen frente a retardos inesperados. σ dependerá del nivel de seguridad del sistema, de manera que cuanto mayor sea el valor de σ , menor será el número de hosts detectados como sospechosos.
 - d) Inicia el contador T_o . Las unidades de T_o deben ser las adecuadas para medir de forma precisa tiempos de ejecución o transmisión. En la implementación de nuestro sistema las unidades del contador son milisegundos.

3.2. Funcionamiento de SDP

- e) Envía un mensaje que contiene K_s y T_o a cada host por separado. Cada mensaje irá cifrado con la clave pública del host destino para dar privacidad a K_s y T_o . Cualquier algoritmo de clave pública puede ser utilizado a tal efecto. En particular, en la implementación de nuestro sistema se dispone de RSA como algoritmo de cifrado de clave pública. Teniendo en cuenta que el itinerario es de N hosts, éste es el formato de los N mensajes que envía el host origen:

$$Message_{O \rightarrow i}(sign_O[E_{K_{pub-i}}[K_s, T_o]]) \quad i \in 1 \dots N$$

2. Cada host que va a ejecutar el agente debe realizar las siguientes tareas:

- a) Recibe el mensaje procedente del host origen, lo descifra e inmediatamente inicia el contador temporal. Denotaremos el contador temporal en el host i -ésimo del itinerario como T_i .
- b) Si el host considera que dispone de suficientes recursos para ejecutar el agente, manda un mensaje de vuelta a modo de reconocimiento. El host incluye en el mensaje el hash de la clave de sesión para dar fe de la recepción de la misma. El mensaje que manda el host i -ésimo del itinerario es el siguiente:

$$Message_{i \rightarrow O}(sign_i[H(K_s)])$$

Si hay hosts que no responden al mensaje, el host origen asume que o bien no están disponibles para ejecutar el agente, o bien no disponen de suficientes recursos, con lo cual deben ser eliminados del itinerario. En ese caso y para asegurar la privacidad del envío, el host origen vuelve a iniciar la fase de configuración inicial con el envío de otro mensaje a los hosts que sí respondieron con una nueva clave de sesión. Este proceso continúa hasta que se recibe el reconocimiento de todos los hosts disponibles. En la Figura 3.1 se muestra el intercambio de mensajes entre el host origen y el resto de hosts.

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

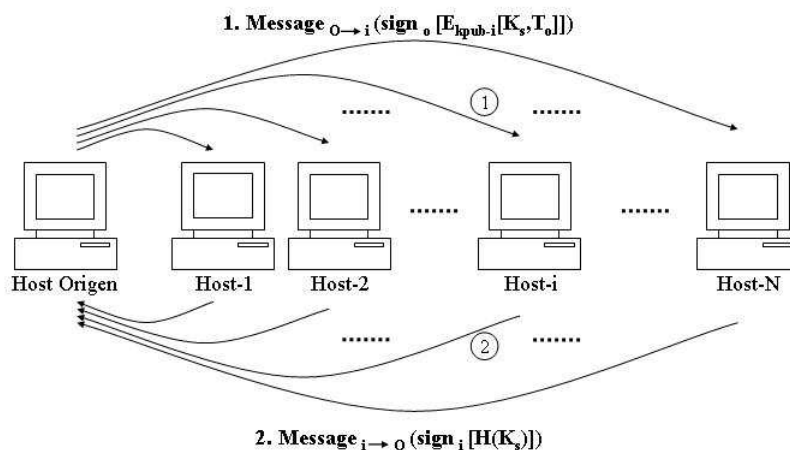


Figura 3.1: Fase de Configuración Inicial

Antes de iniciar la siguiente fase de envío del agente, el host origen debe realizar las siguientes tareas:

- Inicia un Estimador de Tiempos de Transmisión (*Transmission Time Estimator* o TTE). Se pretende con ello estimar el retardo de transmisión entre el host origen y cada host del itinerario. Este retardo incluye tanto los efectos de propagación como de transmisión del mensaje. Denotaremos el retardo de transmisión entre el host origen y el host i -ésimo del itinerario como $T_{delay-o-i}$. Este valor se usará para calcular la diferencia entre la referencia temporal general T_o y las distintas referencias temporales de cada host T_i . A modo de ejemplo, si queremos cambiar un tiempo T medido con T_i a su equivalente medido según la referencia temporal común T_o se debe realizar la siguiente conversión :

$$T|_{respecto T_o} = T|_{respecto T_i} + T_{delay-o-i}$$

- Inicia un Estimador de Tiempos de Ejecución (*Execution Time Estimator* o ETE). Se pretende con ello estimar el tiempo que cada host necesita para

3.2. Funcionamiento de SDP

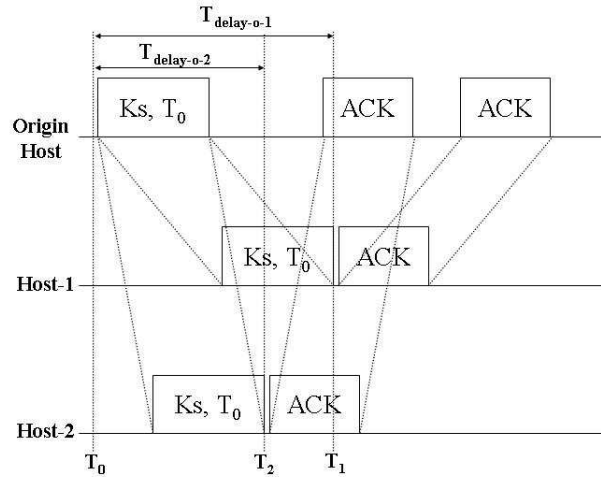


Figura 3.2: Cronograma de la Fase de Configuración Inicial

ejecutar el agente. Denotaremos el tiempo de ejecución estimado del host i -ésimo del itinerario como $T_{\text{exec}-i}$.

- El ETE también estima el tiempo necesario para cifrar los resultados y enviar el agente al siguiente host. Denotaremos este tiempo para el host i -ésimo del itinerario como $T_{\text{send}-i}$.

En la Figura 3.2 se incorpora un cronograma con las relaciones temporales que se dan entre los mensajes para un itinerario simplificado de sólo dos hosts.

A modo de resumen, cuando esta fase de configuración inicial concluye todas las entidades disponen de los parámetros necesarios para el envío y ejecución del agente. El host origen dispone de las estimaciones de los tiempos de ejecución y de transmisión necesarios, mientras que los hosts disponen de una clave de sesión para descifrar el agente, así como una referencia temporal con la cual contar los tiempos de llegada y finalización.

3.2.2. Fase de Envío del Agente Móvil

En esta fase del protocolo se manda el agente móvil para que sea ejecutado en todos los hosts del itinerario. Éstos deben tomar los tiempos de llegada del agente y de finalización de la ejecución para mandarlos de vuelta al host origen junto con los resultados. A continuación se especifican los pasos que deben seguirse en la fase de envío del agente:

3. El host origen realiza las siguientes tareas:
 - a) Inicia el TTE para estimar el tiempo de transmisión entre hosts consecutivos. Denotaremos el tiempo de transmisión entre el host i -ésimo del itinerario y el siguiente host como $T_{trans-i-i+1}$.
 - b) Cifra el agente con la clave de sesión K_s para protegerlo durante el trayecto y lo manda al primer host del itinerario. El tiempo de salida del agente en el host origen $T_{final-o}$ se guarda para su posterior uso en la fase de validación de resultados, ya que constituye el punto de partida de las medidas de tiempo. El agente enviado tiene el siguiente formato:

$$Agent_{O \rightarrow 1}(S_{K_s}[R_O])$$

donde

$$R_O = sign_O[Code, Data_O].$$

4. El primer host del itinerario recibe el agente y realiza las siguientes tareas:
 - a) Guarda el tiempo de llegada del agente con respecto al contador local T_1 . Denotaremos el tiempo de llegada del agente al primer host del itinerario como $T_{arriv-1}$.
 - b) Descifra el contenido del agente R_O con la clave de sesión K_s .
 - c) Ejecuta el código del agente y obtiene los resultados de la ejecución $Results_1$.

3.2. Funcionamiento de SDP

- d) Guarda el tiempo de finalización de la ejecución con respecto T_1 . Denotaremos el tiempo de finalización de la ejecución del agente en el primer host del itinerario como $T_{final-1}$.
- e) Cifra los resultados $Results_1$ y los tiempos de llegada $T_{arriv-1}$ y finalización $T_{final-1}$ usando la clave pública del host origen.

$$E_{K_{pub-o}}[Results_1, T_{arriv-1}, T_{final-1}].$$

- f) Prepara el agente para su migración al siguiente host. El agente debe transportar R_O (donde se encuentra el código y algunos datos de entrada), ciertos datos de entrada $Data_1$ para el siguiente host (si fueran necesarios), así como los resultados y los tiempos que se han cifrado en el anterior paso. Todo ello constituye el contenido del agente R_1 , que debe estar convenientemente firmado para asegurar su autenticidad e integridad. Finalmente, cifra todos estos datos con la clave de sesión K_s y lo envía al siguiente host del itinerario¹:

$$Agent_{1 \rightarrow 2}(S_{K_s}[R_1])$$

donde

$$R_1 = sign_1[R_O, Data_1, E_{K_{pub-o}}[Results_1, T_{arriv-1}, T_{final-1}]].$$

- 5. Cada host del itinerario realiza las mismas tareas del punto 4. Finalmente, el último host del itinerario envía el siguiente agente al host origen:

$$Agent_{N \rightarrow O}(S_{K_s}[R_N])$$

¹Estas dos últimas tareas suponen un cierto tiempo de preparación del agente antes del envío al siguiente host que se había estimado como T_{send-1} .

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

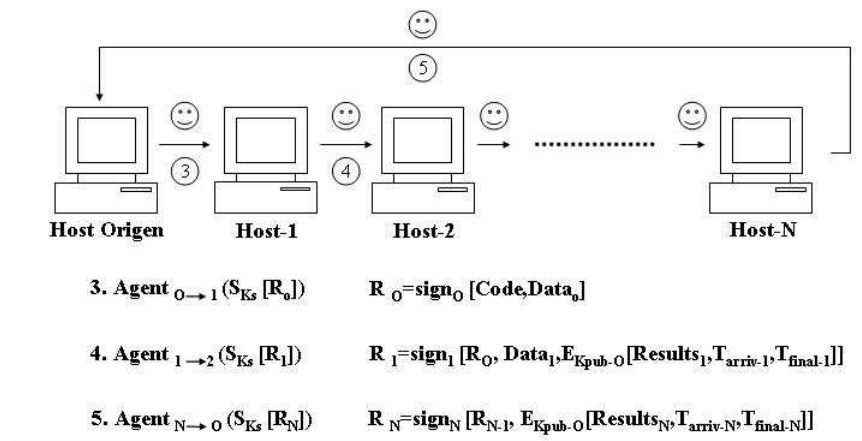


Figura 3.3: Fase de Envío del Agente

donde

$$R_N = \text{sign}_N[R_{N-1}, E_{K_{\text{pub-0}}}[\text{Results}_N, T_{\text{arriv-N}}, T_{\text{final-N}}]].$$

Simplemente resaltar que no es necesario el envío de datos de entrada para el siguiente host, pues éste es el host origen. Finalmente, una vez el agente ha llegado al host origen, éste debe realizar las siguientes tareas:

- Guarda el tiempo de llegada del agente. Denotaremos este tiempo como $T_{\text{arriv-o}}$.
- Descifra los resultados y los tiempos de llegada y finalización de cada host.

En la Figura 3.3 se muestran las distintas migraciones del agente para ejecutarse en un itinerario de N hosts. Por su parte, en la Figura 3.4 se muestra un cronograma con las relaciones temporales entre los distintos envíos del agente para un itinerario simplificado de sólo dos hosts.

A modo de resumen, al finalizar esta fase el agente ha realizado las tareas asignadas y ha vuelto al origen con los resultados de la ejecución y los tiempos de llegada

3.2. Funcionamiento de SDP

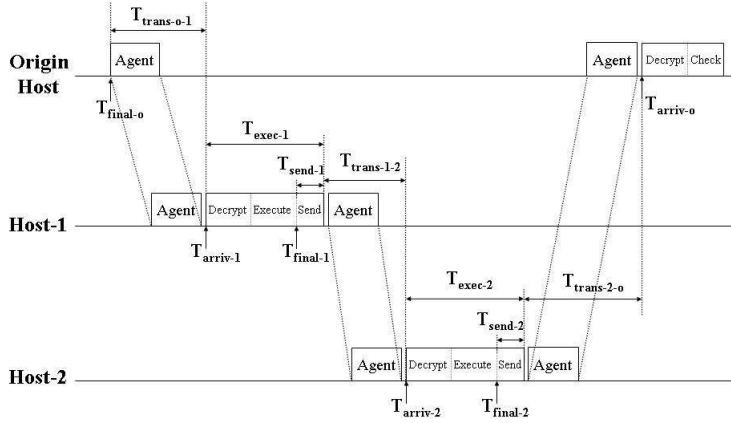


Figura 3.4: Cronograma de la Fase de Envío del Agente

y finalización. Con dichos tiempos el host origen puede verificar la coherencia temporal de la ejecución.

3.2.3. Fase de Validación de Resultados

En esta fase del protocolo, el host origen debe verificar la coherencia temporal de la ejecución. Para ello dispone tanto de los tiempos de llegada y finalización de cada uno de los hosts, como de las estimaciones de los tiempos de ejecución y transmisión realizadas por el ETE y el TTE respectivamente. Para asegurar la coherencia temporal de la ejecución el host origen realiza un conjunto de chequeos temporales. Estos son tres de los chequeos que consideramos esenciales:

- Primer Chequeo: un host es sospechoso si el tiempo que tarda en ejecutar el agente supera el tiempo de ejecución estimado en un valor superior a la tolerancia. Por tanto, todos los hosts honestos deben cumplir la siguiente expresión:

$$(T_{final-i} - T_{arriv-i} + T_{send-i}) \leq T_{exec-i} + \sigma$$

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

Un host es sospechoso cuando sus tiempos de llegada y finalización no cumplen esta expresión.

- Segundo Chequeo: el tiempo de salida de un host no puede ser mayor que el tiempo de llegada del siguiente host. Todos los hosts honestos deben cumplir la siguiente expresión:

$$T_{arri\text{-}i+1}|_{\text{respecto}T_{i+1}} \geq T_{final\text{-}i}|_{\text{respecto}T_i} + T_{send\text{-}i}$$

Antes de aplicar la expresión es necesario unificar la referencia temporal a T_o . Para ello solamente se debe sumar el tiempo de transmisión de cada host respecto el origen. La expresión puede reescribirse de la siguiente forma:

$$T_{arri\text{-}i+1} + T_{delay\text{-}o\text{-}i+1} \geq T_{final\text{-}i} + T_{delay\text{-}o\text{-}i} + T_{send\text{-}i}$$

Si la expresión no se cumple, ambos hosts son sospechosos ya que no es posible discriminar cual de los dos está mintiendo.

- Tercer Chequeo: la diferencia entre los dos términos anteriores debe ser similar al tiempo de transmisión entre hosts. Todos los hosts honestos deben cumplir la siguiente expresión:

$$| [T_{arri\text{-}i+1} + T_{delay\text{-}o\text{-}i+1}) - (T_{final\text{-}i} + T_{delay\text{-}o\text{-}i} + T_{send\text{-}i})] - T_{trans\text{-}i\text{-}i+1} | \leq \sigma$$

Si la expresión no se cumple, ambos hosts son sospechosos ya que no es posible discriminar cual de los dos está mintiendo.

Dadas las dos medidas de tiempos que ha tomado el agente en su trayecto, éstos son tres de los chequeos temporales que consideramos primordiales. Dicho conjunto de chequeos podría verse incrementado fácilmente mediante la toma de medidas

3.3. Integración con Otras Propuestas

adicionales, como por ejemplo tiempos en los cuales se realizaron comunicaciones con entidades externas.

Como se puede apreciar de las expresiones, la tolerancia temporal σ permite unos márgenes de variación a la hora de ejecutar o transmitir el agente. SDP ha sido diseñado para permitir ajustar el nivel de seguridad mediante este parámetro. Cuanto mayor sea σ , mayor será la permisividad del sistema y por tanto menos hosts serán detectados como sospechosos. Por el contrario, cuanto menor sea σ , los hosts disponen de menos tiempo para analizar y modificar el agente y por lo tanto el sistema es más seguro.

A modo de resumen, una vez finalizada la fase de validación de resultados, el host origen dispone de una lista de hosts sospechosos. Cómo utilizar dicha lista de hosts sospechosos conjuntamente con el mecanismo de las trazas criptográficas se especifica en la siguiente sección.

3.3. Integración con Otras Propuestas

La idea inicial de SDP era solucionar los principales inconvenientes de la propuesta de las trazas criptográficas, si bien eso no implica que no pueda ser utilizado conjuntamente con otras propuestas.

3.3.1. Trazas Criptográficas

SDP ha sido diseñado para que sea fácilmente integrable con el mecanismo de las trazas criptográficas [Vig98]. A continuación se muestran las tareas adicionales que se deben realizar si ambos mecanismos se utilizan de forma conjunta en el mismo agente móvil:

- Fase de configuración inicial: no hay tareas adicionales en esta fase, las tareas son las mismas que las especificadas en SDP.
- Fase de envío del agente: durante la ejecución, el agente deberá tomar las trazas en cada host. Dichas trazas quedarán almacenadas en el host durante

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

un tiempo reducido pues el host origen puede pedir las trazas una vez acabada la fase de validación de resultados. Para evitar posteriores manipulaciones, el host envía un hash de las trazas a modo de prueba junto con los resultados y los tiempos.

- Fase de validación de resultados: cuando el host origen recibe el agente y aplica los chequeos temporales, lo único que debe hacer es pedir las trazas a aquellos hosts que han sido detectados como sospechosos por el protocolo. Cuando llegan las trazas, en primera instancia se verifica que corresponden con el hash enviado, y finalmente se verifica la integridad de la ejecución reejecutando el agente.

3.3.2. Ofuscación

El control del tiempo de ejecución puede ser muy útil si se usa conjuntamente con técnicas de ofuscación del código [Hoh98]. Tal como se comentó en el Capítulo 2, Hohl proponía la ofuscación como herramienta para conseguir una *blackbox* limitada en tiempo, esto es, un entorno software donde durante un cierto tiempo no es posible extraer información del mismo, y por tanto es imposible modificarlo para obtener una ejecución favorable. El agente ofuscado asegura una ejecución íntegra y confidencial durante el tiempo necesario para que un host malicioso lo analice y modifique a su favor.

El uso de la ofuscación como mecanismo de protección implica que un host malicioso necesitará mucho más tiempo para analizar y modificar el agente que lo que le llevaría modificar un agente no ofuscado. Por tanto, si utilizamos SDP para controlar el tiempo de ejecución del agente estamos forzando a los hosts maliciosos a ejecutar el agente directamente sin analizarlo si no quieren ser detectados como maliciosos.

3.4. Mejoras

La descripción de SDP trataba de ser de uso genérico para cualquier tipo de plataforma. Sin embargo, es posible introducir ciertas mejoras en dicha especificación

3.4. Mejoras

para hacer al protocolo más sencillo y exacto simplemente con el uso de alguna técnica adicional. Estas son algunas de las mejoras que podrían introducirse sobre SDP:

- Los tiempos de llegada y finalización se miden a partir de una referencia temporal enviada por el host origen. Sin embargo, esta referencia temporal es poco precisa debido a que hay que enviarla a cada uno de los hosts, con lo cual existen variaciones debidas a los tiempos de transmisión y propagación de los paquetes. SDP puede simplificarse notoriamente y hacerse más exacto si introducimos en el sistema una referencia temporal externa calculada por protocolos específicos, como podría ser NTP (*Network Time Protocol*). En este supuesto, tanto T_o como los T_i no serían necesarios y todos los tiempos de llegada y finalización podrían medirse respecto dicha referencia temporal única del sistema. En estos momentos, nuestro grupo de trabajo está trabajando en la adaptación e interacción del sistema de sincronización NTP con la implementación de SDP.
- Debe realizarse un estudio profundo acerca de las políticas a seguir para estimar los tiempos de ejecución en los hosts. Cómo implementar el ETE para que las estimaciones de tiempos de ejecución se ajusten al máximo a la realidad es un tema que no es de fácil solución. En la actual implementación de SDP hemos considerado por simplicidad que dichos tiempos de ejecución son deterministas, si bien se podrían utilizar algoritmos más sofisticados que dependan del código a ejecutar, o de la memoria o CPU disponibles en los hosts.
- Debe realizarse un estudio profundo acerca de las políticas a seguir para estimar los tiempos de transmisión entre hosts. Al igual que el anterior punto, la estimación de tiempos de transmisión que realiza el TTE se debe ajustar al máximo a la realidad. En la actual implementación de SDP hemos considerado por simplicidad que los tiempos de transmisión son deterministas, si bien es posible utilizar algoritmos más sofisticados que dependan del ancho de banda de la red o de los retardos de propagación.

3.5. Ataques al Protocolo

Los ataques a SDP están centrados en tratar de pasar de forma ilegal los chequeos temporales. El objetivo de un hosts malicioso es no ser detectado como sospechoso, involucrando a otros hosts si fuera necesario. Para ello puede modificar los tiempos de llegada o finalización del agente. Asumiremos que los tiempos estimados por el ETE y el TTE son suficientemente cercanos a los tiempos de ejecución o transmisión reales como para impedir el análisis y modificación del agente. En este supuesto, los hosts maliciosos sólo pueden tratar de acusar a un host honesto o bien confabular con otros hosts.

3.5.1. Ataques de un Único Host

A continuación se especifican los ataques que puede realizar un único host malicioso:

- El host malicioso no modifica los tiempos de llegada y finalización: cuando el agente llega al host, en lugar de ejecutar el código directamente, lo analiza y modifica a su favor². Si el ETE ha estimado de forma correcta el tiempo de ejecución, el host malicioso será detectado como sospechoso pues no pasará el Primer Chequeo.
- El host malicioso modifica los tiempos de llegada y finalización: el host miente sobre los tiempos de llegada o finalización para quedar exculpado o inculpar a otro host. Existen dos posibilidades:
 - El host modifica su tiempo de llegada $T_{arri\text{v}-i}$: cambiar el tiempo de llegada a un valor menor no tiene sentido ya que esto incrementaría el tiempo de ejecución. La única posibilidad es guardar un tiempo de llegada

²Dichas modificaciones pueden perseguir un beneficio económico, una ejecución favorable o dañar la reputación de otra entidad. En ningún caso incluimos en esta tipología de ataques las modificaciones de tipo aleatorio, ya que no persiguen ningún objetivo y pueden conducir al agente a comportarse de forma impredecible.

3.5. Ataques al Protocolo

mayor al real. Sin embargo, el host no pasará el Tercer Chequeo si el tiempo de llegada real $T_{arriv-i}$ supera al tiempo de llegada estimado en más de la tolerancia σ .

- El host modifica su tiempo de finalización $T_{final-i}$: cambiar el tiempo de finalización a un valor mayor no tiene sentido ya que esto incrementaría el tiempo de ejecución. La única posibilidad es guardar un tiempo de finalización menor del real. Sin embargo, el host no pasará el Tercer Chequeo si el tiempo de finalización real $T_{final-i}$ es inferior al tiempo de finalización estimado en más de la tolerancia σ .
- El host malicioso guarda una copia del agente: el host recibe el agente y lo ejecuta correctamente, pero guarda una copia del mismo. Esta copia puede ser analizada con detenimiento para buscar las debilidades del código, de manera que si alguien envía un agente similar a éste, el host sólo tiene que sustituir el código original por el código modificado. Si el código modificado se ejecuta en un tiempo similar al código original, este ataque no puede ser detectado. Este ataque puede evitarse bien utilizando agentes distintos para tareas similares, o bien utilizando técnicas de ofuscación del código [Hoh98].

3.5.2. Ataques de Confabulación

Los siguientes ataques al protocolo pueden realizarse por un grupo de hosts maliciosos confabulados:

- Traspaso de código original: un host malicioso recibe el agente y lo ejecuta correctamente, pero manda inmediatamente una copia del código a otro host malicioso. Éste tiene tiempo para analizarlo hasta que recibe el agente. Si en dicho tiempo ha conseguido modificarlo a su favor, lo único que debe hacer es sustituir el código original por el modificado. Si el código modificado se ejecuta en un tiempo similar al código original, este ataque no puede ser detectado.
- Traspaso de código modificado: el host malicioso analiza y modifica el agente,

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

y a continuación manda una copia del agente modificado a otro host malicioso. Cuando recibe el agente, el segundo host malicioso lo único que debe hacer es sustituir el código original por el modificado. El primer host será detectado como sospechoso, sin embargo, si el código modificado se ejecuta en un tiempo similar que el código original, el segundo host malicioso no será detectado. Este ataque es improbable ya que uno de los hosts debe estar dispuesto a “suicidarse”.

Este tipo de ataques pueden evitarse limitando el número de hosts que pueden confabular con técnicas de protección del itinerario [DW99, MB03]. Uno de los requerimientos que persigue la protección del itinerario es forzar que los hosts sólo tengan conocimiento del anterior y del posterior host, con lo cual la posibilidad de confabulación se limita a estos dos hosts. Sin embargo, este tipo de mecanismos no son útiles para proteger el agente frente a confabulaciones realizadas por hosts consecutivos:

- El primer host analiza y modifica el código, pero guarda un valor menor de $T_{final-i}$. El código modificado se envía al siguiente host para que lo ejecute. Para dar coherencia al sistema temporal, el siguiente host guarda un valor menor de $T_{arriiv-i+1}$ y ejecuta el código modificado. Ambos hosts intentan pasar el Tercer Chequeo modificando sus tiempos de llegada y finalización. En cierto sentido, ambos hosts comparten sus tiempos de ejecución, pero el primero de ellos dispone de la mayoría de tiempo para analizar y modificar el código. Si hay coherencia en los tiempos, este ataque no puede ser detectado. Esta situación puede intentarse paliarse separando del itinerario aquellos hosts que son susceptibles de confabular, como por ejemplo hosts que dependan de una misma compañía.

3.6. Inconvenientes

Como es lógico, SDP no está exento de inconvenientes que pueden limitar su uso en todos los entornos. A continuación se enumeran los más relevantes:

3.6. Inconvenientes

- El host origen debe permanecer on-line hasta que finalice la transacción para verificar la validez temporal de la ejecución. Este es el precio que se debe pagar para que los hosts puedan borrar las trazas en el mínimo tiempo. La única forma de permitir al dueño del agente estar off-line pasa por delegar las tareas del protocolo a un servidor de confianza que permanezca on-line mientras dure la transacción.
- SDP sólo puede ser utilizado si el agente vuelve al host origen una vez finalizada la ejecución. En muchas aplicaciones no es estrictamente necesario que el agente móvil vuelva al host origen. Por ejemplo, un agente cuya tarea sea configurar un router no tiene por qué volver ya que no hay resultados. Sin embargo, en nuestra opinión cualquier esquema de detección de ataques debe estar basado en unas pruebas que deben llegar al host origen para ser evaluadas. Dichas pruebas o bien vuelven con el agente, o bien se deberán enviar más tarde en un mensaje.
- El protocolo puede detectar a hosts honestos como sospechosos. El mecanismo no es ni mucho menos infalible, de hecho, en el segundo y tercer chequeo es imposible discriminar cual es el sospechoso entre dos hosts consecutivos. Este factor restará efectividad al sistema de detección.
- El host origen debe estimar los tiempos de ejecución del agente en los hosts, así como los tiempos de transmisión entre ellos. Cómo se realizan estas estimaciones es un tema futuro de investigación en nuestro grupo de trabajo. De hecho, se asume que dichos cálculos han sido delegados a entidades externas, las cuales pueden utilizar muy distintas políticas para realizar el cálculo.

Dados estos inconvenientes, recomendamos el uso de SDP en entornos donde el agente móvil deba volver al host origen obligatoriamente, por ejemplo para devolver los resultados de una ejecución. Asimismo, debe ser posible realizar estimaciones de los tiempos de ejecución y de transmisión, con lo cual se desaconseja el protocolo cuando existan factores que hagan dichas estimaciones difíciles, como por ejemplo que el

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

agente tenga dependencias con otros agentes o deba computar grandes volúmenes de datos.

3.7. Implementación y Evaluación del Rendimiento de SDP

En el resto del capítulo se comentan algunos aspectos relativos a la implementación de SDP, como son las herramientas software utilizadas para la programación del protocolo y las librerías criptográficas utilizadas para dar protección al agente. Asimismo, se incluyen las pruebas realizadas para evaluar el rendimiento de SDP [ESMF03c].

3.7.1. Herramientas Software y Especificaciones Hardware

Dado que buscamos que el software implementado sea portable a cualquier plataforma, utilizaremos Java como lenguaje de programación, más concretamente el *Aglet Software Development Kit* 2.0.1 (ASDK o *Aglets Workbench*) [Sou] diseñado por el *IBM Tokyo Research Laboratory*. ASDK nos permite crear *aglets* (AGent-appLETS), que no son más que objetos Java que pueden utilizarse como agentes móviles. Asimismo, ASDK ofrece un entorno gráfico llamado Tahiti desde donde los *aglets* pueden ser enviados, recibidos y ejecutados. Dicho servidor de agentes móviles Tahiti está soportado por la *Java Virtual Machine* (JVM) incluida en el *Java Development Kit* (JDK) 1.3.1 [Jav]. Una de las razones por las que se eligió ASDK para la implementación es la gran cantidad de información que se puede encontrar en Internet acerca de su uso, como por ejemplo [Agl, Tah], y especialmente [LO98].

Como se ha descrito, SDP ofrece protección criptográfica al agente y a los datos que transporta en caso que sea requerida. Para ello han sido necesarias una serie de librerías criptográficas adicionales, la *Java Cryptography Extension* (JCE) 3.01 [IAI] desarrolladas por el *Institute for Applied Information Processing and Communication* perteneciente a la *Graz University of Technology*. La JCE es un conjunto de librerías donde podemos encontrar la mayoría de herramientas criptográficas, como por ejemplo el cifrado simétrico y asimétrico, la generación de claves o los algoritmos

3.7. Implementación y Evaluación del Rendimiento de SDP

de firma. JCE extiende la *Java Cryptography Architecture* (JCA) que proporciona Sun Microsystems en la JDK.

En particular, en la implementación desarrollada para SDP sólo se van a utilizar las siguientes herramientas criptográficas:

- Algoritmos de cifrado simétrico: en la implementación se utiliza para proteger el agente durante el trayecto. SDP dispone de las siguientes opciones: algoritmo DES con longitud de clave 64 bits³ y Triple DES con clave de 128 bits. Consideramos que mayores longitudes de clave no son necesarias dado que se genera una nueva clave de sesión para cada agente. El sentido de cifrar el contenido del agente es el de protegerlo frente a escuchas de elementos externos, esto es, elementos que no están incluidos en el itinerario del agente.
- Algoritmo de Clave Pública: en la implementación se utiliza para asegurar la confidencialidad de la clave de sesión, los resultados y los tiempos de llegada y finalización, así como para asegurar la integridad de cualquier información enviada mediante la firma digital. En particular, en la implementación se dispone de cifrado RSA con claves de 512, 1024 y 2048 bits, así como SHA-1 para firma. Para utilizar criptografía de clave pública sería conveniente el uso de una PKI con la cual emitir certificados que asocien una clave pública a una identidad. Sin embargo, por simplicidad en la implementación de SDP se ha generado un Certificado de Identidad X.509 [HFPS99, PFS02] por cada servidor de agentes mediante las herramientas que ofrece la JCE, en lugar de pedirlos a una entidad emisora. Asimismo, se ha creado un almacén de claves en cada servidor para poder disponer de todas las claves públicas del resto de máquinas.

En cuanto a las pruebas, han sido realizadas en un laboratorio compuesto por tres computadoras, una trabajando como *Home* o host origen, y las dos restantes actuando como hosts. Host-1 y Host-2 son estaciones fijas Pentium II a 400 MHz, 64 MB de

³Actualmente se desaconseja el uso de DES con clave de 64 bits por ser débil para los servicios actuales. A pesar de ello, incluimos dicha posibilidad en la implementación para compararlos con algoritmos que se consideran más seguros como Triple DES.

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

RAM y Windows 2000 y Linux SuSE 8.1 como Sistemas Operativos, mientras que el Home es un portátil con un procesador a 1GHz, 256 MB de RAM y Windows XP como Sistema Operativo. Todas las estaciones se conectan mediante una red de tipo Ethernet trabajando a 10 Mbps.

3.7.2. Resultados de las Pruebas

Para las pruebas se ha considerado un agente muy sencillo que solamente debe encontrar el precio más barato de un vuelo. Esta decisión se debe a que queremos evaluar únicamente el coste que supone SDP, y no el coste que supone ejecutar el agente. El uso de lógicas de servicio más complicadas podría distorsionar los resultados que buscamos.

SDP ofrece la posibilidad de utilizar herramientas criptográficas para proteger el agente y la información que transporta. Como es lógico, el rendimiento del protocolo dependerá del uso de dichas herramientas y en especial de la longitud de las claves de cifrado. Éstos son los modos de funcionamiento para los cuales se han realizado medidas:

- El agente se envía sin ningún tipo de mecanismo de seguridad, ni siguiera SDP. En este caso no se asegura ni la confidencialidad ni la integridad del agente durante su viaje. Este modo de funcionamiento se denota en las gráficas con una longitud de clave de -50. Obviamente, dicha longitud de clave no tiene sentido, pero de esta manera podemos incluir estas medidas en las gráficas para compararlas con el resto de modos que sí utilizan cifrado.
- El agente utiliza SDP, pero no utiliza ningún tipo de herramienta criptográfica. Al igual que en el anterior caso, no se asegura ni la confidencialidad ni la integridad del agente durante su viaje. Este modo de funcionamiento es especialmente útil para compararlo justo con el anterior, ya que la diferencia entre ambas medidas nos dará específicamente el coste que supone SDP. Las medidas de este modo se denotan en las gráficas con una longitud de clave de 0.

3.7. Implementación y Evaluación del Rendimiento de SDP

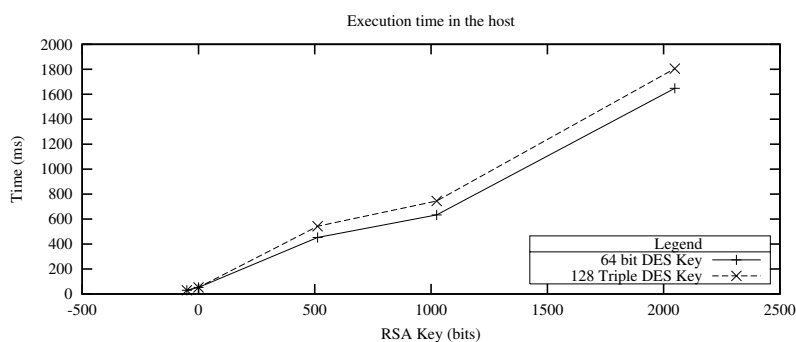


Figura 3.5: Tiempo de Ejecución en los Hosts

- El agente utiliza tanto SDP como herramientas criptográficas. Con ello es posible asegurar la confidencialidad e integridad del agente durante el trayecto, así como la autenticación de las entidades implicadas. Los distintos modos de funcionamiento se denotan en las gráficas en función del tipo de algoritmo criptográfico y de la longitud de las claves.

A continuación se presentan los resultados obtenidos, teniendo en cuenta que las medidas son el valor medio obtenido tras realizar varias pruebas.

Tiempo de Ejecución en los Hosts

En la Figura 3.5 se muestra el tiempo medio necesario para que un host ejecute el agente. Dicho tiempo se ha calculado como la diferencia entre el tiempo de llegada del agente y el tiempo de finalización de la ejecución. Por tanto, las medidas tienen en cuenta el tiempo para descifrar el agente cuando llega, pero no el tiempo requerido para el cifrado antes de enviarlo. En el momento de las pruebas, la CPU del host estaba ocupada a un 50 %.

De la Figura 3.5 se pueden extraer una serie de observaciones. En primer lugar, el uso del protocolo no supone un gran incremento en el tiempo de ejecución del agente, dado que el tiempo sin SDP (30 ms) se incrementa muy poco respecto al tiempo con SDP y sin criptografía (52 ms). Por tanto, las tareas que debe realizar el

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

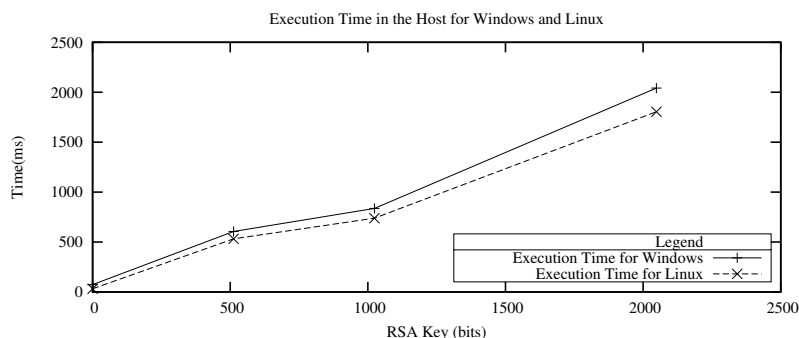


Figura 3.6: Tiempo de Ejecución en los Hosts para Windows y Linux

host relativas a SDP no suponen un gran incremento de tiempo. En segundo lugar, se debe valorar el efecto que produce la aplicación de criptografía. Como se observa en la Figura 3.5, el tiempo de ejecución se incrementa a medida que el tamaño de las claves crece, pero no afecta por igual un incremento de la clave en el cifrado simétrico que en asimétrico. Éste último produce un incremento de tiempo de ejecución mucho mayor frente a la diferencia de tiempos entre el cifrado DES y el Triple DES.

Asimismo, se han realizado pruebas con los dos sistemas operativos disponibles para verificar si hay diferencias notables entre ellos. En concreto, en la Figura 3.6 se muestra una comparativa entre los tiempos de ejecución con Windows 2000 y Linux SuSE 8.1. Al igual que en el caso anterior se ha trabajado con equipos que soportaban una carga añadida entorno al 50 %.

De los valores obtenidos podemos concluir que no hay grandes diferencias entre ambos, existiendo una ligera tendencia a obtener mejores resultados el sistema operativo Linux.

Tiempo de Ejecución en el Host Origen (Home)

En la Figura 3.7 se muestra el tiempo medio necesario para que el host origen recupere los resultados de la ejecución y verifique si hay hosts sospechosos. De la figura se puede inferir que de nuevo el uso del protocolo de detección de sospechosos

3.7. Implementación y Evaluación del Rendimiento de SDP

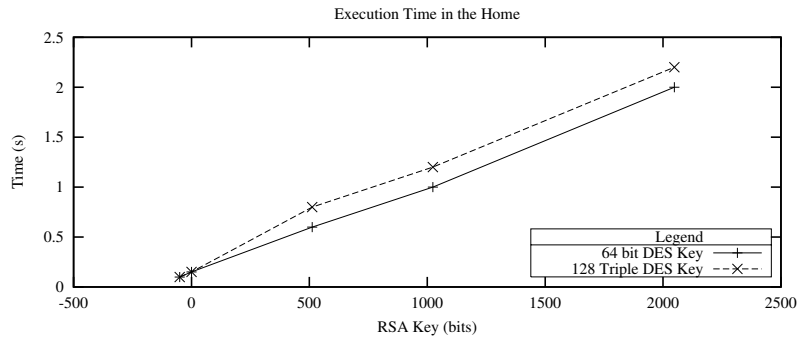


Figura 3.7: Tiempo de Ejecución en el Host Origen

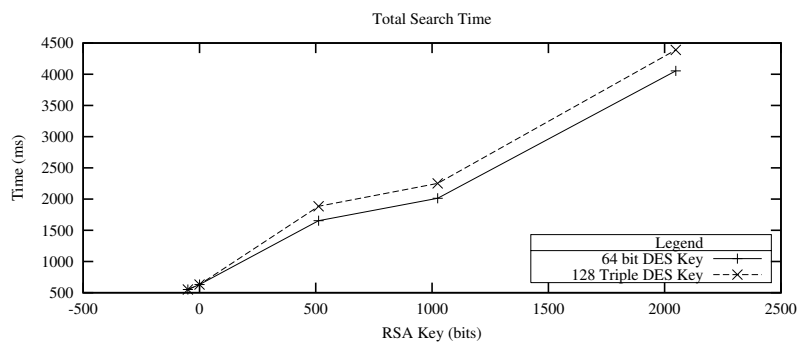


Figura 3.8: Tiempo Total Empleado por el Agente

no supone un gran incremento de tiempo para el host origen ya que los tiempos sin SDP (0,1 s) son muy similares a los tiempos con SDP y sin criptografía (0,15 s).

Tiempo Total de Búsqueda

En la Figura 3.8 se muestra el tiempo que emplea el agente en su viaje desde que es enviado hasta que se recibe en el host origen. En las pruebas la ocupación de la red estaba entorno al 30 %, y la carga de la CPU en los host entorno el 50 %.

Para analizar los resultados debemos considerar que el tiempo total de la búsqueda está formado por una serie de elementos, todos ellos variables. Si N es el número

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

de hosts del itinerario y asumimos que:

- El tiempo de ejecución T_{exec} es similar en todos los hosts.
- El tiempo de transmisión T_{trans} es similar en todos los hosts.
- El tiempo de serialización T_{serial} es igual al de deserialización, y similares en todos los hosts.

Entonces, el tiempo total se puede expresar de la siguiente forma:

$$T_{total} = \sum_{i=1}^N T_{exec-i} + (N + 1)(T_{trans} + 2T_{serial}).$$

Teniendo en cuenta que las medidas han sido tomadas en un itinerario de $N=2$ hosts, de la expresión anterior es posible extraer el tiempo de serialización (o deserialización) T_{serial} . Por ejemplo, T_{serial} está alrededor de unos 76 ms si no se usa SDP ni criptografía, mientras que está alrededor de unos 122 ms para el caso en que se utiliza Triple-DES y RSA con clave de 512 bits. La diferencia entre ambos valores se debe básicamente a que el tamaño del agente es mayor en el caso en que se aplica criptografía.

CPU en los Hosts

En la Figura 3.9 se muestra la carga de CPU media durante el tiempo de ejecución del agente. Igualmente, de la Figura se puede extraer que el protocolo no supone un gran incremento, ya que la carga sin SDP (5 %) no dista mucho de la carga con SDP y sin criptografía (9 %).

Asimismo, se ha considerado interesante poder comparar los dos sistemas operativos utilizados para ver las diferencias en consumo de CPU. De la Figura 3.10 se puede extraer que no existen grandes diferencias entre ambos protocolos, si bien existe una ligera tendencia a consumir menos recursos en Linux.

3.7. Implementación y Evaluación del Rendimiento de SDP

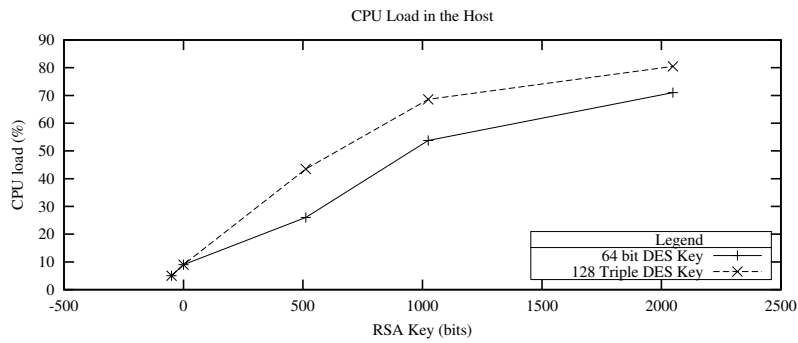


Figura 3.9: CPU en el Host

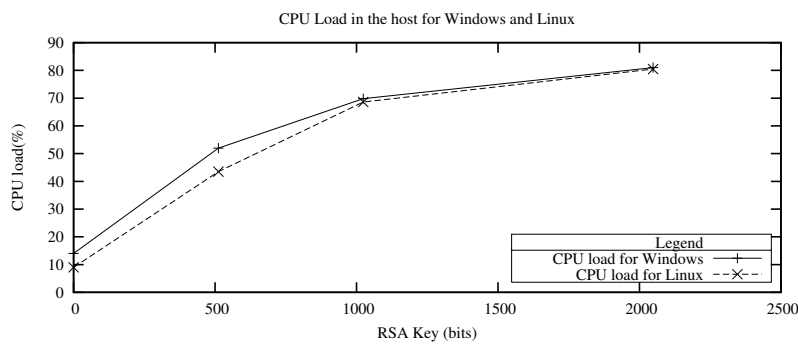


Figura 3.10: CPU en el Host para Windows y Linux

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

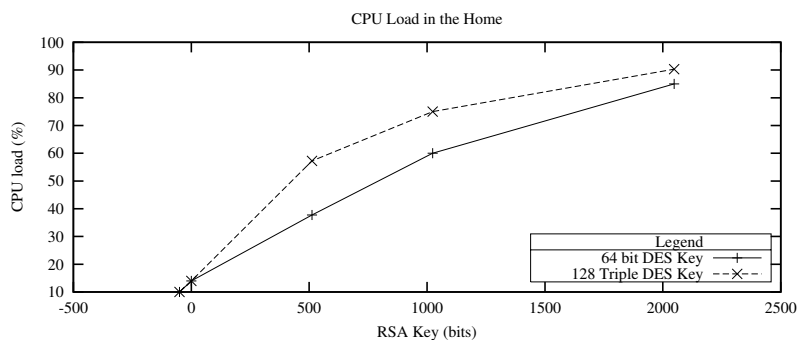


Figura 3.11: CPU en el Host Origen

CPU en el Host Origen

En la Figura 3.11 se muestra la carga de CPU media cuando el host origen recupera los resultados y verifica si hay hosts sospechosos. Del mismo modo que en el resto de parámetros, el uso de SDP tampoco comporta un gran incremento en el consumo de CPU, ya que la carga sin SDP (10%) no dista mucho de la carga con SDP y sin criptografía (14%).

Tamaño del Agente

En la Figura 3.12 se muestra el tamaño del agente en bytes. Esta medida nos puede dar una idea del incremento que sufre la carga de la red debido a SDP. En todas las medidas tomadas se ha tomado Triple DES como algoritmo de clave simétrica. Como se puede observar, el agente móvil incrementa su tamaño a medida que va visitando los host del itinerario. Este incremento se debe a que el agente almacena información durante su trayecto. Tal como se puede apreciar, las diferencias entre el agente sin SDP y el agente con SDP y sin criptografía siguen siendo mínimas. También cabe destacar que la complicación (y por tanto el tamaño) del agente aumenta en el momento que se requiere del uso de criptografía.

3.8. Conclusiones del Capítulo

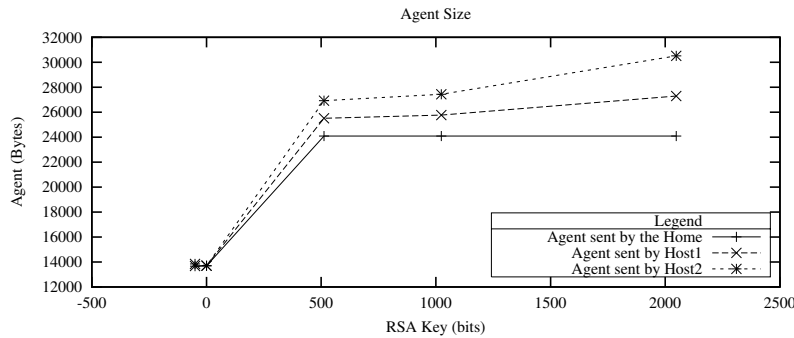


Figura 3.12: Tamaño del Agente

3.8. Conclusiones del Capítulo

Como primera contribución de esta Tesis se han solucionado los principales inconvenientes de la propuesta de detección de ataques más conocida, la de las trazas criptográficas de Vigna[Vig98]. Para ello se ha introducido un nuevo Protocolo de Detección de Sospechosos que utilizado conjuntamente con las trazas nos ofrece un mecanismo de detección de ataques efectivo e implementable.

De forma muy resumida, el protocolo está basado en controlar el tiempo que dispone un host para ejecutar el agente. Todos los hosts deben guardar los tiempos de llegada del agente y de finalización de la ejecución, de manera que cuando el agente vuelve al host origen, se puede verificar la coherencia temporal de la ejecución. Finalizadas las comprobaciones temporales se obtiene una lista de hosts sospechosos, a los cuales se les pueden pedir las trazas. El protocolo también da protección criptográfica al agente y a los datos que transporta, en caso de ser requerida.

Dadas las limitaciones de SDP, se recomienda solamente su uso en entornos donde el agente móvil vuelva al host origen transportando los resultados de la ejecución. Asimismo, se desaconseja su uso cuando sea difícil estimar el tiempo de ejecución del agente, como por ejemplo cuando el agente deba computar grandes volúmenes de datos.

Atendiendo a los resultados presentados en la evaluación del rendimiento de SDP,

Capítulo 3. Protocolo de Detección de Sospechosos (SDP)

el uso del mismo no supone un incremento sustancial para ninguno de los parámetros representativos del sistema y por tanto es viable su aplicación. En caso de ser necesaria protección criptográfica para el agente, se recomienda el uso de Triple DES como algoritmo simétrico, así como RSA como algoritmo de clave pública, con una longitud de clave de al menos 1024 bits.

3.8. Conclusiones del Capítulo

Capítulo 4

Watermarking de Agentes Móviles (MAW)

4.1. Nuevo Esquema de Detección de Ataques

En nuestra opinión, los esquemas de detección de ataques existentes son todavía demasiado costosos. Aún habiendo paliado en el Capítulo 3 los principales inconvenientes de la propuesta de las trazas criptográficas [Vig98] con la adición de un protocolo de detección de sospechosos, consideramos que esta propuesta aun comporta un gran esfuerzo computacional para todas las entidades involucradas. En primer lugar, el tamaño de aquello que constituye la prueba del carácter malicioso de un host (las trazas) depende del volumen de datos de entrada que tenga el agente. Las trazas pueden ser relativamente pequeñas, o bien pueden tener un tamaño considerable que haga prácticamente imposible su envío al host origen. Además, los hosts deben almacenarlas durante un periodo de tiempo, con el fin de dar respuesta a una posible petición de las trazas. Por otra parte, la verificación de la integridad de la ejecución no se realiza sobre todos los hosts sino solamente sobre aquellos que son sospechosos. Asimismo, el proceso que debe realizar el host origen para verificar la integridad de la ejecución en sí ya es complicado, ya que obliga a restablecer exactamente las mismas

4.2. Watermarking de Software

condiciones en las que se encontraba el agente para poder ejecutarlo de nuevo.

Es por ello que como segundo objetivo de esta Tesis se introduce un nuevo mecanismo de detección de ataques alternativo y menos costoso computacionalmente que el de las trazas criptográficas de Vigna. En la nueva propuesta, el tamaño de las pruebas es configurable, con lo cual pueden ser lo suficientemente pequeñas como para que el agente pueda transportarlas durante su trayecto sin que exista una gran pérdida en el rendimiento de la red. Con ello se consigue que el host origen disponga de todas las pruebas cuando regresa el agente, con lo cual ya no es necesario actuar en base a sospechas, sino que es posible verificar la integridad de la ejecución de todos los hosts. Además, tampoco es preciso almacenar dichas pruebas en los hosts. En cuanto al proceso de verificación, éste es más sencillo y no requiere de la reejecución del agente.

El nuevo mecanismo utiliza técnicas de watermarking de software para empotrar una marca digital en el agente, esto es, Watermarking de Agentes Móviles (*Mobile Agent Watermarking* o MAW)[EFS⁺03b, EFS03a]. La ejecución del agente creará un contenedor de datos donde se mezclan los resultados de la ejecución y la marca digital. El contenedor es la prueba que utilizará el host origen para determinar si un host actuó maliciosamente. Cuando el agente vuelve, el host origen busca la marca digital en el contenedor, de manera que si la marca ha sido alterada implica que el host modificó el agente y por tanto ha tenido un comportamiento malicioso.

4.2. Watermarking de Software

Las técnicas de watermarking tienen como objetivo la protección del copyright de contenidos digitales. Para ello, el distribuidor empotra una marca de agua¹ en el envoltorio u objeto digital a proteger, de manera que con ello es posible demostrar la autoría del mismo. La marca en sí suele ser un mensaje secreto que contiene información sobre el copyright del distribuidor o del autor.

La principal vulnerabilidad de los esquemas de watermarking deriva de la posi-

¹A partir de este punto nos referiremos a las marcas de agua simplemente como marcas.

Capítulo 4. Watermarking de Agentes Móviles (MAW)

bilidad que un usuario fraudulento modifique o elimine la marca. Un usuario que conozca la ubicación de la marca puede borrarla o modificarla para redistribuir el objeto asignándose la autoría del mismo. Incluso desconociendo la ubicación de la marca, se puede manipular el objeto protegido para intentar que la marca desaparezca. Es por ello que la mayoría de esfuerzos de investigación se centran en el empotrado de marcas que sean difíciles de localizar o modificar.

Una gran parte de los estudios realizados en el campo intentan proteger la propiedad intelectual de objetos de tipo multimedia, especialmente imágenes. Sin embargo, se ha prestado mucha menos atención a la protección de otro tipo de objetos, como por ejemplo el software. En la literatura se encuentran pocas contribuciones que intenten proteger la propiedad intelectual de programas informáticos mediante la adición de marcas, esto es, watermarking de software [SHKQ99, PKK⁺00, HHJ⁺00]. Sin embargo, en [CT99] encontramos una taxonomía de las principales técnicas de watermarking de software y los ataques que pueden recibir. En el artículo se definen tres principales parámetros a tener en cuenta en un esquema de watermarking de software: (1) el índice de datos (*data rate*), que nos da idea de la cantidad de información que puede ocultarse en el objeto digital o envoltorio; (2) la invisibilidad (*stealth*), que expresa lo imperceptible que es la marca a un observador; y (3) la resistencia (*resilience*), que es el grado de inmunidad que presenta el mensaje oculto a los ataques de un adversario. Por lo general, el incremento de uno de los parámetros acaba repercutiendo en los demás. A modo de ejemplo, es posible aumentar la resistencia de una marca simplemente repitiéndola varias veces a lo largo del documento protegido, pero esto va en detrimento de la cantidad de información adicional que puede ocultarse. Es por ello que cuando se empotra una marca en un código se deben tener en cuenta múltiples factores, como el lenguaje de programación utilizado (máquina o interpretado), la longitud que debe tener la marca respecto a la del código, o incluso el tipo de ataques que puede recibir por parte de un usuario deshonesto.

En [CT99] también se definen los dos principales tipos de watermarking de software: (1) watermarking estático, en el cual las marcas se almacenan en el código

4.3. Watermarking de Agentes Móviles (MAW)

o los datos de la ejecución; y (2) watermarking dinámico, en el cual las marcas se almacenan en el estado de la ejecución del programa. Los esquemas de watermarking estático no son robustos ante ataques basados en el uso de transformaciones que preservan la semántica de los programas, como por ejemplo la ofuscación, la transformación o la optimización de programas. Por su parte, los esquemas de watermarking dinámicos publicados hasta el momento son resistentes a alguna de dichas técnicas, pero no a todas ellas.

4.3. Watermarking de Agentes Móviles (MAW)

La protección de la propiedad intelectual del software y la protección de los agentes móviles frente a ataques de hosts maliciosos son dos problemas que, a pesar de tener múltiples puntos en común, no se habían tratado de conjugar hasta ahora en la literatura. En ambos problemas se tiene un código ejecutable que se mueve por la red y que debemos proteger de manipulaciones que pudiera sufrir. En el primer caso, el objetivo es detectar modificaciones en dicho código para proteger su copyright, mientras que en el segundo el objetivo es disuadir que hosts malintencionados modifiquen el código para obtener una ejecución favorable. Dado que son muchas las similitudes entre ambos problemas, es posible que puedan aprovecharse las soluciones planteadas en uno de los problemas para resolver las carencias del otro. De hecho, en ambos problemas se pueden utilizar técnicas similares para proteger el código, como por ejemplo la ofuscación [CTL97, Hoh98].

En este sentido, la segunda contribución de esta Tesis pretende aplicar la solución del watermarking de software al problema de los hosts maliciosos. En el resto del Capítulo se describe un nuevo esquema de detección de ataques de manipulación que utiliza el watermarking de software en agentes móviles, esto es, Watermarking de Agentes Móviles (*Mobile Agent Watermarking* o MAW). En este caso, la intención no es proteger el copyright, sino asegurar la integridad de la ejecución del agente. Para ello, utilizaremos el watermarking de software para empotrar una marca digital en el agente que más tarde será transferida a los resultados de la ejecución. En cada host,

Capítulo 4. Watermarking de Agentes Móviles (MAW)

el agente crea un contenedor donde va depositando diversos datos (por ejemplo datos de entrada, datos inútiles, datos provenientes de comunicaciones con otras máquinas o agentes, variables intermedias, etc) con una cierta ordenación, y los resultados. La agregación de todos los datos introducidos y sobre todo su distribución constituyen la marca digital. En suma, el contenedor es la ubicación donde se mezclan tanto los resultados como la marca. Cuando el agente vuelve al host origen, se verifica que la marca está impresa en cada uno de los contenedores mediante la aplicación de una serie de reglas de integridad. Si la marca es distinta a la esperada, es decir, si no se cumplen las reglas de integridad, quiere decir que el host ha actuado maliciosamente modificando el agente.

El uso de watermarking de software para resolver el problema de los hosts maliciosos presenta una serie de ventajas que no existen en el escenario de la protección del copyright. El objetivo primordial de un usuario que pretende usurpar el copyright de un código es borrar la marca o sustituirla por otra que indique que la autoría del código es propia. Para ello, los usuarios deshonestos pueden utilizar transformaciones que preservan la semántica de los programas, como por ejemplo la ofuscación, la transformación o la optimización de programas. Por el contrario, el objetivo de un host malicioso será modificar la ejecución del agente pero sin modificar la marca empostrada en el contenedor, ya que cualquier modificación sobre la misma revelaría una actitud maliciosa por parte del host. Es por ello que en el MAW podremos utilizar esquemas de watermarking de software con poca resistencia y mucha invisibilidad, ya que un host sin conocimiento de dónde está la marca sólo puede realizar cambios aleatorios en el agente. Asimismo, el uso de transformaciones que preservan la semántica de los programas (ofuscación, transformación u optimización de programas) no afecta al agente porque éstos no alteran la integridad del contenedor.

Si se pretende reclamar la autoría de un programa mediante watermarking de software, todas las copias distribuidas del programa tienen la misma marca empostrada en el mismo envoltorio. Por el contrario, en MAW el envoltorio es distinto para cada host, ya que la información depositada en el contenedor depende de la ejecución. Por tanto, el empostrado de la misma marca en todos los contenedores haría

4.3. Watermarking de Agentes Móviles (MAW)

al esquema vulnerable frente ataques de confabulación, en el cual varios hosts comparan sus contenedores para determinar qué posiciones son iguales y por tanto son susceptibles de ser marca. Para hacer el esquema resistente a este tipo de ataques, haremos que la marca empotrada dependa de la ejecución. Sin embargo, la forma de verificar la integridad de la marca (las reglas de integridad) sí será la misma para todos los hosts.

4.3.1. Empotrado de la Marca

El objetivo del MAW es detectar manipulaciones que pueda realizar un host durante la ejecución del agente. Para ello se utilizan técnicas de watermarking de software para modificar el agente e incorporarle una marca digital. Sin embargo, hay más aspectos a tener en cuenta antes de realizar la marcación de un agente con las técnicas habituales de watermarking de software. Cuando se marca un código para proteger su copyright, el distribuidor lo que persigue es modificar el código pero sin que ésto suponga ningún cambio respecto a las tareas que debe realizar, o lo que es lo mismo, la ejecución del código original y el código marcado debería tener la misma respuesta ante unos datos de entrada determinados.

Este comportamiento no es exactamente el que esperamos de nuestro esquema de detección MAW. Para asegurar que la ejecución se ha realizado de forma íntegra, o lo que es lo mismo, que el host no ha variado el código para que la ejecución le sea favorable, se necesita algún tipo de prueba que certifique que no han habido modificaciones. Además, esta prueba debe llegar al host origen para que se pueda verificar que en efecto ha sido así. Por tanto, cualquier modificación sobre el agente debe alterar las tareas que iba a realizar el agente original, ya que al menos se deben construir las pruebas de la integridad de la ejecución. Por tanto, modificaremos el agente original utilizando técnicas similares a las de watermarking de software², pero que transfieran la marca al contenedor, que nos servirá de envoltorio y de prueba de

²Hablamos de técnicas “similares” a las de watermarking de software porque, tal como hemos mencionado previamente, estrictamente hablando las modificaciones realizadas por técnicas de watermarking de software no suponen ningún cambio sobre las tareas a realizar por el agente.

Capítulo 4. Watermarking de Agentes Móviles (MAW)

la integridad de la ejecución.

Cualquier modificación sobre el agente para empotrar la marca no puede realizarse al azar, ya que dichas modificaciones dictaminarán la forma en que se transfiere la marca al contenedor. El contenedor resultante debe ser tal que cumpla una cierta ordenación lógica de la información y unas relaciones específicas entre los datos introducidos y los resultados. De tales modificaciones el host origen debe inferir las reglas de integridad que servirán para verificar la bondad de la ejecución de los hosts.

Otro factor importante es dónde se incorpora la marca dentro del agente. Atendiendo a que los agentes están compuestos básicamente de código y datos, estas son las alternativas que se dispone para empotrar la marca:

- **Código marcado:** el host origen empotra la marca en el código original del agente, con lo cual todos los hosts del itinerario comparten el mismo código marcado. Esta es la opción más aconsejable ya que sólo se debe marcar un objeto que es común para todos los hosts.
- **Datos marcados:** el host origen puede asignar distintos datos de entrada para cada host, con lo cual si queremos marcar los datos, el empotrado se deberá realizar sobre varios envoltorios. Es por ello que se desaconseja el empotrado de la marca en los datos de entrada.
- **Código ofuscado marcado:** existe la posibilidad de ofuscar el código y los datos antes de empotrar la marca. Las técnicas de ofuscación hacen que el código sea más difícil de analizar, y por tanto también más difícil de manipular. Marcar un código ofuscado dificulta la tarea de discriminar qué partes del código son marca de aquellas que no lo son. Esta forma es la más segura, pero también es la más cara computacionalmente hablando. De hecho, Hohl en [Hoh98] ya utilizaba la ofuscación como método de protección de agentes.

4.3.2. Transferencia de la Marca

Tal como hemos mencionado previamente, el host origen necesita pruebas que certifiquen que el agente fue ejecutado correctamente. Dichas pruebas se deben generar

4.3. Watermarking de Agentes M3viles (MAW)

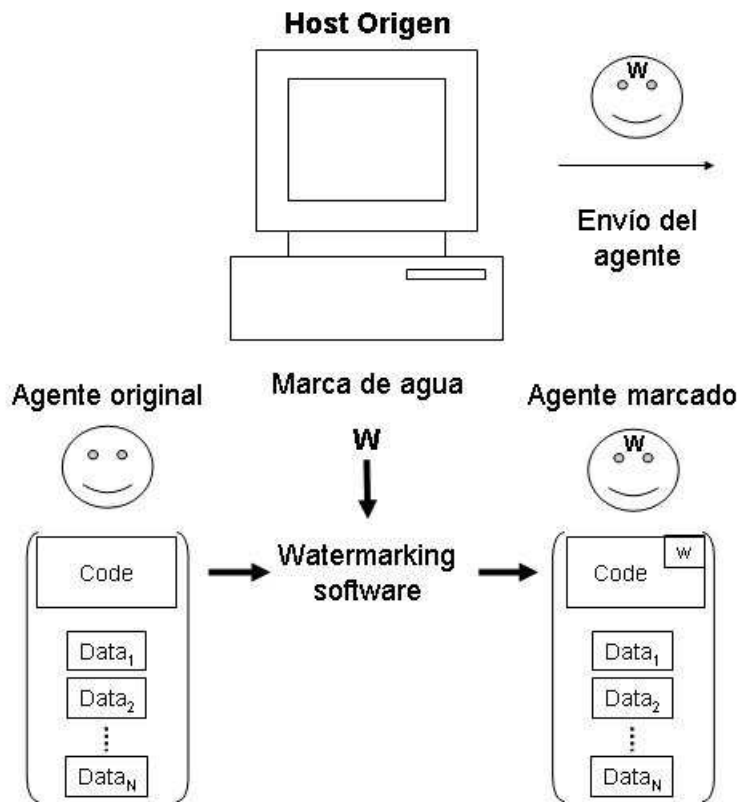


Figura 4.1: Empotrado de la Marca

Capítulo 4. Watermarking de Agentes Móviles (MAW)

durante la ejecución y deben regresar con el agente al host origen. En la propuesta del MAW, las pruebas no son otra cosa que el contenedor donde el agente transfiere la marca que lleva empotrada y donde también se encuentran los resultados. En la Figura 4.2 se muestra dicho proceso de transferencia de la marca. El contenedor no es más que una estructura de datos donde el agente introduce información durante la ejecución. La forma en que se introduce la información constituye en sí el proceso de transferencia de la marca. Obviamente, dicha transferencia se realiza siguiendo unos patrones lógicos que vienen determinados por las modificaciones realizadas sobre el agente. En cuanto a la información introducida en el contenedor, puede provenir de muy distintos sitios y tener distinta naturaleza, como por ejemplo:

- Valores constantes conocidos y prefijados por el host origen.
- Valores que pueden cambiar dinámicamente y que dependen de la ejecución, pero que cumplen una serie de reglas lógicas dentro del contenedor. El agente puede introducir cualquier dato que esté a su disposición, desde datos de entrada, datos inútiles [Mea97], datos provenientes de comunicaciones con otras máquinas e incluso variables intermedias.
- Los resultados de la ejecución³.

Para dificultar el análisis a un posible observador, el agente debe introducir confusión (modificación de valores) y difusión (repetición de valores) en los datos del contenedor. Asimismo, la ubicación de los datos dentro del contenedor puede ser la misma para todos los hosts, o bien depender del flujo de la ejecución.

En la Figura 4.3 se muestra cómo un agente migra de máquina en máquina ejecutando su código y creando el contenedor que transportará los resultados y la marca. Una vez finalizada la ejecución, cada host debe firmar y cifrar el contenedor para asegurar la confidencialidad e integridad del mismo.

³El contenedor puede ser utilizado como prueba de la integridad de la ejecución incluso en aquellos escenarios donde no existan resultados de la ejecución.

4.3. Watermarking de Agentes Móviles (MAW)

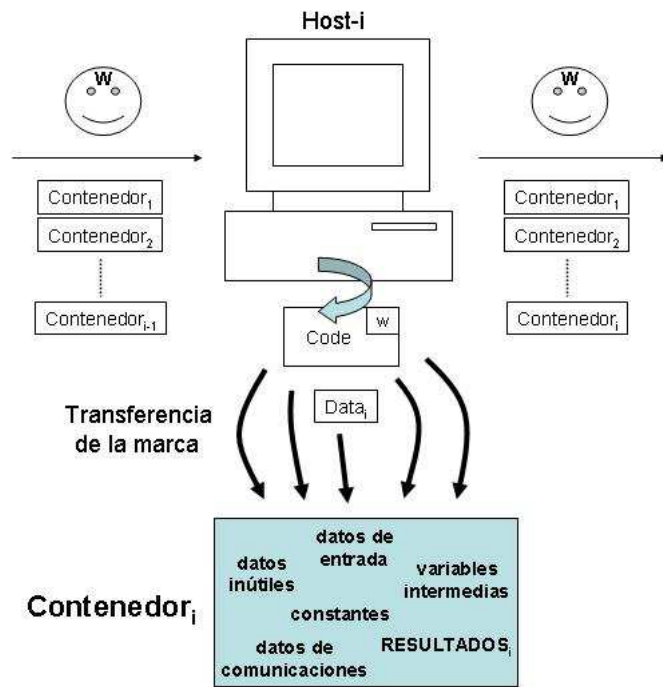


Figura 4.2: Transferencia de la Marca

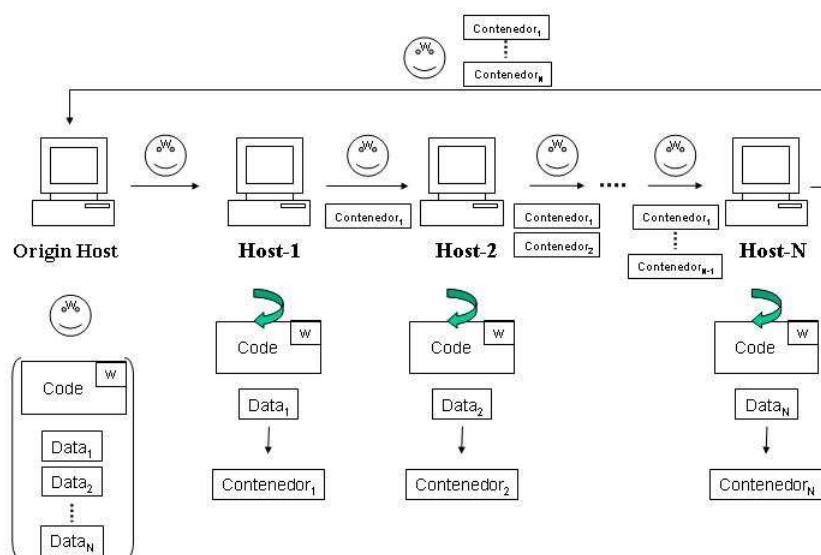


Figura 4.3: Migración del agente con MAW

4.3.3. Verificación de la Marca y Extracción de Resultados

Una vez finalizada la ejecución en el último host del itinerario, el agente debe volver al host origen para presentar los resultados. Sin embargo, antes de hacérselos llegar al usuario, se debe verificar que la ejecución ha sido íntegra, ya que si no lo fuera los resultados podrían verse comprometidos y deberían descartarse. El agente transporta un contenedor como prueba del comportamiento de cada host. Asumiremos que un host actúa honestamente ejecutando el agente si la marca esperada se encuentra en su contenedor, o lo que es lo mismo, que el contenedor de dicho host satisface las reglas de integridad. Si la marca no es la esperada, o lo que es lo mismo, si el contenedor no cumple las reglas de integridad, se puede asegurar que el host ha actuado maliciosamente. La marca empotrada en cada contenedor es distinta para cada host. Sin embargo, las reglas de integridad son las mismas para todos los hosts ya que han sido inferidas del código marcado, que es común para todos los hosts.

En segundo lugar, asumiendo que todos los host han actuado de forma honesta,

4.3. Watermarking de Agentes Móviles (MAW)

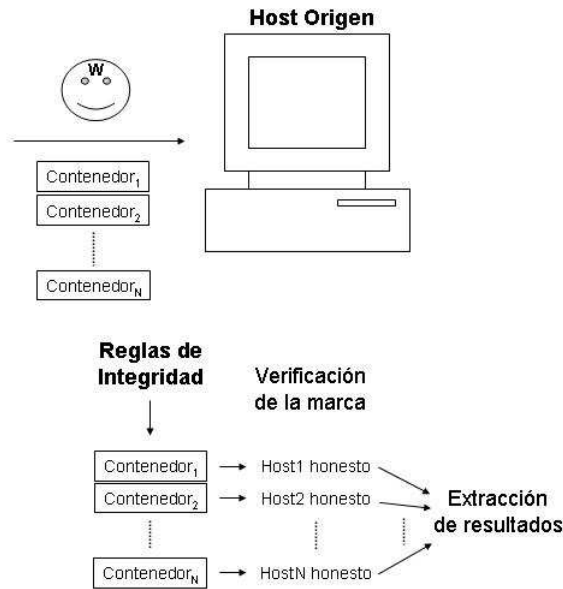


Figura 4.4: Verificación de la Marca y Extracción de Resultados

los resultados deben localizarse dentro de alguno de los contenedores. Los resultados no tienen por qué encontrarse en todos los contenedores simplemente ya que probablemente serán desconocidos hasta que se alcance el último host. Por tanto, los resultados casi siempre se encontrarán en el contenedor del último host, lo cual no implica que también se puedan encontrar en contenedores previos si este último host no ha sido relevante en la ejecución. Para localizarlos, dentro de los contenedores hay reglas específicas para los resultados.

En la Figura 4.4 se muestra el proceso de verificación de la marca en los contenedores, así como la extracción de los resultados de la ejecución para presentárselos al usuario.

4.3.4. Ejemplo de Funcionamiento del MAW

Para ilustrar la utilidad del uso de las técnicas de MAW, a continuación se introduce un ejemplo sencillo de marcado del agente. En la Figura 4.5 se muestran algunas modificaciones simples que pueden aplicarse sobre el ejemplo del agente de búsqueda de un vuelo del Capítulo 2. Es preciso puntualizar que en un escenario real las modificaciones a realizar sobre el agente para protegerlo de ataques deben ser mucho más sofisticadas que las del ejemplo. Asimismo, toda la información que transporta el agente debería protegerse mediante cifrado y firma digital. Es obvio que si nuestro principal objetivo es dar seguridad al sistema de agentes móviles, la primera medida a tomar debe ser dar protección criptográfica a los contenidos que viajan a través de la red. Sin embargo, en el ejemplo de la Figura 4.5 no se ha incluido este tipo de protección criptográfica para hacer el agente más inteligible.

La idea básica es introducir difusión y confusión dentro del contenedor. Entendemos por difusión cualquier tipo de técnica que disperse la información dentro del contenedor, bien sea repitiendo dicha información varias veces, bien sea partiéndola en segmentos y distribuyéndola en distintas ubicaciones. Asimismo, utilizamos el término confusión para denotar cualquier modificación de la información que altere su valor o apariencia para hacerla irreconocible a un observador. Para ello, la información puede enmascarse para evitar búsquedas de patrones en el contenedor, por ejemplo modificando dicha información mediante la adición de una constante.

Estas son algunas técnicas que se utilizan en el ejemplo de cara a introducir difusión y confusión:

- El agente puede introducir en el contenedor cualquier tipo de dato que esté a su disposición, como por ejemplo datos de entrada, valores de variables intermedias, datos inútiles, constantes o datos que provengan de comunicaciones con otras máquinas. Dichos datos y la forma en que se ubican dentro del contenedor forman parte de la marca y servirán para ocultar la ubicación real de los resultados de la ejecución. Cuanto más variada sea la procedencia de los datos, más difícil será discriminar los resultados reales de la marca. En particular, el

4.3. Watermarking de Agentes Móviles (MAW)

DATA BLOCK

```
City c1= Barcelona;
City c2=Madrid;
City c3=Paris;
Adress d=myPC;
Address i]=[Airfrance,Iberia,Spanair];
Integer b=0;
Address a1;
Address a2;
Integer p1=753;
Integer p2=1154;
Integer f[][];
```

CODE BLOCK

```
1 public void example_agent () {
2 Integer h,j,m,n,k=0,z=3;
3 h=i[b].flightprice(c1,c3)*2+107;
4 f[b][k+z]=h+234;
5 j=i[b].flightprice(c2,c3)*3+125;
6 m=p1-h+132+z*3;
7 k++;
8 f[b][k+z]=j-120;
9 z- -;
10 n=p2-j+73+k*5;
11 if (m>87) p1=h-54;
12 if (m<=87) p1=m+h-141;
13 f[b][z-k]=p1+132;
14 z- -;
15 f[b][z-k]=b*4+5;
16 if (n>57) p2=j-21;
17 if (n<=57) p2=n+j-78;
18 z- -;
19 f[b][2*k-z]=p2+56;
20 z++;
21 f[b][3*(z+k)]=h+j;
22 k++;
23 f[b][3*k-z]=p1+p2;
24 if (b >=(i.lenght -1) go(d);
25 go(i[++b]);
26 }
```

Figura 4.5: Agente de Búsqueda con MAW

Capítulo 4. Watermarking de Agentes Móviles (MAW)

agente del ejemplo incorpora dentro del contenedor datos de entrada, variables intermedias, datos inútiles y alguna constante.

- Los nombres de los elementos que componen el código (variables, clases, métodos, etc) no deben dar ningún tipo de información sobre el uso que se da de ellos. Toda la información que pueda ocultarse a un posible observador reduce la posibilidad de análisis y de modificación. Por ejemplo, en el agente de la Figura 4.5 los contenedores se ubican dentro de la matriz de enteros $f[][]$.
- Los datos y la ubicación de dichos datos dentro del contenedor deben depender del flujo de la ejecución. De esta manera se dificulta el posible seguimiento de la ejecución que pueda realizar un observador. En particular, en el ejemplo se utilizan las variables k y z para decidir la ubicación de los datos dentro del contenedor.

En general, el uso de cualquier tipo de técnica que haga más difícil el análisis del agente irá en favor de la seguridad del sistema. De hecho, en el ejemplo se utiliza un mecanismo parecido al de diseminación de intenciones presentado en [Ng02], que consiste en hacer que el agente demande datos o realice tareas en las cuales el usuario no está interesado. En particular, el agente de la Figura 4.5 se interesa por el precio de un vuelo entre Madrid y París cuando en realidad sólo está interesado en el vuelo entre Barcelona y París. A priori un observador no sabe cuales son los objetivos reales del usuario, o lo que es lo mismo, no sabe si al usuario le interesa el vuelo desde Madrid o desde Barcelona, con lo cual los hosts maliciosos deberán trabajar en varios frentes simultáneamente y tratar de modificar el agente para sacar partido en ambos objetivos. Cuantos más objetivos se incorporen, mayor será la dificultad que tendrán los hosts maliciosos para modificar el agente porque deben trabajar para obtener beneficios de todos ellos. Asimismo, las técnicas de ofuscación del código [CTL97] están especialmente indicadas para dificultar el análisis del agente. De hecho, en [Hoh98] Hohl plantea varios posibles algoritmos de enredado (*mess-up*) para dificultar el análisis del código: (1) recomposición de variables, en el cual los contenidos de las variables se separan en distintos segmentos con los cuales se crean nuevas variables

4.3. Watermarking de Agentes Móviles (MAW)

temporales recomponiendo las anteriores. Así el significado real de las variables se pierde e identificar los deseos del agente es mucho más complicado; (2) transformación del flujo de la ejecución, para dificultar el seguimiento de la misma se puede alterar el flujo de la ejecución mediante la adición de saltos que dependan de valores; y (3) cifrados parciales, en el cual se cifran partes del agente, que no estarán disponibles hasta que se comunique la clave de descifrado. Este último mecanismo es similar al de generación de claves dependientes del entorno [RS98].

Para que el ejemplo sea más clarificador, en la Figura 4.6 se muestran algunos parámetros relevantes de una posible ejecución del anterior agente. En la misma se ha asumido que todos los servidores actúan honestamente y dan al agente los precios que están en su base de datos, mostradas también en la Figura. Se incorporan los valores de las variables $p1$ y $p2$ para cada salto del agente, ya que en dichos valores se encuentran enmascarados los precios de los vuelos de Barcelona a París y de Madrid a París respectivamente. Como en realidad el usuario sólo está interesado en el vuelo desde Barcelona a París, la variable $p1$ es la que contiene el resultado real de la búsqueda. De todos modos, el resultado no sólo se encontraba en $p1$ sino que también se encuentra oculto en algunas posiciones dentro de los contenedores ubicados en la matriz $f[][]$.

Atendiendo a las modificaciones realizadas sobre el agente de la Figura 4.5, el host origen podría inferir una serie de reglas con las cuales certificar si la ejecución ha sido íntegra o no. Algunas de dichas reglas de integridad se muestran en la Figura 4.7, las cuales pueden aplicarse sobre los contenedores para verificar que los hosts actuaron honestamente.

Las reglas pueden clasificarse en tres tipos principales: reglas dentro del contenedor, reglas entre contenedores y reglas relativas a los resultados. Las reglas dentro de un contenedor deben aplicarse sobre cada uno de los contenedores por separado, ya que aplican a la ejecución particular de un único host. Éstas nos dan idea de si la ejecución en dicho host ha sido íntegra. Las reglas entre contenedores intentan buscar relaciones entre los distintos contenedores, con lo cual implican la ejecución de varios hosts. Éstas nos dan idea de la coherencia del agente durante todo su trayecto,

Capítulo 4. Watermarking de Agentes Móviles (MAW)

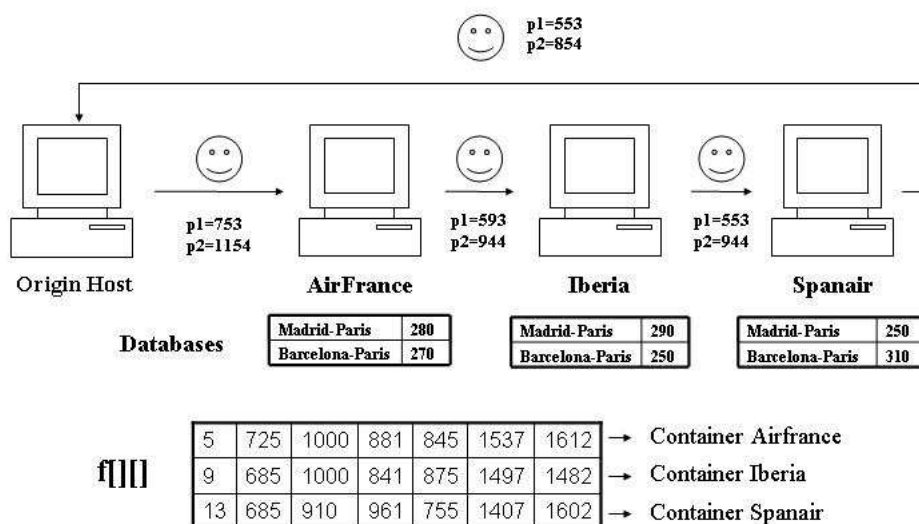


Figura 4.6: Ejecución del Agente de Búsqueda con MAW

teniendo en cuenta todos los hosts del itinerario. Finalmente, las reglas relativas a los resultados son las que utiliza el host origen para extraerlos de la ejecución de las distintas posiciones en que se encuentran ocultos dentro de los contenedores. Como se puede apreciar, el resultado no sólo se encuentra en la variable $p1$ sino que puede también encontrarse en el contenedor del último servidor, más concretamente en las posiciones $f[2][1]$ y $f[2][6]$.

4.3.5. Ventajas

El objetivo perseguido es encontrar una propuesta de detección de ataques computacionalmente menos costosa que las existentes, especialmente más ligera que la propuesta de las trazas criptográficas de Vigna [Vig98]. Ésta es la propuesta de detección de ataques más conocida con diferencia, y bajo nuestro criterio la que más posibilidades de ser implementada tenía de las publicadas hasta el momento. Sin embargo todavía no han sido especificadas las ventajas que presenta la nueva prop-

4.3. Watermarking de Agentes M3viles (MAW)

REGLAS DENTRO DEL CONTENEDOR

$$f[i][0]=i*4+5$$

$$f[i][3]>=f[i][1]+156$$

$$f[i][4]+155>=f[i][2]$$

$$f[i][1]+f[i][2]=f[i][5]+188$$

$$f[i][3]+f[i][4]=f[i][6]+144$$

REGLAS ENTRE CONTENEDORES

$$f[0][1]>=f[1][1]>=f[2][1]>=...>=f[N-1][1]$$

$$f[0][2]>=f[1][2]>=f[2][2]>=...>=f[N-1][2]$$

$$f[0][5]>=f[1][5]>=f[2][5]>=...>=f[N-1][5]$$

REGLAS RELATIVAS A LOS RESULTADOS

$$\text{Result}=(p1-53)/2$$

$$\text{Result}=(f[N-1][1]-185)/2$$

$$\text{Result}=(f[N-1][6]-f[N][2]+3)/2$$

donde N es la longitud del itinerario del agente

Figura 4.7: Reglas de Integridad que Deben Cumplir los Contenedores

Capítulo 4. Watermarking de Agentes Móviles (MAW)

uesta del watermarking de agentes móviles con respecto a la propuesta de la trazas criptográficas. A continuación se exponen dichas ventajas:

- El tamaño de las pruebas que aseguran la integridad de la ejecución debe ser lo suficientemente pequeño como para que el agente pueda transportarlas de vuelta al origen. En el caso del MAW, el tamaño del contenedor es configurable y puede determinarlo el programador. Por el contrario, el tamaño de las trazas depende del volumen de datos de entrada del agente, que puede ser muy grande. Ésta era la razón por la cual era desaconsejable en envío de las trazas al host origen.
- El host origen debe poder verificar la integridad de la ejecución de todos los hosts. Esto es posible con la utilización de MAW ya que el host origen dispone de los contenedores de todos los hosts. Por el contrario, con el uso de trazas el host origen se ve limitado a actuar en base a sospechas, y por tanto sólo se verifica la ejecución de los hosts sospechosos.
- Los hosts no deben almacenar ningún tipo de información. Los servidores difícilmente asumirían la necesidad de reservar una gran capacidad de almacenamiento para guardar las pruebas de su buen comportamiento. Con el uso del MAW, toda prueba se envía con el agente al host origen y los hosts no deben almacenar ningún tipo de información. Por el contrario, las trazas deben ser almacenadas durante un tiempo (aunque este tiempo sea reducido) por si se diera el caso que el host origen las demandase.
- La verificación de la integridad de la ejecución debe ser un proceso sencillo. En el caso de MAW, el host origen debe aplicar unas reglas de integridad sobre los contenedores. Por el contrario, con el uso de trazas es necesario reejecutar el agente de nuevo. Esto puede ser muy difícil, e incluso imposible si dichos datos de entrada dependen de las condiciones en que se encuentra el entorno, el tiempo o datos volátiles desaparecidos del sistema.

4.3.6. Inconvenientes

Tal como se ha especificado anteriormente, la propuesta de MAW presenta una clara ventaja computacional si la comparamos con la propuesta de las trazas criptográficas de Vigna. Como es lógico, el watermarking de agentes no carece de inconvenientes que pueden limitar su uso en determinados escenarios. Si lo comparamos con el envío del agente sin ningún tipo de protección, la propuesta todavía requiere un esfuerzo computacional para aquellas entidades que estén involucradas en la transacción. Es por ello que la mayoría de inconvenientes de la propuesta vienen dados por la pérdida de rendimiento que supone el esquema:

- El host origen debe hacer un esfuerzo inicial para empotrar la marca en el código original. Asimismo, basándose en dichas modificaciones, el host origen debe inferir las reglas de integridad.
- Al empotrar la marca, el tamaño del código sufre un incremento sustancial. Insertar una marca siempre implica que un cierto overhead debe añadirse al código. En primer lugar, este incremento afecta al rendimiento de la red, ya que el agente tiene un tamaño superior, si bien también supone un incremento del coste computacional que supone la ejecución del agente, ya que el código modificado es más complicado que el original.
- En cada ejecución, el agente crea un contenedor que debe transportar de vuelta al host origen, mientras que si mandáramos un agente sin protección sólo debería transportar los resultados. Esto obviamente afecta al rendimiento de la red y supone un incremento en el ancho de banda necesario.

Como se puede apreciar, los inconvenientes de la propuesta tienen que ver con la pérdida de rendimiento. Es por ello que las limitaciones de uso de la propuesta vendrán dadas por la capacidad que tengan las entidades involucradas o la red.

4.3.7. Ataques

El objetivo que persiguen los hosts maliciosos es la vulneración de la seguridad del agente. Estos son los principales ataques que pueden realizar contra la propuesta del MAW:

- Escuchas: cualquier dato no cifrado que contenga el agente puede ser leído o copiado por los hosts. El host origen puede modificar el agente para hacerlo más difícil de analizar, por ejemplo con técnicas como la ofuscación [CTL97]. Sin embargo, un host con suficiente tiempo puede analizar el agente y extraer conclusiones sobre las tareas que debe realizar y los datos que transporta.
- Manipulación: los hosts maliciosos pueden manipular cualquier parte del agente durante la ejecución. Pueden alterar el código, los datos, las comunicaciones o los resultados, pero intentando no modificar el flujo normal de la ejecución para no alterar la marca que se oculta en el contenedor, ya que cualquier alteración sobre la misma hará que el host sea detectado como malicioso. La fuerza del esquema estriba en hacer que la marca sea suficientemente imperceptible para un observador, ya que un host sin conocimiento de qué partes del código afectan a la marca no podrá manipular el agente sin tener peligro de ser detectado. Sin embargo, un host con suficiente tiempo podría llegar a inferir cuáles son las intenciones del agente e identificar qué partes del código afectan a la marca. Incluso podría deducir algunas de las reglas de integridad que deben aplicarse a los contenedores. Aún con esa información, al host malicioso le queda la ardua y no trivial tarea de construir un contenedor que refleje una ejecución favorable y que cumpla con todas las reglas de integridad.
- Confabulación: un grupo de hosts que estén en el itinerario pueden tratar de confabular para vulnerar la seguridad del agente. Por ejemplo, podrían tratar de localizar dónde se encuentra la marca en sus respectivos contenedores simplemente comparándolos entre ellos. Sin embargo, este tipo de acciones no son efectivas ya que los datos incluidos en el contenedor son distintos al depender de la ejecución, con lo cual es difícil extraer información en base a comparaciones.

4.4. Conclusiones del Capitulo

En este capítulo se ha introducido un nuevo método que permite detectar ataques de manipulación realizados por hosts maliciosos. La propuesta utiliza técnicas de watermarking de software para empotrar una marca en el agente, esto es, watermarking de agentes móviles (MAW). Durante la ejecución, el agente crea en cada host un contenedor donde se mezclará la marca y los resultados de la ejecución. Dicho contenedor es la prueba que certifica la integridad de la ejecución del agente. Al llegar al host origen, se verifica que cada contenedor cumple una serie de reglas de integridad. Si el contenedor de un host no cumple con las reglas, quiere decir que dicho host es malicioso. La fuerza de la propuesta del MAW se basa en la dificultad que tendrá un observador en distinguir qué partes del agente afectan a la marca, ya que un usuario sin dicha información difícilmente podrá alterar la ejecución sin ser detectado. Aún sabiendo qué partes son marca dentro del contenedor, si el host quisiera modificar la ejecución para que le sea favorable debe ser capaz de construir un contenedor que cumpla con todas las reglas de integridad.

La propuesta, pese a representar un incremento en el tamaño del código y los datos que debe transportar el agente, supone ciertas ventajas respecto al resto de propuestas de detección de ataques, y en particular la de las trazas criptográficas de Vigna [Vig98]. Con el nuevo mecanismo no es necesario actuar en base a sospecha, pues el host origen dispone de los contenedores de todos los hosts. Adicionalmente, el método no requiere reejecutar el agente para verificar la integridad de la ejecución, sino que sólo es necesario aplicar unas reglas de integridad a los contenedores. Asimismo, libera a los hosts de la obligación de tener que almacenar las pruebas que certifican su buen comportamiento durante la ejecución.

Capítulo 5

Autoridad de Revocación de Hosts (HoRA)

5.1. Políticas de Castigo en Sistemas de Agentes Móviles

El objetivo principal de esta Tesis es el estudio de un sistema de protección de agentes eficaz y computacionalmente factible. La intención es evitar en la medida de lo posible los ataques de hosts maliciosos, y en caso que no fuera posible evitarlos al menos minimizar los efectos que producen. Consideramos que el arma más eficaz para llegar a la consecución de dicho objetivo es disuadir a los hosts a emprender actitudes maliciosas. Para ello hay dos factores que consideramos determinantes para evitar ataques:

- La existencia de un mecanismo de detección de ataques suficientemente eficaz. Como es lógico, cuanto mayor es el control que se ejerce sobre la ejecución, más posibilidades existen de detectar a aquellos que vulneran la seguridad del agente. En los Capítulos 3 y 4 se ha abordado el estudio de dos mecanismos de detección de ataques.
- La implantación de una política de castigo adecuada. Cualquier sistema de

5.1. Políticas de Castigo en Sistemas de Agentes Móviles

protección de agentes basado sólo en métodos de detección de ataques es claramente insuficiente. De poco sirve detectar ataques si no hay posibilidad de practicar algún tipo de castigo a los hosts que han actuado maliciosamente. Por lo general, cuanto mayor es el castigo, más difícil será que los hosts emprendan actitudes maliciosas. Los castigos dependerán del poder que tenga la entidad sancionadora, y pueden ser de tipo económico, administrativo, o simplemente pueden poner en entredicho la reputación del host que realiza el ataque. En este capítulo realizaremos el estudio de un mecanismo de castigo integrable con los dos esquemas de detección estudiados en los Capítulos 3 y 4.

Hasta el momento, poca o ninguna atención se ha prestado a las políticas de castigo en la literatura de sistemas de agentes móviles. En algunos artículos se dejaba entrever la necesidad de una entidad con capacidad para castigar hosts maliciosos [Vig98], sin embargo ningún autor hasta el momento había analizado el problema ni había tratado de implementar un sistema sancionador real. A continuación se enumeran las principales conductas que puede tomar el host origen cuando detecta un ataque:

- El host origen descarta los resultados de los hosts maliciosos. Si hay resultados parciales¹ y alguno de los hosts es detectado como malicioso, automáticamente se considera que dichos resultados parciales están comprometidos y por tanto son descartados directamente. Si por el contrario los resultados dependen de todos los hosts del itinerario (que es el comportamiento general), toda la ejecución se considera comprometida y se descarta cualquier resultado obtenido por el agente. En este caso sólo es posible reenviar el agente para realizar de nuevo la tarea indicada, pero eliminando del itinerario los hosts maliciosos. En esta política de castigo, la impartición de la sanción sólo depende del host origen, con lo cual es muy fácil de utilizar, si bien, es altamente insegura y se desaconseja su uso porque la sanción imprimida es muy débil. Si un host

¹Entendemos por resultado parcial de un host aquel que sólo depende de la ejecución que se realiza en dicho host.

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

vulnera la protección del agente sin ser detectado, automáticamente obtiene el beneficio de una ejecución favorable. Por el contrario, si el host es detectado como malicioso, como castigo simplemente se le deja de tener en cuenta para la transacción en curso. Además, no hay ningún impedimento que evite que dicho host vuelva a atacar a otro agente. En el peor de los casos, puede que el agente simplemente no genere resultados de la ejecución, con lo cual no hay ningún tipo de castigo.

- El host origen crea una lista negra. Dicha lista negra contiene todos los hosts maliciosos que han atacado a un agente propio, de manera que no se enviará ningún agente más a dichos hosts. Esta técnica es una mejora de la anterior, ya que se conserva un cierto histórico de las relaciones establecidas con otros hosts en transacciones previas. De esta manera, un host no podrá realizar más ataques a un determinado host origen a partir que haya sido detectado como malicioso. Por ejemplo, en un escenario de comercio electrónico, un host origen no volverá a realizar transacciones con un servidor que haya actuado maliciosamente con algún agente enviado por él. A pesar de que este tipo de política de castigo sigue siendo de muy sencilla aplicación (pues sólo depende del host origen), no consideramos que sea la adecuada pues los hosts maliciosos pueden continuar atacando a agentes pertenecientes a otros hosts origen.
- Un grupo de hosts origen unen sus listas negras. La nueva lista negra común contiene la unión de las listas negras particulares, con lo cual en ella encontraremos todos los hosts maliciosos que alguna vez atacaron a algún miembro del grupo. Si un host actúa maliciosamente con un agente de cualquier host origen del grupo, es añadido a la lista común y automáticamente deja de recibir agentes de dichos hosts origen. Sin embargo, este tipo de esquema obliga a que todos los hosts del grupo sean entre ellos de confianza. Esto se debe a que un host origen deshonesto puede perjudicar a un servidor honesto incluyéndolo en la lista negra común.
- Una TTP administra la lista negra común. Para evitar que la lista negra sea

5.2. Castigo Basado en Revocación de Hosts

gestionada por los hosts origen y evitar comportamientos deshonestos como el del caso anterior, se delega esta tarea a una entidad independiente y de confianza. El problema que presenta esta política de castigo es que se debe probar ante la TTP el carácter malicioso del host antes que sea añadido a la lista. La detección de ataques ya no es suficiente, los host origen deben presentar pruebas del carácter malicioso del host.

5.2. Castigo Basado en Revocación de Hosts

Como tercera contribución de la Tesis, en este Capítulo presentamos un mecanismo de castigo basado en la adición de una nueva entidad al sistema de agentes móviles, la Autoridad de Revocación de Hosts (*Host Revocation Authority* u HoRA) [ESMF03b]. La HoRA debe considerarse una Tercera Parte de Confianza (TTP) en el sistema de agentes móviles, de la misma manera que se considera una entidad de confianza a la Autoridad de Certificación (*Certificate Authority* o CA) en la Infraestructura de Clave Pública (PKI).

El objetivo de este mecanismo de castigo es discriminar los hosts maliciosos de los honestos. Desafortunadamente, no es posible adivinar si un host que ha sido honesto hasta el momento se tornará malicioso justo en la transacción en curso. Sin embargo, sí es posible conservar un histórico de aquellos hosts que han actuado maliciosamente en el pasado. Si un host actuó maliciosamente una vez, puede volver a hacerlo en el futuro. Es por ello que, para evitar futuros ataques, se debe evitar que dicho host vuelva a recibir agentes. Para conservar dicho histórico, la HoRA será la depositaria y gestora de una base de datos que contendrá toda la información de los hosts revocados, es decir, los hosts que se ha probado que actuaron de forma maliciosa en algún momento.

A continuación se resumen las dos principales tareas que debe realizar la HoRA para que pueda ser utilizada como entidad sancionadora:

- Consulta de Estado: antes de enviar un agente, el host origen consulta el estado de los hosts del itinerario para verificar si están revocados o no. Dicha infor-

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

mación de revocación es accesible bien consultando directamente a la HoRA, o bien consultando una lista de hosts revocados local que ha sido previamente descargada de la HoRA. Una vez se conoce el estado de los hosts, los revocados se eliminan del itinerario del agente, de manera que se impide que un host que actuó maliciosamente en el pasado ejecute de nuevo más agentes.

- **Revocación de Hosts:** la HoRA es la responsable de la información de revocación, y por ello deberá gestionar los contenidos de su base de datos interna para que estén todo lo actualizados que sea posible. En particular es responsable de la adición de nuevos hosts maliciosos. Para ello, los hosts origen deberán demostrar a la HoRA que los hosts que pretenden incorporar a la base de datos actuaron maliciosamente. En ese sentido, son necesarias técnicas de detección y prueba de ataques. En este Capítulo se introducen los protocolos necesarios para revocar hosts tanto si se utiliza el mecanismo de las trazas conjuntamente con SDP [Vig98, ESMF03a], como si se utiliza MAW [EFS⁺03b].

Como se puede apreciar, la finalidad del método de castigo es doble. Por un lado persigue castigar a los hosts maliciosos impidiendo que realicen nuevas ejecuciones, y por otro lado informa al resto de la comunidad que dichos hosts no son honestos y por tanto susceptibles de realizar ataques. Sin embargo, la eficacia de cualquier sistema de castigo basado en revocación de hosts depende en gran medida del uso que se dé de él. Si la inmensa mayoría de hosts origen consultan la información de revocación, los hosts maliciosos difícilmente podrán atacar agentes. Si por el contrario hay un gran volumen de hosts origen que no desean ningún tipo de protección y que por tanto no tienen en cuenta qué hosts han sido revocados, no sólo permiten que sus agentes sean vulnerables, sino que también restan efectividad a los que sí consultan la información de revocación. Esto se debe a que están dejando de informar al resto de hosts origen de los ataques que reciben. Lo mismo sucede cuando un host origen detecta un ataque y no inicia el proceso de revocación del host malicioso. Si no hay denuncia, el host malicioso no sólo queda impune sino que además el resto de hosts origen no tienen conocimiento que dicho host es malicioso.

5.3. Consulta del Estado

Antes de enviar el agente, el host origen debe consultar el estado (*Status Checking*) de los hosts del itinerario para verificar si están revocados o no. El objetivo de consultar el estado es eliminar del itinerario los hosts maliciosos. Sin embargo, cómo se accede a la información de revocación es un tema que debe ser estudiado porque afecta directamente al rendimiento del sistema.

Asumiremos que la HoRA actúa en muchos aspectos de manera similar a la Autoridad de Certificación (CA) de la PKI. Buscar este tipo de similitudes nos permitirá aprovechar toda la experiencia adquirida en la investigación referente a certificación, y particularmente en el ámbito de la revocación de certificados. Teniendo en cuenta la forma en que se realiza la consulta de estado en los entornos de revocación de certificados, asumiremos que existen dos políticas posibles de consulta del estado en el sistema de agentes móviles:

- Política de consulta *Off-line*: está basada en la distribución de la información de revocación en una lista de hosts revocados firmada por la HoRA. Los hosts origen deberán descargarla en su máquina, consultarla localmente e ir actualizándola periódicamente.
- Política de consulta *On-line*: está basada en el acceso directo a la información de revocación mediante el envío de consultas a la HoRA.

La decisión de cual de ambas es la más adecuada para acceder a la información de revocación en un sistema de agentes real dependerá de múltiples parámetros, como por ejemplo el ancho de banda disponible, la población de hosts origen o la capacidad de proceso de las entidades involucradas.

5.3.1. Política de Consulta Off-line

En la política de consulta off-line se asume que un host origen podría no tener conectividad directa con la HoRA. En ese caso le sería imposible consultar la información de revocación. Sin embargo, dichos hosts deben seguir mandando agentes,

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

y por tanto deben decidir a qué hosts mandarlos. La idea es poner la información de revocación a disposición de los hosts origen para que la consulten localmente y puedan tomar la decisión adecuada sobre el itinerario del agente.

La distribución de la información de revocación se realiza mediante una Lista de Hosts Revocados (*Host Revocation List* o HRL), que básicamente no es más que una copia de la base de datos interna de la HoRA. De hecho, la HRL funciona de una manera similar a la Lista de Certificados Revocados (*Certificate Revocation List* o CRL) [X.588, X.597, HFPS99], que es el paradigma de distribución de información de revocación off-line más utilizado en la PKI. Buscando paralelismos con la CRL X.509 v.2 [X.597], definimos los campos necesarios para la HRL, que se muestran en la Figura 5.1:

- *Version*: versión de la HRL. En este caso por tratarse de la primera versión de HRL este campo tendrá como valor 1.
- *Issuer*: identificador de la entidad emisora de la lista, es decir, la HoRA que esté a cargo del sistema de agentes móviles.
- *ThisUpdate*: fecha de creación de la HRL.
- *NextUpdate*: fecha en la que está prevista la emisión de una nueva HRL.
- *RevokedHosts*: encontramos una entrada por cada host revocado que contiene la siguiente información:
 - *HostID*: identificador del host revocado. Esto implica que debe existir un esquema de identificación global para los hosts en el sistema de agentes móviles. Los identificadores de los hosts deben ser únicos e inequívocos, bien sea para no revocar a un host honesto, o bien sea para no dejar sin revocar a un host malicioso².

²El uso de identificadores tales como nombres DNS o direcciones IP está sujeto al hecho que identifiquen plenamente a un único host dentro del sistema. Es por ello que en un sistema abierto como Internet se desaconsejan los identificadores de este tipo ya que existen técnicas, como por ejemplo NAT, que pueden ocultar la identidad de un host malicioso o incluso comprometer la de otros hosts honestos.

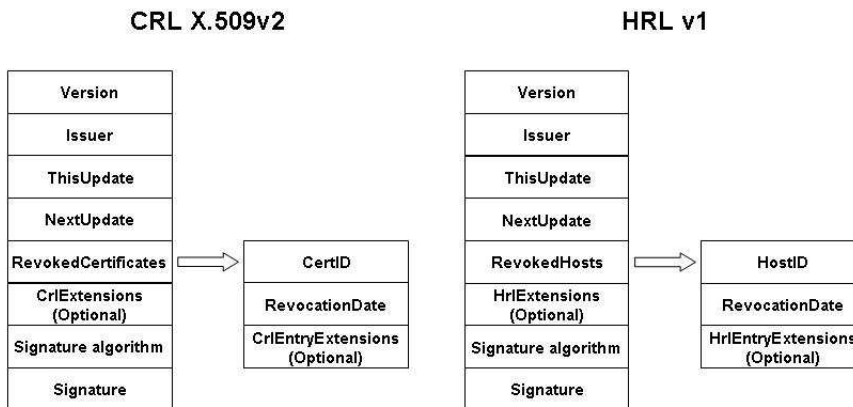


Figura 5.1: Formato de la HRL

- *RevocationDate*: fecha de revocación.
- *HrlEntryExtensions*: en este campo se puede incluir información adicional sobre la revocación, como por ejemplo la razón de la revocación o el host origen atacado.
- *HrlExtensions*: extensiones de la HRL.
- *SignatureAlgorithm*: algoritmo utilizado para la firma.
- *Signature*: firma de la HRL.

Los hosts origen que deseen consultar el itinerario en modo off-line pueden descargar una copia de la HRL para consultarla localmente. Además, deben actualizarla periódicamente para tener en cuenta los nuevos hosts revocados. El proceso de descarga podría realizarse directamente contra la HoRA, pero ello supondría un cuello de botella en el sistema. Para solucionar esta situación la HRL puede ser depositada en

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

repositorios³ para que los hosts origen la descarguen aprovechando que es una lista firmada y por tanto cualquiera puede verificar su autenticidad e integridad. Como es lógico, esto obliga a los repositorios a actualizar también periódicamente la lista que almacenan.

Desgraciadamente, el procedimiento basado en actualizaciones periódicas no es infalible y permite un margen de error. Un host que acaba de ser detectado como malicioso es incorporado a la base de datos interna de la HoRA y será incluido en la próxima HRL que sea emitida. Sin embargo, dicho host malicioso podrá continuar realizando ataques a aquellos hosts origen que no dispongan de la HRL actualizada. Obviamente, cuanto más corto es el periodo entre actualizaciones, menos tiempo están los hosts origen expuestos a los ataques de dichos hosts maliciosos. Por otra parte, actualizaciones demasiado frecuentes suponen un incremento considerable del ancho de banda necesario.

5.3.2. Política de Consulta On-line

En la política de consulta on-line el host origen accede directamente a la información de revocación que se encuentra en la HoRA. Para ello, utiliza el Protocolo del Estado del Host On-line (*Online Host Status Protocol* o OHSP). Tal como se muestra en la Figura 5.2, el host origen envía una petición a la HoRA con los identificadores de los hosts que se encuentran en el itinerario. La HoRA consulta su base de datos interna y devuelve una respuesta firmada indicando el estado de cada uno de los hosts. De hecho, este mecanismo funciona de una manera similar al protocolo OCSP (*Online Certificate Status Protocol*) [MAM⁺99], que es el protocolo de consulta on-line más utilizado en la PKI.

Existen múltiples razones por las cuales los hosts origen pueden optar por este modo de consulta de estado. Por ejemplo, un host origen que manda agentes esporádicamente no deseará mantener una HRL indefinidamente, sino que cuando sea necesario realizará una única consulta OHSP. También es recomendable el uso de este

³Entendemos por repositorio una ubicación en la red no necesariamente de confianza donde se dispone de capacidad de almacenamiento.

5.3. Consulta del Estado

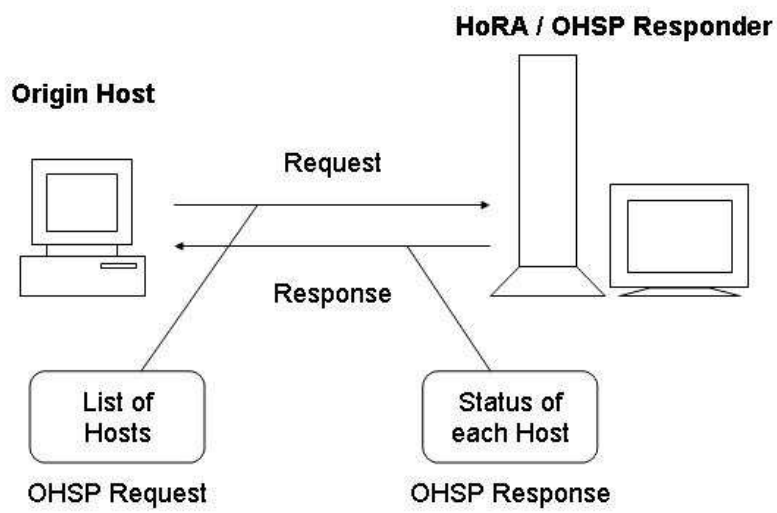


Figura 5.2: Consulta On-line con OHSP

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

tipo de consulta en aquellos hosts origen que tengan graves limitaciones en cuanto a capacidad de almacenamiento o ancho de banda, como sucede en los terminales inalámbricos.

Cabe destacar que, a diferencia de lo que sucedía con las consultas off-line, los hosts origen acceden a información de revocación de primera mano, ya que están accediendo directamente a la base de datos interna de la HoRA. Por tanto, si un host acaba de ser detectado y se incorpora a la base de datos interna de la HoRA, automáticamente dejará de recibir agentes de los host origen que realicen consultas de tipo on-line.

El esquema tal como se describe presenta un serio problema de centralización de recursos. Entre otras muchas tareas, la HoRA debe recibir todas las peticiones que provienen de los host origen y crear respuestas firmadas para cada una de ellas. Esto constituiría un cuello de botella para el sistema de consulta de estado, con lo cual es recomendable delegar esta tarea a un grupo de responders⁴.

5.4. Revocación de Hosts

Para permitir la consulta de estado, la HoRA debe almacenar y gestionar la información de revocación que se encuentra en su base de datos interna. De lo actualizada que esté dicha información puede depender que un host malicioso ataque más agentes. Como los hosts que han sido revocados no se eliminan de la base de datos, todas las actualizaciones de la información de revocación van enfocadas a la adición de nuevos hosts maliciosos.

Los hosts origen son los encargados de iniciar los procesos de revocación una vez se ha detectado un ataque, y es esencial que los inicien inmediatamente después de la detección. Tal como se mencionó anteriormente, la fuerza del mecanismo de castigo reside en el uso que se dé de él, y muy especialmente, de que todos los hosts origen que reciban ataques denuncien a los hosts maliciosos. Para poder revocar un host,

⁴Entendemos por responder una entidad de confianza en la cual podemos delegar algún tipo de tarea, teniendo ésta potestad para enviar respuestas firmadas.

5.5. *Protocolos de Revocación para las Trazas Criptográficas*

el host origen no sólo debe detectar el ataque sino que además debe disponer de pruebas que certifiquen que el ataque se llevó a cabo. Si es así, el host origen puede iniciar un protocolo de revocación que dependerá del mecanismo de detección de ataques utilizado. La función básica de todos los protocolos de revocación es enviar las pruebas a la HoRA para que ésta las evalúe, de manera que el host sólo se añade a la base de datos interna de la HoRA si finalmente las pruebas se consideran válidas. En el resto del Capítulo se introducen los protocolos necesarios para revocar hosts utilizando los dos mecanismos de detección de ataques estudiados en los Capítulos 3 y 4.

5.5. **Protocolos de Revocación para las Trazas Criptográficas**

En el Capítulo 3 se realizó la integración del mecanismo de las trazas criptográficas de Vigna [Vig98] con SDP [ESMF03a]. El uso conjunto de ambos nos ofrecía un mecanismo de detección y prueba de ataques efectivo e implementable. En el presente Capítulo pretendemos integrar el anterior mecanismo de detección con las funcionalidades sancionadoras de la HoRA. De esta manera no sólo se detectan los ataques sino que también es posible revocar a los hosts maliciosos. Para llegar a la integración plena de todos estos mecanismos se debe realizar el estudio de todas las fases por las que pasa el agente:

- Envío del agente: la forma en que se realiza la ejecución del agente es importante para que la HoRA dé por válidas las pruebas en caso de iniciarse un proceso de revocación.
- Verificaciones del host origen: en esta fase el host origen realiza las verificaciones pertinentes para localizar a los hosts maliciosos.
- Protocolos de revocación: en caso de detectar comportamientos maliciosos el host origen inicia un proceso de revocación, que básicamente consiste en el envío de las pruebas a la HoRA.

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

En primera instancia estudiaremos los pasos a realizar en cada una de las anteriores fases cuando la aplicación no requiera confidencialidad del agente [ESMF03e]. En este caso los protocolos de revocación serán más sencillos ya que no es necesario el uso de cifrado de la información. En segundo lugar estudiaremos las medidas adicionales para asegurar la confidencialidad del agente. En ambos casos sí se ha tenido en cuenta la integridad del agente y de los datos. Dichas condiciones son necesarias para que los protocolos de revocación funcionen, ya que sino cualquier entidad podría variar los datos del agente a su favor, impidiendo así el establecimiento de pruebas válidas para revocar un host.

5.5.1. Protocolos sin Privacidad

Envío del Agente

En primer lugar, asumiremos que el host origen ya ha consultado previamente la información de revocación, eliminando del itinerario todos los hosts maliciosos. La forma en que haya consultado el estado es irrelevante para el proceso de revocación, puede haber sido bien utilizando OHSP (on-line), o bien consultando una copia local de la HRL (off-line). Se obvian muchas de las tareas a realizar relativas a SDP [ESMF03a] y a las trazas [Vig98] ya que éstas han sido especificadas previamente en el Capítulo 3 y lo único que harían sería entorpecer la comprensión del ejemplo de funcionamiento que pretendemos explicar.

A continuación se describen los pasos que deben seguir las entidades involucradas para realizar el envío del agente, suponiendo que el itinerario del agente es de N hosts:

1. El host origen (O) envía el agente móvil al primer host del itinerario (Host-1), transportando el código⁵, los datos de entrada⁶ y una caducidad para las trazas o *TST* (*Traces Storage Timestamp*). Por tanto, el host origen envía el siguiente

⁵ Asumimos que el código es el mismo para todos los hosts del itinerario.

⁶ Los datos de entrada pueden ser distintos para cada host. De esta forma, los datos de entrada que envía el host origen corresponden con *Data_O*.

5.5. Protocolos de Revocación para las Trazas Criptográficas

agente al primer host del itinerario:

$$Agent_{O \rightarrow 1}(X_O)$$

donde

$$X_O = sign_O[Code, Data_O, TST].$$

Este tiempo TST indica a los hosts cuándo el host origen pierde el derecho a iniciar un proceso de revocación de hosts. Pasada dicha fecha de caducidad todas las pruebas pueden ser eliminadas porque dejan de tener validez. Obviamente, el valor de TST no debe ser escogido unilateralmente por el host origen, ya que de dicho valor depende el tiempo que los hosts deben tener almacenadas las trazas. Al host origen le convienen valores de TST altos (por ejemplo días o incluso meses a partir de la emisión del agente), ya que así dispone de más margen para iniciar la revocación de los hosts maliciosos. Por su parte, a los hosts les interesan valores de TST bajos (minutos, o como mucho horas), ya que así pueden borrar las trazas antes y liberar capacidad de almacenamiento. Desde el punto de vista de la efectividad del sistema de castigo, son convenientes valores de TST pequeños, ya que obligan a los host origen a revocar a los hosts maliciosos lo antes posible. De todos modos, lo adecuado sería que este tiempo fuera tomado por convenio en el sistema de agentes móviles, o bien se negociara en cada transacción. Independientemente del valor que tenga TST , éste debe incorporarse en el agente para que todas las partes implicadas (y especialmente la HoRA) lo tengan en cuenta.

2. El Host-1 recibe el agente y verifica que la firma de X_O es válida y corresponde al host origen. Si es así, extrae de X_O el código y los datos e inicia la ejecución. Una vez finalizada ésta, el agente se dispone a migrar al siguiente host transportando los resultados, los datos de entrada para el siguiente host (si fueran necesarios), las trazas y los tiempos de llegada del agente y finalización de la ejecución correspondientes a SDP. Por tanto, el agente que se envía al siguiente

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

host tiene el siguiente formato:

$$Agent_{1 \rightarrow 2}(X_1)$$

donde

$$X_1 = sign_1[X_O, Data_1, Results_1, H(Traces_1)].$$

En $Data_1$ encontramos los datos de entrada que el Host-1 deja para el siguiente host, si estos fueran necesarios. En $Results_1$ hemos ubicado tanto los resultados de la ejecución como los tiempos de llegada y finalización. Finalmente, como las trazas pueden tener un tamaño que desaconseja su transporte, se envía un hash de las mismas $H(Traces_1)$ al host origen como prueba. Las trazas completas sólo se enviarán en caso que el host origen considere sospechoso al Host-1. Mientras tanto, el Host-1 guarda las trazas $Traces_1$ hasta que la caducidad TST expire. Asimismo, también guarda un hash de X_1 que será utilizado para certificar que las trazas corresponden a esta ejecución. La firma del Host-1 en X_1 certifica que todos los parámetros (código, datos, tiempos, trazas y resultados) pertenecen a la misma ejecución.

3. Este proceso se repite por cada host del itinerario. De esta manera, el host i -ésimo del itinerario Host- i recibe el agente, lo ejecuta y obtiene los tiempos, las trazas, los datos de entrada y los resultados. Por tanto, el agente que se envía al siguiente host tiene el siguiente formato:

$$Agent_{i \rightarrow i+1}(X_i)$$

donde

$$X_i = sign_i[X_{i-1}, Data_i, Results_i, H(Traces_i)].$$

4. Finalmente, el último host del itinerario Host- N envía el siguiente agente al

5.5. Protocolos de Revocación para las Trazas Criptográficas

host origen:

$$Agent_{N \rightarrow O}(X_N)$$

donde

$$X_N = sign_N[X_{N-1}, Results_N, H(Traces_N)].$$

Como se puede apreciar el campo correspondiente a los datos de entrada $Data_{N+1}$ ya no es necesario.

5. El host origen recibe el agente y verifica que las firmas anidadas en X_N son válidas y corresponden a los hosts que dicen ser. Una vez verificada la integridad y autenticación de los datos que transporta el agente, se extraen todos los tiempos de llegada y finalización⁷ para iniciar el procedimiento de búsqueda de hosts sospechosos con SDP. Una vez se dispone de una lista de hosts sospechosos, a éstos se les demandan las trazas para verificar la integridad de la ejecución. Por ejemplo, suponiendo que sólo hay un host sospechoso y que éste es el j -ésimo host del itinerario Host- j , el host origen le enviaría un mensaje como el siguiente demandándole las trazas:

$$Message_{O \rightarrow j}(sign_O[sendTraces_j, H(X_j)]).$$

Se incorpora en el mensaje el hash $H(X_j)$ para que el Host- j envíe las trazas que corresponden a esta ejecución en particular.

6. El Host- j debe responder con un mensaje firmado que contiene las trazas completas y el mismo hash que certifica que las trazas corresponden con la ejecución llevada a cabo. El formato del mensaje es el siguiente:

$$Message_{j \rightarrow O}(sign_j[Traces_j, H(X_j)]).$$

⁷Simplemente recordar que dichos tiempos se encontraban ubicados en el campo de resultados para cada host.

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

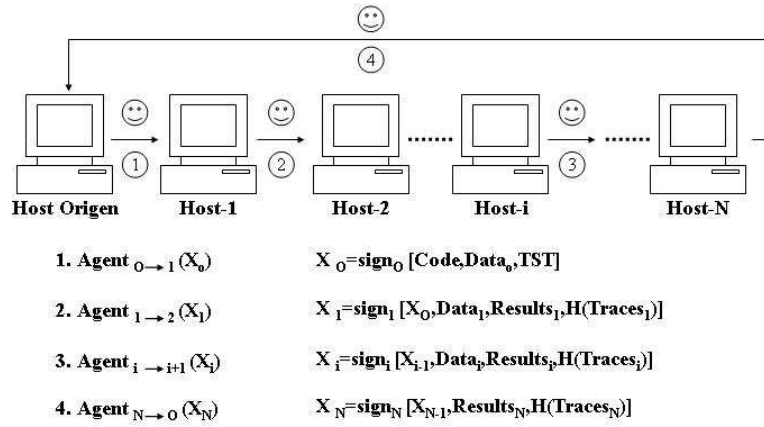


Figura 5.3: Envío del Agente con el Mecanismo de las Trazas

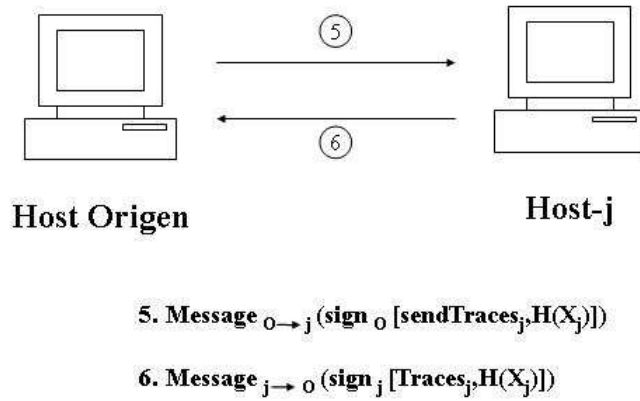


Figura 5.4: Envío de las Trazas

5.5. Protocolos de Revocación para las Trazas Criptográficas

En las Figuras 5.3 y 5.4 pueden apreciarse los procesos de envío del agente y de envío de las trazas respectivamente.

Verificaciones del Host Origen

Llegado este punto, el host origen dispone de todas las pruebas necesarias para verificar la integridad de la ejecución y por tanto puede realizar las siguientes comprobaciones:

- Verifica que el hash $H(X_j)$ enviado en el mensaje del paso 6 es el mismo que había enviado él en el paso 5. Si coinciden, se continúan las verificaciones. Si por el contrario el hash es distinto quiere decir que el Host-j se niega a facilitar las pruebas necesarias, ya que nos está mandando las trazas de otra transacción. En este caso el host origen puede iniciar un proceso de revocación provisional, pues el host se ha negado a enviar las trazas adecuadas. El Protocolo de Revocación Provisional para las Trazas (*Provisional Revocation Protocol for Traces* o PRP-T) se especifica más adelante.
- Verifica que las trazas $Traces_j$ enviadas por el host sospechoso coinciden con el valor hash $H(Traces_j)$ que se encontraba en X_j . Si las trazas coinciden con el valor hash se continúan las verificaciones. Si por el contrario hay alguna inconsistencia quiere decir que existe una prueba que certifica que el Host-j está actuando maliciosamente, pues las trazas han sido modificadas. En este caso el host origen puede iniciar el Protocolo de Revocación de Hosts para las Trazas (*Host Revocation Protocol for Traces* o HRP-T).
- Ejecuta el agente de nuevo y verifica que la ejecución concuerda con las trazas $Traces_j$. El host origen dispone de todos los datos necesarios para restaurar la ejecución⁸, ya que el agente transportaba en X_j el código y los resultados, y en el paso 6 el Host-j ha enviado $Traces_j$ con los datos de entrada del agente. Si al

⁸Para realizar esta verificación el host origen debe disponer de un módulo con capacidad para ejecutar agentes.

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

reejecutar el código, teniendo en cuenta los datos de entrada, nos encontramos con los mismos resultados que en X_j , quiere decir que el host es honesto. Si por el contrario se encuentra alguna inconsistencia durante la ejecución, quiere decir que existe una prueba que certifica que el Host-j actuó maliciosamente y por tanto el host origen puede iniciar HRP-T.

En Resumen, si no existen pruebas del carácter malicioso del host, se pueden dar por válidos tanto la ejecución del agente como los resultados obtenidos, y por tanto finalizarían aquí todas las comprobaciones. Si el host se niega a facilitar las pruebas necesarias para verificar la integridad de la ejecución, se puede iniciar un Protocolo de Revocación Provisional para las Trazas o PRP-T, que se especifica más adelante. Si por el contrario existen pruebas que certifiquen del carácter malicioso de algún host, bien sea porque éste ha manipulado la ejecución o bien porque haya modificado las trazas, el host origen puede iniciar el Protocolo de Revocación de Hosts para las Trazas o HRP-T. Dicho protocolo se especifica justo a continuación.

Protocolo de Revocación de Hosts para las Trazas (HRP-T)

Consideramos que aquí se iniciaría el proceso de revocación propiamente dicho, ya que todas las comprobaciones anteriores se realizan existan o no hosts maliciosos. Asumiremos que el Host-j actuó maliciosamente ejecutando el agente, y el host origen ha detectado el ataque con el mecanismo de las trazas. Es por ello que se dispone a iniciar su revocación, y para ello debe seguir los siguientes pasos:

7. El proceso básicamente consiste en enviar a la HoRA un mensaje con todas las pruebas que certifican que el host ejecutó el agente maliciosamente. Dicho mensaje se especifica de la siguiente forma:

$$Message_{O \rightarrow HoRA}(sign_O[X_j, sign_j[Traces_j, H(X_j)]]).$$

5.5. Protocolos de Revocación para las Trazas Criptográficas

8. La HoRA recibe el mensaje que inicia el proceso de revocación del Host-j, pero antes de revocarlo debe realizar una serie de comprobaciones. La HoRA dispone de toda la información necesaria en X_j y $Traces_j$ para ejecutar el agente de nuevo y verificar la integridad de la ejecución. La idea es que realizará prácticamente las mismas verificaciones que realizó el host origen:

- Como primer paso verifica que la caducidad de las trazas no haya expirado. Para ello busca en X_O el valor de TST , de manera que si es una fecha anterior a la fecha en curso el proceso de revocación finaliza y se envía al host origen un mensaje indicando que la revocación ha sido fallida dado que la caducidad ha expirado:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Not\ revoked, TST\ expired]).$$

Si no es el caso, se pueden continuar las verificaciones.

- Acto seguido, verifica que todos los datos pertenecen a la misma transacción asegurándose que X_j corresponde con el hash $H(X_j)$. Si no es así, se envía un mensaje al host origen indicando que la revocación ha sido fallida ya que los datos enviados son inválidos pues no corresponden a la misma ejecución:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Not\ revoked, Invalid\ data]).$$

Dado que esta comprobación la debería haber realizado el host origen, se le podría imponer algún tipo de sanción por su actitud poco honesta con el host que quería revocar, y por el consumo de recursos innecesarios en la HoRA. Si por el contrario los valores corresponden, se continúan con las verificaciones.

- Verifica que las trazas $Traces_j$ coinciden con el hash $H(Traces_j)$. Si no coinciden, directamente ésto constituye una prueba que el host ha modificado las trazas, con lo cual puede ser revocado. En este caso, la HoRA

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha finalizado correctamente:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered\ traces]).$$

Si por el contrario, las trazas coinciden con el hash, se continúan las verificaciones.

- Finalmente, ejecuta el agente de nuevo teniendo en cuenta el código y los datos de entrada que se encuentran en las trazas⁹. Si los resultados coinciden, el host es honesto y por tanto el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si por el contrario se encuentra algún tipo de inconsistencia en la ejecución, ésto constituye una prueba que el host ha manipulado el agente, con lo cual puede ser revocado. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha finalizado correctamente:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered\ execution]).$$

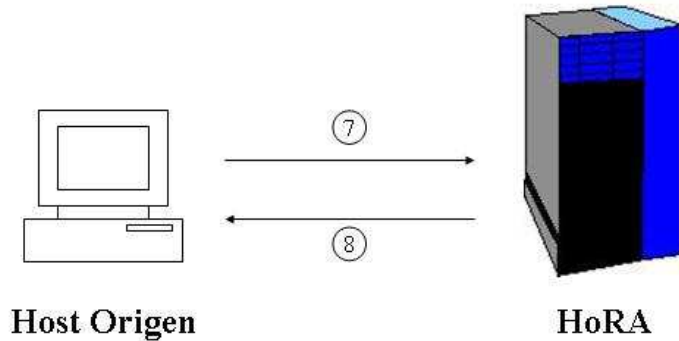
En la Figura 5.5 se muestra el intercambio de mensajes entre el host origen y la HoRA para revocar al Host-j mediante HRP-T.

Protocolo de Revocación Provisional para las Trazas (PRP-T)

Puede darse el caso que un host origen no disponga de las pruebas necesarias para revocar el host, bien porque éste ha quedado fuera de servicio poco después de ejecutar el agente, o bien porque se niega a enviarlas conscientemente. Partiendo de esa base, dicho host puede ser temporalmente revocado hasta que se pruebe que

⁹La HoRA debe disponer de un módulo con capacidad para ejecutar agentes para realizar esta verificación.

5.5. Protocolos de Revocación para las Trazas Criptográficas



7. Message $O \rightarrow HoRA$ ($\text{sign}_O [X_j, \text{sign}_j [\text{Traces}_j, H(X_j)]]$)

8. Message $HoRA \rightarrow O$ ($\text{sign}_{HoRA} [\text{Revoked}, \text{TamperedExecution}]$)

Figura 5.5: HRP-T

ejecutó el agente correctamente. El hecho de revocar un host de forma temporal no debe considerarse una medida desproporcionada. Si la razón por la cual no envía las pruebas es que dicho host ha quedado fuera de servicio, podemos asumir que tampoco está en disposición de ejecutar agentes de la forma adecuada, por tanto la HoRA puede evitar que temporalmente le lleguen agentes hasta que regularice su situación. Si por el contrario, el host se niega a enviar las trazas conscientemente, muy probablemente será debido a que oculta una posible manipulación en el agente, y por tanto puede ser revocado hasta que demuestre su inocencia.

En este caso, el traspaso de mensajes del PRP-T se inicia antes que en el HRP-T, ya que una de las posibilidades es que el host no envíe las trazas en el paso 6, bien sea porque está fuera de servicio, bien porque se niega a hacer el envío, o bien porque envía unas trazas que no corresponden con la ejecución. Sea cual sea la razón, el último mensaje válido fue el mensaje de petición de las trazas del paso 5. Es por ello que iniciaremos este protocolo en el paso 6, como continuación de la fase de envío del agente del ejemplo anteriormente planteado:

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

6. Como el host origen no recibe las trazas del Host-j (o las que ha enviado no son las apropiadas), envía un mensaje a la HoRA informando que las trazas no han sido enviadas por el Host-j. El mensaje además contiene X_j como la única prueba de que dispone el host origen:

$$Message_{O \rightarrow HoRA}(sign_O[X_j, Traces_j \text{ not received}])$$

7. La HoRA recibe el mensaje de petición de revocación provisional de Host-j y, al igual que hizo el host origen, demanda directamente al Host-j las trazas con el envío del siguiente mensaje:

$$Message_{HoRA \rightarrow j}(sign_{HoRA}[sendTraces_j, H(X_j)]).$$

8. El Host-j atendiendo al mensaje anterior puede actuar de la siguiente forma:
 - No respondiendo al mensaje, bien sea porque está fuera de servicio o bien porque deliberadamente no quiere hacerlo.
 - Respondiendo al mensaje con las trazas, con lo cual la HoRA dispone de toda la información necesaria para verificar la integridad de la ejecución.

$$Message_{j \rightarrow HoRA}(sign_j[Traces_j, H(X_j)])$$

9. Atendiendo al comportamiento anterior, la HoRA puede actuar de la siguiente forma:
 - Si el Host-j no respondió al mensaje, su identificador se incluye en la base de datos interna de la HoRA con estado revocado provisionalmente. Durante un cierto periodo de tiempo, el Host-j tiene la posibilidad de enviar las trazas a la HoRA para demostrar su inocencia. El valor de dicho tiempo será una constante en el sistema de agentes móviles, y debe ser suficientemente grande (días o semanas) para que un host que quede fuera

5.5. Protocolos de Revocación para las Trazas Criptográficas

de servicio pueda recuperar el sistema y enviar las pruebas que demuestren su inocencia. Transcurrido dicho tiempo, si el Host-j no ha enviado las trazas, su estado pasa a ser permanentemente revocado en la base de datos interna de la HoRA. Asimismo, para evitar que hosts honestos que han quedado fuera de servicio dejen de recibir agentes durante un periodo de tiempo largo, los hosts revocados provisionalmente no deben incluirse en la HRL hasta que no pasen a permanentemente revocados.

- Si el Host-j respondió con el envío de las trazas, la HoRA puede iniciar las comprobaciones para verificar la integridad de la ejecución.

La idea es que la HoRA realizará prácticamente las mismas verificaciones que realizó el host origen:

- Verifica que la caducidad de las trazas *TST* no haya expirado. Si es así se envía al host origen un mensaje indicando que la revocación ha sido fallida dado que la caducidad ha expirado:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Not\ revoked, TST\ expired]).$$

Si por el contrario la caducidad no ha expirado, se continúan las verificaciones.

- Verifica que todos los datos pertenecen a la misma transacción asegurándose que X_j corresponde con $H(X_j)$ ¹⁰. Si no es así, quiere decir que el Host-j se niega a enviarnos las trazas relativas a esta transacción, con lo cual puede ser revocado. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Traces\ not\ sent]).$$

¹⁰Esta comprobación no la pudo realizar el host origen ya que el host no le envió las trazas.

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

Si por el contrario los datos pertenecen a la misma transacción, se continúan las verificaciones.

- Verifica que las trazas $Traces_j$ coinciden con el hash $H(Traces_j)$. Si no coinciden, directamente ésto constituye una prueba que el host ha modificado las trazas. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered traces]).$$

Si por el contrario, las trazas coinciden con el hash, se continúan las verificaciones.

- Finalmente, ejecuta el agente de nuevo teniendo en cuenta el código y los datos de entrada que se encuentran en las trazas. Si los resultados coinciden, el host es honesto. Si por el contrario se encuentra algún tipo de inconsistencia en la ejecución, ésto constituye una prueba que el host ha manipulado el agente. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

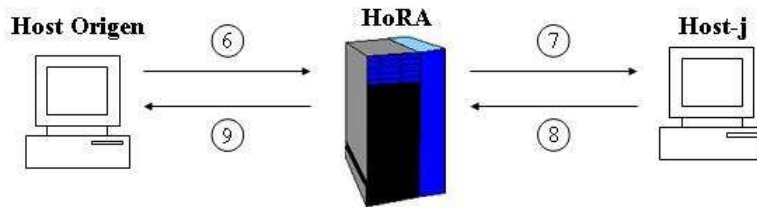
$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered execution]).$$

En la Figura 5.6 se puede apreciar el intercambio de mensajes que se realiza en el protocolo PRP-T teniendo en cuenta que el Host-j envía las trazas y éstas determinan que actuó maliciosamente.

5.5.2. Protocolos con Privacidad

Hasta ahora habíamos obviado el tema de la privacidad de los datos que transporta el agente. Sin embargo, existe la necesidad de dar protección criptográfica a los contenidos del agente, no sólo de elementos externos, sino también de los mismos

5.5. Protocolos de Revocación para las Trazas Criptográficas



- 6. Message $O \rightarrow HoRA$ ($sign_O [X_j, Traces_j, not\ received]$)
- 7. Message $HoRA \rightarrow j$ ($sign_{HoRA} [sendTraces_j, H(X_j)]$)
- 8. Message $j \rightarrow HoRA$ ($sign_j [Traces_j, H(X_j)]$)
- 9. Message $HoRA \rightarrow O$ ($sign_{HoRA} [Revoked, TamperedExecution]$)

Figura 5.6: PRP-T

hosts del itinerario. La solución a este problema pasa por el uso del cifrado de la información, sin embargo su uso en el esquema de revocación de hosts no está exento de inconvenientes. El problema más importante es cómo hacer que la HoRA dé por válida una prueba que debe ser descifrada con la clave privada del host origen sin dar a conocer dicha clave. Para solucionar el problema utilizaremos la técnica del sobre digital, de manera que en ningún punto deberemos revelar la clave privada de ninguna entidad, en todo caso sólo las claves de sesión utilizadas en ese momento.

Asumiremos que el código y los datos de entrada deben ser accesibles sólo por los hosts del itinerario. Por su parte, los resultados de la ejecución sólo deben ser leídos por el host origen.

Envío del Agente

A continuación se describen los pasos que deben seguir las entidades involucradas para realizar el envío del agente, poniendo especial incapié en aquellos factores que aplican sobre la privacidad:

1. Antes de enviar el agente móvil, el host origen (O) genera una serie de claves

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

para cifrar la información con un algoritmo de clave simétrica: la clave K_s se utilizará para cifrar los datos que transporta el agente, y las claves $K_1, K_2 \dots K_N$ para que cada host cifre los resultados de la ejecución. Para ello, el host origen envía el siguiente agente móvil al primer host del itinerario (Host-1):

$$Agent_{O \rightarrow 1}(S_{K_s}[Y_O], Z)$$

donde

$$Y_O = sign_O[Code, Data_O, TST, H(K_1), H(K_2) \dots H(K_N)]$$

$$Z = sign_O[E_{K_{pub1}}[K_s, K_1], E_{K_{pub2}}[K_s, K_2], \dots, E_{K_{pubN}}[K_s, K_N], H(Y_O)].$$

El bloque de datos Y_O contiene básicamente el código, los datos y la caducidad TST . Para darle privacidad, éste se envía cifrado con la clave de sesión K_s . El agente también transporta un vector de claves Z a partir del cual cada host podrá obtener tanto la clave de sesión como la clave de cifrado de los resultados. Z contiene un hash de Y_O para certificar que ambos pertenecen a la misma ejecución. Asimismo, Y_O contiene los valores hash de las claves K_1 a K_N para certificar que las claves introducidas en Z son las que corresponden para esta ejecución. Dichos valores servirán como prueba para que la HoRA dé por válidas dichas claves en caso que sea necesario revocar a algún host.

2. El Host-1 recibe el agente y verifica que la firma de Z es válida y corresponde al host origen. A continuación descifra con su clave privada la primera posición de Z para obtener la clave de sesión K_s y la clave de cifrado de los resultados K_1 :

$$[K_s, K_1] = D_{K_{priv1}}[E_{K_{pub1}}[K_s, K_1]].$$

Una vez obtiene la clave de sesión puede descifrar el código y los datos que se encuentran en Y_O :

$$Y_O = S_{K_s}^{-1}[S_{K_s}[Y_O]].$$

5.5. Protocolos de Revocación para las Trazas Criptográficas

Antes de ejecutar el agente, verifica que Y_O corresponde con el $H(Y_O)$ que contiene Z , y que la clave K_1 corresponde con el hash $H(K_1)$ que ha enviado el host origen en Y_O . Si alguno de ellos no concuerdan, quiere decir que las claves no corresponden a esta ejecución, con lo cual el host puede abortar la ejecución y enviar un mensaje de error al host origen. Si concuerdan, el Host-1 puede iniciar la ejecución del agente. Una vez finalizada ésta, el agente se dispone a migrar al Host-2:

$$Agent_{1 \rightarrow 2}(S_{K_s}[Y_1], Z)$$

donde

$$Y_1 = sign_1[Y_O, Data_1, S_{K_1}[Results_1], H(Traces_1)].$$

Y_1 se cifra con la clave de sesión K_s , con lo cual sólo los hosts del itinerario podrán leer su contenido. Asimismo, los resultados de la ejecución están cifrados con la clave K_1 , de manera que sólo el host origen podrá descifrarlos. Por su parte, los datos de entrada no se cifran porque deben estar disponibles para el resto de hosts del itinerario. El hash de las trazas tampoco es necesario cifrarlo ya que ningún host podrá obtener información a partir de él. Mientras tanto, el Host-1 guarda las trazas $Traces_1$ hasta que la caducidad TST expire, así como la clave K_1 para cifrar las trazas en caso que fuera necesario enviarlas al host origen. También guarda un hash de Y_1 que será utilizado para certificar que las trazas corresponden a la misma ejecución.

3. Este proceso se repite por cada host del itinerario. De esta manera, el host i -ésimo del itinerario Host- i envía al siguiente host este agente:

$$Agent_{i \rightarrow i+1}(S_{K_s}[Y_i], Z)$$

donde

$$Y_i = sign_i[Y_{i-1}, Data_i, S_{K_i}[Results_i], H(Traces_i)].$$

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

4. Finalmente, el último host del itinerario Host-N envía el siguiente agente al host origen:

$$Agent_{N \rightarrow O}(S_{K_s}[Y_N], Z)$$

donde

$$Y_N = sign_N[Y_{N-1}, S_{K_N}[Results_N], H(Traces_N)].$$

5. El agente llega al host origen y éste se dispone a descifrar todos los contenidos que transporta el agente¹¹. En particular, descifra los tiempos de llegada y finalización para utilizar el protocolo de detección de sospechosos. Suponiendo que el Host-j resultase sospechoso, el host origen le envía un mensaje como el siguiente demandándole las trazas:

$$Message_{O \rightarrow j}(sign_O[sendTraces_j, H(Y_j)])$$

6. El Host-j debe responder con un mensaje que contiene las trazas cifradas con K_j y un hash que certifica que las trazas corresponden con la ejecución llevada a cabo. El formato del mensaje es el siguiente:

$$Message_{j \rightarrow O}(S_{K_j}[sign_j[Traces_j, H(Y_j)]])$$

En las Figuras 5.7 y 5.8 pueden apreciarse los procesos de envío del agente y de envío de las trazas respectivamente.

Verificaciones del Host Origen

Una vez recibidas las trazas, el host origen puede realizar las mismas acciones que se realizaban en la fase de verificación de la Sección 5.5.1, cuando la privacidad no era requerida.

¹¹ El host origen puede descifrar todos los contenidos pues conoce todas las claves.

5.5. Protocolos de Revocación para las Trazas Criptográficas

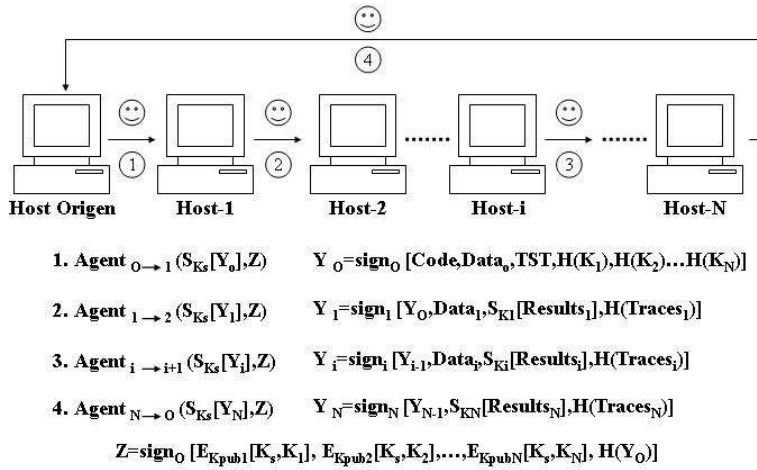


Figura 5.7: Envío del Agente con el Mecanismo de las Trazas y Privacidad

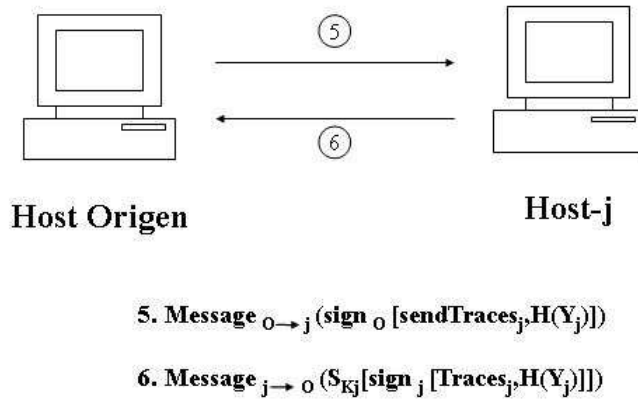


Figura 5.8: Envío de las Trazas Cuando se Requiere Privacidad

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

- Verifica que el hash $H(Y_j)$ del mensaje del paso 6 coincide con el enviado en el mensaje del paso 5. Si no coinciden quiere decir que el Host-j se niega a mandar las trazas que corresponden a esta transacción. En este caso el host origen inicia el Protocolo de Revocación Provisional para las Trazas con Privacidad (*Provisional Revocation Protocolo for Traces with Privacy* o PRP-TP), que será explicado más adelante.
- Verifica que las trazas $Traces_j$ enviadas por el Host-j coinciden con el valor hash $H(Traces_j)$ que se encontraba en Y_j . Si hay alguna inconsistencia en el valor hash de las trazas, quiere decir que existe una prueba que certifica que el Host-j está actuando maliciosamente. En este caso el host origen puede iniciar el Protocolo de Revocación de Hosts para las Trazas con Privacidad (*Host Revocation Protocol for Traces with Privacy* o HRP-TP).
- Ejecuta el agente de nuevo y verifica que la ejecución concuerda con las trazas $Traces_j$. Si se encuentra alguna inconsistencia durante la ejecución, quiere decir que existe una prueba que certifica que el Host-j actuó maliciosamente y por tanto el host origen puede iniciar el HRP-TP.

Protocolo de Revocación de Hosts para las Trazas con Privacidad (HRP-TP)

Con lo visto hasta este punto, podría parecer que el procedimiento para dar privacidad al envío del agente es innecesariamente complejo. Sin embargo, dichas complicaciones no son gratuitas, sino que responden a la necesidad de permitir una posterior revocación del host. Algunos de dichos datos cifrados pueden constituir una prueba para revocar a hosts maliciosos. Obviamente, el conocimiento de dichas pruebas no debería suponer la revelación de la clave privada de ningún host. Es por ello que se han utilizado claves temporales K_i que sí pueden ser reveladas a la HoRA para que pueda descifrar los contenidos del agente y verificar así la integridad de la ejecución.

Supongamos que en la fase de envío del agente se ha detectado que el Host-j

5.5. Protocolos de Revocación para las Trazas Criptográficas

ha manipulado la ejecución del agente y por tanto puede ser revocado. El problema ahora reside en hacer que la HoRA considere como válidas las pruebas enviadas por el host origen. En este caso se debe seguir el siguiente procedimiento:

7. El host origen envía a la HoRA el siguiente mensaje:

$$Message_{O \rightarrow HoRA}(sign_O[E_{K_{pubHoRA}}[Y_j, sign_j[Traces_j, H(Y_j)], K_j]]).$$

Tanto Y_j como $sign_j[Traces_j, H(Y_j)]$ están convenientemente firmados por el Host-j, con lo cual éste no puede realizar un ataque de repudio. Sin embargo, para que la HoRA pueda realizar todas las verificaciones necesarias, debe conocer cuales son los resultados de la ejecución $Results_j$. El problema reside en que dichos resultados se encuentran cifrados con K_j . Por esa razón, dicha clave se envía a la HoRA en el mensaje.

8. La HoRA recibe el mensaje e inicia una serie de comprobaciones antes de revocar el host:

- Como primer paso verifica que la caducidad de las trazas no haya expirado. Para ello busca en Y_O el valor de TST , de manera que si es una fecha anterior a la fecha en curso el mensaje el proceso de revocación finaliza y se envía al host origen un mensaje indicando que la revocación ha sido fallida dado que la caducidad ha expirado:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Not\ revoked, TST\ expired]).$$

Si no es el caso, se pueden continuar las verificaciones.

- Acto seguido, verifica que todos los datos pertenecen a la misma transacción asegurándose que Y_j corresponde con $H(Y_j)$. Si no es así, se envía un mensaje al host origen indicando que la revocación ha sido fallida ya que los datos enviados son inválidos pues no corresponden a la misma

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

ejecución:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Not\ revoked, Invalid\ data]).$$

En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si los valores corresponden, se continúan con las verificaciones.

- Verifica que la clave de descifrado K_j enviada en el mensaje corresponde con el valor hash $H(K_j)$ que se encontraba en Y_O . Si no coinciden quiere decir que el host origen ha comunicado una clave de descifrado errónea. En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si coincide quiere decir que tanto el host origen como el Host-j utilizaron la misma clave¹². Con dicha clave descifra los resultados de la ejecución:

$$Results_j = S_{K_j}^{-1}[S_{K_j}[Results_j]].$$

- Verifica que las trazas $Traces_j$ coinciden con el hash $H(Traces_j)$. Si no coinciden, directamente ésto constituye una prueba que el host ha modificado las trazas, con lo cual puede ser revocado. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

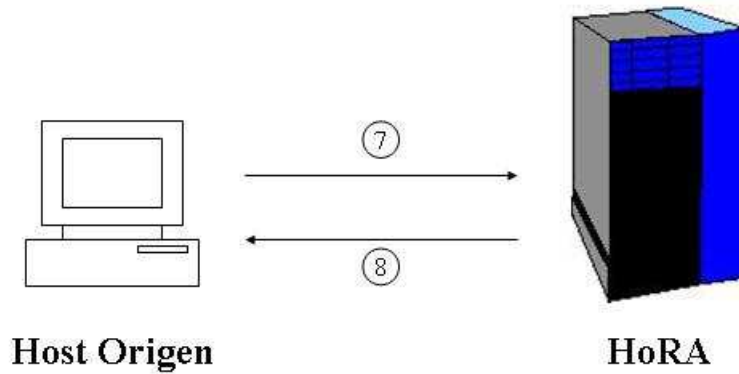
$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered\ traces]).$$

Si por el contrario, las trazas coinciden con el hash, se continúan las verificaciones.

- Finalmente, ejecuta el agente de nuevo teniendo en cuenta el código y

¹²Simplemente recordar que el Host-j hizo la misma comprobación en la fase de envío del agente y por tanto validó que esa era la clave de cifrado que iba a utilizar.

5.5. Protocolos de Revocación para las Trazas Criptográficas



7. Message $O \rightarrow HoRA$ ($sign_O [E_{K_{pubHoRA}}[Y_j, sign_j[Traces_j, H(Y_j)], K_j]]$)

8. Message $HoRA \rightarrow O$ ($sign_{HoRA} [Revoked, TamperedExecution]$)

Figura 5.9: HRP-TP

los datos de entrada que se encuentran en las trazas. Si los resultados coinciden, el host es honesto y por tanto el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si por el contrario se encuentra algún tipo de inconsistencia en la ejecución, ésto constituye una prueba que el host ha manipulado el agente, con lo cual puede ser revocado. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered\ execution])$.

En la Figura 5.9 se muestra el intercambio de mensajes entre el host origen y la HoRA para revocar al Host-j mediante HRP-TP.

Protocolo de Revocación Provisional para las Trazas con Privacidad (PRP-TP)

Puede darse el caso que un host origen no disponga de las pruebas necesarias para revocar un host con lo cual puede intentar hacerlo de forma provisional. El traspaso de mensajes para realizar la revocación provisional es el siguiente:

6. Como el host origen no recibe las trazas del Host-j (o las que ha enviado no son las apropiadas), envía un mensaje a la HoRA informando que las trazas no han sido enviadas por el Host-j. El mensaje además contiene Y_j y K_j como únicas pruebas que dispone el host origen:

$$Message_{O \rightarrow HoRA}(sign_O[E_{K_{pubHoRA}}[Y_j, K_j], Traces_j \text{ not recieved}])$$

7. La HoRA recibe el mensaje que demanda la revocación provisional del Host-j, descifra su contenido y automáticamente demanda las trazas al Host-j con el envío del siguiente mensaje:

$$Message_{HoRA \rightarrow j}(sign_{HoRA}[sendTraces_j, H(Y_j)]).$$

8. El Host-j atendiendo al mensaje anterior puede actuar de la siguiente forma:
 - No respondiendo al mensaje, bien sea porque está fuera de servicio o bien porque deliberadamente no quiere hacerlo.
 - Respondiendo al mensaje con las trazas cifradas, con lo cual la HoRA dispone de toda la información necesaria para verificar la integridad de la ejecución.

$$Message_{j \rightarrow HoRA}(S_{K_j}[sign_j[Traces_j, H(Y_j)]])$$

9. Atendiendo al comportamiento anterior, la HoRA puede actuar de distinta forma. Si el Host-j no respondió al mensaje, su identificador se incluye en la

5.5. Protocolos de Revocación para las Trazas Criptográficas

base de datos interna de la HoRA con estado revocado provisionalmente. Si por el contrario el Host-j envió las trazas, la HoRA puede iniciar las verificaciones:

- Verifica que la caducidad de las trazas TST no haya expirado. Si es así, se envía al host origen un mensaje indicando que la revocación ha sido fallida dado que la caducidad ha expirado:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Not\ revoked, TST\ expired]).$$

Si por el contrario la caducidad no ha expirado, se continúan las verificaciones.

- Verifica que la clave de descifrado K_j enviada en el mensaje corresponde con el valor hash $H(K_j)$ que se encontraba en Y_O . Si no coinciden quiere decir que el host origen ha comunicado una clave de descifrado errónea. En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si coincide quiere decir que tanto el host origen como el Host-j utilizaron la misma clave. Con dicha clave descifra los resultados de la ejecución y las trazas:

$$Results_j = S_{K_j}^{-1}[S_{K_j}[Results_j]]$$

$$sign_j[Traces_j, H(Y_j)] = S_{K_j}^{-1}[S_{K_j}[sign_j[Traces_j, H(Y_j)]]].$$

- Verifica que todos los datos pertenecen a la misma transacción asegurándose que Y_j corresponde con $H(Y_j)$ ¹³. Si no es así, quiere decir que el Host-j se niega a enviarnos las trazas relativas a esta transacción, con lo cual puede ser revocado. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Traces\ not\ sent]).$$

¹³Esta comprobación no la pudo realizar el host origen ya que el host no envió las trazas.

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

Si por el contrario los datos pertenecen a la misma transacción, se continúan con las verificaciones.

- Verifica que las trazas $Traces_j$ coinciden con el hash $H(Traces_j)$. Si no coinciden, directamente ésto constituye una prueba que el host ha modificado las trazas. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered traces]).$$

Si por el contrario, las trazas coinciden con el hash, se continúan las verificaciones.

- Finalmente, ejecuta el agente de nuevo teniendo en cuenta el código y los datos de entrada que se encuentran en las trazas. Si los resultados coinciden, el host es honesto. Si por el contrario se encuentra algún tipo de inconsistencia en la ejecución, ésto constituye una prueba que el host ha manipulado el agente. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

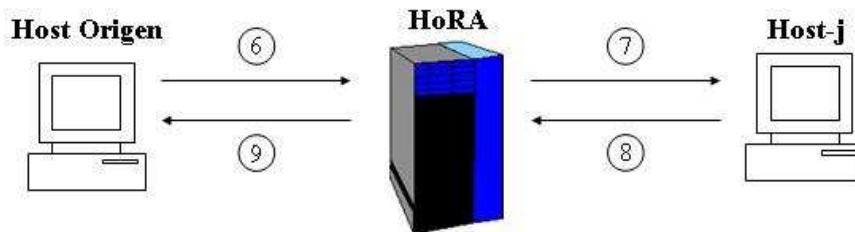
$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered execution]).$$

En la Figura 5.10 se puede apreciar en intercambio de mensajes que se realiza en el PRP-TP:

5.5.3. Ataques a los Protocolos

Los ataques contra este sistema de revocación son, en su mayoría, ataques de repudio en los cuales alguno de los actores implicados niega una acción que sí tuvo lugar. Por lo general, el objetivo que persigue la entidad maliciosa es la exculpación de una acción maliciosa, intentando incluso inculpar a otra entidad honesta. Como

5.5. Protocolos de Revocación para las Trazas Criptográficas



6. Message $O \rightarrow HoRA$ ($sign_O [E_{K_{pubHoRA}}[Y_j, K_j], Traces_j, \text{not received}]$)

7. Message $HoRA \rightarrow j$ ($sign_{HoRA} [\text{sendTraces}_j, H(Y_j)]$)

8. Message $j \rightarrow HoRA$ ($S_{K_s} [sign_j [Traces_j, H(Y_j)]]$)

9. Message $HoRA \rightarrow O$ ($sign_{HoRA} [\text{Revoked}, \text{TamperedExecution}]$)

Figura 5.10: PRP-TP

la revocación sólo puede llevarse a cabo si hay pruebas de actitud maliciosa, los ataques se centran en la ocultación o manipulación de dichas pruebas. Por una parte, el principal objetivo del sistema de castigo es que un host malicioso no quede impune por un fallo del protocolo o por ausencia de pruebas. Por otra parte y como es lógico, no debe existir la posibilidad que un host que ha ejecutado el agente de forma honesta sea involucrado por algún otro host o por el host origen.

A continuación se enumeran los principales ataques que pueden realizarse sobre el sistema de revocación cuando se utilizan las trazas como mecanismo de detección:

- Un host malicioso puede modificar cualquier parámetro que afecte a la ejecución del agente. Por ejemplo, puede alterar el código *Code*, los datos de entrada *Data_j*, los resultados *Results_j*, el hash de las trazas $H(Traces_j)$ o incluso las mismas trazas *Traces_j*. Asumiendo que el SDP funciona correctamente, en caso de manipulación el Host-j será detectado como sospechoso y se le pedirán las trazas. Con dichas trazas se detectará la manipulación de la ejecución, con lo cual el host origen dispone de una prueba que el Host-j es malicioso y por

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

tanto puede ser revocado.

- El host malicioso no envía las trazas $Traces_j$. El host puede no enviarlas por estar fuera de servicio, como también puede negarse a mandarlas de forma consciente. En este caso, el host origen debe iniciar uno de los protocolos de revocación provisional (PRP-T o PRP-TP). En ambos protocolos la HoRA realizará la petición de las trazas, y si el host continúa sin mandarlas será revocado provisionalmente. Finalmente, el estado de dicho host pasará a revocado permanentemente si persiste en la negativa de enviar las trazas.
- El host malicioso envía las trazas correspondientes a otra transacción. Este caso es equivalente al caso anterior, con lo cual el host origen inicia un protocolo de revocación provisional (PRP o PRP-P), y si finalmente el host persiste en su conducta y no envía las pruebas adecuadas, será revocado.
- Un host origen puede intentar involucrar un host honesto iniciando un proceso de revocación, pudiendo ser éste de forma normal (HRP-T o HRP-TP) o provisional (PRP-T o PRP-TP). En ambos casos, este ataque no tendrá consecuencias si el host honesto guarda las trazas hasta que la caducidad TST expire. Incluso el host origen se arriesga a algún tipo de sanción por iniciar un proceso de revocación con datos falsos.
- En caso de ser requerida privacidad, es posible que alguno de los actores pretenda manipular las claves para involucrar a otra entidad. En primer lugar, cualquier actuación sobre la clave de sesión K_s implica de forma directa la imposibilidad de descifrar el contenido del agente, y por tanto, de ejecutarlo. Ello se tomaría como un ataque de negación de servicio, que como tales no pueden ser evitados. En cuanto a las claves que cifran los resultados K_i , el host origen no puede intentar involucrar a un host honesto diciendo que la clave de cifrado era otra, ya que el hash de la clave $H(K_i)$ que se encuentra en Y_O lo delataría. Por su parte, el Host-i tampoco puede utilizar otra clave distinta que K_i , ya que al comprobar si correspondía con el hash $H(K_i)$ había dado la conformidad

5.6. *Protocolos de Revocación para MAW*

a esa clave. Por tanto, si utiliza una clave distinta, al host origen (y a la HoRA) les será imposible descifrar los resultados, que es razón de revocación.

5.5.4. **Inconvenientes**

La propuesta es satisfactoria en cuanto a la integración de los tres mecanismos utilizados, el de las trazas, el protocolo de detección de sospechosos (SDP) y la autoridad de revocación de hosts (HoRA). Sin embargo, esta propuesta de detección y castigo es todavía demasiado complicada y difícil de utilizar. El número de protocolos de revocación a tener en cuenta (HRP-T, HRP-TP, PRP-T y PRP-TP) puede impedir su aplicación en un sistema real. El intercambio de mensajes entre las entidades es costoso y, lo que es peor, requiere de la colaboración de todas las partes implicadas, el host origen, los hosts del itinerario y la HoRA. En definitiva, el coste que tienen que asumir todos los actores del sistema de agentes móviles es grande, lo cual lo hace especialmente desaconsejable para aquellas entidades que tengan limitaciones en cuanto a recursos disponibles, como ancho de banda, CPU o capacidad de almacenamiento.

5.6. **Protocolos de Revocación para MAW**

Tal como se especificó en el Capítulo 4, la propuesta del Watermarking de Agentes Móviles [EFS⁺03b] era, en muchos aspectos, menos costosa computacionalmente que la de las trazas criptográficas de Vigna. Además, en la anterior Sección ha quedado patente que los protocolos de revocación asociados al uso de las trazas también son complicados.

En el resto del Capítulo se pretende poner de manifiesto que MAW no sólo es más sencillo que la propuesta de las trazas a la hora de detectar ataques, sino que también permite disminuir el número de protocolos de revocación necesarios, así como simplificar notablemente el intercambio de mensajes. Para ello, se definirá la forma en que debe realizarse el envío del agente y la recolección de las pruebas necesaria, para finalmente presentar los protocolos de revocación de hosts, teniendo

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

como objetivo la total integración entre MAW y la HoRA.

Al igual que en la Sección 5.5, inicialmente asumiremos que no es necesario asegurar la privacidad de los datos que transporta el agente. Más adelante se incorporarán las modificaciones necesarias para el caso en que fuera necesario incorporar este requisito de seguridad. Sin embargo, en ambos casos sí se ha tenido en cuenta la integridad del agente y de los datos. Es por ello que cualquier información que transporta el agente está convenientemente firmada por la entidad emisora para garantizar su integridad y evitar ataques de repudio. Dichas condiciones son necesarias para que los protocolos de revocación funcionen, ya que sino cualquier entidad podría variar los datos del agente a su favor, impidiendo así el establecimiento de pruebas válidas para revocar un host.

5.6.1. Protocolo sin Privacidad

Envío del Agente

En primer lugar, asumiremos que el host origen ya ha consultado previamente la información de revocación, eliminando del itinerario todos los hosts maliciosos. La forma en que haya consultado dicha información de revocación es irrelevante para el proceso de envío del agente, puede ser bien utilizado consultas OHSP (on-line), o bien consultando una copia local de la HRL (off-line). Se obvian muchas de las tareas a realizar por MAW, ya que éstas han sido especificadas previamente en el Capítulo 4 y lo único que harían sería entorpecer la comprensión del ejemplo de funcionamiento que pretendemos explicar.

A continuación se describen los pasos que deben seguir las entidades involucradas para realizar el envío del agente, suponiendo que el itinerario del agente es de N hosts:

1. El host origen (O) envía el agente móvil al primer host del itinerario (Host-1), transportando el código, los datos de entrada y un hash de las reglas. Dicho hash sirve como prueba para enlazar la ejecución con las reglas que se utilizarán más tarde. Por tanto, el host origen envía al primer host del itinerario el siguiente

5.6. Protocolos de Revocación para MAW

agente:

$$Agent_{O \rightarrow 1}(W_O)$$

donde

$$W_O = sign_O[Code, Data_O, H(Rules)].$$

2. El Host-1 recibe el agente y verifica que la firma de W_O es válida y corresponde al host origen. Si es así, extrae de W_O el código y los datos e inicia la ejecución. Una vez finalizada la ejecución, el agente se dispone a migrar al siguiente host transportando los datos de entrada para el siguiente host (si fueran necesarios) y el contenedor (donde se encuentran los resultados ocultos). Por tanto, el agente que se envía al siguiente host tiene el siguiente formato:

$$Agent_{1 \rightarrow 2}(W_1)$$

donde

$$W_1 = sign_1[W_O, Data_1, Container_1].$$

El host no debe almacenar ningún tipo de prueba de la ejecución, ni más tarde será requerida su intervención en los protocolos de revocación. Esto se debe a que el agente transporta en el contenedor la prueba que certifica la integridad de la ejecución. Simplemente destacar que este hecho hace que no sea necesario incorporar una caducidad a las pruebas, con lo cual el host origen puede iniciar el procedimiento de revocación cuando desee¹⁴.

3. Este proceso se repite por cada host del itinerario. De esta manera, el host i -ésimo del itinerario Host- i recibe el agente, lo ejecuta y obtiene el contenedor. Por tanto, el agente que se envía al siguiente host tiene el siguiente formato:

$$Agent_{i \rightarrow i+1}(W_i)$$

¹⁴Con el uso de MAW no es estrictamente necesario el uso de una caducidad de las pruebas, lo cual no implica que sea aconsejable su uso simplemente para incentivar que los hosts origen lancen los procesos de revocación lo antes posible.

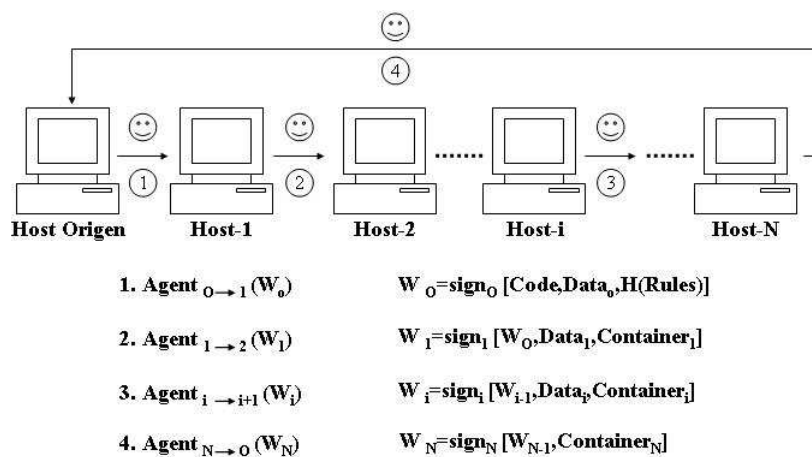


Figura 5.11: Envío del Agente con MAW

donde

$$W_i = \text{sign}_i[W_{i-1}, \text{Data}_i, \text{Container}_i].$$

4. Finalmente, el host origen recibe el agente proveniente del último host del itinerario Host-N con el siguiente formato:

$$\text{Agent}_{N \rightarrow O}(W_N)$$

donde

$$W_N = \text{sign}_N[W_{N-1}, \text{Container}_N].$$

En la Figura 5.11 puede apreciarse el envío del agente si se utiliza MAW como mecanismo de detección.

Verificaciones del Host Origen

La primera acción que realiza el host origen es verificar que las firmas anidadas en W_N son válidas y corresponden a los hosts que dicen ser. Una vez verificada

5.6. Protocolos de Revocación para MAW

la integridad y autenticación de los datos que transporta el agente, se extraen los contenedores. Para verificar que la ejecución ha sido íntegra basta con aplicar las reglas a cada uno de ellos. Si la marca del contenedor está intacta quiere decir que el host ejecutó el agente de forma honesta. Si por el contrario, en alguno de los contenedores la marca ha sido modificada o borrada, quiere decir que el host ha manipulado la ejecución y por tanto puede iniciarse el Protocolo de Revocación de Hosts para MAW (*Host Revocation Protocol for MAW* o HRP-M).

Protocolo de Revocación de Hosts para MAW (HRP-M)

El protocolo de revocación que utiliza MAW es más sencillo que el que se utilizaba en las trazas criptográficas dado que el host origen dispone de todas las pruebas necesarias. Esto implica que no es necesaria ningún tipo de comunicación adicional con los hosts del itinerario, ni tampoco se requiere de un protocolo de revocación provisional. Al igual que en el resto de protocolos de revocación, el proceso consiste en enviar las pruebas a la HoRA para que las dé por válidas.

Asumiendo que el Host-j ha actuado maliciosamente ejecutando el agente, y por tanto el host origen ha detectado el ataque con el mecanismo del MAW, se puede iniciar su revocación:

5. El proceso básicamente consiste en enviar a la HoRA un mensaje con todas las pruebas que certifican que el host ejecutó el agente maliciosamente. Dicho mensaje se especifica de la siguiente forma:

$$Message_{O \rightarrow HoRA}(sign_o[W_j, Rules]).$$

El mensaje contiene W_j , de donde puede extraerse el contenedor del Host-j $Container_j$. Asimismo, se incorporan las reglas que deben cumplir los contenedores.

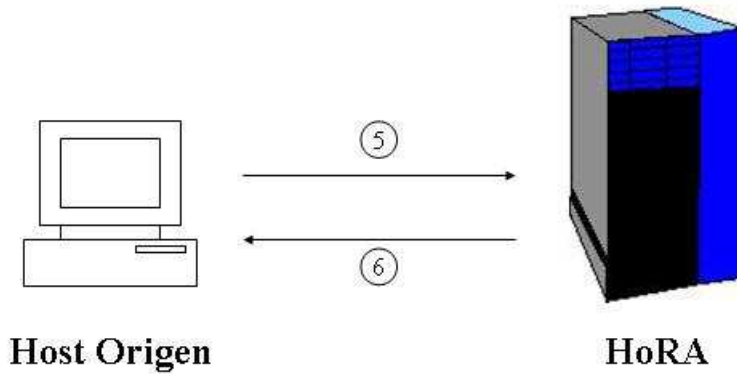
6. La HoRA recibe el mensaje que inicia el proceso de revocación del Host-j, pero antes de revocar el host debe realizar una serie de comprobaciones. Tal

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

como se ha comentado anteriormente, la HoRA dispone de toda la información necesaria para realizar prácticamente las mismas verificaciones que realizó el host origen:

- Verifica que la firma de W_j es válida y corresponde al Host-j. Con ello se valida que el contenedor $Container_j$ ha sido generado por dicho host.
- Verifica que la firma de W_O es válida y corresponde al host origen. Con ello se certifica que el código del agente es el que aplica a la ejecución. Asimismo, se verifica que el hash $H(Rules)$ que contiene W_O concuerda con las reglas enviadas en el mensaje. Con ello se asegura que las reglas enviadas por el host origen son las que aplican sobre la ejecución y no las ha cambiado a posteriori.
- Verifica que las reglas $Rules$ aplican al código. Sin embargo, salvo el host origen ninguna otra entidad conocía las reglas ya que éstas no eran públicas. Las reglas dependían de las modificaciones realizadas en el código original del agente para incorporarle la marca. Sin embargo, la HoRA debe certificar que esas reglas aplicarían a cualquier ejecución honesta. Ésto puede resolverse ejecutando el agente (una o varias veces) con datos de entrada aleatorios. Si los contenedores creados en la ejecución con datos aleatorios no cumplen con las reglas, quiere decir que no corresponden al código del agente. En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si por el contrario sí cumplen las reglas, la HoRA puede darlas por válidas.
- Finalmente, para verificar que la ejecución ha sido íntegra bastaría con aplicar las reglas $Rules$ al contenedor del Host-j $Container_j$. Si la marca del contenedor está intacta, o lo que es lo mismo, si cumple con las reglas quiere decir que el host ejecutó el agente de forma honesta. En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si por el contrario

5.6. Protocolos de Revocación para MAW



5. Message $O \rightarrow HoRA$ (**sign** O [$W_j, Rules$])

6. Message $HoRA \rightarrow O$ (**sign** $HoRA$ [**Revoked, TamperedExecution**])

Figura 5.12: HRP-M

el contenedor no cumple con las reglas, esto es una prueba que actuó maliciosamente y por tanto puede ser revocado. En este caso, la HoRA incorpora los datos del Host-j en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

$$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered execution]).$$

En la Figura 5.12 se muestra el intercambio de mensajes entre el host origen y la HoRA para revocar al Host-j mediante HRP-M.

5.6.2. Protocolo con Privacidad

Hasta ahora habíamos obviado el tema de la privacidad de los datos que transporta el agente. Sin embargo, existe la necesidad de dar protección criptográfica a

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

los contenidos del agente, no sólo de elementos externos, sino también de los mismos hosts del itinerario. La solución a este problema pasa por el uso del cifrado de la información, sin embargo su uso en el esquema de revocación de hosts no está exento de inconvenientes. Al igual que sucedía con los protocolos de revocación para las trazas, la HoRA debe dar por válidas las pruebas sin revelar la clave privada de ninguna entidad.

Asumiremos que el código y los datos de entrada deben ser accesibles sólo por los hosts del itinerario. Por su parte, los resultados de la ejecución sólo deben ser leídos por el host origen.

Envío del Agente

Éstos son los pasos que deben seguir las entidades involucradas para realizar el envío del agente:

1. Antes de enviar el agente móvil, el host origen (O) genera una serie de claves para cifrar la información con un algoritmo de clave simétrica: la clave K_s se utilizará para cifrar los datos que transporta el agente, y las claves $K_1, K_2 \dots K_N$ para que cada host cifre los resultados de la ejecución. El host origen envía el siguiente agente móvil al primer host del itinerario (Host-1):

$$Agent_{O \rightarrow 1}(S_{K_s}[V_O], Z)$$

donde

$$V_O = sign_O[Code, Data_O, H(Rules), H(K_1), H(K_2) \dots H(K_N)]$$

$$Z = sign_O[E_{K_{pub1}}[K_s, K_1], E_{K_{pub2}}[K_s, K_2], \dots, E_{K_{pubN}}[K_s, K_N], H(V_O)].$$

El bloque de datos V_O contiene básicamente el código, los datos y el hash de las reglas. Para darle privacidad, éste se envía cifrado con la clave de sesión K_s . El agente también transporta un vector de claves Z a partir del cual cada host podrá obtener tanto la clave de sesión como la clave de cifrado de los

5.6. Protocolos de Revocación para MAW

resultados. Z contiene un hash de V_O para certificar que ambos pertenecen a la misma ejecución. Asimismo, V_O contiene los valores hash de las claves K_1 a K_N para certificar que las claves introducidas en Z son las que corresponden para esta ejecución. Dichos valores servirán como prueba para que la HoRA dé por válidas dichas claves en caso que sea necesario revocar a algún host.

2. El Host-1 recibe el agente y verifica que la firma de Z es válida y corresponde al host origen. A continuación descifra con su clave privada la primera posición de Z para obtener la clave de sesión K_s y la clave de cifrado de los resultados K_1 . Una vez obtiene la clave de sesión puede descifrar el código y los datos que se encuentran en V_O . Antes de ejecutar el agente, verifica que V_O corresponde con el hash $H(V_O)$ que contiene Z , y que la clave K_1 corresponde con el hash $H(K_1)$ que ha enviado el host origen en V_O . Si alguno de los valores hash no concuerda, quiere decir que alguna de las claves no corresponde a esta ejecución. En este caso el host puede abortar la ejecución y enviar un mensaje de error al host origen. Si por el contrario los valores hash concuerdan, el Host-1 puede iniciar la ejecución del agente. Una vez finalizada ésta, el agente se dispone a migrar al Host-2:

$$Agent_{1 \rightarrow 2}(S_{K_s}[V_1], Z)$$

donde

$$V_1 = sign_1[V_O, Data_1, S_{K_1}[Container_1]].$$

Como se puede apreciar de la expresión, V_1 se cifra con la clave de sesión K_s , con lo cual sólo los hosts del itinerario podrán leer su contenido. Asimismo, el contenedor está cifrado con la clave K_1 , de manera que sólo el host origen podrá leer su contenido. A diferencia de lo que sucedía con el uso de las trazas, ningún host debe almacenar las pruebas de la ejecución, pues éstas viajan con el agente.

3. Este proceso se repite por cada host del itinerario. De esta manera, el host

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

i-ésimo del itinerario Host-i envía al siguiente host este agente:

$$Agent_{i \rightarrow i+1}(S_{K_s}[V_i], Z)$$

donde

$$V_i = [sign_i[V_{i-1}, Data_i, S_{K_i}[Container_i]].$$

4. Finalmente, el host origen recibe el agente proveniente del último host del itinerario Host-N con el siguiente formato:

$$Agent_{N \rightarrow O}(S_{K_s}[V_N], Z)$$

donde

$$V_N = sign_N[V_{N-1}, S_{K_N}[Container_N]].$$

Como el host origen conserva todas las claves, puede descifrar todos los contenidos que transporta el agente.

En la Figura 5.13 puede apreciarse el envío del agente si se utiliza MAW como mecanismo de detección y se requiere dar privacidad al agente.

Verificaciones del Host Origen

Como primera medida descifra el contenido del agente con la clave de sesión K_s para obtener V_N . Una vez realizado esto, verifica que todas las firmas anidadas en V_N corresponden con las entidades que las emitieron. Una vez verificada la integridad de la información que transporta el agente, el host origen puede extraer todos los contenedores ya que dispone de todas las claves $K_1 \dots K_N$. Una vez descifrados, para verificar que la ejecución ha sido íntegra basta con aplicar las reglas a cada uno de los contenedores. Si la marca del contenedor está intacta quiere decir que el host ejecutó el agente de forma honesta. Si por el contrario, en alguno de los contenedores la marca ha sido modificada o borrada, quiere decir que el host ha manipulado la ejecución y por tanto puede iniciarse el Protocolo de Revocación de Hosts para MAW

5.6. Protocolos de Revocación para MAW

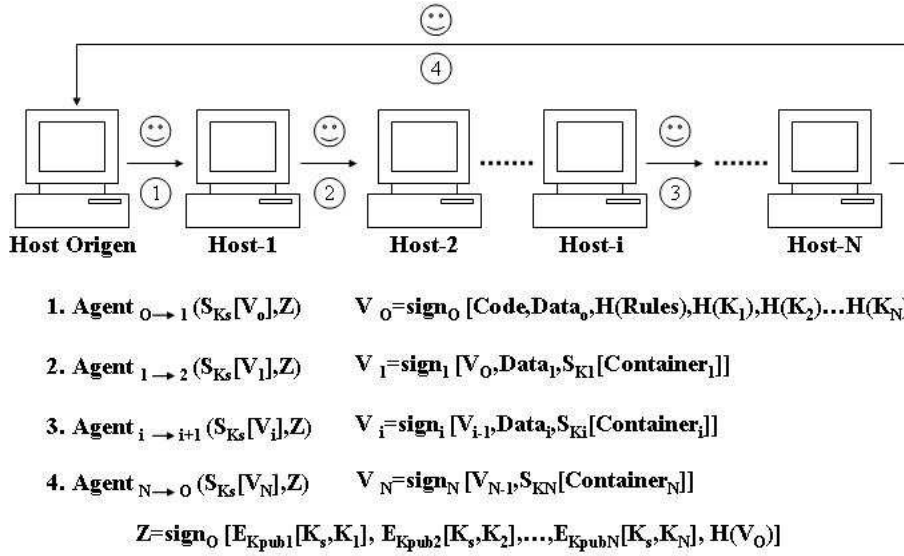


Figura 5.13: Envío del Agente con MAW y Privacidad

con Privacidad (*Host Revocation Protocol for MAW with Privacy* o HRP-MP).

Protocolo de Revocación de Host para MAW con Privacidad (HRP-MP)

Al igual que en anteriores ocasiones, asumiremos que el host origen ha detectado con el mecanismo del MAW que el Host- j ha actuado maliciosamente, con lo cual se puede iniciar su revocación:

5. El host origen envía un mensaje con todas las pruebas que certifican que el host ejecutó el agente maliciosamente:

$$\text{Message}_{O \rightarrow HoRA}(\text{sign}_o [E_{K_{\text{pub}HoRA}} [V_j, \text{Rules}, K_j]]).$$

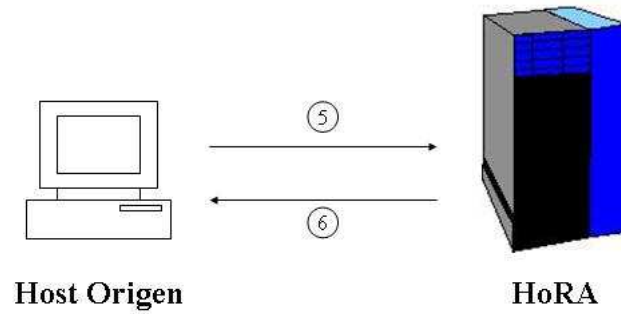
El mensaje contiene las pruebas de la ejecución en V_j , las reglas que deben cumplir los contenedores y la clave de descifrado K_j .

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

6. La HoRA recibe el mensaje que inicia el proceso de revocación del Host-j y realiza las siguientes verificaciones:
 - Verifica que la firma de V_j es válida y corresponde al Host-j. Con ello se certifica que el contenedor ha sido generado por dicho host.
 - Verifica que la firma de V_O es válida y corresponde al host origen. Con ello se asegura que el código del agente es el que aplica a la ejecución. Asimismo, verifica que el hash $H(Rules)$ que contiene V_O concuerda con las reglas enviadas en el mensaje. Con ello se asegura que las reglas enviadas por el host origen son las que aplican sobre la ejecución y no las ha cambiado a posteriori.
 - Verifica que la clave de descifrado K_j enviada en el mensaje corresponde con el valor hash $H(K_j)$ que se encontraba en V_O . Si no coinciden quiere decir que el host origen ha comunicado una clave de descifrado errónea. En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si coincide quiere decir que tanto el host origen como el Host-j utilizaron la misma clave¹⁵. Con dicha clave se descifra el contenedor $Container_j$.
 - Verifica que las reglas $Rules$ aplican al código. Salvo el host origen, ninguna otra entidad conocía las reglas ya que éstas no eran públicas. Sin embargo, la HoRA debe validar que esas reglas aplican a cualquier ejecución honesta. Ésto puede resolverse ejecutando el agente (una o varias veces) con datos de entrada aleatorios. Si los contenedores creados en la ejecución con datos de entrada aleatorios no cumplen con las reglas, quiere decir que no corresponden al código del agente. En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si por el contrario sí cumplen las reglas, la HoRA puede darlas por válidas.

¹⁵Simplemente recordar que el Host-j hizo la misma comprobación en la fase de envío del agente y por tanto validó que esa era la clave de cifrado que iba a utilizar.

5.6. Protocolos de Revocación para MAW



5. Message $O \rightarrow HoRA$ ($sign_O [E_{K_{pubHoRA}}[V_j, Rules, K_j]]$)

6. Message $HoRA \rightarrow O$ ($sign_{HoRA} [Revoked, TamperedExecution]$)

Figura 5.14: HRP-MP

- Finalmente, para verificar que la ejecución ha sido íntegra bastaría con aplicar las reglas *Rules* al contenedor del Host-*j* *Container_j*. Si la marca del contenedor está intacta, o lo que es lo mismo, si cumple con las reglas quiere decir que el host ejecutó el agente de forma honesta. En este caso, el host origen ha iniciado un proceso de revocación con pruebas no válidas, con lo cual se le podría imponer algún tipo de sanción. Si por el contrario el contenedor no cumple con las reglas, esto es una prueba que actuó maliciosamente y por tanto puede ser revocado. En este caso, la HoRA incorpora los datos del Host-*j* en su base de datos interna y envía un mensaje informativo al host origen indicando que la revocación ha sido correcta:

$Message_{HoRA \rightarrow O}(sign_{HoRA}[Revoked, Tampered\ execution])$.

En la Figura 5.14 se muestra el intercambio de mensajes entre el host origen y la HoRA para revocar al Host-*j* mediante HRP-MP.

5.6.3. Ataques a los Protocolos

Al igual que sucedía con el uso de las trazas, la mayoría de ataques que pueden realizarse contra el sistema de revocación son de repudio. Como la revocación sólo puede llevarse a cabo si hay pruebas de actitud maliciosa, los ataques se centran en la ocultación o manipulación de dichas pruebas. A continuación se enumeran los principales ataques que pueden realizarse sobre el sistema de revocación cuando se utiliza MAW como mecanismo de detección:

- Un host malicioso puede modificar cualquier parámetro que afecte a la ejecución del agente. Por ejemplo, puede alterar el código *Code*, los datos de entrada *Data_j*, los resultados *Results_j* e incluso el contenedor *Container_j*. Cualquier manipulación realizada sobre el agente que suponga variaciones sobre la marca del contenedor será detectada al aplicar las reglas de validación, con lo cual el host podrá ser revocado.
- Un host origen puede intentar involucrar al host honesto Host-*i* iniciando un proceso de revocación con pruebas falsas. Asumiendo que una ejecución correcta crea un contenedor que cumple las reglas, el host origen no podrá involucrar al host por tener un contenedor incorrecto. Además, el host origen no puede alterar V_i (y por tanto tampoco *Container_i*) pues está firmado por el Host-*i*. Sin embargo, sí puede intentar que el contenedor no pase los controles de la HoRA enviando unas reglas distintas y que no apliquen al código. En dicho caso, cuando la HoRA ejecute el agente con datos de entrada aleatorios detectará que la ejecución del código del agente *Code* crea contenedores que no cumplen las reglas, y por tanto dichas reglas son falsas. En este caso, el host origen se expone a una sanción por iniciar un proceso de revocación con datos no válidos.
- En caso de ser requerida privacidad, es posible que alguno de los actores pretenda manipular las claves para involucrar a otra entidad. En primer lugar, cualquier actuación sobre la clave de sesión K_s implica de forma directa la

5.7. Inconvenientes de la Revocación de Hosts

imposibilidad de descifrar el contenido del agente, y por tanto, de ejecutarlo. En cuanto a las claves que cifran los contenedores K_i , el host origen no puede intentar involucrar a un host honesto asegurando que la clave de cifrado era otra, ya que el hash de la clave $H(K_i)$ que se encuentra en V_O lo delataría. Por su parte, el Host- i tampoco puede utilizar otra clave distinta que K_i , ya que al comprobar si correspondía con el hash $H(K_i)$ había dado la conformidad a esa clave. Por tanto, si utiliza una clave distinta, al host origen (y a la HoRA) les será imposible descifrar los resultados, que es razón de revocación.

5.7. Inconvenientes de la Revocación de Hosts

La propuesta de castigo basada en revocación de hosts, pese a tratarse de una solución intuitiva y hasta cierto punto lógica, tiene una serie de inconvenientes que pueden limitar su uso en determinados escenarios. En primer lugar, un error no intencionado durante la ejecución de un agente puede provocar que un host sea revocado. Esto a primera vista podría considerarse un castigo desproporcionado, y sin duda en algunos escenarios lo es. Sin embargo, desde nuestro punto de vista, todos los hosts deben asegurar la correctitud de todas las transacciones. Si no fuera así, cualquier error iría en detrimento de los intereses del host origen. A pesar de ello, no hay nada que impida a la HoRA aplicar políticas de castigo más laxas ya que es dicha entidad la que controla quien debe ser revocado y quien no. A modo de ejemplo, se podría amonestar inicialmente a los hosts maliciosos con faltas y sólo revocarlos cuando se han producido un cierto número de ellas. También se podría revocar a los hosts en función de la gravedad de los ataques o del beneficio obtenido por los hosts maliciosos.

En segundo lugar, la base de datos interna de la HoRA crece indefinidamente, ya que los hosts maliciosos una vez revocados no se eliminan nunca. Esto es necesario para que la HoRA pueda controlar todo el histórico de hosts revocados. En particular, se asume que la capacidad de almacenamiento de la HoRA no debe ser una limitación para el sistema, ya que se trata de un servidor cuya principal tarea

Capítulo 5. Autoridad de Revocación de Hosts (HoRA)

es el almacenamiento y gestión de la base de datos. Sin embargo, en el sistema hay otras entidades que dependen de técnicas de consulta de estado off-line y que deben descargarse la HRL. El tamaño de dicha HRL puede ser un serio problema para dispositivos como teléfonos móviles o PDAs cuyas capacidades de almacenamiento están limitadas. Para reducir el tamaño de la lista, la HoRA podría emitir Certificados de Ejecución de Agentes (*Agent Execution Certificate* o AEC) con un cierto periodo de validez. En tal supuesto, sólo podrán ejecutar agentes aquellas entidades que dispongan de un AEC que no haya expirado y no esté revocado¹⁶. En caso de comportamiento malicioso, la HoRA no revocará al host, sino el certificado. De esta manera, la HRL ya no crece indefinidamente porque los hosts cuyos AEC hayan expirado se eliminan de la lista. Con un esquema orientado a certificados es posible aprovechar todos los conocimientos adquiridos en el desarrollo de una PKI, incluyendo protocolos de consulta como OCSP. En particular, se está realizando la integración de la implementación de la HoRA con la del proyecto Cervantes [Cer, MFES04].

Por otra parte, nos encontramos con el problema que plantea la confidencialidad de los datos de la ejecución. A pesar que la HoRA debe considerarse una entidad de confianza, en caso de revocación se le deben enviar los datos para que verifique la integridad de la ejecución, pudiendo ser estos datos confidenciales. La dificultad que puede tener la HoRA a la hora de realizar alguna escucha de dicha información depende en gran medida del mecanismo de detección utilizado. Por ejemplo, en caso de utilizar MAW la HoRA no dispondría de los datos de entrada reales de la ejecución y debería extraer de los contenedores los resultados, cosa a priori no trivial. Sin embargo, en caso de utilizar el mecanismo de las trazas la HoRA dispone de todos los datos de entrada en las trazas, así como los resultados de la ejecución.

¹⁶En realidad, los AEC, más que certificados de identidad serían certificados de atributo (*Attribute Certificate* o AC), ya que su posesión implica la posibilidad de ejecutar agentes.

5.8. Conclusiones del Capítulo

En este Capítulo se ha propuesto un mecanismo de castigo basado en revocación de hosts. Para ello se ha introducido una TTP en el sistema de agentes móviles: la Autoridad de Revocación de Hosts (HoRA). La HoRA es la depositaria y gestora de una base de datos que contiene todos los hosts que se ha demostrado que son maliciosos. El mecanismo de castigo consiste en no volver a enviar agentes a dichos hosts maliciosos. Por tanto, antes del envío de un agente los hosts origen deben consultar el estado de los hosts del itinerario para eliminar aquellos que han sido revocados. Esta consulta de estado puede realizarse mediante una consulta OHSP (on-line), o bien descargando una lista de hosts revocados o HRL (off-line).

Para mantener actualizada la información de revocación los hosts origen inician procesos de revocación cuando detectan el ataque de un host malicioso. El protocolo de revocación a utilizar dependerá directamente del método de detección utilizado. En primer lugar, se han introducido los protocolos de revocación a utilizar cuando se utilizan las trazas y SDP como mecanismo de detección (HRP-T, PRP-T, HRP-TP y PRP-TP). Sin embargo, el número de protocolos y su complejidad hace poco aconsejable su uso en terminales con restricciones de algún tipo. En segundo lugar y en contraposición con los anteriores, se han introducido los protocolos de revocación en caso de utilizar MAW como mecanismo de detección (HRP-M y HRP-MP). Los protocolos de revocación de MAW no requieren de la intervención del host que ejecutó el agente en ningún momento, ni requieren de una fase de envío de las trazas, ni son necesarios protocolos de revocación provisional.

A modo de conclusión, se recomienda el uso del mecanismo de detección MAW (y de los protocolos de revocación asociados) en lugar del de las trazas, ya que MAW no sólo es menos costoso para todas las entidades involucradas, sino que además los protocolos de revocación implicados también son más sencillos.

Capítulo 6

Conclusiones y Líneas Futuras

Los agentes móviles constituyen uno de los paradigmas de diseño de aplicaciones más prometedores dentro de los sistemas distribuidos. Su uso no sólo puede mejorar parámetros de rendimiento como la latencia o el ancho de banda necesario, sino que también puede aumentar la flexibilidad de los servicios. Estos y otros muchos factores los hacen especialmente útiles para realizar de forma automatizada cualquier tipo de tarea que requiera del cómputo de grandes volúmenes de datos dispersos a lo largo de la red. A pesar de las múltiples ventajas que presenta, este paradigma de diseño sigue siendo uno de los menos explotados en los servicios existentes. De hecho, el estudio de los agentes móviles casi no ha pasado del ámbito puramente académico.

Desde nuestro punto de vista, uno de los factores que más ha limitado el despegue de esta tecnología son los problemas relativos a la seguridad, y más particularmente, el problema de los hosts maliciosos. La mayoría de autores versados en el tema coinciden a la hora de afirmar que dicho problema es el más difícil de resolver, con diferencia, de todos los relativos a la seguridad en sistemas de agentes móviles. Sería ingenuo no esperar algún tipo de ataque de la plataforma de ejecución, máxime teniendo en cuenta que los agentes pueden ejecutarse en hosts con muy diversos grados de confianza, pudiendo incluso pertenecer a la competencia. Los hosts pueden atacar al agente de muy diversas formas ya que tienen control total sobre la ejecución, pudiendo tanto leer como modificar el código, los datos, el modo de ejecución, el

estado, las comunicaciones, el itinerario o incluso los resultados. Ésta es la razón por la cual hasta el momento no existe una solución conocida que dé protección global al agente frente a los ataques realizados por un host malicioso.

El objetivo que se planteaba al inicio de esta Tesis era realizar el estudio de mecanismos de protección de agentes eficaces y que no supongan un coste tan alto como para impedir su uso en un sistema real. Como punto de partida, se ha realizado el estudio de las principales propuestas de protección publicadas hasta el momento para analizar los inconvenientes que limitan su uso. Desde nuestro punto de vista, ninguna de dichas propuestas es adecuada para su uso en un sistema real. Por un lado, los mecanismos de prevención de ataques son o bien demasiado costosos, o bien imposibles de implementar. Por su parte las propuestas de detección de ataques existentes, pese a ser las más prometedoras, presentan demasiados inconvenientes funcionales. La carencia de los mecanismos de protección adecuados nos ha inducido a centrar las contribuciones de esta Tesis en la búsqueda de propuestas de detección de ataques eficaces y computacionalmente ligeras. En particular, nuestras aportaciones tratan de asegurar la integridad de la ejecución del agente, esto es, asegurar que la ejecución está libre de ataques de manipulación.

Como primera aproximación al problema de los hosts maliciosos hemos propuesto realizar algunas mejoras sobre una de las propuesta de detección de ataques existentes, la propuesta de las trazas criptográficas. Se ha tomado esta propuesta como base debido a que consideramos que es la propuesta de detección más conocida y la que menos dificultades presenta a la hora de implementarse. La propuesta presenta principalmente dos inconvenientes que limitan su aplicación. En primer lugar, la verificación de las trazas se realiza sólo en caso de sospecha, pero en ningún punto se detalla cómo un host se convierte en sospechoso. En segundo lugar, los hosts deben almacenar las trazas durante un periodo de tiempo indeterminado ya que el host origen puede solicitarlas para verificar la ejecución. Para solucionar ambos inconvenientes es necesario algún tipo de mecanismo que detecte qué hosts han actuado de forma sospechosa, y que lo evalúe en un tiempo razonable para limitar el tiempo que los servidores deben almacenar las trazas. Por ello, como primera aportación de

Capítulo 6. Conclusiones y Líneas Futuras

esta Tesis se introduce un nuevo Protocolo de Detección de Sospechosos (SDP) que permite detectar actitudes sospechosas en los hosts justo cuando el agente llega al origen. SDP se basa en controlar el tiempo que dispone un host para ejecutar un agente. Si a la vuelta del agente el host origen detecta algún tipo de incoherencia temporal en los tiempos de ejecución o transmisión de algún host, éste se convierte en sospechoso y se le pueden solicitar las trazas de ejecución. En definitiva, el uso conjunto de las trazas y SDP nos ofrece un mecanismo de detección y prueba de ataques efectivo y utilizable. La implementación realizada de SDP no se limita a la detección de hosts sospechosos, sino que también ofrece protección criptográfica a los datos que transporta el agente, en caso de ser requerida. Las pruebas realizadas sobre la implementación de SDP han demostrado que su uso no supone un incremento sustancial en ninguno de los parámetros críticos del sistema.

Aún habiéndose resuelto los principales inconvenientes de la propuesta de las trazas criptográficas mediante SDP, consideramos que esta propuesta aun comporta un gran esfuerzo computacional para todas las entidades involucradas. En cuanto a las trazas, su tamaño depende del volumen de datos de entrada que tenga el agente, con lo cual pueden ser relativamente pequeñas, o bien pueden tener un tamaño tan grande que haga muy desaconsejable su envío a través de la red. Asimismo, los hosts están obligados a almacenarlas, aunque sea por un periodo de tiempo pequeño. En cuanto a la detección de ataques, el proceso de verificación en sí ya es complicado, pues obliga al host a reejecutar el agente. Además no se realiza sobre todos los hosts sino que solamente se realiza sobre aquellos que son sospechosos, con lo cual un host que ha actuado maliciosamente y no es sospechoso se escapa del mecanismo de detección.

Es por ello que como segunda contribución de esta Tesis se ha introducido un mecanismo de detección de ataques propio, menos costoso computacionalmente que el de las trazas criptográficas. El nuevo mecanismo utiliza técnicas de watermarking de software para empotrar una marca de agua en el código del agente, esto es, Watermarking de Agentes Móviles (MAW). Durante la ejecución del agente se crea un contenedor de datos donde se transfiere la marca del código y donde se ocultan los

resultados. El contenedor es la prueba que utilizará el host origen para determinar si un host actuó maliciosamente. Cuando el agente vuelve, el host origen busca la marca en el contenedor, de manera que si la marca ha sido alterada quiere decir que el host modificó el agente y por tanto es malicioso. Este proceso de verificación se realiza mediante la aplicación de una serie de reglas de integridad sobre los contenedores. Si el contenedor de un host no cumple con las reglas, quiere decir que dicho host es malicioso. Dichas reglas de integridad pueden inferirse de las modificaciones realizadas sobre el código del agente durante el proceso de empotrado de la marca. La fuerza de la propuesta del MAW se basa en la dificultad que tendrá un observador en distinguir qué partes del agente afectan a la marca, ya que un usuario sin dicha información difícilmente podrá alterar la ejecución sin ser detectado. Aún sabiendo qué partes son marca dentro del contenedor, si el host quisiera modificar la ejecución para que le sea favorable debe ser capaz de construir un contenedor que cumpla con todas las reglas de integridad. La propuesta, pese a representar un incremento en el tamaño del código y los datos que debe transportar el agente, mejora varios aspectos relevantes si la comparamos con la de las trazas. En primer lugar, el tamaño de las pruebas (los contenedores) es configurable, con lo cual pueden ser lo suficientemente pequeñas como para que el agente pueda transportarlas durante su trayecto sin que exista una gran pérdida en el rendimiento de la red. Esto implica que ya no es necesario actuar en base a sospecha y que el host origen pueden verificar la integridad de la ejecución de todos los hosts. Además se libera a los hosts de la obligación de tener que almacenar las pruebas. Asimismo, el proceso de verificación de la integridad de la ejecución no supone la reejecución del agente, sino que sólo es necesario aplicar las reglas de integridad a los contenedores. En nuestro grupo de trabajo continuamos desarrollando el estudio teórico de esta propuesta y de los problemas que plantea. En particular, los esfuerzos se centran en la adaptación de técnicas de watermarking de software para que la marca empotrada en el agente pueda ser luego transferida al contenedor.

Nuestro objetivo inicial era evitar en la medida de lo posible los ataques de hosts maliciosos, y en caso que no fuera posible evitarlos al menos minimizar los efectos que

Capítulo 6. Conclusiones y Líneas Futuras

producen. Para llegar a la consecución de dicho objetivo consideramos que el arma más eficaz es disuadir a los hosts a emprender actitudes maliciosas. Para ello hay dos factores que consideramos determinantes para evitar ataques. En primer lugar, el mecanismo de control de la ejecución no sólo debe detectar ataques de forma suficientemente eficaz, sino que además debe aportar pruebas que dicho ataque se llevó a cabo. En segundo lugar, es necesaria la implantación de una política de castigo para los hosts maliciosos. Cualquier sistema de protección de agentes basado sólo en métodos de detección de ataques es claramente insuficiente. De poco sirve detectar ataques si no hay posibilidad de practicar algún tipo de castigo a los hosts que han actuado maliciosamente.

Es por ello que como tercera contribución de la Tesis hemos presentado una nueva entidad sancionadora al sistema de agentes móviles, la Autoridad de Revocación de Hosts (HoRA). La HoRA debe considerarse una Tercera Parte de Confianza (TTP) en el sistema de agentes móviles cuyo objetivo es discriminar los hosts maliciosos de los honestos. Para ello conserva un histórico de aquellos hosts que han actuado maliciosamente en el pasado para evitar futuros ataques de dichos hosts impidiendo que vuelvan a recibir agentes. La HoRA realiza principalmente dos tareas: permitir la consulta de estado y revocar nuevos hosts maliciosos. Los host origen deben consultar el estado de los hosts del itinerario para no enviar el agente a aquellos que han sido revocados. Dicha información de revocación es accesible bien utilizando el protocolo on-line OHSP, o bien consultando localmente la lista de hosts revocados HRL. En cuanto a la revocación de hosts, la HoRA es la responsable de la adición de nuevos hosts maliciosos. Para ello, los hosts origen deberán utilizar protocolos de revocación para demostrar a la HoRA que los hosts actuaron maliciosamente. El protocolo de revocación a utilizar dependerá del método de detección utilizado. En primer lugar, se han introducido en esta Tesis los protocolos de revocación a utilizar cuando se utilizan las trazas y SDP conjuntamente (HRP-T, PRP-T, HRP-TP y PRP-TP). Sin embargo, el número de protocolos a aplicar y su complejidad hace poco aconsejable el uso de estos mecanismos en terminales con restricciones de algún tipo. En segundo lugar, en esta Tesis también se han introducido los protocolos de revocación asociados

a MAW (HRP-M y HRP-MP), más sencillos que los protocolos equivalentes asociados a las trazas. Estos protocolos no requieren de la intervención del host que ejecutó el agente en ningún momento, ni siquiera son necesarios protocolos de revocación provisional por falta de pruebas. En cuanto a la implementación de esta plataforma de castigo, en estos momentos nuestro grupo de trabajo está finalizando la de la HoRA, así como la de los protocolos de revocación asociados a las trazas.

A modo de conclusión, los mecanismos de prevención de ataques no parecen ser la solución adecuada para proteger agentes. En cambio, los mecanismos de detección de ataques, a pesar de sus limitaciones, pueden ser implementados con relativo éxito. En particular, el mecanismo de detección de las trazas es el más conocido y por tanto a priori será el que menos rechazo podría tener a la hora de utilizarse en un sistema real. Sin embargo, es todavía muy costoso para todas las entidades del sistema (host origen, hosts del itinerario y HoRA), con lo cual debe evitarse su uso en entornos donde alguna de estas entidades tenga limitaciones de algún tipo. Por su parte, MAW es un mecanismo del que apenas se ha realizado el análisis de inconvenientes, y del que no se dispone de ningún tipo de implementación. Por esa razón es difícil estimar el coste real que supondrá a cada una de las entidades implicadas. Sin embargo, intuitivamente parece que es un mecanismo mucho más ligero que las trazas. Asimismo, los protocolos de revocación asociados a MAW son también más sencillos y requieren de menos mensajes que los de las trazas. Sin embargo, hasta no finalizar la fase de implementación y prueba sería atrevido por nuestra parte afirmar que este mecanismo es mejor que el de las trazas criptográficas.

Las líneas futuras de esta Tesis son en su mayoría la continuación del trabajo realizado. En primer lugar, debe finalizarse la implementación de la HoRA y de los protocolos de revocación, tanto para las trazas como para MAW. Una vez finalizada esta implementación se debería pasar a la evaluación del rendimiento de todos ellos. En segundo lugar, deben estudiarse concienzudamente los mecanismos de watermarking de software existentes para decidir cual utilizar en MAW, ya que de la bondad del mecanismo de watermarking de software muy probablemente dependerá la resistencia de MAW frente a ataques de manipulación. Finalmente, queda pendiente también

Capítulo 6. Conclusiones y Líneas Futuras

la integración del mecanismo de castigo con todos los desarrollos realizados dentro del Proyecto Cervantes. En particular, nos interesan las funcionalidades de emisión de certificados para asignar un AEC a cada host. Asimismo, es posible aprovechar protocolos de consulta on-line existentes como OCSP, o mecanismos de distribución de CRLs.

Bibliografía

- [ACCK01] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *IEEE Symposium on Security and Privacy*, 2001.
- [Agl] Aglet tutorial. <http://www.ryerson.ca/~dgrimsha/courses/cps720/>.
- [BRSR99] J. Borrell, S. Robles, J. Serra, and A. Riera. Securing the Itinerary of Mobile Agents through a Non-Repudiation Protocol. In *IEEE International Carnahan Conference on Security Technology*, 1999.
- [CCKM00] C. Cachin, J. Camenisch, J. Kilian, and Joy Müller. One-round secure computation and secure autonomous mobile agents. In *27th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *LNCS*. Springer-Verlag, 2000.
- [Cer] The Cervantes Project Homepage <http://isg.upc.es/cervantes/>.
- [CGH⁺97] David Chess, Benjamin Grosz, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant agents for mobile computing. In *Readings in Agents*. 1997.
- [Che96] D. Chess. Security considerations in agent-based systems. In *First IEEE Conference on Emerging Technologies and Applications in Communications (etaCOM)*, 1996.

- [Che98] D. Chess. Security Issues in Mobile Code Systems. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.
- [CHK95] D. Chess, C. Harrison, and A. Kershenbaum. Mobile Agents: are They a Good Idea. IBM Research Report, 1995.
- [CT99] C. Collberg and C. Thomborson. Software watermarking: Models and dynamic embeddings. In *Principles of Programming Languages 1999, POPL'99*, 1999.
- [CTL97] C. Collberg, C. Thomborson, and D. Low. A taxonomy of obfuscating transformations. Technical Report 148, The University of Auckland, 1997.
- [DF02] J. Domingo-Ferrer. A provably secure additive and multiplicative privacy homomorphism. In *5th Information Security Conference (ISC 2002)*, volume 2433 of *LNCS*. Springer-Verlag, 2002.
- [DW99] C. Unger F. Kaderali D. Westhoff, M. Schneider. Methods for Protecting a Mobile Agent's Route. In *ISW'99*, volume 1729 of *LNCS*. Springer-Verlag, 1999.
- [EFS03a] O. Esparza, M. Fernandez, and M. Soriano. Protecting mobile agents by using traceability techniques. In *International Conference on Information Technology: Research and Education (ITRE 2003)*, *IEEE Communications Society*, 2003.
- [EFS⁺03b] O. Esparza, M. Fernandez, M. Soriano, J.L. Muñoz, and J. Forné. Mobile agent watermarking and fingerprinting: tracing malicious hosts. In *Database and Expert Systems Applications (DEXA 2003)*, volume 2736 of *LNCS*. Springer-Verlag, 2003.
- [ESMF03a] O. Esparza, M. Soriano, J.L. Muñoz, and J. Forné. A protocol for detecting malicious hosts based on limiting the execution time of mobile

Bibliografía

- agents. In *IEEE Symposium on Computers and Communications - ISCC'2003*, 2003.
- [ESMF03b] O. Esparza, M. Soriano, J.L. Muñoz, and J. Forné. Host Revocation Authority: a Way of Protecting Mobile Agents from Malicious Hosts. In *International Conference on Web Engineering (ICWE 2003)*, volume 2722 of *LNCS*. Springer-Verlag, 2003.
- [ESMF03c] O. Esparza, M. Soriano, J.L. Muñoz, and J. Forné. Implementation and Performance Evaluation of a Protocol for Detecting Suspicious Hosts. In *Mobile Agents for Telecommunication Applications (MATA '03)*, volume 2881 of *LNCS*. Springer-Verlag, 2003.
- [ESMF03d] O. Esparza, M. Soriano, J.L. Muñoz, and J. Forné. Limiting the execution time in a host: a way of protecting mobile agents. In *IEEE Sarnoff Symposium, Advances in Wired and Wireless Communications*, 2003.
- [ESMF03e] O. Esparza, M. Soriano, J.L. Muñoz, and J. Forné. Protocols for malicious host revocation. In *International Conference on Information and Communications Security (ICICS 2003)*, volume 2836 of *LNCS*. Springer-Verlag, 2003.
- [FGS96a] W. Farmer, J. Guttman, and V. Swarup. Security for Mobile Agents: Authentication and State Appraisal. In *European Symposium on Research in Computer Security (ESORICS)*, volume 1146 of *LNCS*. Springer-Verlag, 1996.
- [FGS96b] W.M. Farmer, J.D. Guttman, and V. Swarup. Security for mobile agents: issues and requirements. In *19th National Information Systems Security Conference*, 1996.
- [FPV98] A. Fuggetta, G.P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, 1998.

- [Fün99] S. Fünfroeken. Protecting mobile web-commerce agents with smart-cards. In *First International Symposium on Agent Systems and Applications (ASA '99)/Third International Symposium on Mobile Agents (MA '99)*, 1999.
- [HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and CRL Profile, 1999. RFC 2459.
- [HHJ⁺00] G. Hachez, L.D. Hollander, M. Jalali, J.J. Quisquater, and C. Vasserot. Towards a practical secure framework for mobile code commerce. In *Third International Workshop on Information Security (ISW)*, 2000.
- [Hoh98] F. Hohl. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.
- [Hoh00] F. Hohl. A Framework to Protect Malicious Hosts Attacks by Using Reference States. In *International Conference on Distributed Computing Systems (ICDCS)*, 2000.
- [IAI] Java Cryptography Extension (JCE). Institute for Applied Information Processing and Communication of the Graf University of Tecnology. <http://jce.iaik.tugraz.at/download/evaluation/index.php>.
- [Jan00] W. Jansen. Countermeasures for Mobile Agent Security. *Computer Communications, Special Issue on Advanced Security Techniques for Network Protection*, 2000.
- [Jav] Java Development Kit. <http://java.sun.com/downloads/>.
- [JK99] W. Jansen and T. Karygiannis. Mobile Agent Security. Special publication 800-19, National Institute of Standards and Technology (NIST), 1999.

Bibliografía

- [KM01] H. Kim and L. Moreau. Trust Relationships in a Mobile Agent System. In *5th International Conference on Mobile Agents (MA'2001)*, volume 2240 of *LNCS*. Springer-Verlag, 2001.
- [KSB02] G. Knoll, N. Suri, and J.M. Bradshaw. Path-based security for mobile agents. In Klaus Fischer and Dieter Hutter, editors, *Electronic Notes in Theoretical Computer Science*, volume 63. Elsevier, 2002.
- [LO98] D.B. Lange and M. Oshima. *Programming and Deploying Java Mobile Agents with Aglets*. Addison Wesley Professional, 1998.
- [MAM⁺99] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP, 1999. RFC 2560.
- [MB01] J. Mir and J. Borrell. Protecting General Flexible Itineraries of Mobile Agents. In *Information Security and Cryptology (ICISC 2001)*, volume 2288 of *LNCS*. Springer-Verlag, 2001.
- [MB03] J. Mir and J. Borrell. Protecting Mobile Agent Itineraries. In *Mobile Agents for Telecommunication Applications (MATA 2003)*, volume 2881 of *LNCS*. Springer-Verlag, 2003.
- [Mea97] C. Meadows. Detecting attacks on mobile agents. In *Foundations for Secure Mobile Code Workshop*, 1997.
- [MFES04] J.L. Muñoz, J. Forné, O. Esparza, and M. Soriano. CERVANTES - A Certificate Validation Test-bed. In *1st European PKI Workshop Research and Applications*, LNCS. Springer-Verlag, 2004.
- [MvRSS96] Y. Minsky, R. van Renesse, F. Schneider, and S.D. Stoller. Cryptographic Support for Fault-Tolerant Distributed Computing. In *Seventh ACM SIGOPS European Workshop*, 1996.

- [Ng02] Sau-Koon Ng. *Protecting Mobile Agents Against Malicious Hosts*. PhD thesis, The Chinese University of Hong Kong, 2002.
- [NL98] G. Neula and P. Lee. Safe, Untrusted Agents using Proof-Carrying Code. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.
- [Ord96] J. Ordille. When agents roam, who can you trust? Technical report, Computing Science Research Center, Bell Labs, 1996.
- [PFS02] W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2002. RFC 3280.
- [PKK⁺00] J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Shao, and Y. Zhang. Experience with software watermarking. In *16th Annual Computer Security Applications Conference (ACSAC)*, 2000.
- [Rob02] S. Robles. *Mobile Agent Systems and Trust, a Combined View Toward Secure Sea-of-Data Applications*. PhD thesis, Universitat Autònoma de Barcelona, 2002.
- [Rot99] V. Roth. Mutual protection of cooperating agents. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1906 of *LNCS*. Springer-Verlag, 1999.
- [RS98] J. Riordan and B. Schneier. Environmental Key Generation Towards Clueless Agents. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.
- [SHKQ99] J.P. Stern, G. Hachez, F. Koeune, and J.J. Quisquater. Robust object watermarking: Application to code. In *Information Hiding*, volume 1768 of *LNCS*. Springer-Verlag, 1999.

Bibliografia

- [Sou] SourceForge projects, Aglet Software Development Kit (ASDK). <http://sourceforge.net/projects/aglets>.
- [ST97] T. Sander and C. F. Tschudin. Towards mobile cryptography. Technical report 97-049, International Computer Science Institute, Berkeley, 1997.
- [ST98] T. Sander and C.F. Tschudin. Protecting mobile agents against malicious hosts. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.
- [Tah] Tahiti user's guide. <http://www.trl.ibm.com/aglets/>.
- [TvS02] A.S. Tanenbaum and M. van Steen. *Distributed Systems. Principles and Paradigms*. Prentice-Hall, 2002.
- [Vig98] G. Vigna. Cryptographic traces for mobile agents. In *Mobile Agents and Security*, volume 1419 of *LNCS*. Springer-Verlag, 1998.
- [WSB99] U. G. Wilhelm, S. Staamann, and L. Buttyán. Introducing trusted third parties to the mobile agent paradigm. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *LNCS*. Springer-Verlag, 1999.
- [X.588] CCITT Recommendation X.500. The directory overview of concepts, models and services, 1988.
- [X.597] ITU/ISO Recommendation X.509. Information technology Open Systems Interconnection - The Directory: Public Key and Attribute Certificate Frameworks, 1997.
- [Yee97] B.S. Yee. A sanctuary for mobile agents. In *DARPA workshop on foundations for secure mobile code*, 1997.