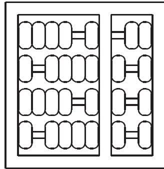

INSTITUTO DE COMPUTAÇÃO

UNIVERSIDADE ESTADUAL DE CAMPINAS

Survey sobre Segurança de Sistemas de Agentes Móveis

Nelson Uto *Ricardo Dahab*

Technical Report - IC-01-008 - Relatório Técnico

July - 2001 - Julho

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

Conteúdo

1 Introdução **1**

2 Segurança de Sistemas de Agentes Móveis **2**
2.1 Tipos de Ataques e Ameaças 3
2.1.1 Ameaças do servidor contra o agente 4
2.1.2 Ameaças do agente contra o servidor 5
2.1.3 Ameaças do agente contra outro agente 6
2.1.4 Outras ameaças contra o sistema de agentes 6
2.2 Requisitos de Segurança 7
2.2.1 Confidencialidade 7
2.2.2 Integridade 8
2.2.3 Autenticação de Entidades 8
2.2.4 Não-repúdio 8
2.2.5 Disponibilidade 8
2.2.6 Anonimato 8

3 Fundamentos **9**
3.1 Fundamentos de Criptografia 9
3.1.1 Ciframento 9
3.1.2 Assinaturas Digitais 9
3.1.3 Funções de Espalhamento Criptográficas 9
3.2 Segurança em Java 10
3.2.1 O *Sandbox* Básico 10
3.2.2 *Class Loader* 11
3.2.3 *Class File Verifier* 11
3.2.4 Características de Segurança da Máquina Virtual Java 11
3.2.5 Gerenciador de Segurança e API Java 12
3.2.6 Assinatura de Código e Autenticação 12

4 Alguns Sistemas de Agentes Móveis **13**
4.1 Telescript 13
4.2 Aglets 14
4.3 Concordia 16
4.4 Ara 19
4.5 D'Agents 20
4.6 Soma 23
4.7 Ajanta 26
4.8 Outros Sistemas 28
4.8.1 Mole 28
4.8.2 Nomads 29

5 Tabela-resumo dos Aspectos de Segurança **29**

A Glossário **31**

Lista de Figuras

1 Ataques a um sistema de agentes móveis. 4
2 Arquitetura do sistema D'Agents. 21

Lista de Tabelas

1 Tabela-Resumo dos Aspectos de Segurança 30

Survey sobre Segurança de Sistemas de Agentes Móveis

Nelson Uto Ricardo Dahab

Instituto de Computação - Unicamp

Resumo

Este trabalho é um *survey* sobre o estado da arte em segurança de sistemas de agentes móveis. Inicialmente, com base em um modelo simplificado de sistemas de agente móveis contendo apenas agentes e servidores, mostramos os problemas de segurança inerentes a estes sistemas e os requisitos de segurança desejáveis. Apresentamos também os fundamentos de Criptografia e de segurança de Java necessários para a compreensão das soluções adotadas pelos sistemas abordados. Dando ênfase aos aspectos de segurança, descrevemos alguns sistemas comerciais e não-comerciais de agentes móveis, apontando deficiências e recomendando algumas soluções. Para finalizar, apresentamos um quadro sinótico que resume as soluções propostas por cada um dos sistemas analisados.

1 Introdução

Agentes móveis são programas que executam tarefas em nome de um usuário e que possuem autonomia para migrar entre os servidores do sistema. Normalmente, são componentes de uma aplicação baseada em agentes e constituem uma entidade ativa que é executada independentemente do processo que as criaram. Há também agentes estacionários que residem em um servidor fixo e oferecem serviços de aplicação aos agentes visitantes. Os servidores fornecem os recursos e o ambiente de execução para os agentes e, juntamente com as entidades necessárias para se prover serviços como resolução de nomes, entre outros, formam uma infraestrutura denominada sistema de agentes móveis.

Os agentes móveis surgiram como uma evolução dos paradigmas cliente-servidor tradicionais usados em sistemas distribuídos como RPC e REV. No modelo RPC (Remote Procedure Call), o cliente efetua uma chamada a um procedimento localizado no servidor. Uma alternativa ao RPC é o modelo REV (Remote Evaluation) no qual o cliente envia uma rotina para ser executada no servidor. O paradigma de agentes móveis apresenta diversas vantagens sobre estes modelos como a redução no tráfego de rede, operação assíncrona com os processos criadores e facilidade para proporcionar tolerância a falhas.

O uso de agentes móveis na criação de aplicações distribuídas tem despertado grande interesse da comunidade científica nos últimos anos; porém, o passo final para a sua adoção comercialmente depende da solução dos problemas de segurança inerentes a estes sistemas: os servidores que executam os agentes estão sujeitos à penetração de agentes maliciosos. Estes podem destruir ou roubar dados importantes ou fazer com que a máquina se torne instável. Outra possibilidade seria um ataque de negação de serviço através do consumo

excessivo de recursos do sistema. Por outro lado, os agentes também estão sujeitos a ataques efetuados pelos servidores ou outros agentes. Servidores maliciosos podem obter dados confidenciais como números de cartão de crédito e dados bancários bem como roubar código proprietário dos agentes.

Este trabalho é um *survey* sobre o estado da arte em segurança de sistemas de agentes móveis. Inicialmente, mostramos os problemas de segurança nestes sistemas e os requisitos de segurança desejáveis. Em seguida, damos alguns fundamentos necessários para a compreensão das soluções adotadas pelos sistemas pesquisados. Por fim, descrevemos alguns sistemas com ênfase nos aspectos de segurança e apresentamos um quadro sinótico das soluções propostas por cada um deles.

O restante deste texto está organizado da seguinte maneira:

- A Seção 2 aborda de forma mais detalhada os problemas de segurança em sistemas de agentes móveis. A partir de um modelo simples, são mostrados os tipos de ataques e ameaças que podem ocorrer nestes ambientes. Em seguida, discutimos os requisitos de segurança que devem ser considerados em uma implementação de tal sistema de forma que o mesmo atenda às necessidades de um sistema real a ser usado comercialmente.

- A Seção 3 introduz fundamentos necessários à compreensão dos mecanismos empregados para solucionar alguns dos problemas de segurança existentes. Descrevemos as ferramentas criptográficas mais comuns como ciframento e assinatura digital e abordamos os pilares da segurança em Java, que é a linguagem mais utilizada nesses sistemas.

- Na Seção 4, discutimos diversos sistemas de agentes móveis mostrando as características gerais de cada um deles e os aspectos de segurança por eles considerados. Procuramos incluir aqueles sistemas que tratam de pelo menos um dos problemas de segurança expostos na Seção 2.

- Finalmente, na Seção 5, apresentamos um quadro sinótico comparando os aspectos de segurança dos sistemas abordados neste *survey*.

2 Segurança de Sistemas de Agentes Móveis

Sistemas de agentes móveis formam uma infraestrutura na qual diversas aplicações pertencentes a diferentes usuários são executadas concorrentemente. Esta infraestrutura é composta pelos servidores e pelas demais entidades necessárias ao sistema para prover serviços como resolução de nomes e aqueles baseados em criptografia de chave pública, entre outros. Os servidores fornecem os recursos e o ambiente de execução para os agentes. Estes, por sua vez, são programas que executam alguma tarefa em nome de um usuário e que podem migrar pelos servidores do sistema, de forma autônoma, para utilizar os recursos necessários à realização da tarefa delegada. Os agentes também podem ser entidades estacionárias e residir em um servidor fixo, oferecendo serviços de aplicação. Neste ambiente, muitos usuários podem não ser confiáveis e querer perpetrar ataques ao sistema. Assim,

o uso de agentes móveis levanta muitas questões de segurança, principalmente quando os mesmos são utilizados em redes abertas como a Internet.

Um exemplo simples de ataque seria contra um sistema que pesquisa o preço de um determinado produto em diversas lojas virtuais com a finalidade de apontar o melhor lugar para compra. A aplicação poderia enviar um agente móvel que percorresse os *sites* das lojas armazenando o preço do produto em cada uma delas. Uma loja desonesta poderia fazer com que o agente a escolhesse simplesmente aumentando os valores coletados nos seus concorrentes.

Não são somente os servidores os responsáveis pelos ataques neste tipo de ambiente. Um agente malicioso pode se aproveitar de falhas de segurança no servidor para alterar ou roubar informações sigilosas. Com tudo isso, fica clara a necessidade de se utilizar mecanismos para detectar e/ou impedir os ataques possíveis. Muitas soluções para esses problemas são baseadas em ferramentas criptográficas, tais como ciframento, assinaturas digitais e protocolos de autenticação de entidades.

2.1 Tipos de Ataques e Ameaças

Para apresentar os principais tipos de ataques e ameaças aos sistemas de agentes móveis vamos utilizar o modelo apresentado na Figura 1. Este modelo é bastante simples e nele o sistema de agentes é composto apenas por dois elementos: agentes móveis e servidores de agentes. Os agentes possuem código e dados e realizam alguma tarefa em nome de um usuário. Para isso, eles podem migrar de forma autônoma para os servidores que ofereçam os recursos necessários à sua execução. A Figura 1 ilustra também as terceiras partes, que são todas as entidades externas ao sistema de agentes que efetuem algum ataque. Os ataques são representados através de setas como a seguir:

$$Atacante \longrightarrow Atacado$$

Cada seta foi rotulada com um número que representa uma das classes de ataques e ameaças possíveis, de acordo com classificação feita pelo NIST [JK99]:

1. ameaças do servidor contra o agente;
2. ameaças do agente contra o servidor;
3. ameaças do agente contra outro agente executando no mesmo servidor; e
4. outras ameaças contra o sistema de agentes.

A grande parte dos sistemas abordados neste *survey* tem boas respostas apenas para as três últimas classes de ameaça. Já o problema de proteção do agente contra servidores maliciosos ainda possui muitos aspectos não resolvidos e é considerado por apenas alguns dos sistemas existentes e de forma parcial. Nas próximas sub-seções fazemos alguns comentários e damos alguns exemplos de cada uma das classes listadas.

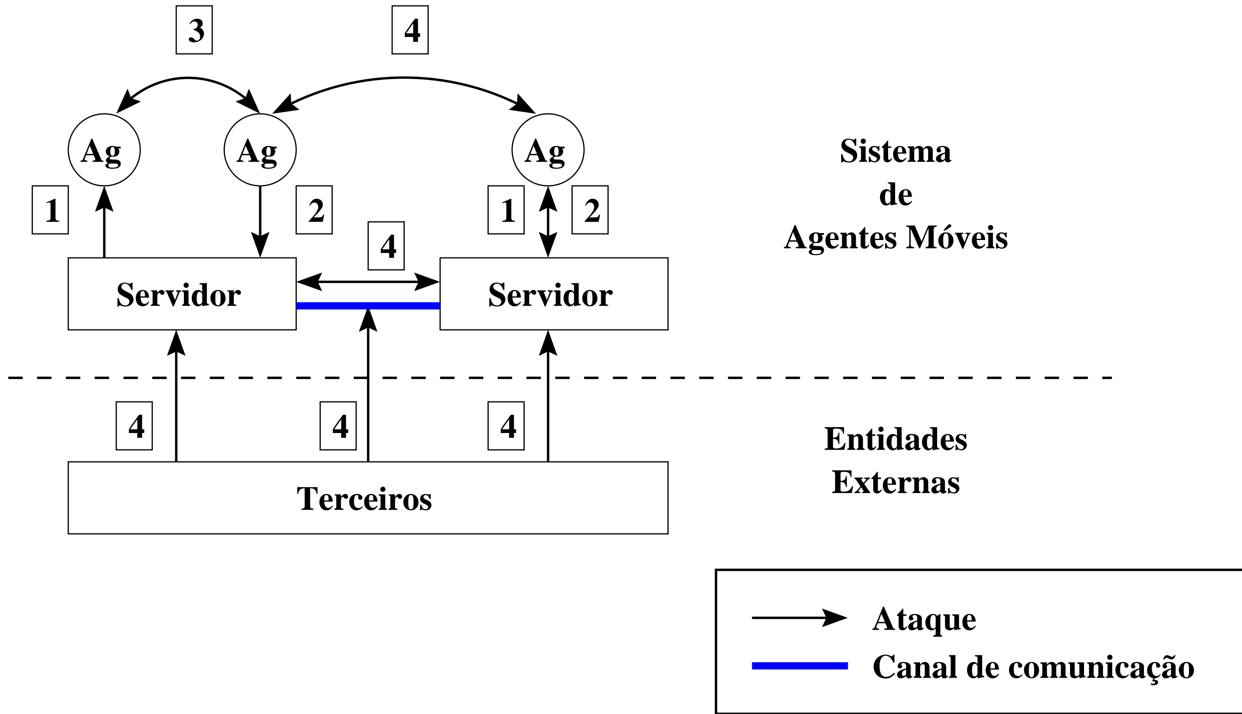

Figura 1: Ataques a um sistema de agentes móveis.

2.1.1 Ameaças do servidor contra o agente

Um agente que executa em um servidor está completamente exposto ao mesmo. Se nenhum mecanismo de proteção for utilizado pelo agente, tanto o seu código como os seus dados serão completamente acessíveis ao servidor, que pode, então, afetar o agente de diversas maneiras. Como, geralmente, parte do estado de um agente precisa ser alterado para armazenar resultados de cálculos ou dados coletados, acredita-se não ser possível garantir, de uma forma geral, que um agente não será alterado de forma maliciosa [FGS96]. Essa classe de ameaças compreende os problemas mais difíceis de serem tratados.

Personificação

Um servidor pode querer personificar um outro servidor, como uma terceira parte confiável, por exemplo, para extrair informações de agentes que tinham como destino o servidor original. Como o servidor malicioso, em caso de um ataque bem sucedido, terá a posse do agente enganado, outros ataques mais danosos poderão suceder à personificação, como os descritos nos parágrafos seguintes.

Negação de Serviço

Quando um agente migra para um servidor, ele espera que o serviço seja executado corretamente e no tempo esperado. Um servidor malicioso pode não atender à solicitação feita, realizar a tarefa delegada em um tempo inaceitável ou finalizar o agente abruptamente.

Este último caso pode fazer a aplicação que enviou os agentes entrar em um estado de *deadlock*, caso ela fique esperando indefinidamente pelo retorno do agente. Uma outra situação é o *livelock* na qual o agente nunca consegue terminar uma tarefa porque o servidor não pára de lhe fornecer serviço.

Eavesdropping

No cenário clássico de "escuta", o atacante intercepta o canal de comunicação estabelecido entre duas partes A e B, que geralmente recebem o nome de *Alice* e *Bob* na literatura. No caso de agentes móveis, além desta possibilidade, há um problema maior: os servidores tem acesso a toda informação não cifrada contida nos agentes visitantes como instruções, dados e resultados. Assim, um servidor malicioso pode obter algoritmos proprietários, números de cartão de crédito ou qualquer outra informação sigilosa.

Alteração

Um agente pode passar por vários servidores, dentre os quais alguns maliciosos; assim, mecanismos que permitam a verificação da integridade do código e dos dados carregados pelo agente devem ser providos aos servidores honestos. Segundo [JK99], não existe ainda uma solução geral para se detectar a alteração maliciosa do estado de um agente sendo executado. Soluções calcadas em linguagens de programação não são suficientes, pois o servidor corrompido pode estar executando uma máquina virtual modificada, por exemplo.

É importante verificar a integridade do agente quando retorna ao *home site*. Embora um processo tenha confiança em seus próprios agentes, estes podem ter sido alterados por algum servidor malicioso, ao longo do itinerário percorrido, com o intuito de causar danos ao servidor de origem.

2.1.2 Ameaças do agente contra o servidor

Nesta categoria de ataques, os agentes procuram explorar fraquezas existentes nos servidores.

Personificação

É o ataque no qual um agente tenta se passar por um outro para utilizar recursos aos quais não tenha acesso ou para não ser responsabilizado por ações que venha a realizar.

Negação de Serviço

Este cenário pode ocorrer quando os agentes consomem excessivamente os recursos do servidor. Isto pode acontecer por *bugs* nos agentes ou propositadamente. Enquanto a primeira causa pode ser atacada através de técnicas de engenharia de software, a segunda requer o uso de mecanismos de segurança. Dependendo dos tipos de recursos que são oferecidos aos agentes, é possível, em alguns casos, tornar o sistema completamente indisponível.

Acesso não Autorizado

O acesso não autorizado a recursos pode comprometer o funcionamento e a segurança de

servidores e dos agentes executando nos mesmos. Por exemplo, em uma aplicação bancária sem o devido controle de recursos, seria possível a um agente malicioso movimentar o dinheiro de contas de terceiros. É necessário, então, estabelecer uma política de segurança e aplicá-la a todo agente que queira utilizar os recursos do servidor em questão.

2.1.3 Ameaças do agente contra outro agente

Esta categoria compreende os ataques em que um agente tenta explorar fraquezas na segurança de outros agentes presentes no mesmo servidor. Os casos de agentes atacando agentes em outras plataformas estão descritos na Subseção 2.1.4, pois estes ataques estão baseados nos serviços de comunicação dos servidores.

Personificação

Um agente pode tentar se passar por outro agente para obter informações confidenciais de um terceiro ou para prejudicar o funcionamento deste último. Pode-se realizar este ataque também com o intuito de manchar a reputação do agente personificado. Um cenário típico deste ataque seria o de um agente personificando um agente de alguma aplicação de comércio eletrônico para obter números de cartão de crédito.

Negação de Serviço

Agentes maliciosos podem realizar um ataque de negação de serviço contra outros agentes, enviando-lhes uma grande quantidade de mensagens desnecessárias. Independentemente de se processar ou bloquear as mensagens, tempo de processamento será desperdiçado pela entidade atacada. Com isso, uma tarefa sendo realizada por esta entidade pode não ser finalizada no prazo estabelecido. Outra forma deste ataque consiste em fornecer informações incorretas para que um agente não funcione apropriadamente.

Repúdio

Quando um agente participante de uma transação ou comunicação nega ter realizado as ações que foram efetuadas tem-se um repúdio. Este pode ocorrer devido a um agente malicioso ou acidentalmente, por perda de uma mensagem. De qualquer forma, grandes disputas podem se originar e, por isso, é importante que a infraestrutura do sistema de agentes tenha mecanismos para poder arbitrar de forma correta tais situações.

Acesso não Autorizado

Se não houver controle de acesso a recursos, um agente malicioso pode alterar dados ou mesmo o código de um outro agente residente no mesmo servidor. O último caso é mais grave ainda pois uma alteração deste tipo provoca uma mudança de comportamento do agente afetado.

2.1.4 Outras ameaças contra o sistema de agentes

Nesta categoria estão os ataques perpetrados por agentes contra agentes situados em outros servidores, ataques de servidores contra servidores e ataques tradicionais em ambiente de

rede explorando falhas de segurança em sistemas operacionais. Um ponto comum nestes ataques é que eles estão baseados nos serviços de comunicação dos servidores e serviços de rede.

Personificação
Um agente pode personificar outro agente para conseguir acesso a um recurso ou a um serviço oferecido por um servidor remoto. Um servidor pode também assumir uma outra identidade para enganar agentes e outros servidores.

Acesso não Autorizado
Agentes remotos ou outras entidades externas podem tentar acessar recursos para os quais eles não possuam os devidos privilégios. É possível também que um atacante consiga acesso ao sistema e destrua os dados da máquina comprometendo o funcionamento do servidor.

Negação de Serviço
Como os serviços dos servidores podem ser acessados remotamente, é possível realizar ataques de negação de serviço comuns contra eles. Pode-se direcionar estes ataques também aos sistemas operacionais e aos protocolos de comunicação.

Eavesdropping
O atacante pode "escutar" o canal de comunicação entre servidores para obter informações de agentes migrantes ou dados armazenados neles. Outra possibilidade seria interceptar um agente ou alguma mensagem para efetuar um ataque por repetição.

Ataque por Repetição
Neste ataque, uma terceira parte maliciosa intercepta um agente ou uma mensagem, realiza uma cópia e depois, clona ou retransmite a mensagem, respectivamente. A interceptação acontece "escutando-se" o canal de comunicação ou em um servidor malicioso, ao receber um agente migrante ou uma mensagem.

2.2 Requisitos de Segurança

A utilização de agentes móveis em redes abertas, como a Internet, gera uma série de problemas de segurança que devem ser abordados. Somente assim estes sistemas poderão deixar o meio acadêmico e ser utilizados de forma ampla no meio comercial. Para que um sistema de agentes móveis seja considerado seguro, ele deve atender a alguns requisitos de segurança como confidencialidade e integridade. Assim, façamos uma breve introdução a esses requisitos.

2.2.1 Confidencialidade

Muitas vezes os agentes carregam consigo código e/ou dados sigilosos e, assim, não devem permitir que servidores e outros agentes não autorizados tenham acesso a essas informações. A mesma consideração é válida para os servidores. Atualmente, existem diversos métodos

para se garantir a confidencialidade de dados. Infelizmente, o mesmo não é verdade para a parte de código, sendo esta, ainda, uma área em desenvolvimento.

É desejável que a confidencialidade seja mantida por toda a infraestrutura de agentes móveis, não se limitindo apenas à relação agente-servidor. Assim, os canais de comunicação pelos quais trafegam agentes e suas mensagens também devem atender a este requisito.

2.2.2 Integridade

Um servidor precisa proteger os agentes contra modificações em seu código, dados e estado e garantir que somente entidades autorizadas possam alterar dados compartilhados. Porém, se o servidor for malicioso, não há meios de se impedir que o agente seja modificado. Neste caso, ao menos, deve-se prover algum mecanismo que permita detectar a modificação indesejada.

2.2.3 Autenticação de Entidades

A autenticação de entidades permite identificar unicamente cada entidade no sistema. Isto é um requisito importante em sistemas de agentes móveis para que se torne possível responsabilizar agentes e/ou servidores pelos atos realizados, bem como conceder privilégios necessários e efetuar cobrança pelo uso de recursos.

2.2.4 Não-repúdio

Não-repúdio é um requisito que não permite que entidades que tenham participado de uma transação possam, em momento algum, negar a sua participação. Para isso, a infraestrutura de agentes móveis deve coletar informações que comprovem qualquer tipo de transação efetuada. É fácil observar como isso é importante, por exemplo, em sistemas de comércio eletrônico.

2.2.5 Disponibilidade

Os servidores devem garantir disponibilidade de dados e serviços aos agentes locais e remotos, prover acesso concorrente ou exclusivo aos recursos e controle de *deadlock*. Se os servidores não estiverem aptos a lidar com a quantidade de requisições realizadas pelos agentes, pode-se criar situações de negação de serviço. Um ataque deste tipo bem sucedido afeta todos os agentes e servidores que dependam da máquina atingida.

2.2.6 Anonimato

Em alguns casos é desejável que a identidade da pessoa responsável pelo agente permaneça no anonimato. Por exemplo, uma pessoa pode querer não se identificar ao responder um questionário de avaliação; ou então ao adquirir algum bem ou ao utilizar um serviço. Casos como o de fornecimento de empréstimos, porém, não permitem anonimato pois é necessário fazer o levantamento do histórico financeiro da pessoa interessada.

3 Fundamentos

3.1 Fundamentos de Criptografia

A seguir daremos uma breve explicação sobre as ferramentas da Criptografia utilizadas para solucionar diversos problemas de segurança encontrados nos sistemas de agentes móveis [MvOV97, Sti95].

3.1.1 Ciframento

Ciframento é um mecanismo criptográfico utilizado para prover confidencialidade de informação. Consiste de funções matemáticas aplicadas a uma mensagem qualquer e a uma chave k_1 que originam uma **mensagem cifrada**. O processo inverso, para recuperar a mensagem original, utiliza uma chave k_2 e é chamado de **deciframento**.

O esquema de ciframento é **simétrico** quando é computacionalmente fácil determinar k_2 a partir de k_1 e vice-versa. Em muitos casos práticos, temos $k_1 = k_2$. Os algoritmos desta classe são rápidos e utilizam chaves relativamente pequenas. Por outro lado, em uma rede grande, há muitas chaves para serem distribuídas e gerenciadas. Exemplos de algoritmos simétricos são DES, AES e IDEA.

Outra classe de agoritmos de ciframento são os **assimétricos** ou de **chave pública**. A característica deste tipo de esquema é que dada a chave k_1 é computacionalmente infactível derivar a chave de deciframento k_2. Os cifradores desta classe são muito mais lentos que os simétricos e necessitam de chaves maiores para garantir o mesmo nível de segurança. Como vantagens temos que somente a chave privada deve ser mantida em segredo e que o número de chaves utilizadas, em uma rede, é pequeno.

Exemplos desses algoritmos são RSA e ElGamal.

3.1.2 Assinaturas Digitais

Assinaturas digitais são primitivas criptográficas utilizadas para garantir integridade, autenticação da origem da mensagem e não-repúdio. Associam a identidade de uma entidade a uma mensagem qualquer. A maior parte dos algoritmos de assinatura digital é baseada em algoritmos de chave pública, que utilizam a chave privada para gerar a assinatura.

Exemplos: RSA, DSA, ElGamal.

3.1.3 Funções de Espalhamento Criptográficas

Estas funções também são chamadas simplesmente de funções *hash*. Segundo definição dada em [MvOV97], uma **função de espalhamento criptográfica** é uma função computacionalmente eficiente que mapeia cadeias binárias de tamanho arbitrário em cadeias binárias de tamanho fixo qualquer, chamadas de **valores *hash***, que se constituem numa espécie de impressão digital da cadeia mapeada.

Uma função de espalhamento criptográfica $h(x)$ deve ter algumas propriedades básicas relacionadas a seguir:

- resistência da pré-imagem - para essencialmente todos os valores *hash* y, é computacionalmente infactível encontrar qualquer pré-imagem x tal que $h(x) = y$ (desde que já não se conheça um tal x, obviamente).

- resistência da segunda pré-imagem - dado um x qualquer é computacionalmente infactível encontrar um $x' \neq x$ tal que $h(x) = h(x')$.

- resistência a colisões - é computacionalmente infactível encontrar dois valores x e x' quaisquer tal que $h(x) = h(x')$.

Esta primitiva criptográfica é normalmente utilizada para detectar modificações ou para gerar códigos de autenticação da origem e conteúdo de dados e é chamada, conforme cada uso, de *modification detection code* (MDC) ou *message authentication code* (MAC) respectivamente; esta última recebe como entrada, além do dado, uma chave secreta.

Exemplos dessas funções são MD4, MD5, SHA-1.

3.2 Segurança em Java

O modelo de segurança de Java é focado em proteger usuários finais contra código malicioso trazido pela rede a partir de servidores não confiáveis. Para isso, todo código externo é executado em um *sandbox* que restringe as operações que podem ser realizadas. O modelo original apresentado na versão 1.0 era extremamente restritivo e isso impedia até programas bem intencionados, mas provenientes de fontes inseguras, de executar qualquer tarefa útil. A versão 1.1 incluía autenticação e assinatura de código e permitia que programas assinados por alguma entidade confiável executassem com todos os privilégios possíveis. Assim, ou depositava-se total confiança em uma classe, ou nenhuma. Com a versão 1.2, finalmente introduziu-se um controle de acesso mais granular estabelecido através de uma política de segurança. A seguir, faremos uma breve exposição dos elementos da arquitetura de segurança de Java ao longo das versões.

3.2.1 O *Sandbox* Básico

O modelo de *sandbox* permite executar código vindo de fontes não confiáveis de forma isolada e impedir que sejam realizadas operações danosas ao funcionamento do sistema. Para implementar o *sandbox*, a arquitetura Java utiliza os seguintes componentes:

- *class loader*;

- *class file verifier*;

- características de segurança da máquina virtual; e

- gerenciador de segurança e Java API.

O primeiro e o quarto componentes podem ser personalizados para se definir políticas de segurança de acordo com a aplicação Java.

3.2.2 *Class Loader*

O *class loader* é responsável por carregar classes na máquina virtual Java. Cada *class loader* define um espaço de nomes separado que é um conjunto de nomes únicos, um para cada classe carregada por este *class loader*. Assim, se a classe X já existe no espaço de nomes Y, é impossível carregar uma classe X diferente em Y. Por outro lado, classes diferentes com o mesmo nome podem coexistir na mesma máquina virtual desde que em espaços de nomes distintos.

Classes em um mesmo espaço de nomes podem interagir diretamente. Por outro lado, classes existentes em espaços de nomes distintos sequer sabem da existência umas das outras. Assim, é possível prevenir que códigos maliciosos interfiram com códigos confiáveis, simplesmente carregando-os através de *class loaders* diferentes.

O *class loader* também previne que classes confiáveis sejam substituídas por código malicioso. Os *class loaders* estão organizados em uma hierarquia cujo topo é o *bootstrap class loader*, responsável por carregar as classes do núcleo da API Java. Quando uma aplicação Java necessita de uma classe pela primeira vez, a máquina virtual solicita que o *class loader* da aplicação a carregue. Esta solicitação é passada nível a nível pela hierarquia até o *bootstrap*. Então, a classe é fornecida pelo nível mais alto que possuí-la. Desse modo, uma aplicação nunca conseguirá utilizar, por exemplo, uma classe `java.util.HashMap` modificada maliciosamente, pois ela será carregada pelo *bootstrap* a partir do núcleo da API Java.

3.2.3 *Class File Verifier*

A função deste módulo é verificar que as classes carregadas atendem a certos requisitos de segurança antes de serem utilizadas pela máquina virtual. Se algo estiver incorreto, uma exceção é gerada.

Num primeiro passo, verifica-se se a classe está em conformidade com a estrutura de uma classe Java: ela deve iniciar com um cabeçalho padrão (bytes 0xCAFEBABE) e possuir o tamanho indicado. Em seguida, são feitos testes de semântica e de tipos de dados que asseguram, entre outras coisas, que descritores de métodos estão corretos e que uma classe segue as especificações da linguagem. O terceiro passo compreende a verificação dos *bytecodes* da classe para garantir que não haja *opcodes* ou operandos inválidos, valores de tipos diferentes do esperado em variáveis e que a execução destes *bytecodes* é segura para a máquina virtual. Por fim, é realizada a verificação de referências simbólicas durante a fase de ligação dinâmica. Nesta fase o verificador confirma a existência de classes, campos e métodos referenciados.

3.2.4 Características de Segurança da Máquina Virtual Java

A máquina virtual Java possui diversas características de segurança que operam enquanto os *bytecodes* são executados:

- *type-safe reference casting* – não é possível utilizar uma referência a um tipo de objeto para manipulá-lo, através de *type-cast*, como um objeto de outro tipo.

- acesso estruturado a memória – Java não permite que o programador declare ponteiros e nem que acesse a memória a partir de uma referência acrescida de um deslocamento. Devido a estas duas últimas características, os programas ficam impedidos de manipular a memória diretamente e corrompê-la de forma acidental ou maliciosa.

- coleta de lixo automática – Java se encarrega de liberar a memória alocada a objetos não utilizados. Desse modo, evita-se que se tente desalocar a memória de um mesmo objeto duas vezes ou que ela seja desperdiçada com objetos não mais referenciados.

- verificação dos limites de vetores – quando um item de um vetor é acessado, a máquina virtual verifica se o índice está dentro dos limites do mesmo. Assim, não é possível armazenar um valor, por exemplo, na nona posição de um vetor com sete elementos, como em C++.

- verificação de referências `null` – toda vez que um programa tenta utilizar uma referência `null`, Java gera uma exceção.

3.2.5 Gerenciador de Segurança e API Java

O gerenciador de segurança determina o que pode ser utilizado por um programa fora do *sandbox* no qual ele é confinado. É possível configurá-lo para estabelecer uma política de segurança particular para uma aplicação. Esta política é garantida pela API Java que sempre consulta o gerenciador antes de executar uma operação potencialmente insegura. Isso é feito chamando métodos de checagem existentes no objeto gerenciador. Por exemplo, o método `checkWrite()` determina quando um *thread* possui permissão para escrever em um determinado arquivo. Quando uma ação não é permitida, ocorre uma exceção de segurança.

Nas versões 1.0 e 1.1, para criar uma política de segurança particular era necessário escrever um gerenciador de segurança a partir da classe abstrata `java.lang.SecurityManager` e implementar seus métodos de checagem. Como isso era uma tarefa difícil e susceptível a erros, na versão 1.2, a classe em questão evoluiu de abstrata para concreta e incluiu uma implementação padrão do gerenciador. Além disso, a política passou a ser especificada em um arquivo ASCII ao invés de código Java.

3.2.6 Assinatura de Código e Autenticação

A versão 1.1 de Java introduziu suporte a autenticação com base em assinaturas digitais. Quando uma parte A deseja atestar que um conjunto de classes não possui código malicioso, ela deve agrupá-las em um arquivo JAR e assiná-lo digitalmente. A autenticação é feita verificando-se a assinatura de A sobre o arquivo e, dependendo da confiança que se deposita em A, pode-se relaxar as restrições do *sandbox* sobre as classes presentes no arquivo JAR.

4 Alguns Sistemas de Agentes Móveis

4.1 Telescript

Informações Gerais

Telescript [TV96] é uma linguagem orientada a objetos para desenvolvimento de aplicações baseadas em agentes. Foi criada pela empresa General Magic, em 1995, com objetivos comerciais. Como a programação de agentes em Telescript requeria o aprendizado de uma linguagem completamente nova, o projeto não obteve o sucesso esperado e foi abandonado. Abordamos Telescript neste *survey* apenas por razões históricas, uma vez que foi um dos primeiros projetos a abordar os problemas de segurança oriundos desta área.

Descrição do Sistema

Os servidores em Telescript são chamados de ***places***. Todo agente que chega a um servidor pode verificar quais serviços são oferecidos por este último. Estes serviços geralmente são fornecidos através de agentes estacionários residentes no servidor que interagem com os agentes recebidos.

Os nomes das entidades em Telescript são dependentes de localização e baseados em nomes de domínio DNS. O modelo de mobilidade é forte permitindo a captura do estado de execução no nível de *thread*. Quando um agente migra para outro servidor, todas as classes relacionadas são transportadas para o novo local. A migração pode ocorrer efetuando-se uma chamada à primitiva `go` que faz com que a próxima instrução do agente seja executada no novo local. Telescript possui também uma migração relativa através da primitiva `meet`. Com ela pode-se fazer o agente migrar para um servidor que esteja hospedando um determinado agente com o qual se queira interagir.

Aspectos de Segurança

Telescript é uma linguagem "segura" do ponto de vista de programação. Não possui ponteiros, como em Java, e qualquer interação com objetos é feita através da interface pública dos mesmos. O interpretador realiza verificação de tipos em tempo de execução, gerenciamento de memória e processamento de exceções. Por fim, o interpretador funciona como um monitor de referências mediando os acessos aos objetos.

Cada processo está associado a uma **autoridade** que é uma identificação única atribuída a um responsável (uma pessoa ou organização). Essa autoridade pode ser autenticada quando necessário, por exemplo, para permitir que um servidor decida se aceita ou não um agente migrante.

Através de **permissões** pode-se limitar o consumo de recursos e restringir a execução de um agente. As permissões podem ser atribuídas pelo criador do agente, pelo *engine* Telescript no momento de criação do processo ou pelo servidor ao aceitar um agente. A intersecção destas permissões atribuídas resulta nas permissões finais do agente. Quando um processo viola um limite estabelecido, uma exceção é sinalizada que, geralmente, causa a finalização do código responsável.

A proteção de agentes durante uma migração de um servidor para outro é realizada através de canais seguros estabelecidos segundo um regime de segurança. Há seis regimes de segurança em Telescript e cada um deles especifica um conjunto de serviços e protocolos criptográficos a ser utilizado na criação do canal. O regime mais simples somente troca as informações das autoridades envolvidas, ao passo que, os mais complexos utilizam autenticação por RSA, troca de chaves através do protocolo Diffie-Helman e ciframento para criar o canal seguro através do algoritmo RC4. O uso de RSA requer uma infraestrutura de chaves públicas que é desempenhada pela **autoridade de nomes**. Seu funcionamento, porém, não é abordado em [TV96]

Comentários

Como Telescript pretendia ser utilizado globalmente, os algoritmos criptográficos foram obrigados a utilizar chaves menores, respeitando-se os limites impostos pela política de exportação dos Estados Unidos nesta área. Com isso, torna-se possível quebrar as chaves utilizadas nestes sistemas.

Todos esses mecanismos apresentados objetivam apenas a proteção do servidor e a proteção do agente contra outros agentes e terceiros. Telescript não possui qualquer recurso para a proteção do agente contra ataques realizados por um servidor.

4.2 Aglets

Informações Gerais

Aglets Software Development Kit (Aglets) [KLO97, LA97, LO98, OKO98], antigamente denominado de Aglets Workbench, é um sistema de agentes móveis desenvolvido pela IBM Tokyo Research Laboratory. O termo *aglet* é uma combinação das palavras *agent* e *applet*. Isso se deve ao fato de Aglets ter sido criado com base no modelo de *applets* de Java. O sistema é baseado na linguagem Java 1.1 e a primeira publicação data de 1996. Recentemente, a IBM disponibilizou o código fonte do sistema sob licença pública da IBM, a qual foi aprovada por Open Source Initiative.

Descrição do Sistema

Os agentes neste sistema recebem o nome de **aglets** enquanto que os servidores são chamados de ***contexts***. Uma máquina na rede pode conter mais de um *context*, cada um sendo identificado por um nome diferente. Um servidor pode ser referenciado pelo URL da máquina adicionado de seu nome. Quando um aglet é aceito em um contexto, ele pode criar outros aglets ou obter a lista dos aglets residindo no mesmo servidor.

O modelo de programação de Aglets é orientado a eventos. Desse modo, para cada evento que pode ocorrer durante o ciclo de vida de um aglet, há um método correspondente que é ativado na ocorrência do mesmo. Por exemplo, quando um aglet é criado e quando chega a um contexto, os métodos `onCreation()` e `onArrival()` são executados, respectivamente. Todos os métodos tratadores de eventos podem ser modificados para atender às necessidades específicas dos agentes.

A mobilidade em Aglets é fraca e baseada na serialização de Java. Portanto, o estado de execução não é capturado no nível de *thread*. Quando o aglet deixa um contexto

e é recriado na máquina destino, o método `run()` é chamado automaticamente. Uma migração ocorre quando um aglet invoca o método `dispatch()` ou quando alguma entidade solicita que o agente retorne a algum servidor específico. Aglets suporta transferência de código sob demanda e completa. Em alguns casos, uma combinação dos dois tipos é utilizada.

Nenhuma referência direta é obtida a um aglet. Aglets utiliza *proxys* que funcionam como representantes do aglet desejado e protegem os métodos públicos do mesmo contra acesso direto. Outra função do *proxy* é garantir transparência de localização. Quando o aglet real estiver localizado remotamente, o *proxy* se encarregará de realizar a comunicação com o servidor remoto. Conforme [OKO98], há um pequeno problema que ocorre quando um aglet migra de servidor. Todos os *proxys* que o referenciavam deixam de ser válidos.

A comunicação entre agentes não é realizada através de invocação de métodos e sim, através da passagem de mensagens. Cada aglet pode possuir um método para tratar as mensagens que recebe de outros agentes. A troca de mensagens pode ser síncrona ou assíncrona.

Aspectos de Segurança

Para descrever os aspectos de segurança de Aglets utilizamos o documento [LO98]. O modelo apresentado em [KLO97] é apenas conceitual, não tendo sido implementado até a versão corrente do sistema.

Um domínio, em Aglets, corresponde a um conjunto de contextos. A abordagem adotada considera que todos os servidores dentro de um domínio são confiáveis. Os servidores pertencentes a um domínio compartilham uma chave secreta que é utilizada para a autenticação. Isto é realizado calculando-se um MAC sobre os dados a serem enviados concatenados com um *nonce*. Esta trinca de informações (dados, *nonce* e MAC) é, então, enviada ao servidor destino que confere o MAC e, se o mesmo estiver correto, obtém a autenticação da parte comunicante como pertencente ao mesmo domínio. Porém, não é possivel determinar exatamente quem é o outro servidor. Este esquema também é utilizado para garantir a integridade dos canais de comunicação.

Quando um servidor recebe um aglet migrante, verifica se o mesmo originou-se de um dos servidores da teia de confiança. Em caso afirmativo, o servidor deposita confiança no aglet recebido, permitindo que o mesmo seja executado.

O acesso a recursos é limitado pelas permissões especificadas no banco de dados de política de segurança. As permissões estão associadas a pares (proprietário, *code base*). Quando um agente tenta acessar algum recurso ou realizar alguma operação, o banco de dados é consultado sobre a permissão correspondente. O par (proprietário, *code base*) utilizado é obtido a partir do objeto `AgletInfo` associado ao agente. As permissões se aplicam a recursos da própria linguagem Java, como leitura e escrita de arquivos, a troca de mensagens e a funcionalidades dos contextos e agentes.

Comentários

Os recursos de segurança de Aglets são escassos e atendem a alguns requisitos apenas

superficialmente. Muito pouco do modelo apresentado em [KLO97] foi implementado. Não há nenhuma preocupação quanto à segurança dos agentes contra servidores maliciosos e todo o sistema está sujeito a diversos ataques.

A autenticação é realizada apenas para identificar se uma máquina pertence a um domínio. Assim, não é possível saber com que servidor a comunicação está sendo realizada. Se uma terceira parte mal intencionada obtiver a senha utilizada para cálculo do MAC, nenhum servidor mais poderá ser autenticado. A solução também é muito pouco flexível e não poderia ser aplicada a redes abertas contendo diversos servidores como a Internet, por exemplo. Usuários e proprietários de agentes não podem ser autenticados e suas identidades são validadas com base na confiança depositada em um servidor.

Os canais por onde trafegam mensagens e agentes não possuem mecanismos para garantir a confidencialidade da transmissão. A única proteção existente é em relação à integridade dos dados enviados.

Não existe controle de acesso sobre a primitiva `retract` e assim, qualquer agente pode solicitar que um agente localizado em um servidor remoto retorne/migre para o servidor no qual o primeiro executa.

4.3 Concordia

Informações Gerais

Concordia [Lab, WpW98] é o sistema de agentes móveis desenvolvido pela Mitsubishi Electric ITA. É baseado na linguagem Java 1.1 e a primeira publicação sobre o sistema data de 1997.

Descrição do Sistema

Um sistema de agentes em Concordia é composto por uma série de servidores executando sobre máquinas virtuais Java. Cada servidor Concordia possui uma série de componentes que são responsáveis por funcionalidades como mobilidade dos agentes, comunicação, administração, gerenciamento de segurança e persistência entre outros.

O componente *Service Bridge* permite a desenvolvedores de agentes/aplicações incluir serviços aos servidores que possam ser utilizados pelos agentes visitantes. Quando um agente deseja localizar um servidor ou um serviço ele consulta o *Directory Manager* que é responsável por manter um registro destas informações. Todo serviço que for disponibilizado a agentes visitantes deve ser registrado previamente com este módulo. No máximo, haverá uma unidade deste módulo por máquina. Os nomes de servidores e serviços são dependentes de localização e baseados em DNS.

Concordia suporta apenas mobilidade fraca valendo-se dos mecanismos de serialização de Java. A transferência de código pode ser feita sob demanda ou transferindo-se todas as classes necessárias ao agente de uma só vez. Para aumentar a confiabilidade na migração, Concordia utiliza uma fila para armazenar os agentes migrantes. Uma cópia adicional também é armazenada em disco e a transmissão do agente é feita segundo o

protocolo *Two Phase Commit*. A cópia do agente só é removida da mídia persistente, após ter sido corretamente transferido para o servidor destino.

O estado interno dos servidores Concordia bem como os estados dos agentes são armazenados em mídia persistente para possibilitar a recuperação do sistema em casos de queda. É permitido também a aplicações e agentes realizar *checkpoints* para, em caso de falhas, reiniciar sua execução a partir de um *checkpoint* desejado. Após uma falha do sistema ou do servidor, o módulo *Agent Manager* é responsável por reinicializar cada agente a partir da mídia persistente. Todo este aparato implica uma degradação do desempenho do sistema. Para garantir uma maior flexibilidade, Concordia permite que os administradores tenham um compromisso entre desempenho e confiabilidade determinando quais módulos do servidor serão armazenados em mídia persistente.

Aspectos de Segurança

A proteção do agente, em Concordia, se preocupa com os problemas de segurança decorrentes da migração dos agentes e de seu armazenamento em disco. Concordia não implementa nenhuma funcionalidade adicional para proteger agentes enquanto estejam em memória, confiando esta tarefa, completamente, aos mecanismos de segurança de Java e do sistema operacional. Quanto à segurança do agente contra servidores maliciosos, não há nada desenvolvido.

Concordia provê comunicação segura entre os servidores através do protocolo SSLv3 (Secure Sockets Layer version 3). Este protocolo fornece serviços de autenticação e ciframento para conexões TCP. Como Concordia realiza comunicação de rede utilizando RMI de Java, que é baseado em TCP/IP *sockets*, basta trocar as bibliotecas padrões de *sockets* por SSL, para se obter transmissão segura nos níveis superiores de rede, de forma transparente.

Um agente é armazenado em disco em decorrência da política adotada por Concordia para conseguir servidores mais confiáveis. Obviamente, isso representa uma brecha para ataques. Assim, para evitar o acesso não autorizado a um agente, as informações do mesmo são cifradas com o uso de um algoritmo simétrico, antes de serem armazenadas. Diversos algoritmos podem ser utilizados (IDEA, DES, RC4 entre outros) e a chave privada é gerada aleatoriamente para cada agente recebido pelo servidor. Esta chave é cifrada por um algoritmo assimétrico e armazenada junto com o agente.

A proteção de recursos dos servidores é baseada em extensões aplicadas ao modelo de segurança de Java. Este modelo é bastante restritivo e concede acessos somente a objetos classificados como confiáveis ou cujos códigos tenham sido escritos e digitalmente assinados por um autor considerado confiável. Em Concordia, a relação de confiança desejada não acontece com o autor e sim, com a pessoa pela qual o agente realiza alguma tarefa.

Todo agente em Concordia está associado a um usuário em particular através de uma **identidade de usuário**. Esta identidade corresponde a um objeto Java contendo um nome de usuário, um grupo de usuário e uma senha. Na realidade, este último corresponde a um valor *hash* calculado sobre a senha de fato. A identidade é carregada

pelo agente durante todo o tempo e é verificada contra uma lista de usuários válidos em cada servidor visitado. A lista de usuários válidos é armazenada em um **arquivo de senhas** que pode ser localizado centralizadamente ou de forma replicada. Este arquivo contém os nomes de usuários e os valores *hash* das senhas. Um *hash* é calculado sobre o arquivo e este é assinado digitalmente pelo servidor para evitar modificação não autorizada do arquivo. A validação de um agente se dá comparando o valor *hash* contida na identidade com o correspondente no arquivo de senhas.

O uso de recursos é condicionado por permissões concedidas de acordo com a identidade associada a um agente. Concordia possibilita que as permissões sejam dadas para aceitar ou negar acesso a recursos tão granulares como a leitura de um arquivo em particular. As permissões se aplicam a todos os recursos controlados pela classe `SecurityManager` e algumas funcionalidades adicionais como criação e suspensão de agentes. Para isso, a classe `SecurityManager` é estendida por Concordia. Quando uma classe tenta utilizar um recurso, o sistema permite a operação se:

- a classe for local; ou
- a classe pertencer a um agente, sua identidade for validada e o mesmo possuir as permissões necessárias no **arquivo de permissões**.

De outro modo, a classe executa dentro de um *sandbox*, com o mínimo de privilégios. O *sandbox* pode ser configurado para permitir o nível de acesso desejado.

Como já mencionado, as permissões sobre recursos são armazenadas em um arquivo de permissões. Este arquivo, igualmente ao arquivo de senhas, é protegido contra modificações indesejadas através de um valor *hash* assinado digitalmente pelo servidor.

Comentários

Concordia não possui nenhum mecanismo para proteger agentes contra servidores maliciosos. A proteção de agentes só ocorre durante a migração e na cópia armazenada para aumentar a confiabilidade.

Em [WpW98] comenta-se que a construção de um agente necessita da senha original, não sendo suficiente apenas o valor *hash* da mesma contida no objeto identidade. Não está claro como é este processo, mas nossa impressão é que a associação da identidade a um agente não é feita de forma segura. Considerando que a validação de um agente compara o valor *hash* presente na identidade com o *hash* pré-calculado contido no arquivo de senhas, conclui-se que o *hash* independe de qualquer código ou dado do agente. Embora seja difícil a uma terceira parte maliciosa obter o objeto identidade de um agente, uma vez que a transmissão de agentes entre servidores é feita por canais seguros e sua cópia armazenada em mídia persistente é cifrada, um servidor malicioso tem acesso a essa informação. De posse deste dado, é possível forjar agentes e associá-los ao usuário relacionado, simplesmente adicionando-se o objeto identidade adquirido.

Para garantir a integridade dos arquivos de senha e de permissões poder-se-ia usar, no lugar do *hash* com a assinatura digital, um MAC, simplesmente.

4.4 Ara

Informações Gerais

O sistema de agentes móveis Ara (Agents for Remote Action) [PS97, Pei97, Pei98] é um trabalho desenvolvido na Universidade de Kaiserslautern e iniciado em 1997. A idéia principal do projeto é adicionar mobilidade às linguagens de programação existentes. Ara suporta as linguagens Tcl, C e C++ e as plataformas Sparc Solaris, Intel Linux e Sparc SunOS. O código fonte é disponibilizado gratuitamente para uso não comercial.

Descrição do Sistema

Os servidores em Ara são chamados de *places* como em Telescript (Seção 4.1). A arquitetura do sistema é composta por um núcleo e por interpretadores para cada linguagem suportada. No núcleo, estão concentradas as funcionalidades independentes de linguagem, como mobilidade e comunicação. Já as questões dependentes de linguagem, como a captura e a restauração do estado de um agente, são resolvidas pelos interpretadores. Serviços de mais alto nível são disponibilizados através de agentes estacionários.

Esta arquitetura torna o sistema flexível e permite adicionar suporte a novas linguagens. Para tanto é necessário criar interfaces (*stubs*) para que agentes escritos na nova linguagem possam efetuar chamadas às rotinas do núcleo. Por outro lado, o interpretador deve fornecer rotinas a serem executadas pelo núcleo (*upcalls*) em casos como a migração, onde a captura do estado de um agente é necessária.

O nome de um servidor é constituído de listas de URLs correspondentes aos protocolos de transporte que podem ser utilizados para se comunicar com o servidor. Já o nome de um agente consiste de um identificador único global, uma identificação do usuário responsável e um nome simbólico opcional.

Ara apresenta mobilidade forte em todas as linguagens atualmente suportadas. Quando um agente deseja migrar para um novo servidor, ele efetua uma chamada à primitiva `go` do núcleo através da interface provida pela linguagem. Na migração, todas as classes necessárias ao agente são transferidas para o novo local.

A comunicação entre agentes é feita por troca de mensagens ao estilo cliente-servidor. Ara encoraja o uso de comunicação local fornecendo, nos servidores, pontos de encontro onde esta interação entre agentes pode ocorrer. Comunicações globais não tem suporte do sistema e devem ser tratadas pelos próprios agentes através do uso dos recursos de rede.

Por fim, Ara permite que os agentes criem um *checkpoint*, quando desejado, para armazenar seu estado interno atual em mídia persistente. Assim, se ocorrer algum problema, é possível restaurar o estado anterior do agente preservado pelo *checkpoint*.

Aspectos de Segurança

O modelo de segurança de Ara considera três entidades para serem autenticadas: usuário do agente, fabricante do agente e *host machines*. O primeiro é o usuário do

sistema responsável pelas ações do agente. O fabricante corresponde ao desenvolvedor do código do agente. Finalmente, *host machine* é uma máquina da rede que abriga um ou mais *places*.

Cada agente carrega consigo um **passaporte** que contém informações necessárias a sua autenticação. Entre estas informações estão um identificador, nome do agente e uma assinatura digital do fabricante sobre o código do mesmo. O passaporte nunca é alterado e é assinado digitalmente pelo usuário durante a criação do agente. Há também atributos de segurança variáveis como o registro de servidores visitados, atualizado a cada migração. Este registro é assinado digitalmente por cada nó visitado após a autenticação da identidade do servidor remetente.

Durante a migração, o agente pode optar se deseja ou não realizar autenticação da origem e do destino bem como o ciframento dos dados transmitidos. Estes recursos são implementados com base no protocolo SSL e o agente deve ter um compromisso entre desempenho e segurança ao utilizá-los.

Uma região em Ara é composta por um conjunto de máquinas que depositam confiança umas nas outras. As regiões são transparentes aos agentes e conhecidas apenas pelo sistema de agentes móveis. Quando um agente migra entre máquinas pertencentes a uma mesma região, os serviços de autenticação e ciframento são automaticamente desabilitados.

Cada *place* em Ara possui um função de admissão que é responsável pelo processo de autorização. Estas funções recebem como parâmetros o passaporte do agente, atributos de segurança e quantidade desejada de cada recurso. Com base nestas informações, aceita-se ou não a entrada do agente no servidor. Neste último caso, um vetor contendo os limites no uso de cada recurso é retornado ao agente. Os limites impostos são definidos pelas quantidades solicitadas e pela política de segurança adotada.

Comentários

Dos sistemas analisados, Ara foi o primeiro a oferecer alguma proteção ao agente contra servidores maliciosos. Ara possibilita verificar a integridade e autenticidade do código do agente, bastando para isso, conferir a assinatura digital que é gerada pelo fabricante sobre o código e que faz parte do passaporte.

Quanto ao uso de criptografia de chaves públicas, em [PS97], os autores relatam que Ara supõe a pré-existência de uma infraestrutura de chaves públicas e, assim, não aborda esta questão. Para a verificação das assinaturas digitais do usuário responsável por um agente e o fabricante de seu código são utilizados os certificados contidos no próprio passaporte.

4.5 D'Agents

Informações Gerais

D'Agents [Gra96, Gra97, GKCR98], antigamente denominado de Agent Tcl, tem sido desenvolvido na Dartmouth College desde 1995 e é patrocinado por diversas entidades como DARPA e Office of Naval Research. O objetivo principal do sistema é suportar

aplicações de recuperação, organização e apresentação de informação distribuída em redes arbitrárias. Os agentes podem ser escritos em Tcl, Java e Scheme. O código fonte do sistema é disponível para uso não comercial.

Descrição do Sistema

A arquitetura do sistema D'Agents é composta de quatro níveis hierárquicos, conforme exibida na Figura 2. O nível mais baixo é uma API para os mecanismos de transporte disponíveis. O segundo nível é o servidor D'Agents que executa em cada máquina. Ele é responsável por manter a lista dos agentes executando na máquina, prover primitivas para comunicação inter-agentes e receber um agente migrante, reinicializando-o, após autenticado, no interpretador apropriado. O terceiro nível contém os ambientes de execução para cada linguagem suportada e a interface deste nível com o inferior é feita através de bibliotecas escritas em C/C++. Por fim, a última camada é composta pelos agentes, que podem ser móveis ou estacionários.

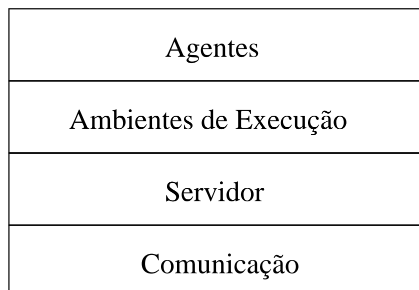

Figura 2: Arquitetura do sistema D'Agents.

O esquema é flexível e permite a inclusão de novas linguagens. Para isso basta criar um novo módulo, no terceiro nível, para a linguagem a ser incluída. Cada módulo é composto de um interpretador para a linguagem, funções para captura/recuperação do estado de um agente, módulo para evitar ações maliciosas dos agentes e uma API para acessar os serviços do servidor.

Quando um agente deseja migrar para um novo servidor, a primitiva `agent_jump` é utilizada. Como o tipo de mobilidade depende das capacidades do interpretador utilizado, D'Agents apresenta mobilidade fraca ou forte, dependendo da linguagem com a qual o agente é escrito. Agentes escritos em Tcl e Java [1] têm seu estado capturado no nível de *thread* de execução, o que configura mobilidade forte. Já agentes escritos em Scheme apresentam apenas mobilidade fraca. Durante a migração, todo código necessário ao agente é enviado ao servidor destino.

A comunicação entre agentes é realizada de duas maneiras: passagem de mensagens e conexão direta. A primeira é utilizada através das primitivas `agent_send` e `agent_receive` que enviam e recebem uma mensagem, respectivamente. A segunda é realizada através da primitiva `agent_meet` que estabele uma conexão com o agente

[1] Uma versão modificada da máquina virtual Java 1.0 é utilizada

desejado. Esta forma é mais eficiente que a primeira para interações mais longas. As mensagens não possuem nenhum formato pré-definido e correspondem simplesmente a cadeias de caracteres. A semântica e a sintaxe das mensagens deve ser combinada pelas partes comunicantes. Agentes escritos em linguagens diferentes podem se comunicar normalmente.

Aspectos de Segurança

Os mecanismos de segurança em D'Agents envolvem, basicamente, as tarefas de autenticação e autorização. Estas tarefas são realizados pelos subsistema de Criptografia, módulo de segurança dependente de linguagem e módulo de política de segurança independente de linguagem.

D'Agents classifica os agentes em *owned* e anônimos. Os primeiros são agentes cujos proprietários podem ser autenticados e constam da lista de usuários autorizados pelo servidor. Os segundos são os agentes para os quais qualquer uma das duas situações anteriores não é satisfeita. Agentes anônimos podem ou não ser aceitos por um servidor, dependendo da sua política de segurança.

Cada usuário e servidor do sistema possui um par de chaves RSA utilizadas para autenticação. D'Agents utiliza PGP para prover os serviços de ciframento e assinatura digital. Estes serviços podem ser usados pelos agentes durante migrações e troca de mensagens. Como é possível desabilitar estes serviços, os agentes devem considerar seu uso conforme cada caso. Uma vez que os algoritmos de chave pública são lentos, a desativação destes recursos implicaria um ganho de desempenho.

Para mostrar como um usuário é autenticado, vamos imaginar que um serivdor X envia um agente pertencente ao usuário Y para o servidor Z. O proprietário do agente, Y, é autenticado se: (1) o agente é digitalmente assinado por Y; ou (2) o agente é digitalmente assinado por X, Z confia em X e X foi capaz de autenticar Y. É importante notar que X poderia ter autenticado Y da mesma forma que em (1) ou (2). Assim, temos que a confiança é transitiva.

Quando um agente registra-se em seu servidor base com o comando `begin` ou quando ele migra pela primeira vez com o comando `jump`, a requisição e o agente, respectivamente, são assinados digitalmente com a chave privada do proprietário do agente. Opcionalmente é possível cifrar os dados envidos com a chave pública do servidor destino. Nas migrações seguintes, o agente é assinado com a chave privada do servidor atual e a autenticação se dá como explicado anteriormente.

Enquanto um agente migra por servidores dentro do conjunto considerado confiável, é possível autenticar seu proprietário direta ou indiretamente da maneira apresentada. Tão logo o agente deixe este conjunto de servidores, ele torna-se anônimo por todo o resto de sua existência.

Em D'Agents há dois tipos de recursos: indiretos, que podem ser utilizados somente por meio de um agente, e nativos (*built-in*), que são acessíveis diretamente através de primitivas da linguagem. Entre exemplos deste último tipo estão recursos como memória e uso de UCP.

O uso de recursos indiretos é controlado diretamente pelo agente responsável. Já os recursos nativos são controlados pelo servidor, pelos módulos de segurança específicos de linguagem e pelos gerenciadores de recursos independentes de linguagem. Estes são compostos por agentes estacionários que são consultados toda vez que um agente quiser acessar um recurso nativo. A consulta é realizada por intermédio dos módulos específicos de cada linguagem. Em ambos os casos, o acesso é concedido se o proprietário do agente possuir permissão para utilizar o recurso, conforme a política de segurança adotada. D'Agents possui seis gerenciadores de recursos distintos. Cada um deles controla uma classe de recursos diferente como o sistema de arquivos e serviços de rede.

Finalmente, D'Agents possui um mecanismo para proteção de um conjunto de máquinas quando as mesmas se encontram sob um mesmo domínio administrativo. Cada agente carrega um vetor que determina, para cada recurso existente, o máximo que pode ser usado e o quanto já foi usado. Se o agente ultrapassar o limite estabelecido ele é terminado. A atualização correta dos valores e o cumprimento dos limites são garantidos pelos módulos de segurança e gerenciadores de recursos .

Comentários

O artigo [GKCR98] cita diversas soluções para o problema de proteção do agente contra servidores maliciosos. Porém, como na maioria dos sistemas, nenhum destes mecanismos propostos foi implementado.

O modelo utilizado para autenticação, assim como em Aglets (Seção 4.2), depende de uma confiança estabelecida entre as máquinas. Isso torna o sistema pouco flexível e de difícil aplicação a muitos sistemas que utilizam redes abertas como a Internet. Além disso, não existe uma infraestrutura de chaves públicas. Os servidores são obrigados a saber previamente as chaves públicas de todos os servidores nos quais depositam confiança.

Quando um agente registra-se no seu servidor base, ele pode ou não estar na mesma máquina. No primeiro caso, a chave privada do proprietário do agente não estará disponível no servidor, após o registro, pois a mesma não é enviada junto com o agente. Mesmo assim, como explicado em [GKCR98], supõe-se que o agente deva ser assinado digitalmente com a chave privada do proprietário, na primeira vez que ele migrar, o que é impossível no caso apresentado.

4.6 Soma

Informações Gerais

O sistema de agentes móveis SOMA (Secure and Open Mobile Agent) [BCS99, CMS99, CCMS99] é um projeto em desenvolvimento no DEIS - Universidade de Bologna e iniciado em 1998. Os objetivos principais do projeto são segurança e interoperabilidade. SOMA suporta a linguagem Java (JDK 2) e as plataformas Win95/NT e Solaris. O binário e o código fonte do sistema estão disponíveis para uso não comercial e podem ser solicitados aos autores do projeto.

Descrição do Sistema

Um servidor em SOMA recebe o nome de *place* como em outros sistemas de agentes móveis. Cada servidor é dividido em diversos módulos: o *Agent Manager*, responsável pela mobilidade dos agentes e pela comunicação entre agentes; o *Local Resource Manager*, que controla o acesso a recursos; o *Distributed Information Service*, que busca informações sobre servidores e agentes localizados remotamente; e o *CORBABridge*, para suporte a CORBA e MASIF.

O sistema SOMA utiliza uma hierarquia de abstrações de localidade para organizar tarefas de gerenciamento e definir políticas de segurança. Cada nó da rede possui pelo menos um servidor para executar agentes. Os servidores são agrupados em **domínios** e cada um destes possui um servidor que funciona como um *gateway* para comunicação entre domínios.

SOMA suporta mobilidade fraca através da primitiva `go` disponível aos agentes. Quando executada, esta primitiva faz com que o agente seja transferido para o novo servidor e que o método especificado seja executado após a migração. A transferência de código para o servidor destino é realizada sob demanda, conforme a necessidade do agente.

A comunicação entre agentes localizados no mesmo servidor é feita através do compartilhamento de recursos como um *buffer*, por exemplo. Já a comunicação com um agente remoto é realizada através da troca de mensagens. Uma mensagem é enviada a um agente mesmo que ele migre para outro servidor.

Para atingir o objetivo de interoperabilidade, SOMA é compatível com CORBA e MASIF. Assim, um agente pode acessar objetos CORBA externos e operar como servidores CORBA, ao passo que o sistema SOMA pode tornar-se acessível a entidades externas através da interface MASIF padrão.

Aspectos de Segurança

SOMA estrutura as políticas de segurança de forma hierárquica baseando-se nas abstrações de localidade. Assim, os domínios determinam autorizações e proibições gerais que podem ser restringidas por cada servidor de acordo com a política de segurança local adotada.

Cada agente carrega consigo suas **credenciais** que são as informações necessárias à autenticação do mesmo. Dentre estas informações estão os nomes do domínio e do servidor origem, a classe que implementa o agente e o usuário responsável por ele. A autenticação ocorre tanto no domínio quanto no servidor e a aceitação de um agente migrante dependerá da política de segurança utilizada.

A autorização para utilizar um recurso depende dos papéis (*roles*) associados ao usuário proprietário do agente. Os papéis definem quais tipos de operações podem ser realizadas sobre os recursos do servidor e sua utilização visa tornar o gerenciamento de permissões mais flexível. Os papéis e as políticas de controle de acesso são especificados na linguagem Ponder [DDLS00] e armazenados em arquivos cifrados. SOMA possui uma ferramenta gráfica para gerenciar estes arquivos em tempo de execução.

A proteção do agente durante a migração entre dois servidores utiliza canais cifrados e

autenticados. É possível utilizar o algoritmo DES para cifrar as informações enviadas ou adotar o protocolo SSLv3. No primeiro caso, a chave utilizada é enviada de um servidor a outro cifrada por RSA ou então estabelecida pelo protocolo Diffie-Hellman.

SOMA apresenta duas soluções para proteger o estado de um agente contra ataques realizados por servidores maliciosos: a primeira depende de uma terceira parte confiável (TTP) que valida os dados coletados pelo agente; e a segunda é um protocolo distribuído, chamado de *Multiple-Hops*, que dispensa o uso da TTP.

Na solução baseada na terceira parte confiável, o agente sempre carrega um MDC (Modification Detection Code) calculado pela TTP sobre os dados coletados. Toda vez que um agente for migrar para um servidor possivelmente malicioso, ele deve, primeiramente, visitar a TTP. Esta verifica o MDC previamente calculado para assegurar que nenhum dado coletado anteriormente foi modificado maliciosamente pelo último servidor. Depois disso, ela recalcula o MDC sobre o MDC anterior e sobre os dados coletados no servidor remetente e envia o agente para o servidor destino.

No protocolo *Multiple-Hops*, o agente pode migrar para quaisquer servidores sem a necessidade de utilizar a TTP. Inicialmente, o *home server* do agente calcula $C_1 = h(C)$, onde C é um número aleatório mantido secreto e $h(C)$ é uma função *hash* livre de colisões. C_1 é enviado cifrado para o primeiro servidor com a chave pública do mesmo. Cada servidor S_i visitado pelo agente calcula um MDC sobre o MDC anterior, os dados incluídos por S_i, C_i e sobre S_{i+1}. O MDC é digitalmente assinado por S_i e armazenado com a assinatura pelo agente. Após isso, S_i calcula $C_{i+1} = h(C_i)$ e o envia cifrado, juntamente com o agente, para o próximo servidor. Quando o agente retornar ao *home server*, este poderá verificar a integridade dos dados coletados recalculando e comparando a cadeia de MDCs.

Comentários

O sistema SOMA apresenta uma grande diversidade de recursos de segurança abrangendo todas as classes de ataques listadas na Seção 2.1. Além disso, é um dos poucos sistemas a se preocupar com a segurança do agente contra servidores maliciosos.

O protocolo baseado em TTP, como apontado pelos autores, não é adequado a redes abertas como a Internet devido a necessidade da existência de nós confiáveis que implementem a TTP. Além disso, o protocolo causa um impacto direto no desempenho do sistema ao obrigar que o agente interaja com a TTP antes de cada migração.

A segurança do protocolo está baseada na suposição de que apenas as TTPs conhecem a função *hash* $h(x)$ utilizada para gerar a prova criptográfica de integridade do estado do agente. Isso é um problema, pois normalmente, a segurança de primitivas criptográficas não deve depender de tal suposição. Se uma entidade maliciosa descobre a função $h(x)$ empregada, ela pode gerar ou apagar itens do estado do agente sem que se possa detectar a modificação indesejada. Ao invés de um MDC, seria recomendado o uso de um MAC cuja chave fosse compartilhada pelo conjunto de TTPs.

Outro problema encontrado é quando o agente visita um mesmo servidor malicioso S_M duas vezes. Durante a primeira visita, o servidor copia e armazena todo o estado

do agente recebido. Como este estado, incluindo a prova criptográfica calculada pela TTP, é válido, basta que durante a segunda visita, S_M substitua o estado atual do agente por aquele armazenado anteriormente. Assim, todos os dados coletados entre as duas migrações são removidas com sucesso. Um ataque semelhante é possível se um servidor malicioso S_i envia o estado do agente recebido a um outro servidor, também malicioso, S_j. Se o agente visitar S_j, este pode substituir o estado daquele pelo enviado por S_i.

O protocolo *Multiple-Hops* é menos custoso que o baseado em TTP, mas ele também é vulnerável a ataques se um servidor puder ser visitado mais que uma vez por um mesmo agente. O servidor malicioso pode armazenar o segredo C_i recebido durante a primeira visita do agente e utilizá-lo, na segunda migração, para remover os dados inseridos pelos servidores S_j, com $j > i$. Um ataque similar pode ser realizado por dois servidores maliciosos S_i e S_k: S_i fornece a S_k o segredo C_i, o dado incluído, o MDC e a sua assinatura digital sobre o MDC. Com esses dados, S_k é capaz de remover os dados incluídos pelos servidores S_j, com $i < j < k$, quando o agente passar por S_k.

Finalmente, como não há controle sobre qual servidor está atualmente hospedando um agente, é possível a um servidor malicioso introduzir uma cópia de um agente simplesmente enviando-o a dois servidores ao mesmo tempo.

4.7 Ajanta

Informações Gerais

O sistema de agentes móveis Ajanta [Kar98, KT98, KT99, TKV+99, TK00, KT00] é um trabalho desenvolvido na Universidade de Minnesota e iniciado em 1997. Suporta a linguagem Java 1.1.5 em plataformas Unix/Linux e necessita de Perl para ser instalado. O binário é disponibilizado gratuitamente para uso não comercial.

Descrição do Sistema

Em Ajanta, um servidor de agentes executa sobre uma máquina virtual Java e deve estar presente em todo nó da rede que deseje suportar aplicações baseadas em agentes. Cada servidor, além de executar agentes visitantes, oferece primitivas para comunicação, migração, controle e monitoração de agentes.

Ajanta apresenta mobilidade fraca como todos os sistemas baseados na máquina virtual Java original. Quando um agente deseja migrar, ele determina o método de sua classe a ser executado após a chegada no servidor destino. A transferência de código necessário à execução do agente é feita sob demanda a partir do servidor base do mesmo.

O tratamento de exceções é realizado pelo próprio agente se possível. Quando o agente não consegue tratar uma exceção, Ajanta o transporta até o **objeto guardião** responsável por ele e notifica o problema. Normalmente, toda aplicação cria um objeto guardião que é incumbido de tratar as exceções não resolvidas pelos agentes por ela utiilizados.

Todas as entidades do sistema (como agentes, servidores e recursos) recebem um nome que as identificam unicamente. Estes nomes são independentes de localização e baseados no modelo URN (Uniform Resource Name) [Moa97, SM94]. Ajanta disponibiliza um **serviço de nomes** responsável por fazer o mapeamento dos nomes das entidades para sua localização atual. Outra função deste serviço é atuar como uma entidade certificadora das chaves públicas utilizadas pelo sistema.

A comunicação entre agentes executando no mesmo servidor é realizada através da invocação de métodos ou do acesso compartilhado a algum recurso. No primeiro caso, um agente se registra no servidor como um recurso e fornece *proxies* (explicado no próximo item) aos agentes que desejem comunicar-se com ele. Além destes mecanismos de comunicação local, Ajanta permite que agentes interajam com pares localizados remotamente através de RMI autenticado.

Aspectos de Segurança

Cada entidade do sistema, como servidores e usuários, pode registrar suas chaves públicas com o serviço de nomes, conforme já mencionado. Estas chaves são utilizadas para assinaturas digitais (DSA) e ciframento (ElGamal) e são associadas ao URN de cada entidade.

A autenticação de entidades é feita com base em um mecanismo simples de desafio-resposta utilizando assinaturas digitais. Ao todo, são trocadas apenas três mensagens e a solicitação de serviço é enviada junto com a última mensagem. Este protocolo opera no nível de aplicação e permite autenticação mútua dos URNs.

Em Ajanta, a migração de agentes entre servidores deve seguir o protocolo de transferência de agentes. A autenticação dos servidores envolvidos e o ciframento dos dados transmitidos podem ser habilitados pelo agente migrante, conforme especificado em suas credenciais. O protocolo permite ao agente determinar o método a ser executado após iniciado no servidor destino. Este verifica as assinaturas nas credenciais e se pode ou não aceitar o agente conforme sua política de segurança. No último passo do protocolo, o servidor origem atualiza a localização do agente com o serviço de nomes.

O acesso a recursos de sistema é controlado pelo **Gerenciador de Segurança do Ajanta** que estende o Gerenciador de Segurança de RMI de Java. O gerenciador utiliza listas de controle de acesso baseados nos URNs dos usuários para liberar o acesso a recursos como o sistema de arquivos e serviços de rede. Já a proteção de recursos de aplicação é realizada através da interposição de um objeto *proxy* entre o recurso e o cliente. O *proxy* possui a mesma interface que o recurso que ele protege, mas quaisquer métodos podem ser desabilitados conforme a política de segurança adotada. Os servidores também podem habilitar/desabilitar os métodos do *proxy* dinamicamente.

Para proteger os agentes contra ataques realizados por servidores maliciosos, Ajanta fornece três mecanismos que permitem detectar modificações indesejadas ao estado do agente. O primeiro consiste de **dados somente para leitura**, inicializados durante a criação do agente. O segundo é um **contêiner somente para inclusão**, que armazena objetos incluídos pelos servidores visitados. Por fim, o último mecanismo é

composto de um vetor de **dados direcionados** em que cada posição tem servidores específicos como destinatários.

Os dados somente para leitura compreendem a parte do estado do agente criada pelo seu proprietário e que não deve ser modificada. Primeiramente é calculado um *hash* de 128 bits sobre estes o qual é assinado digitalmente pelo proprietário do agente com o algoritmo DSA. Esta assinatura é carregada com o agente e pode ser verificada por qualquer servidor utilizando-se a chave pública do proprietário do agente.

O contêiner apenas para inclusão permite coletar dados dos servidores visitados e protegê-los contra remoções e modificações por servidores maliciosos. Para isso, o servidor que desejar incluir um objeto deve assiná-lo digitalmente para associá-lo a sua identidade e garantir a integridade do objeto incluído. Após toda inclusão, um *checksum* é calculado que permite detectar remoções, modificações e inclusões maliciosas.

Para evitar que um dado direcionado seja acessado por outros servidores além do destinatário do dado, o agente, durante sua criação, cifra o dado em questão com a chave pública do servidor destino. Assim, somente este será capaz de decifrar a informação a ele destinada. Os dados direcionados são assinados digitalmente pelo proprietário do agente para impedir que sejam modificados.

Comentários

O sistema Ajanta é o mais completo dos sistemas analisados do ponto-de-vista de segurança, além de possuir ótima documentação dos mecanismos empregados. O sistema se preocupa com todas as classes de ataques apresentadas na Seção 2.1 inclusive ataques perpetrados por servidores maliciosos contra agentes. Isso é um ponto positivo para Ajanta, uma vez que esta classe de ataque, como visto nos outros sistemas, ou não é abordada ou é de forma bem superficial, com exceção de SOMA.

4.8 Outros Sistemas

4.8.1 Mole

O sistema de agentes móveis Mole [BHRS97, SBH97, BHR+98] é um trabalho de mais de cinco anos desenvolvido no IPVR (Institute for Parallel and Distributed Computer Systems) da Universidade de Stuttgart. A última versão é a 3.0, cujo código fonte é disponibilizado gratuitamente para uso não comercial.

Como Mole utiliza a máquina virtual Java padrão, o sistema suporta apenas mobilidade fraca. Quando um agente migra, todas as classes utilizadas são transferidas de uma vez para o servidor destino. A comunicação entre agentes é feita estabelecendo-se uma sessão entre eles, após o que, a interação ocorre através de RMI ou passagem de mensagens.

Mole adota o modelo *sandbox* de segurança no qual os agentes de usuários não podem acessar diretamente nenhum recurso do sistema. O acesso é feito de forma controlada e segura por intermédio de agentes estacionários existentes nos servidores. Para evitar ataques de negação de serviço, o gerenciador de recursos controla o uso de tempo de CPU,

uso da rede, número de agentes filhos criados e tempo total no servidor. Além disso, cada servidor pode especificar o tipo de agente que aceita para execução.

4.8.2 Nomads

O sistema Nomads [SBB$^+$, SBB$^+$00a, SBB$^+$00b] é um projeto em andamento na Universidade de West Florida e suportado pelo DARPA. Nomads tem como objetivo fornecer mobilidade forte e controle granular de recursos. Ele é baseado na lingugem Java e é compatível com as plataformas Windows NT, Solaris e Linux.

O sistema utiliza a máquina virtual Aroma que é compatível com a máquina virtual Java padrão e que foi projetada e implementada para atingir os objetivos listados anteriormente como mobilidade forte. Em uma migração, todas as classes necessárias são transferidas de uma vez só para o servidor destino. A comunicação entre agentes é feita através da troca de mensagens que podem conter qualquer objeto Java serializável.

A biblioteca de código nativo é capaz de limitar o uso de disco e de rede, evitando assim alguns ataques de negação de serviço. Três limites podem ser impostos: taxa de transferência, quantidade e espaço. Por exemplo, pode-se impor a um agente uma taxa de escrita na rede de 100KB/s e determinar que o espaço total em disco utilizado não ultrapasse 5MB.

A política de segurança pode ser estabelecida sobre um usuário individual ou sobre um grupo de usuários. Ela é responsabilidade do módulo *Policy Manager* do servidor (chamado de Oasis) que, na inicialização, lê um arquivo `policies` contendo a política sobre transferência de agentes, controle de execução, controle de acesso e uso de recursos.

5 Tabela-resumo dos Aspectos de Segurança

Para finalizar este trabalho, resumimos os aspectos de segurança de cada sistema analisado agrupando-os na Tabela 1 (página 30).

Sistema de Agentes	Proteção do Agente		Proteção do Servidor		Comunicação Segura
	x Agente	x Servidor	x Agente	x Servidor	
Telescript	Acesso a outro objeto somente através da interface pública	N/A	Limite no uso de recursos; Autorização baseada na autoridade do agente	Autenticação de entidades	Autenticação com RSA e ciframento via RC4
Aglets	Mecanismos Java???	N/A	Autorização baseada no *code base* do aglet	Autenticação de um servidor como pertencente a teia de confiança	Autenticação e integridade via MAC em teia de confiança
Concordia	Mecanismos Java e ciframento dos agentes enquanto em disco	N/A	Estende gerenciador de segurança de Java; Autorização baseada na identidade do usuário responsável pelo agente	Autenticação de entidades	Autenticação e ciframento via SSLv3
Ara	Cada agente executa em um interpretador separado. Comunicação inter-agentes controlada e segura	Integridade e autenticidade do código através de assinaturas digitais; Registro dos servidores visitados	Limitação no uso de recursos determinado por uma função de admissão	Autenticação dos servidores; Regiões de confiança	Autenticação e ciframento via SSL
D'Agents	Acesso a outros agentes somente através de passagem de mensagens	N/A	Autorização baseada no proprietário do agente; Limite no uso de recursos em uma máquina ou em um conjunto de máquinas	Autenticação de entidades	Autenticação e ciframento via PGP
Soma	Cada agente utiliza um espaço de nomes separado	Integridade do código por assinatura digital; Integridade do agente através dos protocolos Multiple-Hops e TTP	Política de segurança especificada na linguagem Ponder e autorização baseada em *roles* associadas a usuários	Autenticação de entidades	Ciframento por DES ou autenticação e ciframento por SSLv3
Ajanta	Domínios de proteção de Java	Integridade do código e estado do agente	Autorização baseada no proprietário do agente; Proteção de recursos por interposição *proxy*	Autenticação de entidades	Ciframento com El-Gamal e autenticação com DSA

Tabela 1: Tabela-Resumo dos Aspectos de Segurança

A Glossário

applet pequenos programas Java que são executados por um navegador Web. Por motivos de segurança, as operações que eles podem realizar são limitadas. Apesar disso, são uma ferramenta poderosa para programação das aplicações clientes em Web.

bytecode linguagem de máquina para uma máquina abstrata que é interpretada por uma máquina virtual.

classe abstrata é uma classe que define apenas uma parte da implementação e serve de base para outras classes que devem prover a implementação de alguns ou todos os métodos.

CORBA (Common Object Request Broker Architecture) infraestrutura para objetos distribuídos padronizada pela OMG (Object Management Group). CORBA automatiza muitas tarefas de programação como registro e localização de objetos entre outras.

deadlock é uma situação na qual tem-se um conjunto de processos que não podem prosseguir porque cada um deles espera por um evento (como a liberação de um recurso, por exemplo) que depende de um processo do próprio conjunto. Como todos os processos estão esperando, nenhum dos eventos necessários ocorrerá.

DNS (Domain Name System é um modelo para gerenciamento de nomes de domínios da Internet de forma hierárquica e distribuída. O DNS é responsável por mapear nomes de máquinas para endereços IP e vice-versa.

gateway dispositivo que interliga duas redes e pode operar em níveis a partir da camada 3 do modelo RM-OSI. Quando ele se encarrega do roteamento de pacotes é chamado de roteador e opera na camada de rede. Quando sua função é traduzir mensagens de uma rede para outra que utiliza um protocolo diferente para a camada em questão é chamado de tradutor de protocolo.

livelock situação em que um processo não consegue terminar uma tarefa porque o servidor lhe fornece serviço indefinidamente.

MASIF (Mobile Agent System Interoperability Facility) é um padrão da OMG para um *middleware* que possibilita interações transparentes de localização entre agentes/objetos estáticos e móveis. A idéia é alcançar interoperabilidade entre sistemas de agentes móveis de diferentes fabricantes sem a necessidade de grandes modificações nas plataformas.

mobilidade forte modelo de migração em sistemas de agentes móveis no qual o estado do agente é capturado no nível de *thread* de execução ou processo. Assim, tanto o estado da pilha como o *program counter* são transportados para o servidor destino além das estruturas definidas pelo agente. Com isso, a migração pode ocorrer em qualquer ponto da execução do agente e o mesmo continuar, na máquina destino, a partir do ponto em que foi interrompido.

mobilidade fraca modelo de migração em sistemas de agentes móveis no qual o estado que é capturado do agente consiste essencialmente das estruturas de dados definidas pelo mesmo. O estado da pilha de execução e o *program counter* não são transportados para o servidor destino e, assim, o agente não reinicia do ponto em que ocorreu a migração.

nonce um valor utilizado não mais que uma vez para o mesmo propósito. Tipicamente é usado para prevenir ataques por repetição [MvOV97].

opcode parte de uma instrução de computador que define a operação a ser realizada.

RMI (Remote Method Invocation) possibilita efetuar uma chamada a um método de um objeto localizado remotamente e receber o resultado de volta, como se o objeto estivesse na máquina local.

serialização de objetos em Java permite transformar qualquer objeto que implementa a interface `Serializable` em uma sequência de bytes independente de plataforma.

sockets é uma interface de programação desenvolvida para a linguagem C e utilizada para comunicação. Ela utiliza chamadas de sistema do Unix e adiciona algumas outras para suportar TCP/IP.

thread também chamado de *lightweight process*, é uma subtarefa de um processo que possui PC, registradores e pilha próprios. Compartilha com os demais *threads* que compõem o processo a seção de dados, código e recursos de sistema operacional.

Referências

[BCS99] Paolo Bellavista, Antonio Corradi, and Cesare Stefanelli. A Secure and Open Mobile Agent Programming Environment. In *4th International Symposium on Autonomous Decentralized Systems (ISADS'99)*, Mar 1999.

[BHR+98] Joachim Baumann, Fritz Hohl, K. Rothermel, M. Schwehm, and Markus Straβer. Mole 3.0: A Middleware for Java-Based Mobile Software Agents. In *Middleware'98*. Springer-Verlag, 1998.

[BHRS97] Joachim Baumann, Fritz Hohl, K. Rothermel, and Markus Straβer. Mole - Concepts of a Mobile Agent System. Technical report, University of Stuttgart, 1997.

[CCMS99] Antonio Corradi, Marco Cremonini, Rebecca Montanari, and Cesare Stefanelli. Mobile Agents and Security: Protocols for Integrity. In *2nd International Working Conference on Distributed Applications and Interoperable Systems*, 1999.

[CMS99] Antonio Corradi, Rebecca Montanari, and Cesare Stefanelli. Mobile Agents Protection in the Internet Environment. In *COMPSAC'99*, Oct 1999.

[DDLS00] Nicodemos Damianous, Naranker Dulay, Emil Lupu, and Morris Sloman. Ponder: A Language for Specifying Security and Management Policies for Distributed Systems – The Language Specification – Version 2.3. Technical report, Department of Computing – Imperial College of Science, Technology and Medicine, Oct 2000.

[FGS96] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for Mobile Agents: Issues and Requirements. In *19th National Information Systems Security Conference*, pages 591–597, Oct 1996.

[GKCR98] Robert S. Gray, David Kotz, George Cybenko, and Daniela Rus. D'Agents: Security in a multiple-language, mobile-agent system. In Giovanni Vigna, editor, *Mobile Agents and Security*, volume 1419 of *Lecture Notes in Computer Science*, pages 154–187. Springer-Verlag, 1998.

[Gra96] Robert S. Gray. Agent Tcl: A flexible and secure mobile-agent system. In *1996 USENIX Tcl/Tk Workshop*, pages 9–23, 1996.

[Gra97] Robert S. Gray. *Agent Tcl: A flexible and secure mobile-agent system*. PhD thesis, Dartmouth College, Jun 1997.

[Jan00] Wayne Jansen. Countermeasures for Mobile Agent Security. *Special Issue on Advanced Security Techniques for Network Protection, Elsevier Science BV*, Nov 2000.

[JK99] Wayne Jansen and Tom Karygiannis. Mobile Agent Security. Technical report, National Institute of Standards of Technology, Aug 1999.

[Kar98] Neeran Karnik. *Security in Mobile Agent Systems*. PhD thesis, University of Minnesota, 1998.

[KLO97] Gunter Karjoth, Danny B. Lange, and Mitsuru Oshima. A Security Model for Aglets. *IEEE Internet Computing*, 1(4):68–77, 1997.

[KT98] Neeran M. Karnik and Anand R. Tripathi. Design Issues in Mobile-Agent Programming Systems. *IEEE Concurrency*, pages 52–61, Jul–Sep 1998.

[KT99] Neeran M. Karnik and Anand R. Tripathi. Security in the Ajanta Mobile Agent System. Technical report, University of Minnesota, May 1999.

[KT00] Neeran M. Karnik and Anand R. Tripathi. A Security Architecture for Mobile Agents in Ajanta. In *20th IEEE International Conference on Distributed Computing Systems*, 2000.

[LA97] Danny B. Lange and Yariv Aridor. Agent Trasnfer Protocol – ATP/0.1. Mar 1997.

[Lab] Mitsubishi Electric ITA Horizon Systems Laboratory. Technology at a Glance – Concordia - Java Mobile Agent Technology. In http://www.concordiaagents.com/documents.htm.

[LO98] Danny B. Lange and Mitsuru Oshima. A Security Model for Aglets. *World Wide Web*, Special Issue:111–121, 1998.

[Moa97] R. Moats. RFC 2141: URN Syntax, 1997.

[MvOV97] A. Menezes, P. van Oorchot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[OKO98] Mitsuru Oshima, Guenter Karjoth, and Kouichi Ono. Aglets Specification 1.1 Draft. Sep 1998.

[Pei97] Holger Peine. Ara - Agents for Remote Action. In William R. Cockayne and Michael Zyda, editors, *Mobile Agents: Explanations and Examples with CD-ROM*, pages 96–164. Manning Publications Co., 1997.

[Pei98] Holger Peine. Security Concepts and Implementation in the Ara Mobile Agent Systems. In *WETICE'98*, Jun 1998.

[PS97] Holger Peine and Torsten Stolpmann. The Architecture of the Ara Platform for Mobile Agents. In *First International Workshop on Mobile Agents MA'97*, Apr 1997.

[SBB+] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, Allan R. Ditzel, Gregory A. Hill, Brian R. Pouliot, and David S. Smith. NOMADS: Toward an Environment for Strong and Safe Agent Mobility.

[SBB+00a] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, Paul T. Groth, Gregory A. Hill, and Renia Jeffers. Strong Mobility and Fine-Grained Resource Control in NOMADS. In *Second International Symposium on Agent Systems and Applications*, Sep 2000.

[SBB+00b] Niranjan Suri, Jeffrey M. Bradshaw, Maggie R. Breedy, Paul T. Groth, Gregory A. Hill, Renia Jeffers, and Timothy S. Mitrovich. An Overview of the NOMADS Mobile Agent System. In *6th ECOOP WORKSHOP ON MOBILE OBJECT SYSTEMS: Operating System Support, Security and Programming Languages*, Jun 2000.

[SBH97] Markus Stra*β*er, Joachim Baumann, and Fritz Hohl. Mole - A Java Based Mobile Agent System. In M. Muehlhaeuser, editor, *Special Issues in Object Oriented Programming*, pages 301–308. Springer-Verlag, 1997.

[SM94] Karen Sollins and Larry Masinter. RFC 1737: Functional Requirements for Uniform Resource Names, 1994.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, Mar 1995.

[TK00] Anand R. Tripathi and Neeran M. Karnik. Delegation of Priviliges to Mobile Agents in Ajanta. In *First International Conference on Internet Computing*, Jun 2000.

[TKV+99] Anand R. Tripathi, Neeran M. Karnik, Manish K. Vora, Tanvir Ahmed, and Ram D. Singh. Mobile Agent Programming in Ajanta. In *19th International Conference on Distributed Computing Systems*, pages 190–197, May 1999.

[TV96] Joseph Tardo and Luis Valente. Mobile Agent Security and Telescript. In *IEEE COMPCON Spring 96*, pages 58–63, 1996.

[Ven99a] Bill Venners. *Inside the Java 2 Virtual Machine*, chapter 1 – Introduction to Java's Architecture. McGraw-Hill Professional Publishing, 2nd edition, Dec 1999.

[Ven99b] Bill Venners. *Inside the Java 2 Virtual Machine*, chapter 3 – Security. McGraw-Hill Professional Publishing, 2nd edition, Dec 1999.

[WpW98] Tom Walsh, Noemi paciorek, and David Wong. Security and Reliability in Concordia™. In *31st Hawaii's International Conference on System Sciences (HICSS'98)*. IEEE Computer Society, Jan 1998.