Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Patrícia Bonezi Nunes da Mota

Estudo comparativo entre modelos de segurança para sistemas de desenvolvimento de agentes móveis

São Paulo
2009

Patrícia Bonezi Nunes da Mota

Estudo comparativo entre modelos de segurança para sistemas de desenvolvimento de agentes móveis

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do titulo de Mestre em Engenharia de Computação.

Data da aprovação ____/____/_____

_____
Prof. Dr. Wagner Luiz Zucchi
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Wagner Luiz Zucchi (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Profª. Drª. Pollyana Notargiacomo Mustaro (Membro)
USP- Universidade São Paulo

Prof. Dr. Marcelo Novaes Rezende (Membro)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Patrícia Bonezi Nunes da Mota

Estudo comparativo entre modelos de segurança para sistemas de desenvolvimento de agentes móveis

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para a obtenção do titulo de Mestre em Engenharia de Computação.

Área de Concentração: Redes de Computadores

Orientador: Prof. Dr. Wagner Luiz Zucchi

São Paulo

Nov./2009

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

M917e **Mota, Patrícia Bonezi Nunes da**
Estudo comparativo entre modelos de segurança para sistemas de desenvolvimento de agentes móveis. / Patrícia Bonezi Nunes da Mota. São Paulo, 2009.
98p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Redes de Computadores.

Orientador: Prof. Dr. Wagner Luiz Zucchi

1. Segurança 2. Agentes móveis Ajanta 3. Engenharia de software 4. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Coordenadoria de Ensino Tecnológico II.Título

10-27 CDU 004.056'41(043)

Dedicatória

Dedico este trabalho a todos que me apoiaram nesta longa trajetória.

Aos meus pais que me deram base nos anos iniciais da minha vida, para chegar ao final deste mestrado, me colocando em boas escolas e me incentivando em pesquisas e muita leitura.

Ao meu marido Roberto que, com muita paciência me viu entrar em desespero, apagar e recomeçar todo o trabalho inúmeras vezes e sempre esteve ao meu lado, acreditando que isso seria possível e que me tornaria mestre.

Aos meus filhotes peludos, Antônio e Alice que estiveram ao meu lado durante todo o desenvolvimento do trabalho, ronronando e arranhando a tela do meu micro, na tentativa em vão de pegar o “rato” e me fazer parar de trabalhar para dar um mínimo de atenção para eles.

Agradecimentos

Agradeço à paciência do meu professor e mestre Wagner Luiz Zucchi e ao incansável apoio nas leituras e correções intermináveis do meu trabalho e a todos esses anos em que acreditou que eu conseguiria terminar esta dissertação.

E ao pessoal da secretaria do IPT que me apoiou, aguentou meus pedidos de extensão de prazo e sempre, com sorriso estiveram prontos para ouvir e ajudar para que no final tudo desse certo. Obrigada a todos!

Resumo

A utilização de agentes móveis em ambientes com sistemas distribuídos é recomendada para que as plataformas de redes com baixa latência e alta disponibilidade sejam consideradas seguras. Os agentes móveis têm como características: facilidade de programação, leveza na execução do programa na plataforma e maior segurança de código móvel. Trafegam na Internet, em ambiente de rede sem controle e em ambiente computacional com sistemas distribuídos. Entretanto, a utilização de agentes móveis vem sendo postergada devido à necessidade de garantir a segurança, pois mesmo com toda a eficiência em contornar os riscos, vulnerabilidades e falhas da tecnologia, os métodos de desenvolvimento clássicos de programas distribuídos não são suficientes para dirimir todas as ameaças existentes. Não pode existir um ambiente completamente seguro, onde as falhas e ameaças de segurança estão presentes. Além da preocupação com a segurança, os agentes móveis contribuem para restrições de desempenho, que muitas vezes, limitam os seus benefícios para certas aplicações. Neste trabalho são avaliados os aspectos de segurança da plataforma de desenvolvimento de agentes móveis Ajanta, em laboratório, com a finalidade de validar os aspectos de segurança desta plataforma e discutidos aspectos pertinentes à segurança.

Palavras-chaves: agentes móveis; rede; JAVA; sistemas distribuídos; ambiente seguro; segurança.

Abstract

Comparative study of security models for systems development for mobile agents

The use of mobile agents in environments with distributed systems is recommended so that the platforms of nets with low latency and high availability become safe. The mobile agents have as characteristic: easiness of programming, the slightness in the execution of the program in the platform and greater security of mobile code. They pass through in the Internet, environment of net without control and computational environment with distributed systems. However, the use of mobile agents comes being delayed due to necessity to guarantee the security, therefore exactly with all the efficiency in skirting the risks, vulnerabilities and imperfections of the technology, the classic methods of development of distributed programs are not enough to nullify all the existing threats. A completely safe environment cannot exist, where the imperfections and threats of security are gifts. Beyond the concern with the security, the mobile agents contribute for performance restrictions, that many times, deny its benefits for certain applications. In this work the aspects of security of the platform of development of mobile agents Ajanta are evaluated, in laboratory, with the purpose to validate the aspects of security of this platform and presented pertinent conclusions to the security.

Keywords: mobile agents; net; JAVA, distributed systems; secure environment; security.

Lista de Ilustrações

Figura 1 – Mecanismos de mobilidade e classes..... 21
Figura 2 – Mecanismos de delegação dos certificados SPKI/SDSI..... 25
Figura 3 - Arquitetura de Sistemas distribuídos com e sem agentes móveis..... 26
Figura 4 – Agente móvel em um ambiente computacional e com a presença de atacante ativos e passivos..... 29
Figura 5 – Tráfego do agente do servidor 1 para o servidor 2 – validação de segurança dos agentes móveis..... 50
Figura 6 - Configuração de variáveis de ambiente..... 51
Figura 7 – Diretórios de configuração do Ajanta..... 52
Figura 8 – Arquivos de instalação do Ajanta..... 53
Figura 9 – Configuração de variáveis de ambiente..... 54
Figura 10 – Configuração do User Setup..... 54
Figura 11 – Configuração do nome do usuário na máquina cliente..... 55
Figura 12 – Configuração do Name Registry Administrador ou Usuário..... 56
Figura 13 – Arquivos de configuração do Ajanta..... 56
Figura 14 – Janela de criação de chaves..... 57
Figura 15 – Colocação da chave 512..... 57
Figura 16 – Modelo de Segurança JDK..... 60
Figura 17 – Arquitetura de comunicação do serviço RMI..... 64
Figura 18 – Funcionamento do *rmiregistry* em aplicações distribuídas..... 65
Figura 19 – Topologia da rede do Laboratório..... 74
Figura 20 – Tráfego do protocolo RMI..... 87
Figura 21 – ProtocolAck..... 88
Figura 22 – Serialização de Objeto..... 89
Figura 23 – Serialização de Objeto II..... 89
Figura 24 – Serialização de Objeto III..... 90
Figura 25 – PingAck..... 90
Figura 26 – Retorno dos objetos remotos..... 91

Lista de Tabelas

Tabela 1 – Fluxo de saída 76

Tabela 2 – Mensagem de saída..... 78

Tabela 3 – Fluxo de entrada 79

Tabela 4 – Método de serialização 80

Tabela 5 – Retorno de um código..... 80

Tabela 6 – Lista de símbolos utilizados pelo RMI WIRE..... 81

Tabela 7 – Registros de tamanho variável..... 83

Tabela 8 – Operação CLOSE 84

Tabela 9 – Operação CLOSEACK..... 85

Tabela 10 – Operação REQUEST 85

Tabela 11 – Operação TRANSMIT 86

Lista de abreviaturas e siglas

AES - *Advanced Encryption Standard*

AES - *Advanced Encryption Standard*

APIs - *Application Programming Interface*

CORBA - *Common Object Request Broker Architecture*

CPU - Unidade Central de Processamento

DAC - *Discretionary Access Control*

DCOM - *Distributed Component Object Model*

DNS - *Domain Name System*

***DoS* -** *Denied of Service*

GUI - Interface Gráfica do Usuário

HTML - *HyperText Markup Language*

HTTP - *Hypertext Transfer Protocol*

IBM - *International Business Machines*

IPVR - *Institute for Parallel and Distribute Computer Systems*

ISO - *International Organization for Standardization*

JDK - *Java Development Kit*

JVM - *JAVA Virtual Machine*

MAC - *Mandatory Access Control*

MASIF - *Mobile Agent System Interoperability Facility*

OMG - *Object Management Group*

ORB - *Object Request Broker*

RBAC - *Role-Based Access Control*

RMI - *Remote Method Invocation*

RPC - *Remote Procedure Call*

RSA - *Rivest Shamir Adelman*

RSA - *Rivest Shamir Adelman*

SOMA - *Secure Open Mobile Agent*

SPKI/SDSI - *Simple Public Key Infrastructure/Simple Distributed Security Infrastructure*

URN - *Uniform Resource Name*

WEB - *World Wide Web*

SUMÁRIO

1 INTRODUÇÃO 14

1.1 MOTIVAÇÃO 14
1.2 OBJETIVO 17
1.3 CONTRIBUIÇÕES ESPERADAS 18
1.4 METODOLOGIA 19
1.5 ORGANIZAÇÃO DO TRABALHO 19

2 A SEGURANÇA NA MOBILIDADE DE CÓDIGO E OS AGENTES MÓVEIS 20

2.1 CÓDIGOS MÓVEIS 20
2.2 MOBILIDADE DE CÓDIGO 20
2.3 AGENTES MÓVEIS 22
2.4 A MOBILIDADE DE CÓDIGO EM SISTEMAS DISTRIBUÍDOS COM AGENTES MÓVEIS 25
2.5 POLÍTICAS, MODELOS E MECANISMOS DE SEGURANÇA 30
2.6 MECANISMOS DE SEGURANÇA 32
2.6.1 AUTENTICAÇÃO E AUTORIZAÇÃO 32
2.6.2 CONTROLES DE ACESSO 33
2.6.3 CONTROLES CRIPTOGRÁFICOS 34
2.7 A LINGUAGEM DE DESENVOLVIMENTO JAVA 36
2.8 PLATAFORMAS DE DESENVOLVIMENTO DE AGENTES MÓVEIS DISPONÍVEIS NO MERCADO 38
2.8.1 AJANTA 39
2.8.2 AGLET 39
2.8.3 CONCÓRDIA 40
2.8.4 GRASSHOPPER 41
2.8.5 VOYAGER 41
2.8.6 MOLE 41
2.8.7 SOMA 42
2.9 ASPECTOS DE SEGURANÇA JAVA E COMPONENTES DAS FERRAMENTAS 44

3 SISTEMAS COMPARADOS E MEIOS DE AVALIAÇÃO 46

3.1 A PLATAFORMA AJANTA 46
3.2 CARACTERÍSTICAS E CAPACIDADES DE SEGURANÇA DA PLATAFORMA DE DESENVOLVIMENTO AJANTA 47
3.3 INSTALAÇÃO DO AJANTA E LABORATÓRIO 48
3.4 SEGURANÇA DE EXECUÇÃO NO AMBIENTE JAVA 58
3.5 INVOCAÇÃO DE MÉTODOS REMOTOS 61
3.6 CONCEITO DO RMI 62
3.7 APLICAÇÕES CLIENTE E SERVIDOR 62
3.8 ARQUITETURA RMI 63
3.9 IMPLANTAÇÃO DO RMI 66
3.10 JAVA.RMI 66
3.11 CLASSES E SUBCLASSES NO JAVA.RMI 68
3.12 JAVA. RMI.REGISTRY 68
3.13 JAVA.RMI.SERVER, SUAS CLASSES E SUBCLASSES 69
3.14 IMPLEMENTANDO INTERFACES REMOTAS 69
3.15 LIMITAÇÕES DAS FERRAMENTAS DE SEGURANÇA EM AGENTES MÓVEIS 71
3.16 ASPECTOS DE SEGURANÇA EM PLATAFORMAS DE AGENTES MÓVEIS DISPONÍVEIS E LABORATÓRIO 73

4 ANÁLISE EXPERIMENTAL DOS MECANISMOS DE SEGURANÇA DA PLATAFORMA AJANTA 74

4.1 O PROTOCOLO RMI WIRE 75
4.1.1 VALORES DO RMI WIRE 81

4.1.2 PROTOCOLO DE MULTIPLEXAÇÃO 81
4.1.3 CONTROLE DE FLUXO DOS DADOS 82
4.1.4 FORMATO DO PROTOCOLO 83
4.1.5 OPEN 84
4.1.6 CLOSE 84
4.1.7 CLOSEACK 85
4.1.8 REQUEST 85
4.1.9 TRANSMIT 86
4.2 ANÁLISE DO TRÁFEGO DOS DADOS EM LABORATÓRIO 86
5 CONCLUSÃO 92
5.1 TRABALHOS FUTUROS 93
REFERÊNCIAS 94
GLOSSÁRIO 96

1 INTRODUÇÃO

Esta seção tem como objetivo apresentar a motivação para o desenvolvimento deste trabalho.

1.1 Motivação

Os computadores pessoais surgiram na década de 1970[1], e possuíam cerca de 8 bits de memória interna e 8 bits de memória externa[2], trabalhando com a tecnologia de processamento 8080, e mesmo assim já havia indício de mobilidade de código com a submissão de arquivos em lote para os supercomputadores.

Com a evolução da computação, a possibilidade da interligação de sistemas computacionais distintos, formando uma rede de comunicação de dados, permitiu que se passasse a pensar, a imaginar e a perceber a necessidade real da mobilidade de código. Estes processos em execução trabalham em uma máquina cliente transferindo os dados para o servidor, normalmente podem ter sistemas operacionais distintos. Estas atividades são chamadas de migração de objetos ou de código, surgindo o conceito de mobilidade de código.

Esta mobilidade de código é a capacidade de um sistema interligado a outro, de trocar informações para atualização de uma única base de dados, mesmo que estes sistemas estejam em computadores distintos, como em um servidor que armazena os dados e na máquina cliente que executa o sistema.

A migração dos dados dentro de um sistema é a capacidade de movimentação de partes do sistema entre servidores e clientes. Em aplicações interativas, principalmente em ambiente da internet, a transferência de dados da origem para o destino exige que o banco de dados seja acionado muitas vezes.

Para que estes sistemas não sejam acionados muitas vezes, necessitando de mais tempo, os usuários preenchem formulários na tela que são traduzidos em operações dentro do sistema e demandam ações de idas e voltas ao banco de dados, evitando o prejuízo do desempenho da máquina e da aplicação. O modo de processar este formulário no lado cliente e enviar de uma única vez o formulário

[1] Apple Corporation, www.apple.com
[2] Intel Corporation, www.intel.com

preenchido faz com que se ganhe tempo nesse processo. Este é o meio de buscar o aumento do desempenho da máquina, na tentativa de diminuir a utilização de processamento desnecessário. Isto não garante que esses dados cheguem ao servidor em segurança, o que acontece normalmente em sistemas distribuídos.

Sistemas distribuídos são conjuntos de tecnologias de linguagens e plataforma que suportam a construção de programas de computador ou conjunto de computadores independentes, que se apresentam ao usuário como um sistema único e consistente. Antes do desenvolvimento dos sistemas distribuídos uma das necessidades mais importantes para os sistemas de códigos móveis era a solução de problemas de balanceamento de carga nas redes (Tanembaum, 2008).

O balanceamento de carga nas redes era uma tentativa de deixá-las mais ágeis, com capacidade maior de trafegar informações do cliente para o servidor e vice versa e também implantar mecanismos de tolerância à falhas permitindo que uma máquina assumisse rapidamente as funções de uma outra faltosa.

Com o desenvolvimento dos sistemas distribuídos no ambiente de redes, a preocupação maior é com a chegada em segurança dos dados do cliente para o servidor e no retorno do servidor para o cliente. Os usuários de computadores e dispositivos móveis esperam que os dados e informações trafegadas do cliente para o servidor, em um ambiente sem controle como a internet, estejam e cheguem em segurança e de forma controlada ao seu destino, bem como o retorno delas.

Os sistemas distribuídos surgiram devido à necessidade das empresas em obterem sistemas que trafegassem em rede de alta velocidade, com segurança, alta disponibilidade e baixa latência, para tornarem-se competitivas. Em grandes corporações, com alto tráfego de informações sigilosas, manterem-se a segurança das informações dentro de sistemas distribuídos é uma necessidade primordial.

A computação com sistemas distribuídos consiste em diversas máquinas interligadas em rede ou em mais de uma rede trabalhando em conjunto no mesmo computador para processar uma mesma tarefa, de forma transparente e ágil. A junção destes computadores tem como objetivo compartilhar a execução de diversas tarefas. As dúvidas e os desafios para a implantação de um sistema distribuído é que o mesmo seja criado com capacidade de expansão e que estes possam evoluir ao longo do tempo, mantendo o mesmo desempenho.

Segurança de informações é a garantia que os dados serão trocados ou mesmo que um usuário receberá um *e-mail*, sem se preocupar com a intromissão de agentes externos, códigos maliciosos, que não fazem parte desta mensagem ou transação. As transações ou mensagens entram de forma silenciosa nas redes por meio dos agentes móveis que servem de transporte para os agentes maliciosos, sem intenção de transportá-los.

Não existem garantias que o agente móvel não está sendo utilizado como um meio para a chegada de agentes maliciosos. Nenhuma técnica garante a segurança dos sistemas e redes, visto que este ambiente pode ser completamente livre, em um meio como a internet, passando por diversos ambientes, sem controle.

O trabalho com códigos e agentes móveis utiliza um ambiente com sistemas distribuídos, conhecidos como sistemas de códigos móveis que possuem características, como tráfego de dados entre diferentes países, transparência nas transações e tolerância a falhas.

A capacidade de permissão de acesso a conteúdos executáveis de páginas da Web e o avanço da tecnologia para a utilização de dispositivos móveis, permite o acesso a sites na internet onde se pode obter qualquer informação, sem saber se é segura ou se a informação está livre de códigos maliciosos (Jaeger e Rubin, 1996).

O envio de um agente móvel por meio da internet deve permitir ao proprietário deste agente que ele seja protegido contra hospedeiros mal intencionados que podem roubar, modificar ou danificar as informações transportadas por esses agentes, sem que os mesmos percebam (Tanembaum, 2008).

Com a recente ênfase dada a novas aplicações baseadas em multimídia, levar em conta requisitos de segurança e confiabilidade no ambiente de internet torna-se necessário para a correta especificação de sistemas de produção e de execução de agentes móveis.

Em ambiente de rede sem fio esta necessidade de segurança nos sistemas aumenta, devido a restrições inerentes ao próprio meio, implicando diretamente na segurança da informação trafegada.

O meio de transmissão da rede sem fio é considerado não confiável pelas perdas e interceptação que os dados estão sujeitos durante o tráfego na rede, comprometendo mais a segurança dos dados que trafegam do cliente para o servidor.

Os programas maliciosos podem ser copiados ou instalados, independente do meio de comunicação ser com ou sem fio e nem sempre são perceptíveis pelo usuário, comprometendo assim o acesso á internet, à lojas virtuais e aos bancos eletrônicos, com segurança e confiabilidade.

Ameaças, vulnerabilidades, riscos e ataques que comprometem os agentes móveis, são evidentes e as limitações técnicas de proteção comprovam a necessidade de análise dos modelos existentes, composto de prevenção e detecção que possam garantir a segurança de sistemas de agentes móveis.

Os modelos analisados devem compor e garantir a escalabilidade, mobilidade, autenticidade, autorização e facilidade de operação dos agentes móveis em ambientes computacionais com ou sem sistemas distribuídos.

A motivação deste trabalho é validação dos aspectos de segurança da plataforma de desenvolvimento de agentes móveis Ajanta, analisando as características de segurança para o desenvolvimento de agentes móveis, visando permitir somente o acesso autorizado aos recursos do hospedeiro, buscando maior segurança na utilização de aplicações na internet, salas de bate-papos, comunicações *on line* e sistemas distribuídos.

Foi considerada neste trabalho, a avaliação do ambiente heterogêneo como a internet, suporte a um conjunto de mecanismos compatíveis para a construção de aplicações com agentes móveis protegidos, avaliação do comportamento correto de agentes em um ambiente de rede, avaliação de soluções a fim de minimizar problemas de desempenho e latência de rede, esquemas bem definidos de segurança a fim de garantir a segurança e integridade dos dados para códigos móveis na plataforma de desenvolvimento de agentes móveis Ajanta.

1.2 Objetivo

O objetivo deste trabalho é analisar e comparar aspectos de segurança das arquiteturas de desenvolvimento e de execução de agentes móveis na plataforma de desenvolvimento de agentes móveis Ajanta.

A facilidade de desenvolvimento e as potencialidades amplas de execução implicam em sistemas de baixa segurança, para ganhos em desempenho, onde os agentes sofrem a possibilidade de ser modificados por um ou mais atacantes para

realizarem tarefas indesejáveis ou enviarem informações para pessoas ou sistemas não autorizados.

O uso de mecanismos de segurança, tais como senhas criptográficas secretas, resulta numa maior dificuldade de difusão do agente, pois somente os hospedeiros que compartilham tais senhas poderão executar o código móvel.

Esse trabalho deve avaliar como o compromisso entre segurança e funcionalidades são tratadas nos sistemas de desenvolvimento de agentes móveis Ajanta.

1.3 Contribuições Esperadas

A contribuição esperada neste trabalho é identificar em ambiente controlado as características do modelo de desenvolvimento de agentes móveis Ajanta, avaliados com foco em aspectos de segurança. O resultado deste trabalho permitirá apresentar o resultado da análise dos agentes móveis e avaliar cada método de desenvolvimento proposto em função do tipo de aplicação e do ambiente de execução alvo dos agentes móveis.

Contribuições esperadas:

- Para os desenvolvedores de software, o presente trabalho contribuirá nos aspectos de segurança dos agentes móveis que estarão presentes nos sistemas distribuídos;

- Para os profissionais de rede, o trabalho tem como objetivo ajudar na validação das redes de alto desempenho, baixa latência e alta disponibilidade, que necessitam de agentes móveis nas aplicações que trafegam na rede;

- Para as empresas de desenvolvimento de aplicações e arquitetura de sistemas, este trabalho poderá ser utilizado como premissa para avaliar a segurança e o tráfego de informações das aplicações em ambientes *Web*;

- Para os estudiosos da área, este trabalho se apresenta como premissa para novas considerações e trabalhos futuros.

1.4 Metodologia

O plano para a elaboração desse trabalho contempla atividades de pesquisa e identificação de sistemas de desenvolvimento ou modelos de segurança de agentes móveis com características similares comparáveis e avaliadas em aspectos de segurança, escalabilidade, portabilidade, mobilidade e facilidade de operação dos agentes móveis em sistemas distribuídos.

Utilizando prova de funcionamento dos agentes móveis (Ajanta) em ambiente controlado (laboratório) para avaliar e validar a pesquisa.

Foram feitas análises, avaliações e apresentação dos resultados obtidos por meio de análise do ambiente de desenvolvimento de agentes móveis Ajanta e redação do texto com as especificações, características e o resultado do teste em ambiente controlado.

1.5 Organização do Trabalho

Na seção 1 é descrito o cenário atual com aspectos de segurança dos agentes móveis existentes e considerados, objetivo, contribuições esperadas com a conclusão e motivação quanto ao tema escolhido.

A seção 2 apresenta os principais conceitos e estado da arte em relação à segurança na mobilidade de código e agentes móveis. Da mesma forma também são identificadas no cenário atual as deficiências dos agentes móveis em sistemas distribuídos.

Na seção 3 são tratados os critérios de avaliação e sistemas comparados para os testes de validação das plataformas de desenvolvimento dos agentes móveis.

A análise e avaliação prática desenvolvida em ambiente controlado do ambiente de desenvolvimento de agentes móveis Ajanta, são tratadas na seção 4 e a conclusão na seção 5.

2 A SEGURANÇA NA MOBILIDADE DE CÓDIGO E OS AGENTES MÓVEIS

O objetivo desta seção é apresentar o conceito de código móvel, mobilidade e os agentes móveis, com suas principais características, suas facilidades e suas dificuldades na garantia da segurança.

2.1 Códigos Móveis

Os códigos móveis são programas desenvolvidos na linguagem de programação JAVA, conhecidos como *applets*, criados pela Sun Microsystems em 1995[3]. Os *applets* são hoje os principais responsáveis pela disseminação do conceito e das tecnologias de códigos móveis em geral. Estes são programas ágeis, pequenos, fáceis de trabalhar e criados em um ambiente considerado seguro por trafegar com os dados criptografados conhecidos como plataforma JAVA, é popular no ambiente de desenvolvimento de softwares e aplicações em âmbito mundial.

O uso de código móvel no desenvolvimento de sistemas distribuídos e em ambiente Web permite a comunicação entre os dados dos sistemas, adicionalmente ao uso de páginas HTML e figuras; possibilitando que os recursos transferidos entre servidores e clientes que utilizam a internet sejam os programas de computadores.

O aspecto de segurança garante que o código móvel vindo de qualquer servidor da Web (chamado e conhecido como código estrangeiro) não tenha acesso restrito de recursos presentes à plataforma do cliente, permitindo falhas ou entrada de vírus.

2.2 Mobilidade de Código

O objetivo dos ambientes computacionais é permitir o suporte necessário às aplicações com a capacidade de realocar dinamicamente seus componentes sobre diferentes sites ou endereços de rede. Estes ambientes podem ter ou não sistemas de códigos móveis.

Os ambientes computacionais que possuem sistemas de códigos móveis permitem basicamente duas formas de mobilidade, que se caracterizam pela

necessidade de componentes que processam os dados de uma máquina para outra, conforme apresentado na figura a seguir:

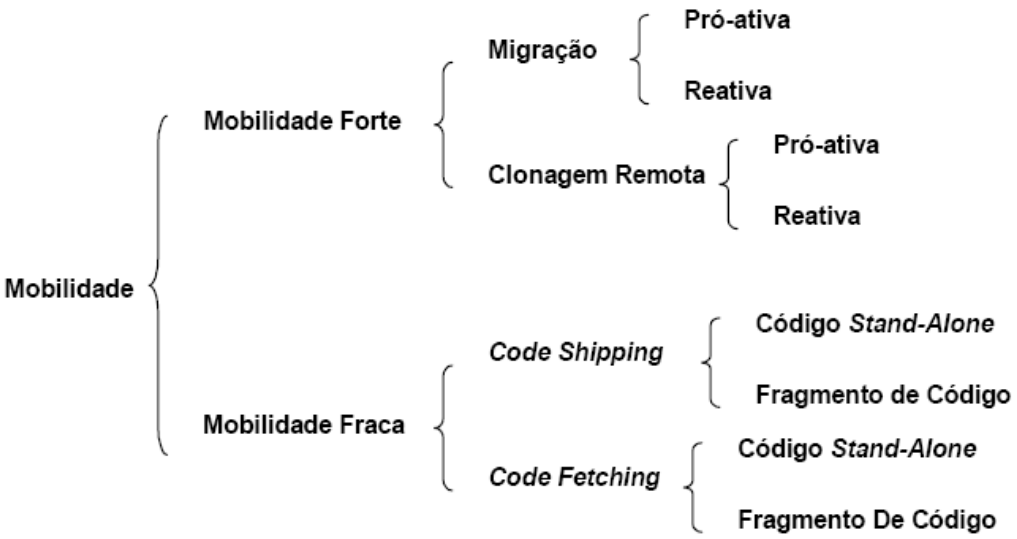

Figura 1 – Mecanismos de mobilidade e classes
Fonte: Costa (1999)

Os ambientes computacionais permitem a mobilidade, tanto do código quanto do estado de execução de uma máquina, para o mesmo ambiente computacional ou para outro, caracterizando-se desta forma por possuir a propriedade de mobilidade forte.

A implantação da mobilidade forte é feita pelo meio de migração. O mecanismo de migração tem como objetivo suspender a máquina e transmiti-la para o ambiente computacional de destino onde é retomada a sua execução, exatamente do ponto onde foi interrompida. As migrações podem ser pró-ativa ou reativa. A migração pró-ativa funciona de maneira autônoma sobre o destino da migração. Na reativa, quando a transferência é provocada por outro computador, desde que essa transferência tenha permissão para a operação. Esta migração reativa opera em sistemas que delegam o gerenciamento de todos os componentes móveis a uma entidade, caracterizando a mobilidade fraca, que estão presentes nos sistemas de desenvolvimento de agentes móveis.

A mobilidade fraca tem como propriedade a capacidade de transferência do código de apenas uma máquina. Na mobilidade fraca pode-se ainda transferir com o

[3] www.sun.com

código de dados e variáveis de inicialização dos códigos, mas em nenhum momento o estado de execução de uma máquina é movido para outra.

Os mecanismos que implantam a mobilidade fraca, ao transferirem o código para o ambiente computacional destino (*code fragment*), associam este código dinamicamente a uma unidade de execução ativa no mesmo ambiente ou aproveitam o código para gerar uma nova unidade de execução (*stand-alone code*). A transferência do código pode ser iniciada pela unidade de execução proprietária do código (*code shipping*), ou ainda, requisitada por outra unidade de execução que precise incorporá-la ao seu próprio código (*code fetching*).

As plataformas de desenvolvimento de agentes móveis escolhidas para análise trabalham com mobilidade fraca.

2.3 Agentes Móveis

Os agentes móveis são programas desenvolvidos em linguagem JAVA, presentes nas redes de comunicação, alternativas para ambientes distribuídos, principalmente em aplicações para comércio eletrônico e internet e possuem o código móvel. Os agentes móveis podem estar presentes nas redes de baixa latência e alta disponibilidade, onde a aplicação teria que fazer buscas no banco de dados para atualização dos dados na tela. Podem ser definidos como observadores, buscadores e organizadores de dados que são identificados em sistemas que precisam ser constantemente verificados e necessitam de baixa latência de rede.

O agente observador tem como objetivo a monitoração de ferramentas para operação no mercado eletrônico, até que um produto atinja um determinado preço competitivo, notificando o usuário para a rápida tomada de decisão. As empresas que utilizam agentes móveis nas redes como agentes observadores são empresas do mercado financeiro, que precisam de baixa latência e informações rápidas, com pouco acesso ao banco de dados.

Quando o agente tem a característica de um buscador, o usuário pode delegar a um agente a busca de um produto dentro de um site de busca de um determinado valor. O agente móvel procura, dentro de vários servidores, alternativas para localizar e trazer o melhor preço. Esse processo é utilizado pelos usuários da

internet em sites de compra, onde o objetivo é identificar dentre algumas lojas, as que têm o produto com o menor preço.

O agente móvel que tem a função de um organizador de uma agenda, interage e retorna o melhor resultado para abranger os requisitos determinados pelo cliente também em sistemas distribuídos no ambiente da internet.

Os agentes móveis atuam em sistemas distribuídos, na comunicação com a troca de dados entre processos da migração entre máquinas em uma mesma rede (Tanembaum, 2008).

Os agentes móveis podem oferecer um paradigma uniforme para objetos distribuídos; englobando passagens de mensagens síncronas e assíncronas, passagem de objetos, móveis e estacionários e não estão restritos aos sistemas que iniciaram a execução. Os mesmos possuem a habilidade de se transportar de um sistema para outro por meio de uma rede cabeada ou mesmo sem fio.

Existem algumas vantagens na utilização dos agentes móveis, como abordado por Costa (Costa,1999):

Redução no tráfego da rede: os sistemas distribuídos demandam tráfego de dados nas redes para a realização de tarefas, principalmente quando há restrições de segurança envolvidas. Agentes móveis podem ainda reduzir o tráfego de dados na rede, pois permitem mover o processamento para o local onde os dados estão armazenados ao invés de transferir os dados para depois processá-los. O princípio do agente móvel é mover o processamento para os dados ao invés de mover os dados para o local de processamento.

Redução de latência de rede: os agentes móveis oferecem uma solução que contribui para que esta baixa latência consiga ser alcançada, pois podem ser despachados pelo controlador central para realizarem suas tarefas localmente, sem demandar tráfego na rede.

Encapsulamento de protocolo: cada máquina em um sistema distribuído possui seu próprio programa que é necessário para realizar a transferência de dados. Porém, novos requisitos de segurança e eficiência demandam mudanças no protocolo que podem ocasionar problemas na manutenção do código existente. Os agentes móveis podem mover-se para máquinas remotas a fim de estabelecer canais de comunicação baseados em protocolos proprietários.

Execução assíncrona e autônoma: tarefas podem ser embutidas em agentes móveis que podem ser despachados pela rede. Após serem despachados, os agentes são autônomos e independentes da criação do processo, podendo executar assincronamente. Este recurso é útil principalmente porque dispositivos móveis, como notebooks, podem se reconectar na rede mais tarde, para coletar o agente de forma síncrona.

Adaptação dinâmica: agentes móveis possuem a habilidade de perceber mudanças no ambiente de execução e reagir autonomamente. Múltiplos agentes podem interagir entre si e se distribuir pela rede, de modo a buscar uma configuração ótima para resolver um problema em particular.

Independência de plataforma: redes de computadores, geralmente são heterogêneas, tanto na perspectiva de hardware como a de software. Agentes móveis são independentes da máquina e também da rede, sendo dependentes somente do seu ambiente de execução, não dificultando a integração de sistemas.

Robustez e tolerância a falhas: a habilidade dos agentes móveis de reagirem dinamicamente a situações e eventos desfavoráveis torna fácil a construção de sistemas distribuídos robustos e tolerantes a falhas. Se uma máquina está para ser desligada, todos os agentes em execução na máquina podem ser advertidos para que possam ser despachados e continuarem suas tarefas em outra máquina da rede, sem afetar a conectividade ou mesmo o tempo de resposta das aplicações.

Na tentativa de viabilizar a implantação dos agentes móveis em aplicações distribuídas, foi desenvolvido um esquema de segurança, a fim de atender a esta plataforma. Baseia-se em um controle descentralizado de autenticação e autorização que se utiliza dos mecanismos de delegação dos certificados SPKI/SDSI *(Simple Public Key Infrastructure/Simple Distributed Security Infrastructure).* Conforme figura 2:

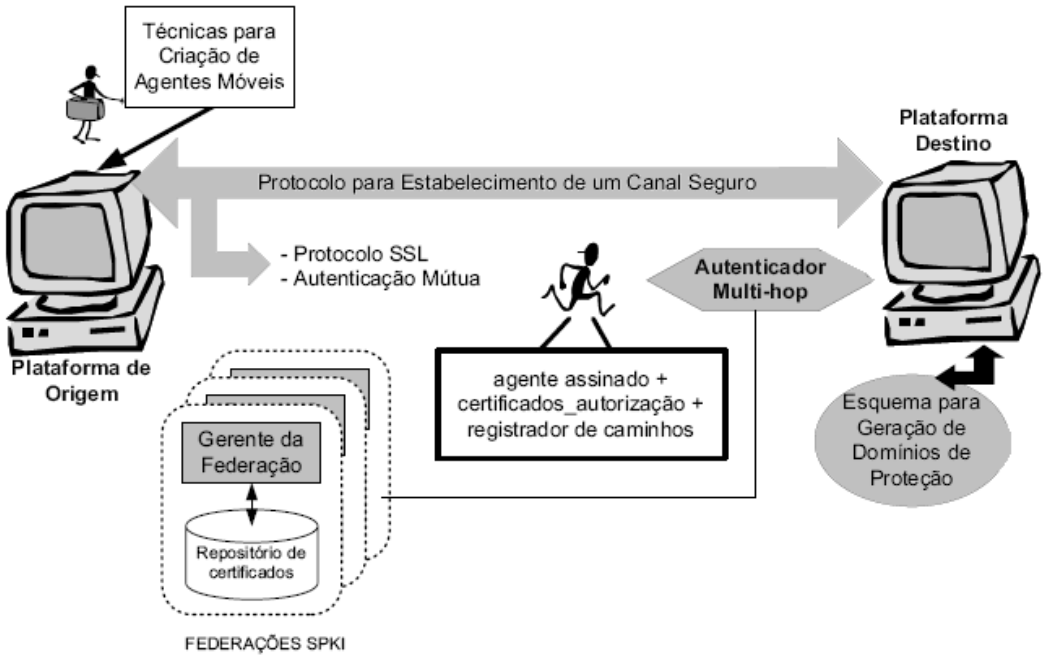

Figura 2 – Mecanismos de delegação dos certificados SPKI/SDSI
Fonte: Wangham (2004)

Após o processo de criação e programação de um agente móvel protegido, antes de enviá-lo às emissoras de origem dos agentes móveis devem passar por um protocolo de autenticação entre ambas, origem e destino, para que um canal seguro seja estabelecido entre os agentes no envio. Na seqüência, o agente é enviado com as suas credenciais para ser autenticado pela plataforma destino e então o seu domínio de execução será criado.

Desta forma, quando um agente chega ao destino, ele apresenta a sua chave pública e um ou mais certificados SPKI/SDSI para que o verificador da plataforma realize a checagem dos certificados. A partir destas informações, este verificador deve gerar as permissões necessárias ao agente para sua execução na plataforma destino.

2.4 A Mobilidade de Código em Sistemas Distribuídos com Agentes Móveis

Um sistema distribuído é um conjunto de computadores independentes que se apresentam a seus usuários como um sistema único e coerente (Tanembaum, 2008).

Existem diferenças entre os sistemas distribuídos tradicionais e os que trabalham com código móvel. Os sistemas distribuídos tradicionais são compostos

por uma camada na arquitetura computacional que oferece uma transparência ao usuário no processamento das aplicações. A arquitetura de um sistema que utiliza o código móvel especifica o local onde serão executados os componentes (partes) do sistema. Este ambiente é conhecido como ambiente computacional.

A figura 3 ilustra o ambiente com e sem agentes móveis. A partir da visualização de baixo para cima, tem-se a parte inferior como hardware da máquina, acima o sistema operacional responsável por prover as suas funcionalidades básicas, como sistema de arquivos, gerenciamento de memória e gerenciamento de processos. Neste ponto, não existe suporte para comunicação ou ligação entre as máquinas. Estas funcionalidades são providas pela camada do ambiente operacional de rede, onde serviços de endereçamento possibilitam a comunicação entre duas ou mais máquinas conectadas fisicamente.

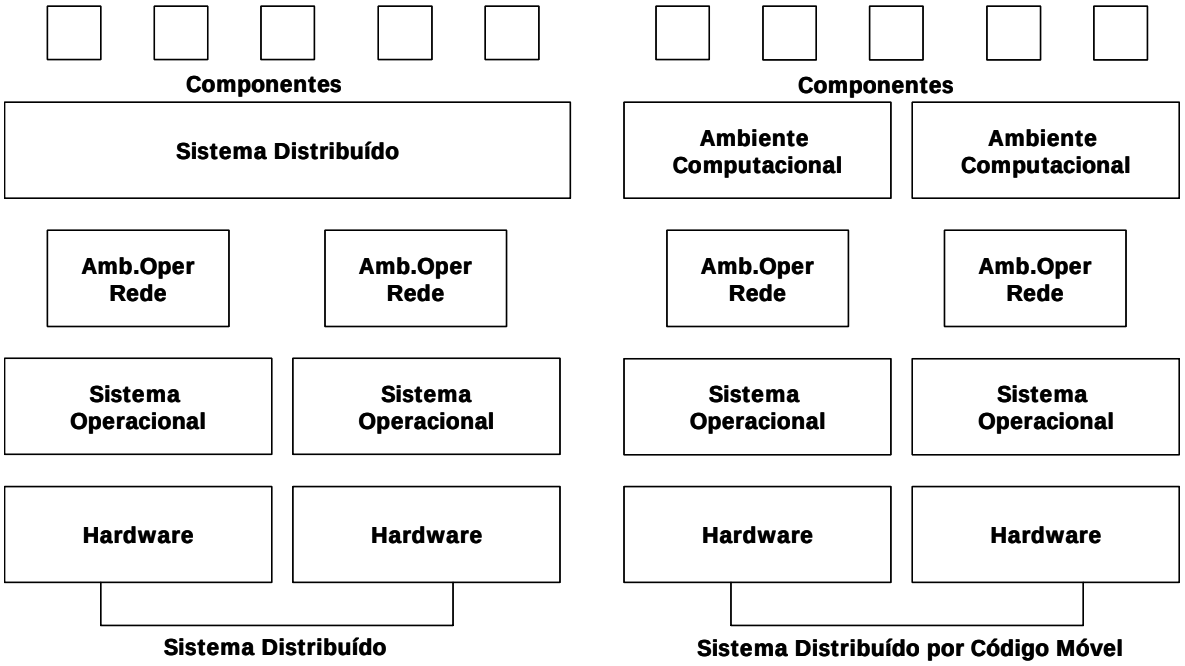

Figura 3 - Arquitetura de Sistemas distribuídos com e sem agentes móveis.
Fonte: adaptado de Wangham (2004)

Os serviços são muito semelhantes. Em um sistema tradicional sem agentes móveis não é necessário o conhecimento da topologia da rede onde o sistema é executado. E, ao convocar um serviço qualquer não se pode determinar qual nó da rede irá oferecer e executar o serviço. Apesar da arquitetura ser muito similar,

aquela que trabalha com código móvel provê o suporte aos serviços distribuídos do sistema. A diferença entre uma arquitetura e outra está em diversos ambientes computacionais que os agentes móveis dispõe. Neste caso a rede deixa transparente a identidade da máquina onde estão sendo executados os componentes e sistema.

Em um ambiente em que o sistema distribuído trabalha com os códigos móveis, a estrutura do ambiente computacional é distinta para melhorar o desempenho e tentar garantir flexibilidade (Tanembaum, 2008).

O objetivo do ambiente computacional é determinar a identidade da máquina, onde o agente móvel deve ser executado e oferecer às aplicações, serviços de alocação e transferência dos componentes para outros nós da rede. A arquitetura dos sistemas tradicionais e da pilha de protocolos é consumidora do ambiente computacional para realizar a comunicação entre as máquinas, para preparar o código e os dados para uma transferência qualquer.

O código móvel normalmente é utilizado em ambientes que utilizam a internet como meio de comunicação, em modelos de *applets* JAVA, o que permite aos recursos a transferência dos dados entre cliente e servidores. Os códigos são instalados em servidores onde são feitas as transferências dos dados sob demanda do servidor para o cliente. Estes dados são executados de forma automática, segura sobre plataforma de computadores clientes, a fim de que não trafeguem na rede aumentando a latência de forma desnecessária. Desta forma, os códigos trabalham no cliente e somente ao final, com todos os dados coletados, são transferidos de uma única vez para o servidor (Tanembaum, 2008).

Em todo o ambiente de rede, programas maliciosos não autorizados podem espionar seu tráfego de dados. Os ataques mais freqüentes são modificações de códigos e leitura de dados confidenciais da aplicação, com interrupção do serviço. A forma de proteção é por prevenção ou detecção, conforme descrito (Costa,1999):

- **Mecanismos de prevenção** – tentam impedir que acessos e/ou modificações indevidas sejam feitas nas unidades de execução. O mecanismo mais fácil para este objetivo é o uso de um hardware seguro que traz algumas funções criptográficas. Uma outra abordagem menos eficiente é a engenharia de software reversa, onde o código é escrito contrariando-se as regras: nomes de variáveis sem sentido e códigos redundantes, tornando assim a lógica da

unidade de execução quase que incompreensível para a pessoa que efetuou o ataque. O funcionamento desse mecanismo é limitado, pois com o tempo o código poderá ser decodificado. Mecanismos de criptografia parcial são mais limitados ainda. Eles cifram apenas alguns dados de forma que só possam ser utilizados por um ambiente computacional escolhido. Estes dados são cifrados com a chave pública do ambiente computacional destino, tornando-os ilegíveis para qualquer outro ambiente computacional por onde passar.

Mecanismos de detecção – tentam descobrir quando e se um ataque foi realizado após a execução da unidade. O mecanismo de análise do estado de execução estabelece algumas constantes para sua unidade e após ser executada, verifica seu estado para descobrir se o valor de alguma dessas constantes foi modificado. Estas constantes podem ser incluídas em algumas funções *hash*[4], tornando o mecanismo ainda mais eficiente. Um outro mecanismo de detecção de ataques é o rastreamento ou *tracing*, que através da criptografia e assinaturas digitais permitem que quaisquer modificações ilegais feitas na unidade de execução, sejam detectadas.

Essas aplicações que trafegam na rede podem conseguir mais vantagens, alegando pertencer a um grupo de usuários privilegiados no ambiente computacional, com o intuito de acessarem recursos e informações críticas dos sistemas. Um programa malicioso, que também pode ser chamado de aplicação, pode tentar altera sua identidade, fazendo-se passar por outro, com o objetivo de se alojar em aplicações alheias, conseguindo desta forma acessar os dados pessoais dos usuários.

Uma vez que esta aplicação ou programa malicioso foi aceito por um ambiente computacional, pode atacar o próprio *host* ou não. Sempre a aplicação ou programa malicioso tenta acessar informações confidenciais do seu próprio hospedeiro, como acessos a senhas ou chaves secretas.

Em um segundo momento o programa malicioso tira proveito do ambiente do hospedeiro para atacar outros *hosts* da rede, com ou sem *firewall*. Nos ataques é comum a interrupção dos serviços. Um programa malicioso pode degradar o sistema em um ou mais recursos de um ambiente computacional ultrapassando o limite de

memória, espaço em disco e processamento, fazendo com que o computador suspenda a execução de todos os processos.

As falhas de segurança dos sistemas distribuídos, como falta de controle de acesso e possibilidade de alteração de dados são conhecidas pelos desenvolvedores de sistemas. Essas falhas podem comprometer ações, compras pela internet, busca pelo melhor preço de mercadorias, etc. Elas também permitem a manipulação de códigos distribuídos e abre as portas para perigosos ataques.

Por outro lado, o ambiente computacional pode desobedecer ao fluxo do programa modificando sua lógica e alterando o código para executá-lo na próxima máquina a ser visitada e tentar enganá-lo e minimizar o problema. Estes ofensores obviamente podem comprometer o objetivo final de uma aplicação na internet. Como pode ser visto na figura 4:

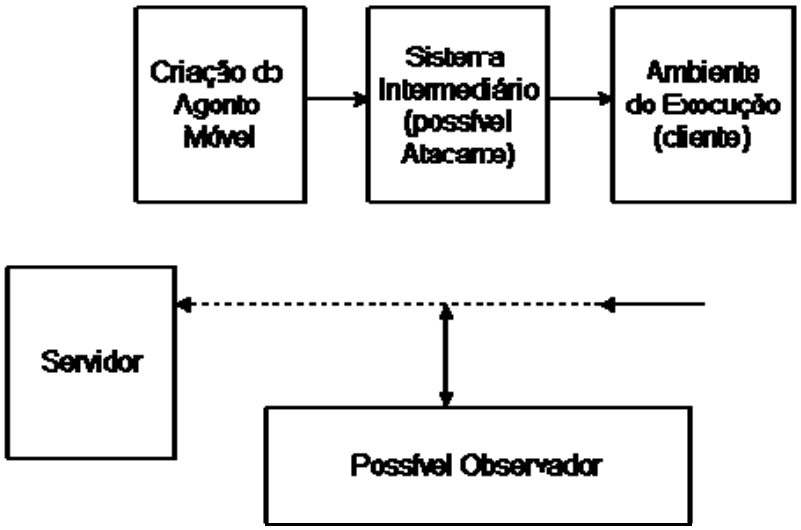

Figura 4 – Agente móvel em um ambiente computacional e com a presença de atacante ativos e passivos
Fonte: Elaborado pela autora

A classificação dos aspectos de segurança dos agentes móveis deve ser observada sob dois aspectos:

[4] Método digital para assinar digitalmente documentos. Método em constante desenvolvimento.

- Intra-ambiente computacional: abrange as falhas e mecanismos de segurança dos elementos localizados em um mesmo ambiente computacional;
- Inter-ambiente computacional: as vulnerabilidades das interações são identificadas entre elementos remotos ou sistemas em ambientes distintos.

Dentre os aspectos endereçados, estão privacidade, integridade e autenticação da comunicação entre dois ambientes. É importante a implantação de mecanismos que atendam aos requisitos de integridade e privacidade das informações que são transmitidas de um lado para outro.

Estes mecanismos de integridade são algoritmos de técnica de soma verificadora (checksum), e são fundamentais para que as informações não sejam modificadas por falsos sinais ou mesmo por erros no processo. O cheksum ou soma verificadora é um código utilizado para verificar a integridade de dados transmitidos através de um canal com ruídos ou armazenados em algum meio por algum tempo.

A técnica de soma verificadora é utilizada em comunicação de dados e armazenamento sequencial. Esta técnica possui três desvantagens a serem consideradas; a primeira desvantagem é que ela funciona bem apenas com blocos grandes. A segunda desvantagem é o fato da detecção não ser *on-line*, pois todas as palavras do bloco têm que ser lidas para calcular a técnica e o tempo de detecção será o mesmo para um erro na primeira palavra ou na última. A terceira desvantagem é a dificuldade de diagnóstico do erro, que em memórias podem ocorrer no bloco de palavras, na técnica armazenada ou no circuito de verificação. Na transmissão de dados, o erro pode estar na fonte dos dados, no meio de comunicação ou no circuito de verificação[5].

2.5 Políticas, Modelos e Mecanismos de Segurança

Os conceitos de políticas, modelos e mecanismos de segurança são muito confundidos na literatura e existe uma norma para a padronização desses conceitos. Estes conceitos estão padronizados pela norma ISO 17799:2005, baseados no

padrão inglês BS 7799. Atualmente esta norma está sendo utilizada para fins históricos, sendo substituída pela ISO27000.

De acordo com Wangham (2004), *política de segurança é o conjunto de regras e práticas que estabelecem os limites de operação dos usuários do sistema e que determinam o modo pelo quais as informações e os recursos são administrados, protegidos e distribuídos no interior de um sistema específico.*

A política de segurança se classifica em três formas distintas (Nicomette, 1996):

- **física**: trata todos os aspectos que se referem a situação física do sistema ou rede que se quer proteger. Nesta categoria, são definidas as medidas contra a violação de segurança física provocadas por incêndio, catástrofes naturais etc.
- **administrativa**: trata a segurança sob o aspecto organizacional, como seleção de pessoal responsável pela segurança dos sistemas informatizados e quem será o responsável pelos sistemas e senhas.
- **lógica**: define regras de controle de acesso e de circulação das informações no sistema. Políticas de segurança lógica podem ser decompostas em políticas de autenticação e de autorização. Quando um usuário é identificado e autenticado, a política de autorização deve especificar quais são as operações que este usuário particular pode realizar no sistema.

O modelo de segurança normalmente é restrito ao comportamento dos sistemas e o das redes dentro de uma empresa. São úteis como guias para a definição de políticas específicas e de todas as formas de descrever tanto o comportamento de entidades quanto as regras que definem a evolução de uma política de autorização e autenticação.

Os mecanismos de segurança em um sistema constituem a implementação das políticas de autorização e de autenticação definidos para o referido sistema. Esses mecanismos asseguram que todos os acessos a objetos no sistema sejam autorizados pela política definida.

---

[5] www.microsoft.com

Desta forma, conclui-se que toda política de segurança é um conjunto de regras, padrões e objetivos bem fixados, que definem um comportamento ideal e determinam o perfil de um comportamento não aceitável pelos administradores e usuários de um sistema ou rede.

2.6 Mecanismos de Segurança

Segundo Denning, 1982, os mecanismos de segurança presentes em sistemas distribuídos com agentes móveis, que implantam políticas de autorização e de autenticação fazem uso de controles de acesso e criptográficos.

2.6.1 Autenticação e Autorização

Autenticação e autorização permitem que uma entidade, chamada de principal (um usuário, um ambiente de rede ou outro sistema), comprove sua identidade perante um sistema e seja validado em um ambiente de rede. Para atestar sua veracidade, a entidade deve fornecer uma prova de identificação apresentando evidências. Na maioria dos sistemas operacionais é fornecido aos usuários um identificador (*login*) para que estes possam ser reconhecidos pelo sistema, por meio de autenticação. Para garantir que a identidade de um usuário esteja correta, uma senha (*password*) é associada ao identificador do usuário.

Outras provas que podem e devem ser pedidas e validadas, são dispositivos que apenas o dono da identidade o possua. Nesta categoria estão incluídos o cartão magnético ou um *smartcard*[6] ou ainda um traço pessoal que seja uma característica única, como impressões digitais ou íris dos olhos.

Nos sistemas distribuídos ou não, é necessária a autenticação entre dois principais comunicadores entre eles, autenticação da origem de dados que permite comprovar a identidade do “chamador” e do “demandador” de uma determinada mensagem. A atribuição destes serviços de autenticação normalmente é feita por controles criptográficos. A autorização, em sistemas de informação, é a capacidade de garantir que apenas usuários autorizados consumam os recursos protegidos de

um sistema operacional. A autenticação é a capacidade de possibilitar acesso liberado por meio de autorização. A autorização e a autenticação só são possíveis se existir o controle de acesso, como garantia da segurança.

2.6.2 Controles de Acesso

O controle de acesso, dentro da segurança da informação é composto por processos de autenticação, autorização e auditoria, com a habilidade de permitir ou negar acesso à informações de sistemas, informações controladas e confidenciais. Os controles de acesso determinam e garantem quais recursos podem e devem ser utilizados. E de acordo com Costa (Costa, 1999), são:

- **Estáticos** – permite a verificação do código de acesso e se respeita a política de controle de acesso estabelecida para os recursos do ambiente computacional hospedeiro. Se essa política for respeitada, então nenhuma outra verificação será feita durante o processamento. Um mecanismo utilizado para a verificação estática do controle de acesso são os verificadores de código que usam técnicas de prova de teoremas para determinar se o código preserva alguns itens de segurança.
- **Dinâmicos** – conjunto de direitos de acesso alinhados à política de segurança. Cada tentativa de acesso aos recursos do ambiente computacional no decorrer da execução são interceptadas e verificadas na lista de controle de acesso local. Desta forma, cada processamento pode ser aceito ou negado. Este mecanismo é geralmente chamado de execução segura ou, *"caixa de areia"*.

A caixa de areia ou *sandboxing*, como é conhecida em inglês é a forma que a plataforma JAVA trabalha com os aspectos de segurança. É característica de segurança das plataformas de desenvolvimento de agentes móveis e será tratado com detalhes no próximo ítem deste trabalho.

---

[6] *Smart card*, *chip card*, cartão de circuito integrado de bolso com informações pessoais, para diversos fins [http://en.wikipedia.org/wiki/Smartcards]

O controle de acesso nos casos de segurança de sistemas distribuídos e rede são usados para implantar políticas de autorização de sistemas, considerando controle de acesso seletivo obrigatório e baseado em papéis, conforme defendido por Wangham e apresentado a seguir (Obelheiro, 2001; Wangham, 2004):

- **Controle de Acesso Seletivo (Discretionary Access Control - DAC):** as informações são manipuladas livremente pelos responsáveis pela informação gerada, normalmente o proprietário. Um controle deste permite a cópia livre dos dados, mesmo que o acesso aos dados originais seja negado, um sujeito sempre pode obter acesso a uma cópia.
- **Controle de Acesso Obrigatório (Mandatory Access Control - MAC):** administração centralizada de segurança. O controle obrigatório supõe que os usuários e objetos (recursos do sistema) tenham sido etiquetados. As etiquetas dos objetos seguem uma classificação específica, enquanto os usuários ou sujeitos de acesso possuem níveis de habilitação. Os controles que verificam a autorização do acesso são baseados na comparação da habilitação do sujeito e da classificação do objeto.
- **Controle de Acesso Baseado em Papéis (Role-Based Access Control - RBAC):** Exige que os direitos de acesso sejam atribuídos a papéis e não a usuários. Os usuários obtêm estes direitos em virtude de terem papéis a si atribuídos.

2.6.3 Controles Criptográficos

Controles criptográficos têm como base os algoritmos criptográficos, que cifram ou decifram o texto. O algoritmo deve ter como parâmetro uma chave, que deve ser secreta e de conhecimento somente dos comunicantes.

Os algoritmos podem ser simétricos ou assimétricos. Os chamados algoritmos simétricos são aqueles que usam a mesma chave para cifrar ou decifrar as mensagens, que também são chamados de algoritmos de chave secreta. Como exemplo, AES (*Advanced Encryption Standard*), que usa chaves de 128,192 ou 256

bits e o RC5 que usa chaves de tamanho definido pelo usuário. Os assimétricos usam chave pública para a comunicação entre os dois lados. E entre eles existe uma chave privada que permite que a mensagem seja decifrada. O RSA (*Rivest Shamir Adelman*)[7] destaca-se como algoritmo assimétrico. Este algoritmo de criptografia de chave pública, utilizado em protocolos no comércio eletrônico, utiliza chaves seguras para implementações de soluções no ambiente da internet. O RSA envolve uma chave pública e uma chave privada. A chave pública pode ser conhecida de todos e é utilizada para criptografar mensagens. As mensagens criptografadas com a chave pública só podem ser decodificadas usando a chave privada.

Os controles criptográficos são usados em sistemas computacionais e provêem a confidencialidade, autenticidade de mensagens e integridade das informações armazenadas ou transmitidas.

A confidencialidade é garantida através da cifragem de dados e de mensagens. Visando permitir a autenticidade de mensagens e a sua integridade, a técnica de assinatura digital baseada em chave pública é a mais utilizada.

Na assinatura de uma mensagem, um emissor deve gerar um resumo criptográfico da mensagem e este resultado deve estar cifrado com uma chave privada. A assinatura sempre é enviada junto com a mensagem, que deve ser verificada, por meio da cifragem da assinatura pública do emissor, comprovando a autenticidade da mensagem e recalculando o resumo da mensagem. Quando comparados, se os resumos (o cifrado e o calculado) forem iguais, a assinatura será autêntica, desta forma a integridade da mensagem está preservada.

A fim de garantir a irretratabilidade, recipientes ou armazenadores dos dados criptografados são criados para que os autores das mensagens não possam negar o envio dessas mensagens que devem ser aceitas. A importância dos controles criptográficos para a implantação das políticas de segurança em sistemas distribuídos ou mesmo em redes de dados preocupa quanto ao armazenamento de dados e distribuição das chaves criptográficas dentro dos sistemas.

Quanto maior o controle de acesso por meio de chaves criptográficas, mais burocrático torna-se o sistema, podendo causar uma latência de rede maior do que a desejada, mas necessária, quando se quer um nível de segurança maior nos acessos que se tem aos sistemas.

---

[7] Algoritmo para criptografia de chave pública

Essa latência pode ser contornada por outros meios que não comprometam a segurança, como equipamentos de rede de alto desempenho e velocidade, onde o investimento em tecnologia de rede, por vezes deve ser maior que o aportado, para manter a segurança do negócio e estabilidade nas aplicações.

Para minimizar os problemas, as distribuições destas chaves baseiam-se em certificados digitais, que trabalham com vínculos entre chaves públicas e privadas, já detalhadas em ítem anterior. Para garantir que a chave pública contida no certificado realmente pertence ao sujeito, é preciso que este certificado seja assinado por uma entidade confiável, chamada de autoridade certificadora (ISO 27000, 2008).

Os processos que permitem a autenticação e a autorização utilizam como base, controles criptográficos onde são gerados os certificados de nomes e autorizações. Estes certificados de nomes são correspondentes à informações que garantem a autenticidade dos objetos do sistema e rede de dados. Os certificados de autorização garantem a autenticidade de permissões de acesso. O serviço de autenticação, na produção de certificados, pode envolver assinaturas digitais e a distribuição de chaves de sessão (Wangham, 2004).

2.7 A linguagem de desenvolvimento JAVA

As características da linguagem JAVA permitem dizer se ela é a melhor solução para o desenvolvimento de aplicações móveis com controle de acesso e segurança, devido a grande facilidade de aprendizado pelos desenvolvedores, portabilidade e suporte a requisitos da rede.

Agentes móveis fornecem um ambiente adequado e mais completo para o desenvolvimento de aplicações distribuídas. A utilização da linguagem JAVA permite que as aplicações sejam desenvolvidas de modo rápido, obtendo código simples e compacto, explorando os conceitos de orientação a objetos. Mesmo o JAVA sendo uma linguagem voltada para aplicações móveis, a preocupação em garantir e permitir o acesso às aplicações de forma segura existe e ainda não contempla um modelo perfeito ou ideal, conforme informações oficiais do site do fabricante[8]. O

[8] http://java.sun.com/ : JAVA, linguagem de programação orientada a objeto, desenvolvida na década de 90 por equipe chefiada por James Gosling, Sun Microsystems. Diferente das linguagens convencionais, JAVA é compilada para um “bytecode”, executada por uma máquina virtual.

código JAVA é simples, fácil, seguro, compacto e por ser orientado a objetos permite reuso e formas coerentes de explorar recursos como interfaces e poliformismo.

As características da linguagem JAVA fazem dela a melhor opção para o desenvolvimento de aplicações em rede, como a seguir:

- **Independência de plataforma** - o código JAVA é compilado em um formato independente de arquitetura chamado *bytecode*[9], permitindo a execução de aplicações em sistemas heterogêneos, sendo, portanto, independente de plataforma que é executada. Isto permite criar agentes móveis sem conhecimento prévio e em qual tipo de computadores eles irão executar ou trabalhar com o código.

- **Execução segura** – JAVA possui mecanismos de segurança principalmente pelo fato da linguagem ser voltada para internet e intranet. Programas feitos nessa linguagem não permitem acesso a dados privados de objetos e muito menos violar a semântica básica da linguagem. Este fato torna possível construir um ambiente mais seguro aos ataques de códigos mal intencionados.

- **Carga dinâmica de classes** - a máquina virtual JAVA carrega e define classes em tempo de execução, fornecendo um espaço de endereçamento privado para cada agente, que pode executar independentemente e com segurança em relação a outros agentes. Este mecanismo é extensível e permite que as classes possam ser carregadas via rede.

- **Programação Multitarefa -** o modelo de programação multitarefa permite a implementação de agentes como entidades autônomas. As primitivas de sincronização nativas da linguagem JAVA habilitam a interação entre agentes.

- **Serialização de objetos** - a linguagem JAVA permite a serialização e deserialização de objetos e que objetos sejam empacotados com informação suficiente para posterior reconstrução. Esta é uma característica chave para implementação de agentes móveis.

[9] Código de programa de computador escrito na linguagem JAVA. Interpretada por máquinas virtuais. Caracterizada por independência de plataforma. Cada opcode tem o tamanho de um byte, bem como número diferentes de códigos de operação está limitada a 256.

- **Reflexão** – a linguagem JAVA possui mecanismos para obter informações sobre classes carregadas, permitindo construir agentes com maior conhecimento de si próprios e de outros agentes.

Embora apresente diversos aspectos positivos para implementação de agentes móveis, esta linguagem possui deficiências, apresentadas a seguir:

- **Suporte inadequado para controle de recursos** - a linguagem não disponibiliza mecanismos para controle de recursos alocados por um objeto. Portanto recursos podem ser alocados inadvertidamente para agentes, e possivelmente permanecerem alocados após serem despachados para uma outra máquina.

- **Referências sem proteção** – a linguagem JAVA não possui mecanismo de proteção às referências possibilitando que métodos públicos de um objeto sejam acessados por outros. Proteção é um fator crítico para implementação de agentes, sendo este problema contornado através do uso de um objeto *proxy,* entre os agentes que interagirão.

Ausência de objeto é proprietário de referências – a linguagem JAVA permite que qualquer objeto referencie um outro. O mecanismo de *garbage collector*[10] não desaloja um objeto até que não haja mais referências à ele, tornando possível que um agente permaneça no sistema caso um outro agente o referencie.

Ausência de suporte para preservação e recuperação do estado de execução - JAVA não possui mecanismo para salvar o estado completo de execução de um objeto. Portanto, para que um agente móvel consiga restaurar seu estado de execução, são necessários a utilização de atributos internos e eventos externos para realizar esta tarefa.

2.8 Plataformas de desenvolvimento de Agentes Móveis disponíveis no mercado

No mercado existem algumas plataformas de desenvolvimento de agentes móveis, com características distintas, bem como diversos fabricantes. Analisando os

[10] Chamado de coletor de lixo, processo usado no gerenciamento de memória. Um erro comum que pode levar ao término não desejado do programa por esgotamento da memória livre.

aspectos de segurança de cada uma das plataformas, todas possuem características semelhantes, com algumas diferenças e algumas deficiências ou facilidades. A escolha pelo Ajanta foi pelo fato desta plataforma ser acadêmica e podermos livremente fazer o estudo e ter o apoio do fabricante. Independente da escolha da plataforma de desenvolvimento escolhida para o estudo, seguem algumas disponíveis no mercado.

2.8.1 Ajanta

O Ajanta surgiu de um projeto iniciado em 1997 e está sendo desenvolvido pela Universidade de Minnesota nos EUA[11]. Baseada na linguagem JAVA, esta plataforma tem seu código binário disponível de forma gratuita não comercial. A maior vantagem do Ajanta é ele ter sido desenvolvido com a capacidade de contornar problemas de segurança quando os agentes móveis fazem parte de um sistema.

Os aspectos de segurança das aplicações que usam agentes móveis mantêm uma infra-estrutura para distribuição de chaves públicas, que são utilizadas em assinaturas digitais e cifragem do canal de comunicação. Este protocolo permite a autenticação mútua entre os agentes móveis para garantir a confidencialidade e integridade dos dados do agente. A plataforma Ajanta possui mecanismos de proteção de agentes contra códigos maliciosos, como encapsulamento de informações e dependência de partes confiáveis entre os agentes.

2.8.2 Aglet

Desenvolvido pela IBM[12], com o nome de API Java Aglets (J-AAPI), pelo laboratório de pesquisa da IBM em Tóquio, Japão e tem como exemplo o modelo de *applet* do JAVA. Com a iniciativa de trabalhar como código aberto a IBM disponibilizou o código fonte do sistema sob licença pública e famosa pela plataforma de agentes baseada na linguagem JAVA.

---

[11] http://www.cs.umn.edu/Ajanta.
[12] www.ibm.com

Não há proteções contra plataformas maliciosas. A autenticação é realizada apenas para identificar se uma plataforma pertence a um domínio. Não é possível identificar e verificar os sites de comunicação. Se a chave compartilhada for roubada de uma plataforma, não há como distinguir as plataformas válidas de uma plataforma que possui a chave roubada. A abordagem baseada em confiança, empregada no Aglets, não pode ser aplicada em redes abertas que contêm diversos domínios e plataformas. Usuários e proprietários de agentes não podem ser autenticados e suas identidades válidas com base na confiança depositada em uma plataforma. Os canais, por onde trafegam as mensagens e agentes, não possuem nenhuma proteção que garanta a confidencialidade da transmissão. Nenhum mecanismo de contabilização dos recursos do sistema é fornecido.

A IBM apresenta um modelo de segurança para a plataforma Aglets, porém este modelo não se encontra totalmente implantado na versão corrente do Kit de Desenvolvimento de Software Aglets (ou ASDK). Algumas características de segurança desta plataforma, como autenticação de usuários e integridade da comunicação entre plataformas garantem a segurança.

2.8.3 Concórdia

A plataforma Concórdia iniciou a sua primeira versão em 1997 pela Mitsubishi Electric ITA. Sistema baseado na plataforma de desenvolvimento JAVA, suporta apenas mobilidade fraca, usando a serialização de objetos do Java e os seus mecanismos de carregamento de classes. A transferência de código entre máquinas de mesmos sites ou outros pode ser feita somente sob demanda ou transferindo todas as classes necessárias de uma só vez. Cada objeto agente está associado com um objeto que especifica o caminho da migração do agente (usando nomes DNS) e os métodos que serão executados em cada site. A versão corrente da plataforma, 1.1.2, possui mais de 2Mb e contém a documentação e a GUI (Interface Gráfica do Usuário) pobres.

O modelo de segurança da plataforma Concórdia fornece as características a serem consideradas, como proteção do canal de comunicação para a transferência e de dados dos agentes e agentes maliciosos.

2.8.4 Grasshopper

A plataforma Grasshopper foi desenvolvida pela GMD Fokus e foi construída voltada para um ambiente de processamento distribuído. Foi o primeiro sistema de agentes desenvolvidos de acordo com o padrão proposto pela OMG, isto é, o *Mobile Agent System Interoperability Facility* - MASIF, atualmente chamado de MAF. A conformidade com o MASIF torna o Grasshopper muito atraente para pesquisadores da tecnologia de agentes móveis. A versão atual, 2.1, possui uma boa documentação e GUI (Interface Gráfica do Usuário), porém, a plataforma é muito grande com mais de 4 Mb.

O serviço de segurança do Grasshopper suporta dois tipos de mecanismos: internos e externos. Os mecanismos externos protegem as interações remotas entre os componentes Grasshopper distribuídos e os internos protegem os recursos das agências contra acessos não autorizados. O controle de acesso do Grasshopper é fortemente orientado pelos mecanismos de segurança do JAVA 2. Também não há proteção dos agentes contra plataformas maliciosas.

2.8.5 Voyager

A plataforma Voyager foi desenvolvida pela ObjectSpace. Voyager e é uma plataforma para computação distribuída em JAVA, com mobilidade fraca. O Voyager combina as propriedades de um ORB, baseado em JAVA, com uma abordagem simples para transferência de agentes, que permite aos programadores trabalharem com aplicações e técnicas tradicionais de programação. O ORB universal do Voyager combina técnicas de RMI, CORBA e DCOM. O Voyager visa uma programação distribuída fácil e transparente. É considerado um produto profissional e possui uma excelente documentação. ObjectSpace foi comprada pela Recursion Software. A versão atual é a 4.1 e possui aproximadamente 700kb

2.8.6 Mole

A plataforma MOLE é um estudo acadêmico de mais de cinco anos, desenvolvido desde 1996 no IPVR (*Institute for Parallel and Distribute Computer*

Systems) da Universidade de Stuttgard, na Alemanha. A versão atual, 3.0, possui código fonte disponível gratuitamente para uso não comercial, ocupa aproximadamente 360Kb e uma boa documentação, consistindo de manual de usuários, JAVAdoc API e diversos artigos.

2.8.7 SOMA

A plataforma de agentes móveis SOMA (Secure Open Mobile Agent) é um projeto que está sendo desenvolvido desde 1998 no DEIS da Universidade de Bologna, na Itália. O código binário e o código fonte do sistema estão disponíveis para uso não comercial. A plataforma está baseada em JAVA (JDK2) e tem como objetivos principais a segurança e a interoperabilidade. Para atingir o objetivo da interoperabilidade, a plataforma SOMA é compatível com o CORBA e com a especificação MAF.

Tabela – 1 Comparativa entre aspectos de segurança de sistemas de Agentes Móveis disponíveis no mercado

	Sandbox	Mobilidade	Autenticação	Autorização	Canal Seguro	Tolerância a Falhas
Ajanta	Sim	Fraca	Mecanismo de desafio-resposta	Protocolo de transferência de agentes (ATP)	Confidencialidade e autenticação via DAS.	Sim.
Aglet		Fraca	Usuários de um mesmo domínio	Baseado no Security Manager do Java.	Autenticação e Integridade (MAC).	Sim.
Concordia		Fraca	--	Baseado no Security Manager do Java.	Autenticação e confidencialidade via SSL	Sim.
Grasshopper		Fraca	Mútua entre as plataformas (utilizando certificados X.509)	Baseado no Security Manager do Java.	Autenticação e confidencialidade via SSL.	Sim.
Voyager		Fraca	--	Baseado no Security Manager do Java.	Autenticação e confidencialidade via SSL.	Sim.
Mole	Sim	Fraca	--	Agentes estacionários.	NÃO POSSUI.	Não implementada.
Soma		Fraca	Assinatura do proprietário do agente.	Hierárquicas.	Autenticação e confidencialidade via SSL.	Sim.

Fonte: Elaborado pela autora

2.9 Aspectos de Segurança JAVA e componentes das ferramentas

Os principais componentes presentes na plataforma JAVA, considerando os aspectos de segurança, são:

- Código intermediário proveniente da característica da linguagem de programação, independente de arquitetura, chamado *bytecode*; a **máquina virtual JAVA** ou JVM (*JAVA Virtual Machine*) que interpreta o *bytecode* e atua como um sistema operacional para executar objetos (interpretador JAVA);
- Conjunto de classes ou **APIs**[13] ***(Application Programming Interface***) que compõem a plataforma JAVA. Executadas com o interpretador, fornecendo classes básicas para a construção de aplicações, que permitem que aplicações JAVA sejam executadas em qualquer sistema que possua uma implementação da máquina virtual JAVA (portabilidade).
- Linguagem elaborada para facilitar a criação de aplicações e *applets* para estes serem usados em rede de computadores. O grande diferencial do JAVA é torná-la independente de plataforma, robusta, segura e fácil de ser trabalhada, usando o modelo cliente-servidor.

A plataforma JAVA é considerada um padrão para a programação de aplicações distribuídas e excelente linguagem para o desenvolvimento de ferramentas de agentes móveis devido às suas características e propriedades, que são: interpretação segura de código, portabilidade, programação multitarefa, serialização de objetos, reflexão estrutural, suporte para programação distribuída, carregamento dinâmico de código e assinatura de código (Wangham, 2004).

Estas características permitem que a linguagem JAVA e o seu modelo de segurança sirvam de base para a concepção de diversos mecanismos e de esquemas de segurança.

---

[13] Application Programming Interface (Interface de Programação de Aplicativos) conjunto de rotinas e padrões estabelecidos por um software para utilização de suas funcionalidades por aplicativos.

Esta seção tem por objetivo apresentar, de forma resumida, o modelo de segurança de sistemas de criação de agentes móveis, para comparação e avaliação de suas limitações e para o desenvolvimento e apresentação das considerações em análise laboratorial, referentes a este trabalho.

3 SISTEMAS COMPARADOS E MEIOS DE AVALIAÇÃO

O objetivo deste tópico é apresentar os sistemas de programação de agentes móveis na plataforma JAVA, que são comparados neste trabalho e o laboratório para validação de uma das plataformas nos aspectos de segurança.

3.1 A plataforma AJANTA

O Ajanta é uma plataforma de desenvolvimento de agentes móveis em JAVA, desenvolvido em uma universidade em Minnesota nos Estados Unidos. O projeto foi iniciado em 1997, e está em constante desenvolvimento, é utilizado livremente para pesquisa e desenvolvimento de agentes móveis no meio acadêmico.

Essa plataforma possui um ambiente de execução de agentes, com a capacidade de permitir a criação de um ambiente seguro para execução de agentes móveis, utilizando um protocolo de transferência e protegendo a integridade e a confidencialidade do mesmo. A plataforma usa a serialização de objetos padrão do JAVA para capturar parte do estado de execução para a migração, característico de mobilidade fraca do agente desenvolvido pelo Ajanta (Wangham, 2004).

A mobilidade fraca, característica da plataforma Ajanta, permite que a transferência do código seja feita sob demanda, a partir da origem do agente, usando itinerários de tráfego como uma forma de definir para onde o agente será enviado. A comunicação entre os agentes dentro de um mesmo ambiente é feita por meio de acessos a métodos remotos e uso de recursos compartilhados. No acesso aos métodos remotos, um agente sempre deve registrar um recurso de um servidor e permitir condições de comunicação entre agentes externos.

A escolha da plataforma Ajanta, para a validação dos aspectos de segurança das plataformas de agentes móveis deste trabalho deve-se à facilidade de uso e a montagem do ambiente, apoio da equipe de desenvolvimento por e-mails e por esta plataforma ser livre: não é exigida licença nem permissão de uso. Além de possuir características de segurança que nos permitem validar os aspectos de segurança de forma ampla.

3.2 Características e capacidades de segurança da plataforma de desenvolvimento Ajanta

O objetivo deste item é descrever tecnicamente a arquitetura da plataforma de desenvolvimento de agentes móveis Ajanta. Um agente móvel desenvolvido por esta plataforma, é um objeto que permite a migração das operações para um sistema distribuído executar as tarefas. Com relação à segurança, proteção de recursos, bem como a proteção do agente, ao identificarem obstáculos significativos na aplicação, têm a função de facilitar o entendimento das funcionalidades de um agente móvel, bem como sua plataforma.

Esta arquitetura permite que se crie um mecanismo de segurança para proteger os recursos do servidor de agentes maliciosos, e que o mesmo seja protegido quando esteja sendo manipulado por esses agentes que trafegam por meio dos servidores e canais da comunicação durante o tráfego de um servidor para outro. É necessário um mecanismo de *proxy* para garantir o acesso ao servidor pelos agentes.

O Ajanta suporta a comunicação entre agentes usando RMI, que será descrito nos próximos tópicos e o mesmo pode ser controlado se solicitado pelos servidores.

Um agente é criado por algum programa aplicativo, que pode ser tanto um agente servidor, ou mesmo um programa cliente, ou até mesmo outro agente. Normalmente, uma aplicação cria um objeto estacionário, que lida com as condições encontradas pelos agentes, que trabalham com eles.

O Ajanta usa nomes baseados no URN (*Uniform Resource Name*) do modelo. O nome do serviço é usado para o mapeamento e a localização física das várias entidades, tais como os usuários, agentes, servidores agentes, recursos e clientes. A segurança desta plataforma é sempre muito polêmica, ainda mais quando estes agentes trafegam em redes abertas e sem controle como a internet, anteriormente já citadas. Isto acontece em todas as plataformas de desenvolvimento de agentes móveis e o Ajanta não é diferente delas.

O comprometimento dos dados trafegados por agentes móveis pode ser por vários motivos. Dentre eles servidores expostos a acesso de agentes maliciosos acarretando o vazamento de informações, acesso e modificação de dados contidos dentro de um agente (como número de cartão de crédito do usuário) que podem ser

abertos ou lidos devido à escuta em redes inseguras ou agentes maliciosos. Como exemplo, pode ser citado a modificação da chave pública de usuários críticos, como diretores que podem comprometer os dados armazenados com as respectivas senhas ou permitir o acesso de agentes maliciosos que podem criar nomes e características de outros usuários.

É esperado que o sistema de agentes móveis, como o Ajanta forneça mecanismos de segurança para a detecção de ataques. Estas características incluem a confidencialidade (para proteção de dados e utilização de códigos secretos), mecanismos de autenticação (para estabelecer a identidade de comunicação de partes) e mecanismos de autorização (para fornecer agentes com acesso controlado ao servidor de recursos).

A comunicação segura deve permitir que o agente transfira com segurança um agente pela rede. Como um agente móvel percorre a rede, o seu código e seus dados são vulneráveis a diversos tipos de ameaças de segurança. Estas incluem ataques, tais como escuta passiva e análise de tráfego e ativos e ataques com mensagens de modificação. Ataques passivos são difíceis de detectar, pois nem sempre eles são compreendidos como um ataque, mas geralmente podem ser protegidos , utilizando mecanismos criptográficos. Os ataques ativos são relativamente fáceis de detectar criptograficamente, mas difíceis de evitar.

Concluindo, a confidencialidade, a integridade e mecanismos de autenticação devem estar presentes nos aspectos de segurança do protocolo de transferência de agentes móveis.

3.3 Instalação do AJANTA e Laboratório

Para executar a plataforma Ajanta, é necessário os seguintes softwares e seus componentes: JDK 1.1.X e Perl.

De acordo com a recomendação do fabricante, a configuração do ambiente de desenvolvimento deve possuir a variável CLASSPATH e AJANTA_HOME. Para a instalação do Ajanta, deve ser escolhidos um administrador e um usuário.

Ainda de acordo com o manual dos desenvolvedores da plataforma, é proposta a instalação administrativa que inclui todos os módulos, como a criação da configuração do registro do nome (*name registry*), que permite que o administrador

execute e registre outros usuários no registro do nome. A opção "instalação do usuário" irá desconsiderar o processo de configuração de registro do nome, onde normalmente há um único nome, de um domínio. Um usuário é responsável por saber a sua localização DNS (nome do host) e perguntar o nome do administrador (*admin name*) para registrar-se. Executando a instalação do script "userSetup", deve ser usado o script perl "userSetup" para instalar em seu usuário Ajanta ambiente[14]:

O script % Perl $ AJANTA_HOME / setup / userSetup (dentro do ambiente UNIX), irá lhe pedir para escolher entre duas opções de instalação: (1) Administrador e (2) Usuário.

Se outro usuário está servindo como um administrador, pode-se escolher a opção (2) Usuário. Nesse caso, será necessário perguntar ao administrador de registro de nomes para registrar-se no local nome-registro. Este script criará um diretório ".Ajanta" em seu diretório *home*. Este diretório mantém todas as informações de configuração necessárias para diferentes entidades como usuário, registro do nome (*name registry*), os agentes-servidores no Ajanta.

O ambiente de laboratório foi montado com três máquinas, sendo, dois servidores e uma máquina cliente, para avaliar o tráfego do agente móvel entre os servidores, vindo da máquina cliente. E para monitorar e avaliar o conteúdo dos pacotes que trafegam na rede, foi colocada uma máquina com a ferramenta de monitoração WireShark. A proposta é que a máquina cliente envie um agente para o primeiro servidor e depois este agente trafegue para o segundo servidor, como ilustrado a seguir:

[14] http://ajanta.cs.umn.edu/papers/Guide/toc.html

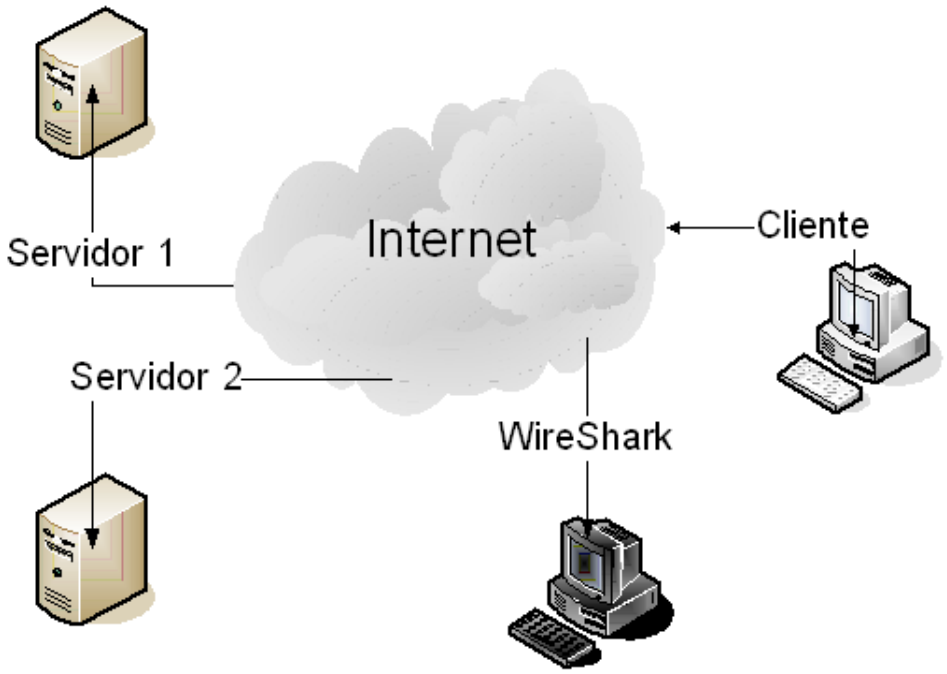

Figura 5 – Tráfego do agente do servidor 1 para o servidor 2 – validação de segurança dos agentes móveis
Fonte: Elaborado pela autora

Dentre as ferramentas de desenvolvimento de agentes móveis disponíveis, a escolhida para validar este conceito e fazer o laboratório é o AJANTA, por motivos descritos na seção anterior.

Alinhado às instruções dos desenvolvedores da plataforma de agentes móveis Ajanta, o laboratório foi instalado e colocado em funcionamento de acordo com os passos apresentados a seguir. Esta plataforma permite a instalação em ambiente Linux ou Windows. O laboratório foi instalado no ambiente Windows XP, acompanhando as seguintes telas.

O primeiro passo é preparar o ambiente para a instalação do AJANTA, configurando as variáveis do ambiente, criando um diretório chamado HOME e AJANTA_HOME. Dentro do ambiente Windows, entre no PAINEL DE CONTROLE, na guia SISTEMAS, conforme a figura 7 :

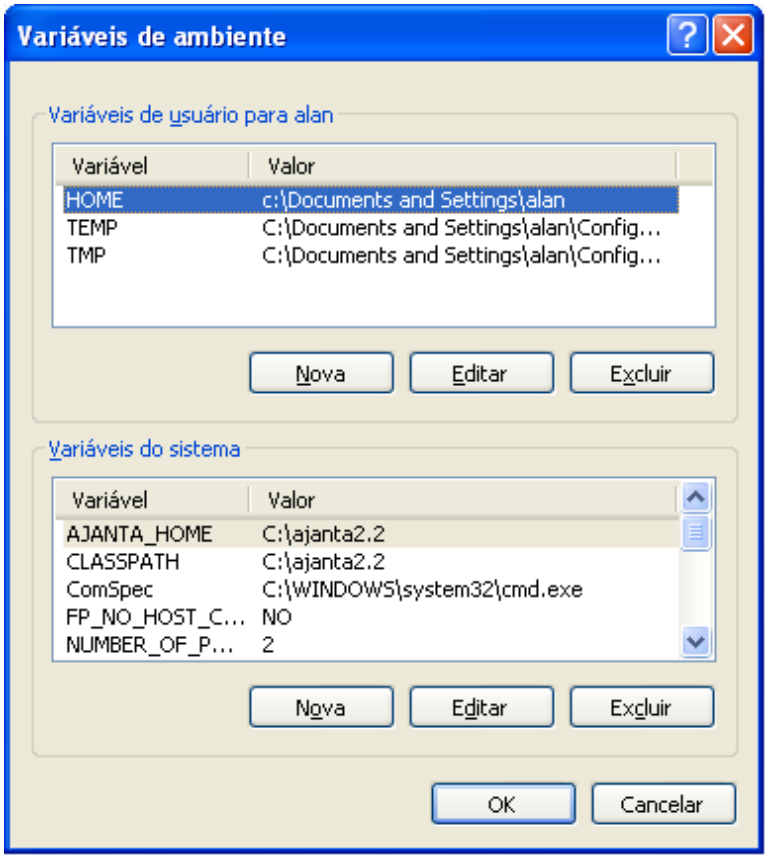

Figura 6 - Configuração de variáveis de ambiente
Fonte: Windows

Para o passo seguinte é necessária a extração dos arquivos e criação dos diretórios, conforme mostrado a seguir:

AJANTA2.2

AJANTA

EXAMPLE

SETUP

WINSETUP

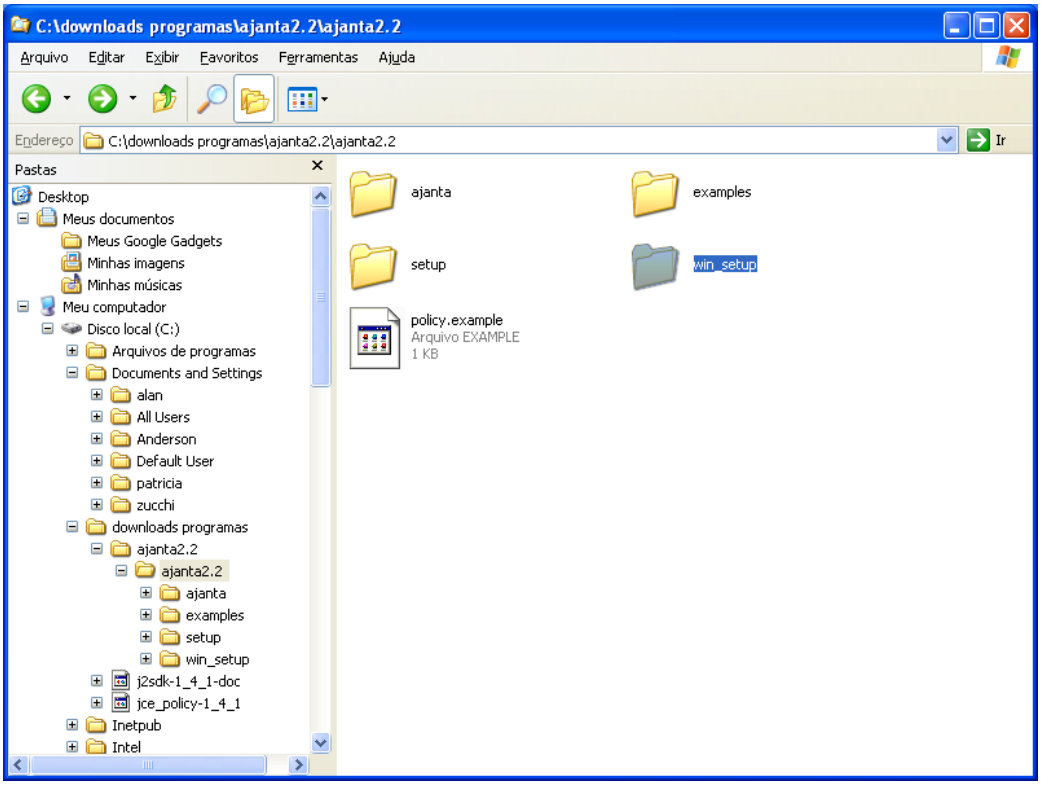

Figura 7 – Diretórios de configuração do Ajanta

Fonte: Windows

No passo 3, entrar no diretório WINSETUP. Entrar no AJANTA CONFIG e executar o USER SETUP.

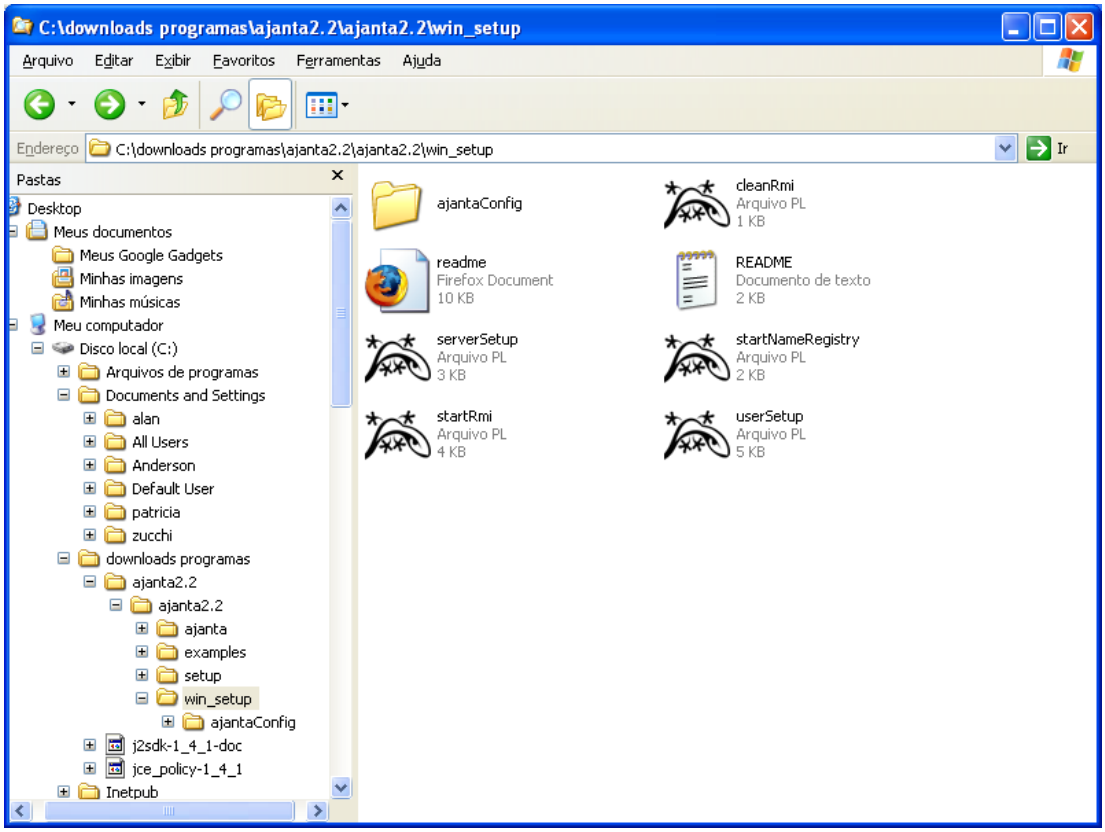

Figura 8 – Arquivos de instalação do Ajanta

Fonte: Windows

O próximo passo é o 4, adicionar o CLASSPATH no servidor 1:

Entre no PAINEL DE CONTROLE, na guia SISTEMAS e em VARIÁVEIS DE AMBIENTE, como mostra a figura a seguir.

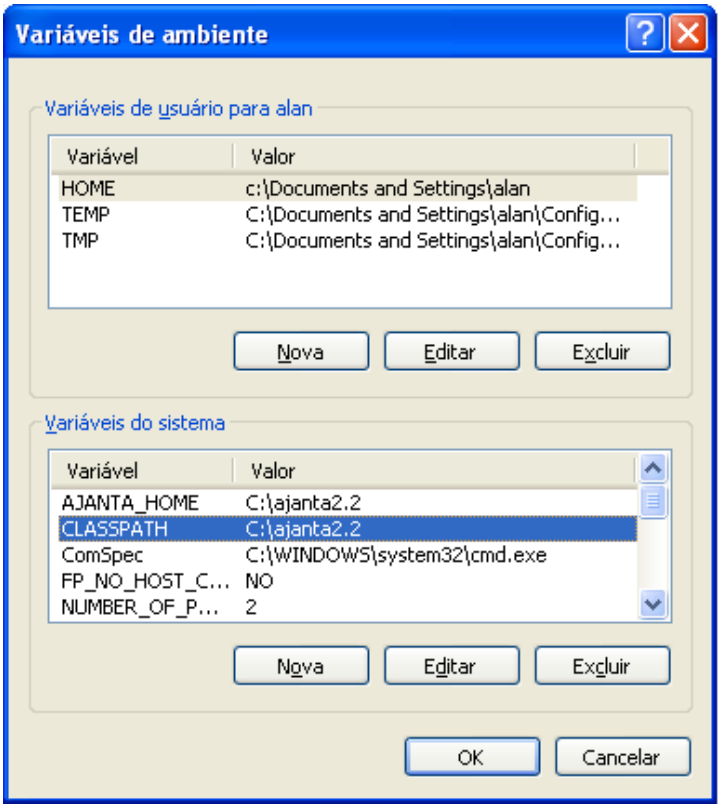

Figura 9 – Configuração de variáveis de ambiente

Fonte: Windows

No passo 5, executar o USER SETUP.

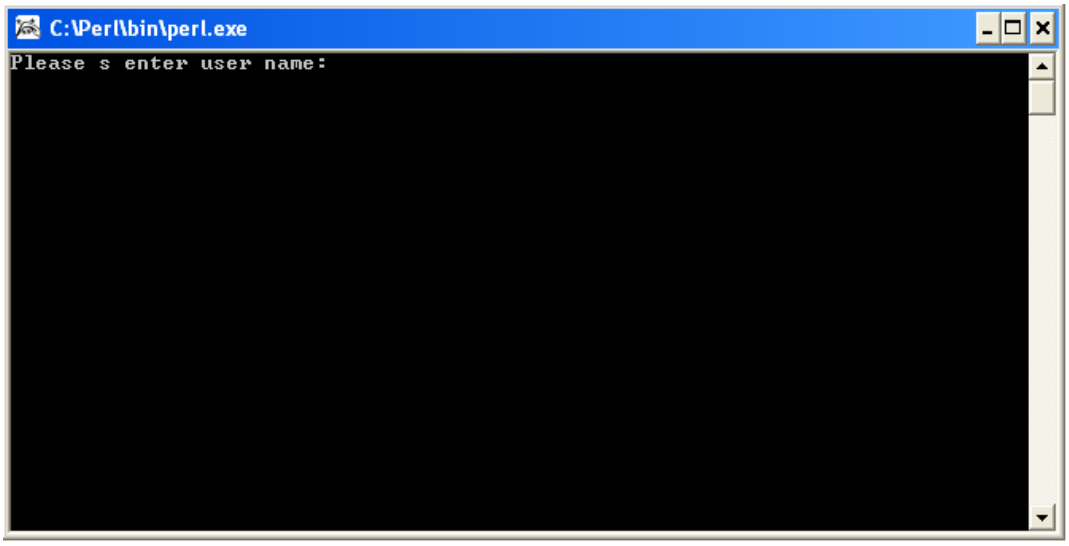

Figura 10 – Configuração do User Setup

Fonte: Windows

Passo 6: Colocar o usuário, no caso deste laboratório o nome ALAN no servidor 1.

Esse processo deve ser repetido nas 3 máquinas que fazem parte do laboratório.

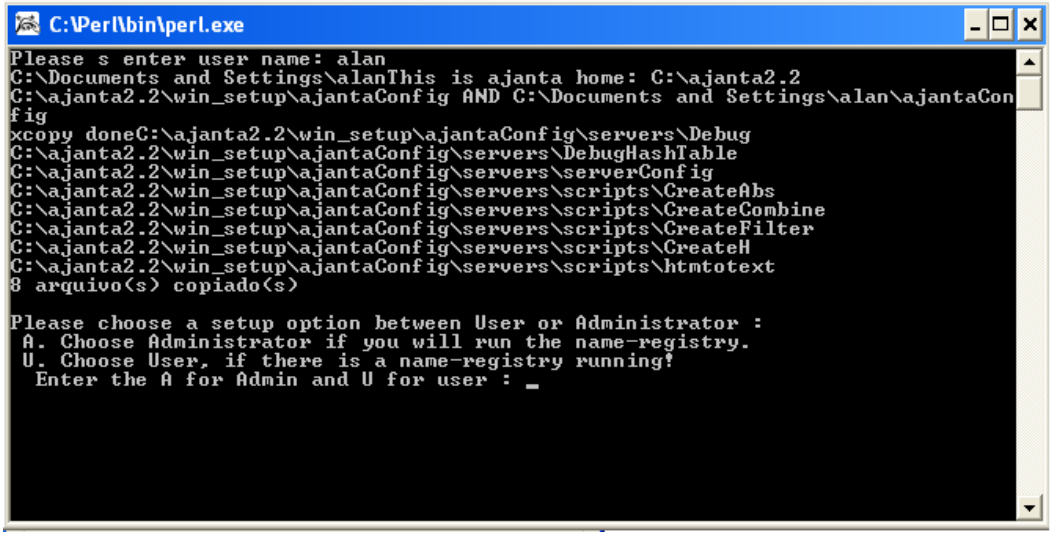

Figura 11 – Configuração do nome do usuário na máquina cliente

Fonte: Windows

Passo 7: NAME REGISTER

Conforme ilustra a figura 13, deve ser colocado “U” ou “A” no NAME REGISTER.

U = usuário

A = Administrador

Lembrando que somente uma das máquinas pode ser “A”.

Depois desta definição, tecle “Y”.

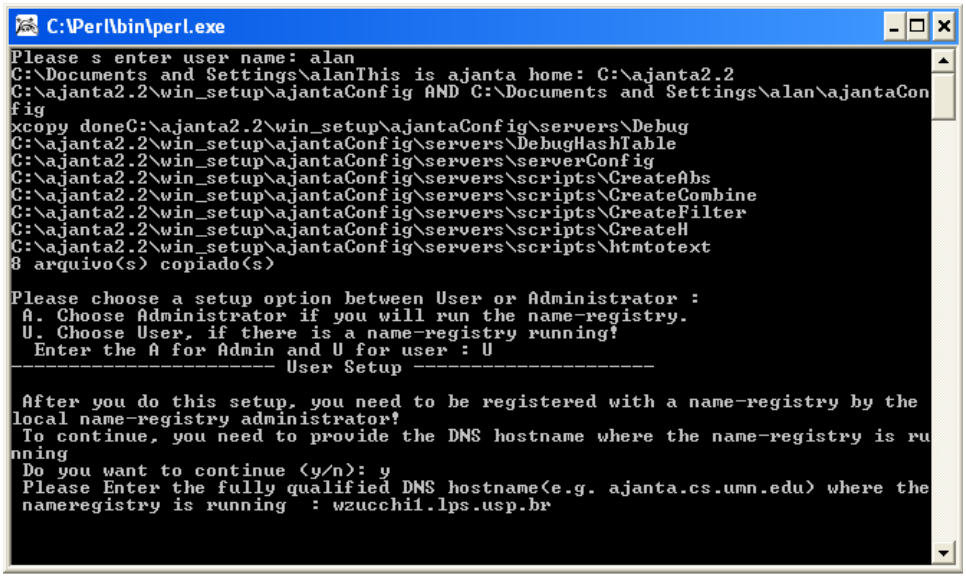

Figura 12 – Configuração do Name Registry Administrador ou Usuário

Fonte: Windows

Passo 8: Coloque o nome DNS onde está sendo executado o NAME REGISTER: wzucchi.lps.usp.br, no caso foi usado este laboratório para a conclusão deste trabalho. Desta forma foram criados todos os diretórios instalados na plataforma AJANTA.

Passo 9: apresenta a tela com o resultado

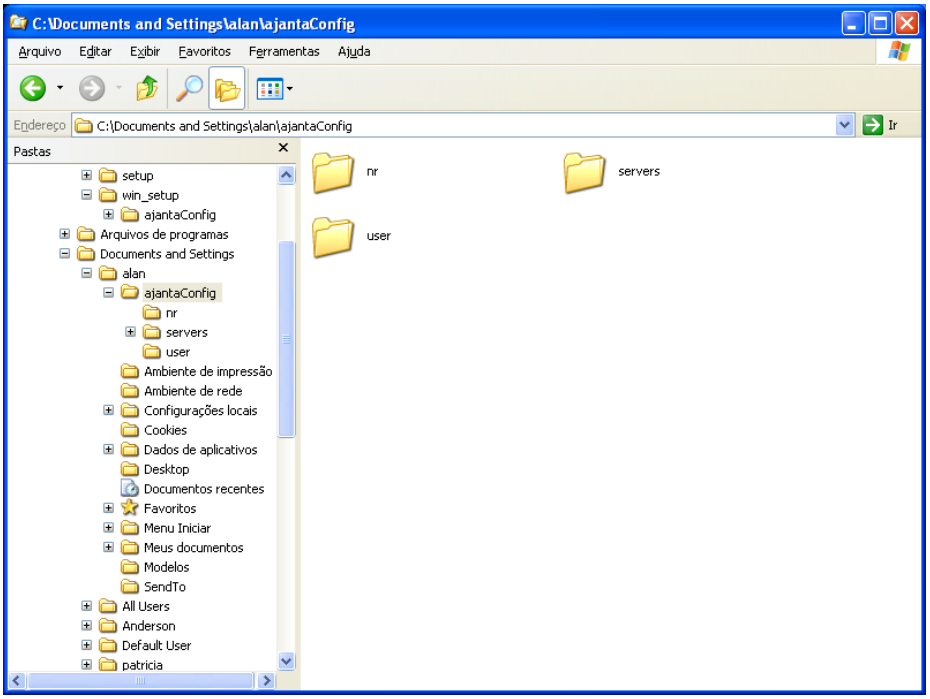

Figura 13 – Arquivos de configuração do Ajanta

Fonte: Windows

Passo 10: Depois de todos esses passos, o ambiente está criado e o próximo passo é a criação das chaves.

Entrar no NR e teclar JAVA AJANTA.CRYPTO.GENERATEKEYPAIRS.

E teclar enter e ".".

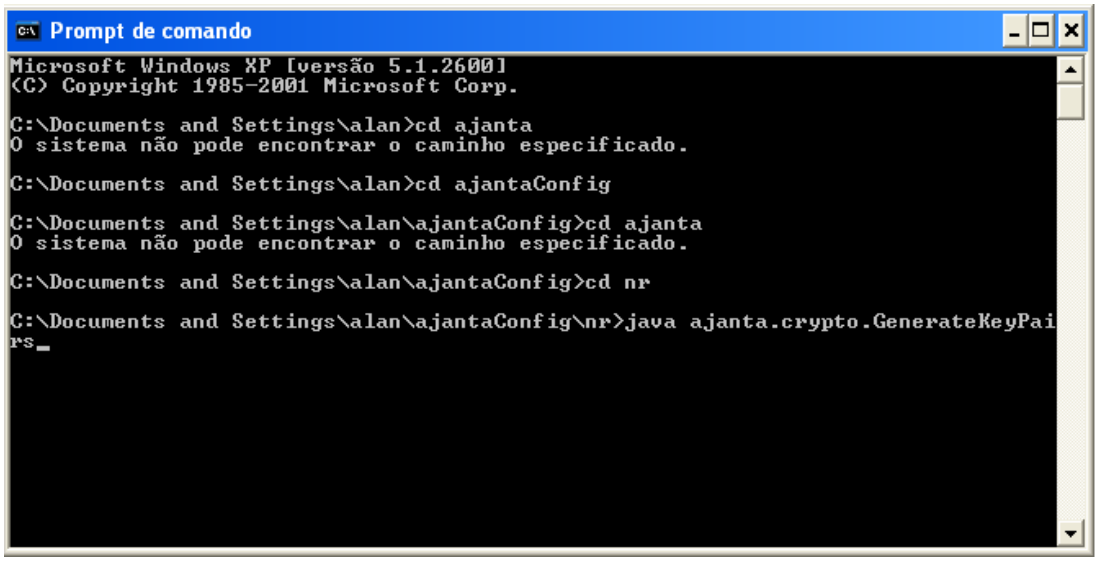

Figura 14 – Janela de criação de chaves

Fonte: Windows

Passo 11: Na próxima tela, tecle o tamanho da chave: 512

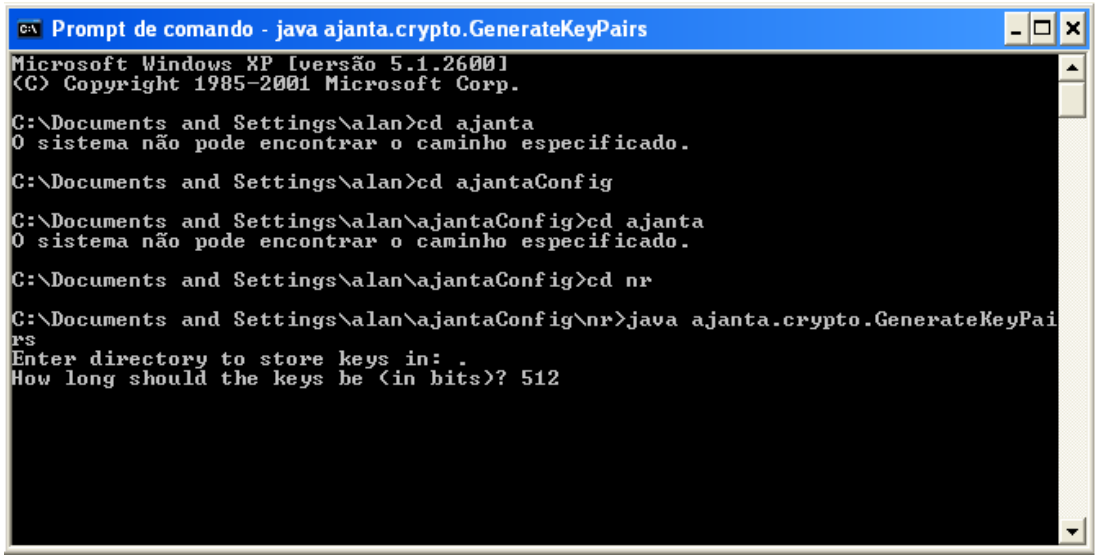

Figura 15 – Colocação da chave 512

Fonte: Windows

Esse processo deve ser repetido nos diretórios e em todos os outros diretórios dos programas que serão utilizados.

SERVERS/

HELLOSERVER1

HELLOSERVER2

HELLOWORLD

3.4 Segurança de Execução no ambiente JAVA

Códigos executados em qualquer máquina que não seja a mesma na qual foram criados, possibilitam vulnerabilidades quanto à segurança, uma vez que as ações realizadas não podem ser previstas antes da execução do código. Uma abordagem para os códigos móveis, utilizados em sistemas distribuídos, tais como os *applets* JAVA, que podem ser citados como exemplo, consiste em limitar severamente as ações que tais códigos podem realizar, para melhorar a segurança do processamento remoto. Sem tais limitações a existência de *applets* seria impossível na Internet, pois ao acessar uma página qualquer poder-se-ia estar recebendo e executando automaticamente um programa potencialmente perigoso.

Por esse motivo, um *applet* sempre executa no contexto de um programa hospedeiro e limitado por esse programa a uma “caixa de areia” (*sandbox*) que impede o *applet* de ter acesso aos dados da máquina onde ele executa. Pode-se dizer então que, um *applet* Java é um aplicativo que usa a JVM (Java Virtual Machine) que fica localizado na máquina do cliente. Não é um agente móvel, mesmo tendo a característica de ser executado em uma “caixa de areia”, não permitindo o acesso à máquina onde eles são executados. É utilizado no ambiente Web, conforme informação da SUN[15].

O desenvolvimento de agentes móveis, abordados neste trabalho é mais complexo, pois pretende-se transferir métodos, ou sub-rotinas para uma máquina remota, com o programa hospedeiro que executa no cliente. Dessa forma, o método remoto executa diretamente sobre a JVM, sem estar embutido em um programa hospedeiro.

[15] www.sun.com

O fato de um código móvel ser desenvolvido dentro de uma plataforma segura, ou dentro de uma máquina confiável, não o torna invulnerável a ataques, pois este código pode se tornar um agente malicioso pelo simples fato do seu estado ter sido corrompido por plataformas visitadas anteriormente. Garantir que este agente é seguro, manter a confiabilidade com itinerários livres, sem conhecer as plataformas e caminhos visitados não permite afirmar que esta solução seja segura.

As proteções inerentes a um agente móvel devem permitir que este trafegue em uma rede sem danificar ou alterar o seu conteúdo perdendo-se a confiabilidade.

É possível ilustrar esse aspecto com a necessidade de compra de uma passagem aérea. O papel do agente móvel neste exemplo é buscar dentro do ambiente da internet a melhor companhia aérea com o melhor preço e condições desta compra. Este agente solicita informações e tem permissão para reservar as passagens assim que encontrá-las, dentro das condições e critérios exigidos. Para que esta compra seja efetuada, o agente móvel deve transportar um cartão de crédito eletrônico. Neste caso, é fundamental a segurança deste cartão, pois sempre que um agente passa por um hospedeiro, este não deve ter permissão para roubar, alterar, ou criar novas informações no cartão de crédito. Além disso, o agente deve ser protegido contra alterações nos dados que façam com que o comprador pague mais por uma passagem aérea. Finalmente, o servidor precisa de garantias de que o agente irá executar apenas operações permitidas (Tanembaum, 2008).

A plataforma JAVA, bem como a plataforma de desenvolvimento de agentes móveis Ajanta, utiliza um modelo de segurança conhecido como*"caixa de areia (sandbox)*, já citado anteriormente. Este modelo de segurança caracteriza-se por ser um ambiente restrito, como uma "caixa de areia", para que os códigos remotos, que possam ser considerados perigosos ou maliciosos consigam trafegar com acesso controlado e recursos limitados. Dentro desta "caixa de areia", o código tem acesso restrito aos recursos de sistema, por meio de um gerenciador de segurança *(security manager)*, responsável por essa gestão e por determinar quais acessos são permitidos e como é controlado esse acesso. A técnica utilizada nesta "caixa de areia" é descarregar e executar um programa de modo que as instruções possam ser controladas. Caso exista a tentativa de executar uma operação proibida pelo hospedeiro, esta execução é interrompida, caracterizando a violação da segurança (Tanembaum, 2008).

Na plataforma JAVA, desde a versão JDK 1.0 (*JAVA Development Kit*) o modelo de caixa de areia é usado. Adotado inicialmente pelas aplicações de códigos móveis, como *Applets* e *JavaScript* que executam no contexto de um navegador Web, com um *plug-in* que os habilita a executar códigos JAVA.

A figura 5 ilustra o modelo de *"caixa de areia"* original:

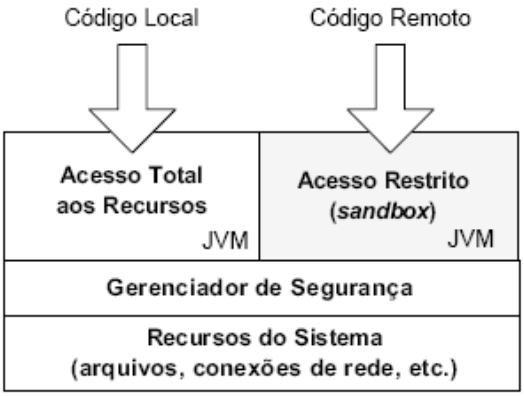

Figura 16 – Modelo de Segurança JDK
Fonte: Wangham (2004)

A implantação desta técnica de "caixa de areia" na plataforma JAVA é possível por transformar o programa-fonte em pseudo-códigos que são interpretados por uma máquina virtual (JVM: Java Virtual Machine). Essa linguagem virtual foi projetada para ser segura em relação aos tipos de dados utilizados e às operações realizadas pelas aplicações nela desenvolvidas.

Um programa desenvolvido em JAVA é compilado e interpretado pelo JVM (*Java Virtual Machine*). A JVM fará o manuseio da execução, interpretando cada uma das instruções deste programa descarregado.

A "caixa de areia" tem como objetivo garantir a segurança das operações realizadas pelos programas remotos para uma máquina cliente. Dentro desta "caixa de areia" existem os carregadores de classes que fazem a carga de programas. O carregador de classe é responsável pela busca no servidor e colocação no cliente, permitindo o controle dos agentes móveis e a criação de objetos. Esses carregadores de classe são exclusivamente de confiança e estes programas podem possuir seus próprios carregadores, mas não podem criá-los e manipulá-los.

Outro componente de segurança para o desenvolvimento de códigos móveis utilizando a plataforma JAVA são os verificadores de *bytecode*. Os compiladores e

verificadores garantem que os *bytecodes* JAVA sejam executados nestas aplicações e permitem a verificação de uma classe carregada obedecer às regras de segurança impostas pela “caixa de areia”. O verificador de *bytecode* verifica o código antes da execução sobre a máquina virtual.

O carregador de classes recupera a classe correspondente, possivelmente de um site remoto e então carrega a classe na JVM. O carregador de classes define um espaço de nomes distintos para códigos não confiáveis para que estes não tenham interferências na execução de outros programas. Depois que uma classe é descarregada e verificada sua segurança, o JVM pode alocar os objetos. Para dificultar mais ainda o acesso a recursos não autorizados do cliente, é usado um gerenciador de segurança.

Os programas JAVA que serão carregados passam obrigatoriamente por esse gerenciador que valida o acesso ou não e assim não permitindo que programas não autorizados tenham acesso a informações que não são devidas (Tanembaum, 2008).

Essa política de gerenciamento de segurança é característica da plataforma JAVA. Estes gerenciadores são bastante restritivos e não fazem distinção entre programas carregados, ou mesmo aqueles que são originários de diferentes servidores. Gerenciamento automático de memória, coleta automática de lixo (*garbage collection*), acesso estruturado a memória, verificação dos limites de vetores são características dos códigos móveis em ambiente de desenvolvimento do código com segurança (Tanembaum, 2008).

3.5 Invocação de Métodos Remotos

Com o objetivo de tornar a linguagem mais potente, incorporando recursos de ativação de objetos distribuídos, outra característica do ambiente de desenvolvimento JAVA para aplicações distribuídas é a utilização de um método de invocação de código remoto chamado RMI (*Remote Method Invocation*).

O RMI ou *Remote Method Invocation* é uma variante do RPC *(Remote Procedure Call)*. O RPC é um mecanismo utilizado em algumas linguagens de programação que permite que códigos remotos sejam utilizados por outros códigos que estejam localizados em outras máquinas na mesma ou em outra rede. Quando o

código chama uma classe remota exige-se dois elementos: um RMI-Client onde a classe é invocada e um RMI-Server onde a classe é executada. O método RMI inclui duas porções de software, o chamado “stub”, trabalha no servidor para tornar transparente a execução das classes remotas e o “skeleton”, que trabalha na máquina cliente.

3.6 Conceito do RMI

Na linguagem de desenvolvimento Java, em ambientes de sistemas distribuídos, um programa remoto é um método que deve ser invocado a partir de outra máquina virtual JAVA, potencialmente diferente em um *host*. Um programa deste tipo é descrito por uma ou mais interfaces remotas, que são interfaces escritas na linguagem de programação JAVA que declaram os métodos do objeto remoto.

A invocação remota é a ação de invocar um método de uma interface remota em um objeto remoto. Mais importante ainda, um método de invocação remota em um objeto tem a mesma sintaxe como um método de invocação sobre um objeto local.

3.7 Aplicações Cliente e Servidor

Após a definição da interface remota, a classe deve implementar o serviço remoto, e para isso ela deve ter sido criada. O passo seguinte no desenvolvimento da aplicação distribuída é desenvolver o servidor RMI, uma classe que crie o objeto que implementa o serviço e cadastre esse serviço na plataforma de objetos distribuídos (Ricarte, 2002).

O desenvolvimento do cliente RMI necessita da obtenção de uma referência remota para o objeto que implementa o serviço, o que ocorre por meio do cadastro feito pelo servidor. Com esta referência funcionando, a operação com o objeto remoto fica idêntica à operação com um objeto local. Para que um serviço oferecido por um objeto possa ser acessado remotamente através de RMI, são necessárias também as classes auxiliares internas de *stubs* e *skeletons*, responsáveis pela comunicação entre o objeto cliente e o objeto que implementa o serviço, conforme descrito na arquitetura RMI (Ricarte, 2002).

3.8 Arquitetura RMI

A arquitetura RMI permite a transparência de localização por meio da organização de três camadas entre os objetos cliente e servidor, sendo que a camada de *stub/skeleton* permite que os objetos da aplicação interajam entre si. A camada de referência remota é o *middleware* entre a camada de *stub/skeleton* e o protocolo de transporte. E a camada do protocolo de transporte permite que o protocolo de dados binários envie as solicitações aos objetos remotos pela rede.

O *stub* é um programa local que implementa a interface remota do objeto remoto. O programa cliente passa a acreditar que está acionando um método em um objeto local, mas na verdade está chamando um método equivalente no *stub.* O *stub* permite que o cliente se comunique diretamente com o servidor, que fala junto à camada de referência remota e conversa com a camada de transporte. A camada de transporte sobre o cliente envia os dados por meio da internet para a camada de transporte no servidor. A camada de transporte do servidor se comunica com o servidor remoto da camada referenciada, que fala com uma parte do software do servidor chamado o *skeleton.* O *skeleton* comunica-se com o servidor e na outra direção (servidor-a-cliente), desta forma, invertendo-se o fluxo (Harold, 2004).

O RMI consiste em uma arquitetura cliente-servidor, onde o servidor cria novas instâncias de objetos remotos e os referencia com um nome. Este objeto espera que clientes convoquem seus métodos, que podem ser um ou mais. Essas aplicações são conhecidas como aplicação de objetos em sistemas distribuídos e precisam localizar objetos remotos, por meio do *rmiregistry.*

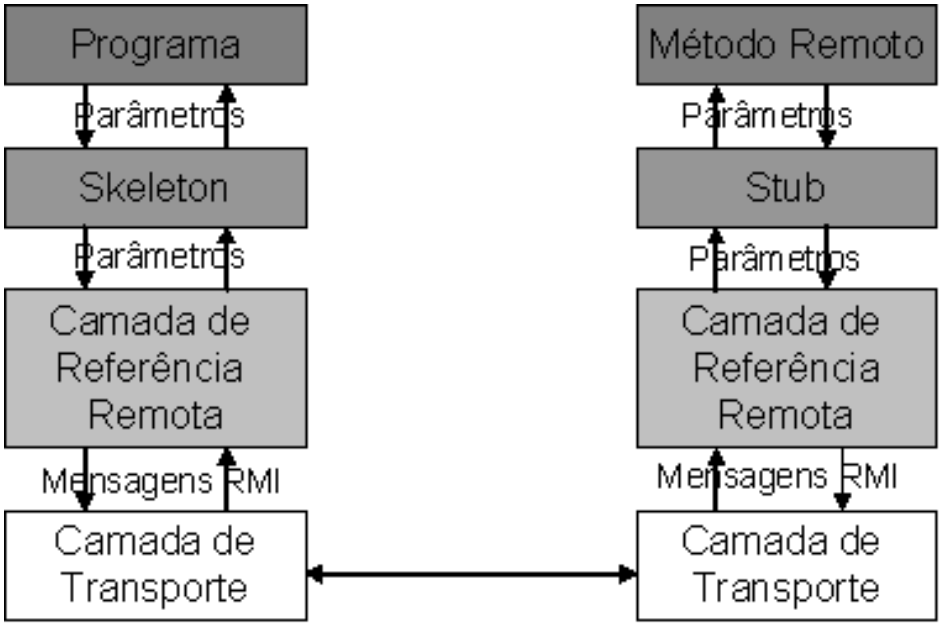

Figura 17 – Arquitetura de comunicação do serviço RMI
Fonte: Elaborado pela autora

O objetivo do RMI é permitir que o seu programa passe argumentos por meio de métodos para retornar valores, que se movimentarão pelo ambiente de rede. Ele permite que um objeto localizado em uma Máquina Virtual JAVA (interpretador de *bytecode*) acione métodos de um objeto, instanciado em outra Máquina Virtual JAVA, permitindo a comunicação entre objetos distribuídos de forma similar à comunicação entre objetos locais, simplificando a programação de sistemas distribuídos. Estas operações são realizadas na forma cliente-servidor. O servidor cria instâncias de objetos remotos e espera por clientes que invoquem seus métodos. Pelo lado cliente, este referencia remotamente um ou mais métodos de um objeto remoto. O RMI contém mecanismos para a comunicação entre cliente e servidor. Os sistemas distribuídos precisam localizar os objetos, se comunicar e carregar os *bytecodes.*

A aplicação usa mecanismos de localização de objetos remotos e os registra com a ferramenta de nomes que se chama *rmiregistry.* Os endereços dos objetos remotos podem ser localizados por meio de uma consulta ao *rmiregistry*. A comunicação com os objetos remotos é tratada pelo RMI, de forma semelhante a uma chamada local (Harold, 2004).

A figura 18, ilustra como o RMI atua em uma aplicação distribuída *rmiregistry* e como é feita a referência ao objeto remoto.

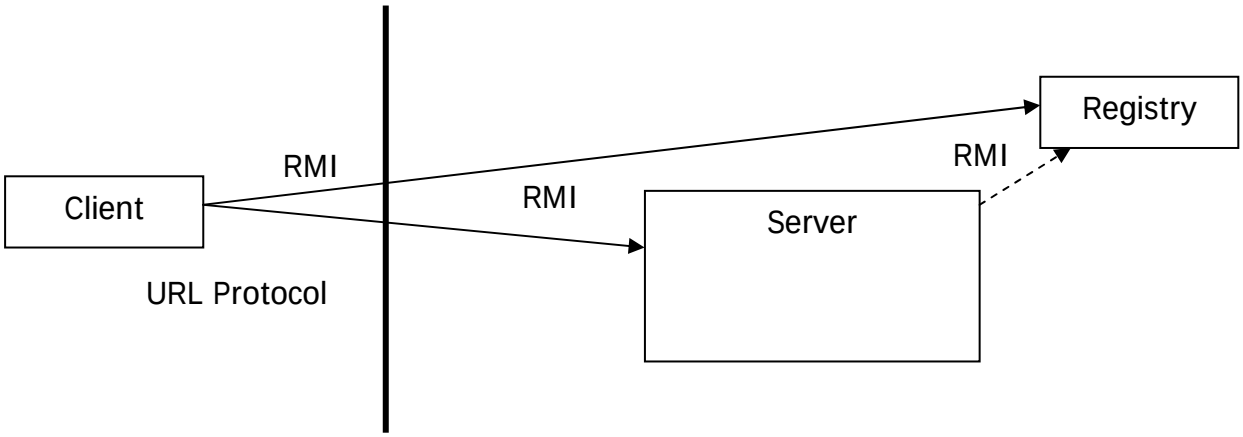

Figura 18 – Funcionamento do *rmiregistry* em aplicações distribuídas
Fonte: Elaborado pela autora

O objeto remoto permanece no servidor e cada objeto remoto implementa uma interface remota pela qual esse método pode ser chamado pelo cliente. O cliente chama o objeto remoto da mesma forma como se estivesse na máquina local. Por exemplo, um objeto em execução em uma máquina local (cliente) passa por uma consulta de dados como um argumento para um método String. Estes dados em execução de um servidor remoto solicitam a soma destes registros como se estivesse em uma máquina local. O servidor retorna o resultado deste cliente de forma mais eficiente do que somá-los localmente. Normalmente essa interface limita outros programas a serem executados nesta mesma máquina ou numa máquina virtual. Entretanto, este método remoto solicita que outras máquinas virtuais executem o código em outras máquinas virtuais ou em qualquer máquina, onde estiver, trazendo estes dados para a máquina local. A interface remota é similar a qualquer interface em JAVA, diferenciada somente pelo fato de que a interface deverá ser direta ou indireta.

A estrutura de uma interface remota para um objeto que contém um contador interno é mostrada a seguir, manipulado por quatro métodos (Ricarte,2002):

- **set()**, definição de um valor inicial para o contador;
- **reset()**, reinício do contador com o valor 0;
- **get()**, consulta do valor do contador sem alterá-lo; e

- **increment()**, leitura do valor atual do contador e incrementa-o.

3.9 Implantação do RMI

A implantação do serviço é feito por meio da definição de uma classe para a implantação de uma interface e a inclusão de funcionalidades deste serviço para que um objeto dessa classe possa ser acessado remotamente sendo um servidor. Esta implantação remota se dá do mesmo modo que a implantação de qualquer classe que implanta uma interface JAVA, ou seja, a classe fornece a implantação para cada um dos métodos especificados na interface.

É necessário implantar três pacotes: java.rmi, java.rmi.server e java.rmi.registry, para que funcione. E deve ser implantado da seguinte forma, o java.rmi define classes, interfaces e exceções que estão no cliente. O java.rmi.server define classes, interfaces e exceções no lado servidor. E o java.rmi.registry são usados para localizar e nomear os objetos remotos.

As funcionalidades deste servidor remoto são especificadas na classe abstrata RemoteServer, do pacote java.rmi.server. Um objeto servidor RMI deve referenciar essa classe ou, mais especificamente, uma de suas subclasses. Uma subclasse de RemoteServer oferecida no mesmo pacote é UnicastRemoteObject, que representa um objeto de uma única implementação em um servidor e mantém uma conexão ponto-a-ponto com cada cliente que o referencia (Ricarte,2002).

3.10 Java.RMI

É um mecanismo que fornece condições a um objeto em uma máquina virtual JAVA para invocar métodos sobre um objeto em uma outra máquina virtual. Qualquer objeto que possa ser invocada desta forma deve implantar a interface remota. Quando um objeto é invocado, os seus argumentos são *condensados* e deslocam-se a partir do local da máquina virtual para um remoto, onde os argumentos são *separados.* Quando o método termina, os resultados se deslocam a partir da máquina remota e enviada para o chamador da máquina virtual. Este mecanismo está presente no JDK a partir da versão 1.1.

Uma interface Java.RMI remota deve estender a interface java.rmi.Remote. Entretanto, em uma interface remota é permitido que exista uma interface não-remota.

Uma interface remota também pode estender outra interface não-remota, desde que todos os métodos da prorrogação da interface satisfaçam os requisitos de um método remoto. Como exemplo, a seguinte interface de um banco define uma interface remota para acessar uma conta bancária. Contém métodos remotos para se fazer um depósito na conta, para obter o saldo da conta, e para retirar da conta:

```
public interface BankAccount extends java.rmi.Remote {
        public void deposit(float amount)
                throws java.rmi.RemoteException;
        public void withdraw(float amount)
                throws OverdrawnException, java.rmi.RemoteException;
        public float getBalance()
                throws java.rmi.RemoteException;
}
```

O próximo exemplo ilustra uma interface remota “X” que se estende por uma distância não-remota “Y”, que tem métodos remotos da interface java.rmi.Remote:

```
public interface Y {
        public final String okay = "constants are okay too";
        public Object foo(Object obj)
                throws java.rmi.RemoteException;
        public void bar() throws java.io.IOException;
        public int baz() throws java.lang.Exception;
}
public interface X extends Y, java.rmi.Remote {
        public void ping() throws java.rmi.RemoteException;
}
```

Os exemplos citados anteriormente são retirados do site oficial da SUN[16], a fim de ilustrar o Java.RMI.Remote, o primeiro exemplifica um método remoto e o outro, o segundo, um método não-remoto.

3.11 Classes e SubClasses no Java.RMI

Alinhado ao fabricante JAVA, SUN, pode ser dito que a classe e subclasse Java.rmi.RemoteException é a superclasse de exceções lançadas no *runtime* do RMI durante a chamada do método remoto. Com o objetivo de permitir que as aplicações atestassem a veracidade das aplicações que utilizam o sistema RMI.

Cada método remoto declarado em uma interface remota deve especificar java.rmi.RemoteException (ou uma de suas superclasses, como java.io.IOException ou java.lang.Exception) na sua cláusula.
A exceção java.rmi.RemoteException é lançada quando um método remoto falhar por algum motivo. Como motivos para que isso aconteça, podemos dizer que uma falha na comunicação onde o servidor remoto está inacessível ou se recusar conexões, ela é fechada pelo servidor, ou mesmo uma falha durante um parâmetro ou valor retornado ou erro nos protocolos. Este controle é feito pelo controlador e não pelo Runtime Exception.

3.12 Java. RMI.Registry

É um mecanismo que fornece uma classe e duas interfaces para a RMI Registry. Um objeto remoto é um registro que mapeia nomes de objetos somente e um servidor registra os objetos remotos, para que possam ser consultados. Quando um objeto necessita invocar um método remoto em um objeto, este deve primeiramente procurar o objeto remoto usando seu nome. O registro retorna para o objeto remoto ao chamar um objeto de referência para este, que utiliza um método remoto para ser chamado. Este mecanismo está presente no JDK a partir da versão 1.1.

[16] http://java.sun.com/j2se/1.4.2/docs/guide/rmi/spec/rmi-objmodel5.html#3459

3.13 Java.RMI.Server, suas classes e subclasses

Este mecanismo fornece classes e interfaces de apoio ao servidor de RMI. Um grupo de classes é utilizado pelas *stubs* e *skeleton* gerados pelo compilador do RMI.Stub. Outro grupo de classes implementa o protocolo da camada de transporte do RMI e tunelamento do HTTP. Este mecanismo está presente no JDK a partir da versão 1.1.

O Java.RMIserver tem funções que são fornecidas pela java.rmi.server.RemoteObject e suas subclasses, java.rmi.server.RemoteServer e java.rmi.server.UnicastRemoteObject e java.rmi.activation.Activatable.

A classe java.RMI.server.RemoteObject fornece implementações para os métodos do java.lang.Object, código hash, equal e ToString, que são sensíveis a objetos remotos.

Os métodos obrigatórios para a criação de objetos são fornecidos pelas classes UnicastRemoteObject e Activatable. A subclasse identifica a semântica da referência remota, como se o servidor remoto fosse um simples objeto remoto, sendo chamado quando necessário.

A classe java.rmi.server.UnicastRemoteObject define um único (unicast) objeto remoto cujas referências são válidas somente enquanto o processo no servidor estiver no ar.

A classe java.rmi.activation.Activatable é uma classe abstrata que define um objeto remoto ativo que inicia quando a sua execução de métodos remotos são invocados.

3.14 Implementando interfaces remotas

Para a implementação de classes remotas, deve ser entendido que a classe java.rmi.server.UnicastRemoteObject, herda o controle remoto fornecido pelo comportamento das classes java.rmi.server.RemoteObject e java.rmi.server.RemoteServer. Esta classe implementa qualquer número de interfaces remotas. A classe incrementa a execução remota e esta deve definir métodos que não aparecem na interface remota, mas esses métodos a utilizam apenas localmente e não estão disponíveis remotamente. A fim de exemplificar o

seu comportamento, a seguir é apresentado um modelo do fabricante SUN onde é implementada a classe BankAcctImpl, com a interface remota bankaccount e classe java.rmi.server.UnicastRemoteObject:

```
package mypackage;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class BankAccountImpl
        extends UnicastRemoteObject
        implements BankAccount
{
        private float balance = 0.0;

        public BankAccountImpl(float initialBalance)
                throws RemoteException
        {
                balance = initialBalance;
        }
        public void deposit(float amount) throws RemoteException {
                ...
        }
        public void withdraw(float amount) throws OverdrawnException,
                RemoteException {
                ...
        }
        public float getBalance() throws RemoteException {
                ...
        }
}
```

O RMI dentro do ambiente Ajanta, inicia o ambiente agente-servidores, onde o usuário precisa executar um RMI. Como exemplo, é escolhido executar agente-servidores em máquinas chamadas "Máquina1", "Máquina2" e "Máquina3", é necessário a execução do script "startRmi" em todas estas máquinas como exibido a seguir:

Máquina1> perl) ($ AJANTA_HOME / setup / startRmi

Máquina2> perl) ($ AJANTA_HOME / setup / startRmi

Máquina3> perl) ($ AJANTA_HOME / setup / startRmi

O script irá iniciar uma RMI Registro no atual da máquina, onde ele está sendo executado. Mantém-se também a informação no

"$ HOME / .ajanta / servidores / serverConfig".

Caso exista qualquer problema ou dificuldade em acessar esse código, o script eliminará o RMI Registry anterior e irá executar um novo.

3.15 Limitações das ferramentas de Segurança em Agentes Móveis

O controle de acesso de segurança do JAVA apresenta algumas limitações a serem consideradas. Em vez de possuir uma característica distribuída do seu modelo de execução, o modelo de segurança do JAVA possui um esquema de segurança centralizado. Está baseado no controle do tempo de execução e no conceito de monitor de referência implementado pela classe do Gerenciador de Segurança. Durante a sua execução, cada objeto é rotulado como pertencente a um domínio de proteção.

No JAVA, os direitos de acesso de cada componente, os *applets*, são definidos em um arquivo de configuração tratados em um ambiente em tempo de execução. Incluindo o mapeamento entre os códigos móveis e as permissões concedidas para execução deste código no ambiente local.

A dificuldade na programação é o impedimento do desenvolvimento de um ambiente distribuído e dinâmico, exigindo uma programação estática de todas as partes do programa distribuído e suas características de segurança. E também da alta utilização de CPU (Unidade Central de Processamento) e memória que

comprometem o modelo de segurança do JAVA em ambiente de desenvolvimento. Como conseqüência desta habilidade de utilização indiscriminada de CPU e memória, ataques de negação de serviço (*DoS*, *Denied of Service*) que tem como objetivo consumir toda a memória disponível ou permitir que o sistema torne-se extremamente lento, comprometendo o desempenho.

Nas aplicações desenvolvidas em JAVA é difícil identificar o motivo e a causa de problemas que possam causar lentidão, visto que a dificuldade está exatamente em identificar o código malicioso que está demandando a memória e a máquina. De toda forma, mesmo com dificuldades existentes na segurança, muitas aplicações distribuídas são desenvolvidas utilizando JAVA como linguagem de programação. Na versão da plataforma JAVA mais atualizada é permitido diversas funcionalidades de segurança que proporcionam aos programadores desenvolver aplicações com um nível aceitável de segurança.

3.16 Aspectos de Segurança em plataformas de Agentes Móveis disponíveis e laboratório

Como os agentes móveis podem viajar por redes e sites não confiáveis, o programador de agentes necessita de conceitos e premissas para garantir a confidencialidade dos dados (primitivas para cifragem e decifragem) e a integridade dos dados (primitiva para fornecer *digests*). Bem como assinaturas digitais e verificação destas assinaturas também são necessárias para estabelecer comunicações entre pares autenticados, criptografia de chave pública e uma infraestrutura para gerenciar os possíveis certificados associados a estas chaves.

Como forma de validar os mecanismos de segurança proposto nas plataformas de desenvolvimento de agentes móveis, foi desenvolvido um laboratório. A ferramenta de desenvolvimento de agentes móveis escolhida para validar os aspectos de segurança é o Ajanta. Esta plataforma de desenvolvimento é uma ferramenta acadêmica e o laboratório permite que seja feita a análise do tráfego dos pacotes entre uma máquina cliente e dois servidores e apresentar um resultado sobre mecanismos de confidencialidade, integridade e segurança, por meio da monitoração de tráfego por uma ferramenta livre, WireShark.

4 ANÁLISE EXPERIMENTAL DOS MECANISMOS DE SEGURANÇA DA PLATAFORMA AJANTA

Esta seção apresenta o estudo laboratorial realizado para análise dos mecanismos de segurança da plataforma Ajanta.

O objetivo básico é verificar quais informações um eventual atacante com capacidade de capturar e interpretar mensagens trocadas através da internet poderia obter, somente observando as mensagens trocadas entre os clientes e os servidores da plataforma Ajanta.

Para tanto foi montado a seguinte topologia de rede:

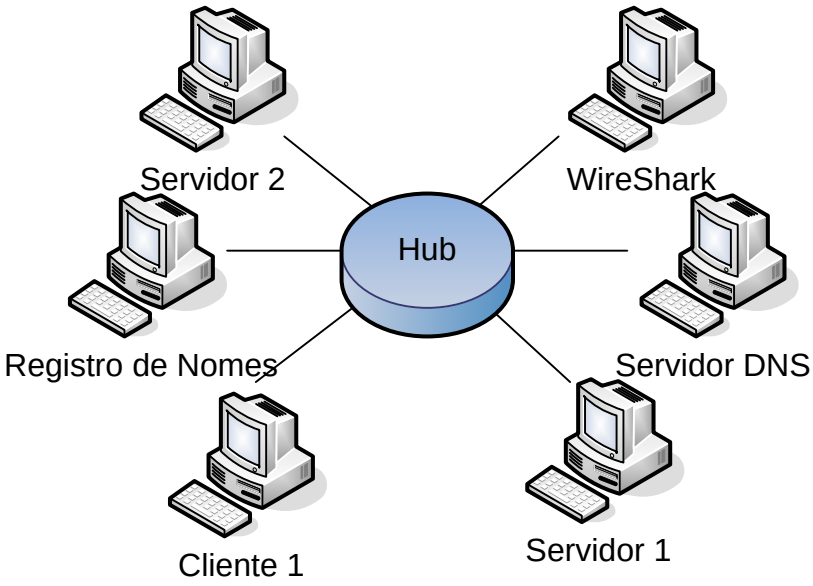

Figura 19 – Topologia da rede do Laboratório
Fonte: Elaborado pela autora

A plataforma Ajanta utiliza o protocolo “RMI WIRE” para a sinalização de objetos e transporte de mensagens (Ricarte, 2002).

O ítem 4.1 descreve o protocolo RMI WIRE. O ítem 4.2 apresenta as telas capturadas durante uma transação RMI WIRE comprovando o trânsito dos pacotes utilizando o protocolo de comunicação no laboratório experimental. O ítem 4.3 apresenta a análise dos mecanismos de segurança do ponto de vista dos serviços de autenticação, autorização, confidencialidade e conclusões.

4.1 O Protocolo RMI WIRE

O Protocolo RMI WIRE é composto de dois outros protocolos para compor o seu formato: Java Object Serialization e HTTP. O primeiro protocolo é utilizado para envio e retorno de dados. O HTTP é usado para invocação remota por acesso à internet.

O formato do protocolo RMI WIRE é representado por um “Stream”, conforme documentado pela SUN. As terminologias utilizadas refletem a perspectiva do cliente, onde o “Stream” é composto por Out (*OutputStream*) e In (*InputStream*). Out refere-se à mensagens de saída e In para mensagens de entrada em relação ao cliente (JAVA, 2008).

Para cada *Out* existe um *In* equivalente e eles trafegam na rede em paralelo. Apenas o cabeçalho possui informações necessárias sobre um fluxo de entrada e outras informações, como número de versão do protocolo.

As mensagens de saída (*Out*) possuem informações no cabeçalho RMI, seguidos por uma seqüência de mensagens. Um fluxo de saída pode conter embutida uma solicitação para o protocolo HTTP.

O fluxo de saída *OutputStream* é composto pelos seguintes métodos:

```
public void close( ) throws IOException
public void flush( ) throws IOException
public void write(byte[] buffer) throws IOException
public void write(byte[] buffer, int startingOffset, int numberOfBytes)
throws
        IOException
public void write(int value) throws IOException
```

O fluxo de saída é composto pelos itens descritos a seguir, onde a variação das opções depende do tráfego do protocolo que se tem na rede:

Tabela 1 – Fluxo de saída

Out:

Header Messages

HttpMessage

Header:

0x4a 0x52 0x4d 0x49 Version Protocol

Version:

0x00 0x01

Protocol:

StreamProtocol

SingleOpProtocol

MultiplexProtocol

StreamProtocol:

0x4b

SingleOpProtocol:

0x4c

MultiplexProtocol:

0x4d

Messages:

Message

Fonte: Ricarte (2002)

O *HTTP Protocol* é um protocolo de comunicação, e o *HTTP Message* faz a comunicação entre o cliente e o servidor, onde o cliente envia uma mensagem e o servidor envia uma mensagem de retorno ao cliente.

As mensagens de entrada e saída são descritas e utilizadas em *Protocol*. No protocolo *SingleOpProtocol*, existe apenas uma mensagem após o cabeçalho e na ausência de dados adicionais para que a mensagem seja envolta pelo *SingleOpProtocol*, este solicita o HTTP e a mensagem 0x4c.

No *StreamProtocol, SingleOpProtocol* ou *MultiplexProtocol*, no *OutputStream*, o servidor responde com o byte 0x4b, 0x4c ou 0x4d, reconhecendo como a resposta do protocolo. E coloca o *EndpointIdentifier* ao final da mensagem, que contém o nome do host e o número da porta do servidor utilizado pelo cliente.

Após a negociação pelo *StreamProtocol*, as mensagens são retornadas, sem qualquer alteração dos dados trafegados. O *MultiplexProtocol* é o meio de conexão utilizado como ligação para uma multiplexação, conforme descrito na documentação do protocolo RMI multiplexador. Para estabelecimento das conexões virtuais iniciadas durante a multiplexação é necessária uma série de mensagens como descritas na sequência.

Existem três tipos de mensagens de saída: *Call, Ping* e *DgcAck*. O *Call* codifica um método de invocação. Um *Ping* é o transportador da mensagem de testes de conexão e máquinas virtuais, indicando o envio e recebimento. Um *DGCAck* é um aviso dirigido a um *Garbage Colletor* que indica os valores de retorno de um objeto remoto a partir da mensagem de um servidor ter sido recebida pelo cliente. O *DGCAck* será detalhado mais adiante nesta seção.

Tabela 2 – Mensagem de saída

Message:

Call

Ping

DgcAck

Call:

0x50 CallData

Ping:

0x52

DgcAck:

0x54 UniqueIdentifier

Fonte: Ricarte (2002)

A classe de entrada ou *InputStream* consiste nos seguintes métodos:

```
public int available( ) throws IOException
public void close( ) throws IOException
public void mark(int numberOfBytes) throws IOException
public boolean markSupported( ) throws IOException
public abstract int read( ) throws IOException
public int read(byte[] buffer) throws IOException
public int read(byte[] buffer, int startingOffset, int numberOfBytes)
throws
        IOException
public void reset( ) throws IOException
public long skip(long numberOfBytes) throws IOException
```

Na tabela a seguir a descrição do fluxo de entrada e suas opções:

Tabela 3 – Fluxo de entrada

In:

ProtocolAck Returns

ProtocolNotSupported

HttpReturn

ProtocolAck:

0x4e

ProtocolNotSupported:

0x4f

Returns:

Return

Returns Return

Return:

ReturnData

PingAck

ReturnData:

0x51 *ReturnValue* *(opcional)*

PingAck:

0x53

Fonte: Ricarte (2002)

No fluxo de entrada existem três tipos de mensagem de entrada: *ReturnData*, *HttpReturn* e *PingAck*. O *ReturnData* é o resultado de uma chamada do RMI WIRE. Um *HttpReturn* é o resultado da invocação no protocolo HTTP. Um *PingAck* é o reconhecimento da entrada da mensagem por meio de um *ping*.

Os dados que são enviados e recebidos no protocolo RMI WIRE, são formatados usando a capacidade de serialização da plataforma JAVA. Serialização é um processo de conversão de objetos que contém referências, onde os dados podem ser enviados, armazenados em arquivo ou simplesmente manipulados como dados “streams” (Grosso, 2001).

Um método de serialização é conhecido como *CallData e* foi escrito para atender um objeto desenvolvido no ambiente JAVA. O retorno dos dados e o envio no RMI WIRE são formados usando o protocolo Java Object Serialization. Cada método chama um *CallData* que é representado por: *ObjectIdentifier* (o alvo da chamada), um *Operation* (um número representando o método acionado), um *Hash*

(um número que verifica o cliente *Stub* e o *Skeleton* usado no protocolo *Stub*), seguidos de uma lista de zeros, que são os argumentos e que são opcionais.

Tabela 4 – Método de serialização

CallData:

ObjectIdentifier Operation Hash Arguments *(opcional)*

ObjectIdentifier:

ObjectNumber UniqueIdentifier

UniqueIdentifier:

Number Time Count

Arguments:

Value

Arguments Value

Value:

Object

Primitive

Fonte: Ricarte (2002)

No RMI WIRE o *ReturnValue* significa o retorno de um código para indicar qualquer retorno normal ou opcional e um *UniqueIdentifier* para codificar o valor de retorno (usado para enviar uma DGCAck se necessário), seguido pelo retorno do resultado, que podem ser 0X01 ou 0X02, como descritos abaixo.

Tabela 5 – Retorno de um código

ReturnValue:

0x01 *UniqueIdentifier Value*

0x02 *UniqueIdentifier Exception*

Fonte: Ricarte (2002)

A DGCAck é uma notificação dirigida ao *Garbage Collector* para indicação do valor de retorno dos objetos remotos vindos de um servidor que tenha sido recebido por um cliente.

Os objetos *ObjectIdentifier*, *UniqueIdentifier*, e *EndpointIdentifier* não são escritos utilizando a serialização padrão, mas cada um utiliza o seu próprio método.

4.1.1 Valores do RMI WIRE

A tabela a seguir, lista os símbolos que representam os valores específicos das aplicações utilizadas pelo RMI WIRE. A tabela mapeia cada símbolo para seu respectivo tipo e valores (Ricarte, 2002):

Tabela 6 – Lista de símbolos utilizados pelo RMI WIRE

Count	short
Exception	java.lang.Exception
Hash	long
Hostname	String
Number	int
Object	java.lang.Object
ObjectNumber	int
Operation	int
PortNumber	int
Primitive	byte, int, short, long
Time	long

Fonte: Ricarte (2002)

4.1.2 Protocolo de Multiplexação

O objetivo da multiplexação é permitir que vários canais trafeguem por um único meio. É fornecer um meio para que dois parâmetros possam abrir múltiplas conexões *full duplex,* para a outra extremidade, permitindo que apenas um dos parâmetros seja capaz de abrir esta conexão bidirecional usando algum outro mecanismo (por exemplo, uma conexão TCP). O RMI WIRE utiliza este protocolo para que seja feito a multiplexação a fim de um cliente conectar-se a um servidor RMI WIRE (Grosso, 2001).

Este protocolo permite o uso de conexões virtuais, bidirecionais, que possibilitam uma conexão entre dois pontos. O estado de uma conexão virtual e os parâmetros de conexão é definido pelos elementos do protocolo, que são enviados e recebidos através de uma conexão.

Cada conexão virtual multiplexada em uma mesma conexão física é identificada por um número de 16 bits, chamado identificador de conexão virtual. Uma conexão física suporta até 65.536 conexões virtuais.

Por motivos de segurança é desautorizada a criação do servidor para estabelecer as conexões entre o cliente e o servidor, impedindo que os *applets* que exportam objetos RMI WIRE e serviço de chamadas de conexões remotas possam comunicar-se. Se o *applet* abrir uma conexão, este pode usar o protocolo durante a multiplexação para permitir a conexão e invocar métodos dos objetos do RMI WIRE exportados pelo *applet*.

4.1.3 Controle de Fluxo dos dados

As operações que utilizam as conexões definidas pela multiplexação são: OPEN, CLOSE, CLOSEACK, REQUEST e TRANSMIT.

O formato e as regras utilizadas nestas operações estão no formato do protocolo, detalhados e explicados no item 4.1.4. O controle das conexões das operações OPEN, CLOSE e CLOSEACK permitem a abertura e o fechamento, enquanto REQUEST e TRANSMIT são usados para a transmissão dos dados trafegados e abertura das conexões de restrições controladas pelos mecanismos.

Uma conexão virtual é permitida ser aberta e finalizada por meio do *endpoint*. O fechamento de uma operação OPEN pode ser feito com uma operação CLOSE ou CLOSEACK.

Uma conexão virtual está pendente de fechamento, quando um *endpoint* não foi fechado por uma operação CLOSE, ou CLOSEACK ou quando o mesmo nunca foi aberto.

O protocolo de multiplexação usa um simples mecanismo de controle de fluxo que permite múltiplas conexões virtuais, acontecendo em paralelo durante a mesma conexão. Por exigência desse mecanismo de controle de fluxo, não é permitido que o comportamento de uma conexão afete a outra. Se os armazenadores de dados, ou buffers manipulados ficarem cheios, isto não impede a transmissão dos dados.

Em chamadas recursivas, onde o processo é repetido por algumas vezes, é necessário que este processo seja fechado. O resultado é que a aplicação deve ser

sempre capaz de consumir e processar os protocolos da multiplexação para a entrada e saída de dados.

Cada *endpoint* tem dois valores associados com cada conexão, indicando quantos bytes são requeridos, mas não recebidos *(input request count)* e quantos bytes no outro *endpoint* são solicitados, mas não fornecidos (*output request count*).

O contador de *endpoint* no output é contabilizado, incrementado quando recebe um REQUEST de outro *endpoint* e este contador é diminuído quando é enviado um TRANSMIT. Este segundo processo, na diminuição do contador é conhecido como violação de protocolo de ambos, tornando o valor negativo. Esta violação de protocolo para o envio de um *endpoint* para a operação REQUEST que poderia incrementar este contador. Deve ser verificado que a contagem do pedido de entrada é superior a zero se o usuário da conexão estiver aguardando para ler os dados.

Se a violação de um protocolo ou se uma comunicação de erro é detectada, em seguida a conexão da multiplexação é encerrada. O verdadeiro propósito é armazenado e todas as conexões virtuais tornam-se imediatamente fechadas.

4.1.4 Formato do Protocolo

O formato do protocolo de multiplexação consiste em uma série contínua de registros de tamanho variável. O primeiro byte do registro é uma operação código que identifica o funcionamento do registro e determina o formato do restante do seu conteúdo. Os códigos da operação são definidos da seguinte forma:

Tabela 7 – Registros de tamanho variável

Valor	Nome
0xE1	OPEN
0xE2	CLOSE
0xE3	CLOSEACK
0xE4	REQUEST
0xE5	TRANSMIT

Fonte: Ricarte (2002)

4.1.5 OPEN

Tamanho (bytes)	*Nome*	*Descrição*
1	opcode	*operation code (OPEN)*
2	ID	*connection identifier*

Um *endpoint* envia uma operação OPEN para abrir a conexão indicada. Após a conexão aberta, a entrada para a conexão é igual à zero em ambos os casos. Uma violação do protocolo ID refere-se a uma conexão que abre ou fecha as conexões pendentes.

Para evitar colisões entre dois parâmetros, o espaço de conexão válido é dividido ao meio, dependendo do valor do bit mais significativo. Só é permitido a cada parâmetro abrir conexões com um valor específico para o bit. A conexão só deve ser aberta com o bit fixado no identificador e os outros parâmetros só devem abrir conexões com um zero (Ricarte, 2002).

4.1.6 CLOSE

Tabela 8 – Operação CLOSE

Tamanho (bytes)	*Nome*	*Descrição*
1	*opcode*	*operation code (OPEN)*
2	*ID*	*connection identifier*

Fonte: Ricarte (2002)

Um *endpoint* envia um aviso para que seja fechada a conexão indicada. Uma violação de protocolo ID refere-se a uma conexão que se encontra fechada ou pendente no que diz respeito ao envio de parâmetro. Após o envio do CLOSE, a ligação torna-se pendente para fechamento. Assim, não se pode reabrir a conexão até que ele tenha recebido um CLOSE ou um CLOSEACK a partir da outra extremidade.

A recepção de uma operação CLOSE, indica o fechamento da outra extremidade encerrando a conexão indicada. Apesar de não receber ou enviar mais operações para esta conexão (até que se abra de novo), ele ainda deve fornecer

dados na execução de entrada. Se a conexão está aberta em vez de fechar as pendentes, recebendo o *endpoint* deve-se responder com uma operação CLOSEACK para o fechamento da conexão (Ricarte, 2002).

4.1.7 CLOSEACK

Tabela 9 – Operação CLOSEACK

Tamanho (bytes)	*Nome*	*Descrição*
1	opcode	*operation code (OPEN)*
2	ID	*connection identifier*
4	count	*number of additional bytes requested*

Fonte: Ricarte, 2002

Para que esta operação funcione, uma operação *endpoint* envia um pedido para que seja enviado um contador de conexão. Uma violação de protocolo ID não se refere a uma conexão que está aberta no que diz respeito ao envio. O parâmetro de entrada é incrementado pelo valor contagem. Esse valor é de 32 bits inteiros, e é um protocolo violação, se for negativo ou zero. A recepção de um pedido de operação CLOSEACK faz com que o pedido de saída para a contagem indicada aumente. Se uma conexão estiver pendente pela recepção parâmetro, qualquer pedido pode ser ignorado (Ricarte, 2002).

4.1.8 REQUEST

Tabela 10 – Operação REQUEST

Tamanho (bytes)	Nome	Descrição
1	*opcode*	operation code (OPEN)
2	*ID*	connection identifier
4	*count*	number of additional bytes requested

Fonte: Ricarte, 2002

Uma *endpoint* envia um REQUEST para incrementar o contador, indicando uma conexão. Uma violação do protocolo ID não se refere a uma conexão que está aberta no que diz respeito ao envio. O parâmetro de entrada é incrementado pelo valor de contagem. O valor é assinado com 32 bits inteiros, e é um protocolo

violação, se for negativo ou zero. A recepção de um REQUEST faz com que uma solicitação de output incremente a conexão. Se a ligação estiver pendente, pela recepção de um parâmetro, então qualquer pedido pode ser ignorado (Ricarte, 2002).

4.1.9 TRANSMIT

Tabela 11 – Operação TRANSMIT

Tamanho (bytes)	Nome	Descrição
1	*opcode*	operation code (OPEN)
2	*ID*	connection identifier
3	*count*	number of bytes in transmission
count	*data*	transmission data

Fonte: Ricarte (2002)

Um *endpoint* envia um TRANSMIT para iniciar a transmissão de dados através de uma conexão. Se um protocolo de violação ID não se refere a uma conexão que está aberta no que diz respeito ao envio, o término da contagem do pedido de output é diminuído. O valor da contagem é de 32 bits inteiros, e é uma violação de protocolo, se for negativo ou zero, assim como se a operação provocar o envio do pedido de output para se tornar negativo. A recepção de um TRANSMIT faz com que a contagem de bytes de dados seja adicionada à fila de bytes disponíveis para leitura a partir da conexão. Quando a recepção do contador do sinal é diminuída pela contagem, a contagem do pedido de input torna-se o usuário da conexão, e este tenta ler um maior número de dados. Se a ligação estiver pendente pela recepção parâmetro, então qualquer envio de TRANSMIT pode ser ignorado (Ricarte, 2002).

4.2 Análise do Tráfego dos dados em Laboratório

O tráfego gerado pelo sistema Ajanta foi capturado para verificar como as funções de segurança, criptografia e autorização são utilizadas e qual o seu comportamento.

Nas telas que seguem, é possível ver que o protocolo RMI trafega de uma máquina a outra entre o cliente e servidor. A figura 22 apresenta no *StreamProtocol* onde o servidor responde com o byte 0x4e, reconhecendo a resposta do protocolo e o *EndpointIdentifier* que contém o nome do host e o número da porta do servidor utilizado pelo cliente. Desta forma garante-se a mútua autenticação entre as partes.

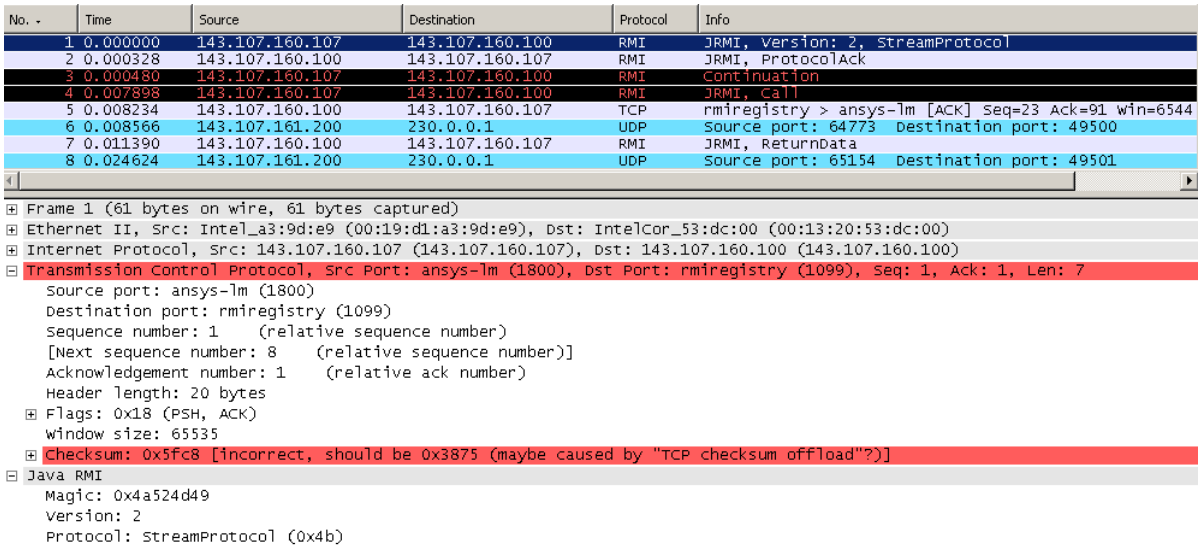

Figura 20 – Tráfego do protocolo RMI
Fonte: Analisador de protocolo WireShark

As mensagens geradas pelo protocolo abrem a conexão e na seqüência é enviado um *ProtocolAck* para a confirmação do estabelecimento da conexão e passagem para a fase de transmissão. A mensagem *ProtocolAck* é mostrada na figura 21.

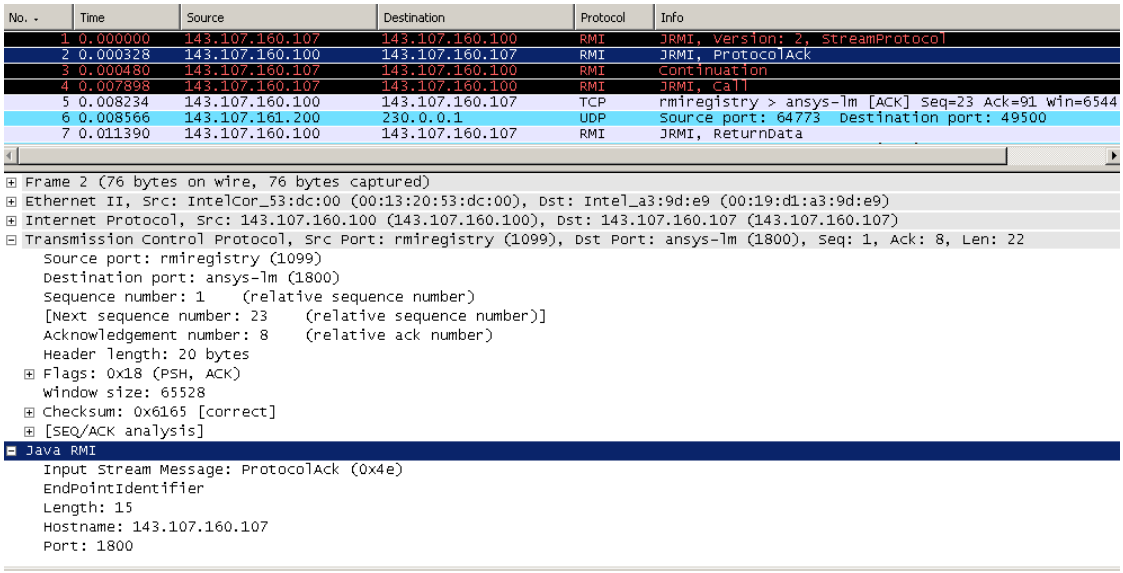

Figura 21 – ProtocolAck
Fonte: Analisador de protocolo WireShark

O pacote mostrado na Figura 22, continuação do tráfego, mostra a serialização de um objeto.

Os campos da mensagem permitem identificar o objeto de modo confiável e verificar a integridade do mesmo, como visto na discussão do protocolo RMI WIRE.

Com a serialização, apresentada a seguir, é permitida a codificação automática, para que os objetos possam ser transmitidos ao longo da rede, utilizando um método chamado *CallData,* e desta forma é possível confirmar o tráfego do protocolo RMI.

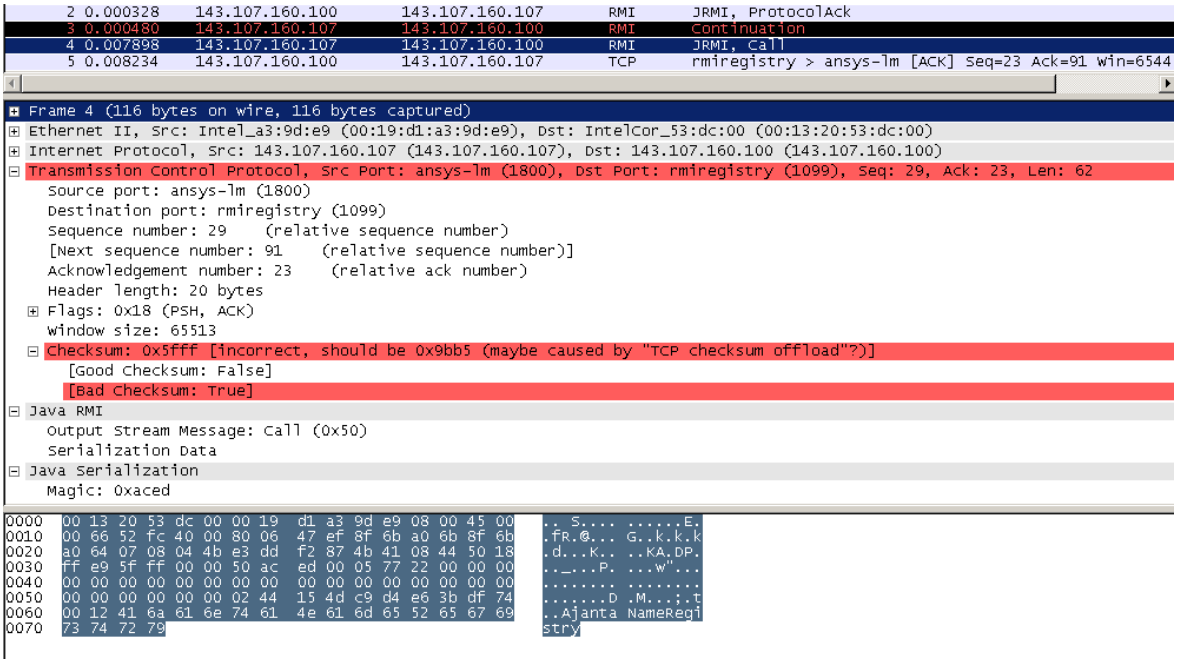

Figura 22 – Serialização de Objeto
Fonte: Analisador de protocolo WireShark

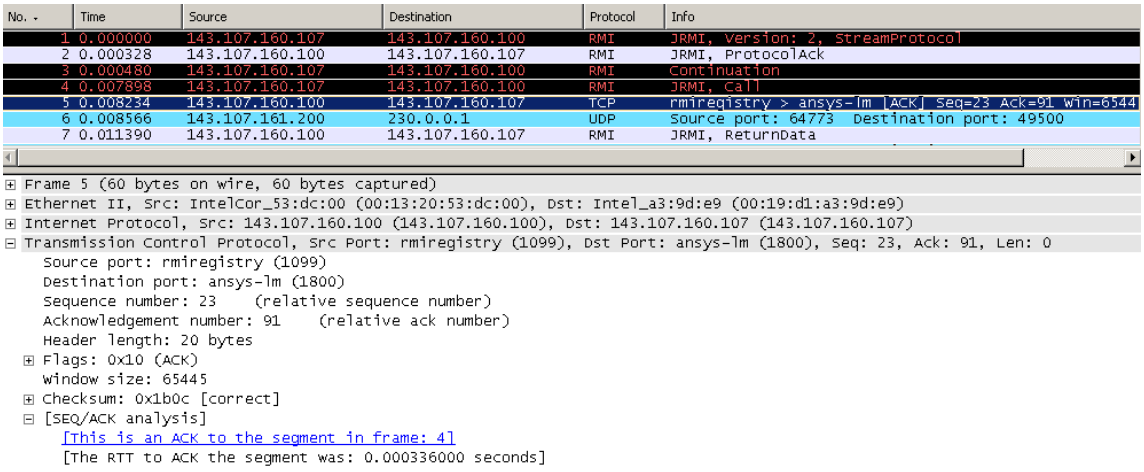

Figura 23 – Serialização de Objeto II
Fonte: Analisador de protocolo WireShark

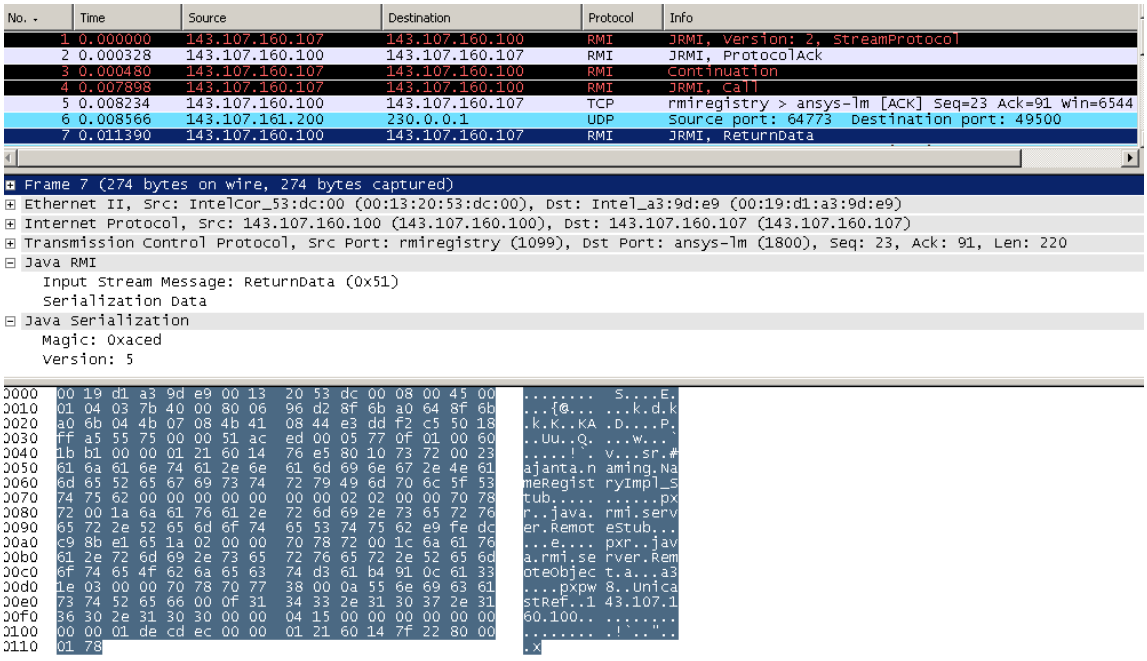

Figura 24 – Serialização de Objeto III
Fonte: Analisador de protocolo WireShark

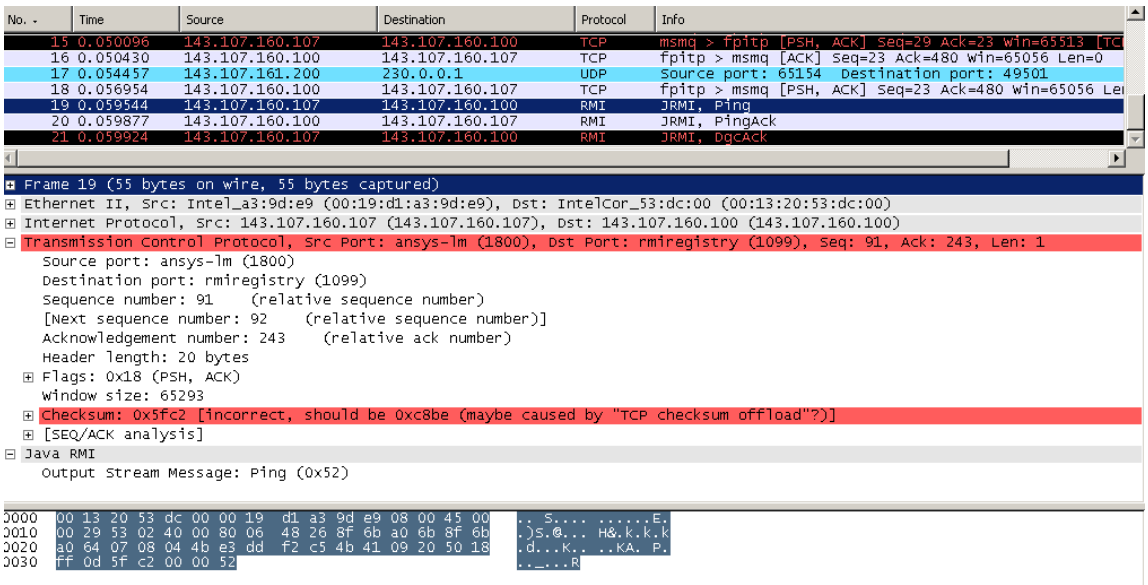

Figura 25 – PingAck
Fonte: Analisador de protocolo WireShark

A figura 25 mostra o reconhecimento da mensagem por meio de um ping *(PingAck (0x52)*)

A figura 26 mostra a indicação dos valores de retorno dos objetos remotos. A autenticação da mensagem é garantida pelo campo *UniqueIdentifier*, que aprova a resposta ao pedido efetuado pelo cliente.

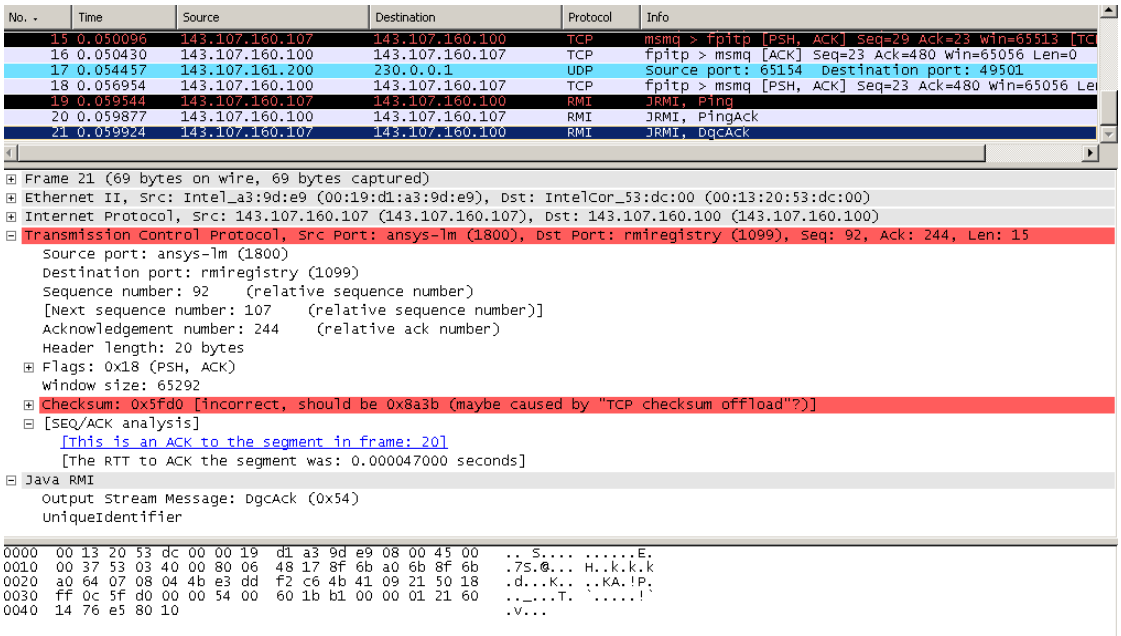

Figura 26 – Retorno dos objetos remotos
Fonte: Analisador de protocolo WireShark

5 CONCLUSÃO

A análise laboratorial mostrou como as funções de segurança consideradas críticas nesse trabalho: confidencialidade, integridade e autenticação, são utilizadas no sistema Ajanta e pode se concluir a partir dos pacotes que trafegam na rede, que apesar deles não estarem criptografados essas funções de segurança são atendidas, embora tais conclusões sejam parciais em função das limitações do ambiente de laboratório e dos recursos de autenticação utilizados.

Com as limitações existentes no ambiente, a utilização das soluções que envolvam agentes móveis fica restrita a um ambiente acadêmico, pois o ambiente corporativo necessita de alta disponibilidade dos dados e a segurança é imprescindível, além do ambiente controlado. O uso da identidade do proprietário do agente, como forma de delimitar os direitos de acesso, torna-se um fator limitante da plataforma. A limitação se deve ao fato do controle de segurança da plataforma Java ser feito em um modelo de autorização centralizado, por meio de um único arquivo de configuração de segurança.

A autenticação que é função fundamental na manutenção da segurança é realizada sem usar chaves públicas, permitindo um maior controle dentro do ambiente, facilidade de configuração e visibilidade das características de segurança neste ambiente.

Neste trabalho, com base nas análises feitas em laboratório, é possível concluir que os resultados obtidos nos permitem garantir apesar de forma limitada a segurança da plataforma de agentes móveis Ajanta, sendo mais crítico a essa limitação nos casos de ambientes corporativos, que necessitam de escalabilidade e alta disponibilidade. É possível afirmar que no ambiente corporativo, a utilização dos agentes móveis ainda é limitada e não utilizada. É necessário aprimorar a plataforma de desenvolvimento de agentes móveis tornando-a segura o suficiente para que seja utilizado em larga escala em ambientes corporativos.

5.1 Trabalhos Futuros

Uma importante lacuna desse trabalho é a extensão da análise realizada e dos resultados para além da fronteira do mundo corporativo.

Na internet aberta outros tipos de ameaças devem ser considerados e, consequentemente, as limitações dos resultados aqui obtidas são ainda mais evidentes.

A realização deste trabalho exigirá um laboratório de redes que possa emular eficazmente a internet aberta ou realiza um protótipo do serviço Ajanta ou outra plataforma nesse ambiente.

REFERÊNCIAS

ARIDOR, Y., LANGE, D.B., Agent Design Patterns: Elements of Agents Application Design. In: *INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS, 2. 1998.* Minnepolis, USA. **Proceedings...** ACM Press, New York. v.2. p.1-8.

CAMPIONE, M., WALRATH, K., E HUML, A. **The Java Tutorial.** *3.ed. Addison Wesley.* 2000. Disponível em : http://java.sun.com/docs/books/tutorial. Acesso em: 13 jul. 2009.

COSTA, T. F. S., **Avaliação Analítica do uso de Agentes Móveis na Gerência de Redes**, Florianópolis, 1999. 117f. Dissertação (Mestrado) - Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Santa Catarina, Florianópolis, 1999.

GROSSO, W., **JAVA RMI.** O'Reilly Media : New York. 2001. 572p.

HARALD, E.R. **Java Network Programming***, 3*. ed. O´Reilly, 2004.

IEEE INTERNET COMPUTING: INTERNET BASED AGENTS. New York, v.1, n.4, Jul/Aug. 1997.

ISO 27000 (2008). Disponível em: http://www.27000.org/ . Acesso em: 15 jun. 2008

JAEGER, T., PRAKASH, A., RUBIN, A., A System Architecture for Flexible Control of Downloaded Executable Content. **IEEE Computer Society.** 1996.

JAVA. Disponível em: http://java.sun.com/. Acesso em: 03 ago. 2008

LEE, VALENTINO, SCHNEIDER, HEATHER E SCHELL, ROBBIE. **Aplicações móveis: arquitetura, projeto e desenvolvimento**. São Paulo: Pearson Education do Brasil, 2005. 352p.

MATOS, L., ALMEIDA, M., SÁ, M. **Agentes Móveis – Aspectos de desenvolvimento**, Universidade Federal da Bahia, Salvador, 2007.

NICOMETTE, V., **La Protection dans les Systèmes à Objets Répartis***.* França, 1996. Tese (Doutorado) - Institut National Polytechnique de Toulouse, 1996.

OBELHEIRO, R. R., **Modelos de segurança baseado em papéis para sistemas de larga escala**: a proposta rbac-jacoweb. Florianópolis, 2001. 89f. Dissertação (Mestrado) - Centro Tecnológico. Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, 2001.

OYAMADA, M.S., ITO, S.A., ***Aglets: Agentes Móveis em Java.*** Porto Alegre : UFRGS, 1998. Disponível em: http://www.inf.ufrgs.br/gppd/disc/cmp134/trabs/T2/981/Aglets/aglets.html. Acesso em: 13 jul. 2009

RIBEIRO, F.J.L., LOPES, J.C.R., PEDROZA, A.C.P., **Análise dos Processos de Segurança em Sistemas Móveis de 3a Geração**. Rio de Janeiro: UFRJ, 2000. p.1-6.

RICARTE, I.L.M**. Curso on-line em JAVA RMI.** UNICAMP. 2002 Disponível em: http://www.dca.fee.unicamp.br/cursos/PooJava/objdist/javarmi.html Acesso em: 28 mar. 2009.

TANENBAUM, A. S., **Sistemas Distribuídos – Princípios e Paradigmas** *2.* ed. São Paulo: Prentice-Hall, 2008. 416p.

WANGHAM, M.S., **Esquema de Segurança para agentes móveis em sistemas abertos**. Florianópolis, 2004. 206f. Tese (Doutorado) – Centro de Tecnológico. Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal de Santa Catarina, Florianópolis, 2004.

GLOSSÁRIO

Advanced Encryption Standard (AES) - Padrão de Criptografia Avançado.

AJANTA - Nome dado a um sistema de programação, framework de agentes móveis desenvolvido na Universidade de Minesota

Bytecode - Código de programa de computador escrito na linguagem JAVA. Interpretada por máquinas virtuais. Caracterizada por independência de plataforma. Cada opcode tem o tamanho de um byte, bem como números diferentes de códigos de operação e está limitada a 256.

Checksum - soma de verificação é um código usado para verificar a integridade de dados transmitidos através de um canal com ruídos ou armazenados em algum meio por algum tempo.

Common Object Request Broker Architecture (CORBA) - arquitetura padrão criada pelo Object Management Group para estabelecer e simplificar a troca de dados entre sistemas distribuídos heterogêneos.

Framework (ou arcabouço, na tradução livre para o português) - significa um conjunto de conceitos usados para resolver um problema específico. Possui sua funcionalidade em comum a várias aplicações, porém para isso elas deverão ter algo trivial e desenvolvido, para que o mesmo possa atingir sua meta.

Garbage Collection – coletor de lixo, processo usado para o gerenciamento de memória nos sistemas computacionais.

Hash - Método digital para assinar digitalmente documentos.

Host – qualquer máquina ou computador conectado a uma rede.

Java Development Kit (JDK) - modelo de segurança, adotado pelas aplicações de agentes móveis, incluindo os navegadores *Web*, habilitados a executar códigos JAVA

Object Request Broker (ORB) - Sistemas distribuídos, orientados à objetos, chamados de sistema distribuído tradicional.

Proxy – tipo de servidor que atende a requisições repassando os dados a outros servidores.

Rivest Shamir Adelman (RSA) – Algoritmo para criptografia de chave pública.

Sandboxing ou **Sandbox –** forma de proteção aplicada a agentes móveis na plataforma JAVA.

Smartcard - Smart card, chip card, cartão de circuito integrado de bolso com informações pessoais, para diversos fins.

Tracing – rastreamento, no caso da computação, rastreamento de dados ou redes.