Projeto e Desenvolvimento de um Sistema Baseado em Agentes Móveis Autônomos para Detecção de Intrusos

José Duarte de Queiroz*, Luiz Fernando Rust da Costa Carmo, Luci Pirmez
{jqueiroz,rust,luci}@nce.ufrj.br

Núcleo de Computação Eletrônica – UFRJ
Cx. P. 2324 – Rio de Janeiro – RJ – CEP 20001-970 – Brasil

Resumo

Este artigo descreve um projeto de pesquisa onde se objetiva a implantação de um Sistema de Detecção de Intrusos com o uso de Agentes Móveis de Software Autônomos. Um Sistema de Detecção de Intrusos é uma ferramenta de administração de sistemas que identifica e reage às tentativas de intrusão e uso não autorizado. Esses agentes usarão mecanismos de mobilidade, possibilitando uma melhor distribuição dos recursos utilizados com degradação mínima do desempenho percebido pelos usuários.

Abstract

Here we present a research project to create and deploy an Intrusion Detection System based on Autonomous Mobile Software Agents. An Intrusion Detection System is an administration/management tool that identifies and reacts to intrusion and unauthorized use attempts. These agents will use mobility facilities, allowing an efficient use of resources, by dynamically distributing processing tasks, with a minimal degradation of the performance perceived by users.

Palavras Chave: Segurança; Agentes Móveis; Agentes de Software; Redes de Computadores.

1 Introdução

Hoje em dia é extremamente comum encontrarmos, nas empresas dos mais variados ramos e portes, computadores, ligados a redes, realizando funções essenciais e/ou armazenando informações que, de alguma forma, são cruciais para o desempenho das suas atividades-fim. Estas informações, muitas vezes, são de caráter sigiloso, ou seja, sua divulgação não autorizada pode causar sérios prejuízos para a empresa. Em outras vezes elas precisam estar disponíveis imediatamente, sem o que se tornam inúteis; já em outros casos, os recursos computacionais são escassos, por isso necessitando ter seu uso rigorosamente controlado. Seja qual for o motivo, esses sistemas estão sujeitos ao risco de serem acessados de maneira não autorizada, intencionalmente ou não, resultando, para seus usuários legítimos, em deturpação, divulgação indevida, ou impedimento de acesso.

A ligação do computador a uma rede, local ou de longo alcance, potencializa esses riscos, uma vez que o agressor tem, ao mesmo tempo, a proteção do véu do anonimato no momento do ilícito, e a facilidade de acesso proporcionada pelas ferramentas disponibilizadas pela própria rede. Para evitar que esses riscos se tornem realidade, num mundo em que cada vez mais os computadores estão interligados através de redes, é necessário que os administradores desses sistemas tenham

* Aluno do curso de Mestrado em Informática do Instituto de Matemática da UFRJ e funcionário do Ministério Público Federal – Procuradoria da República no Estado do Rio de Janeiro, lotado no Grupo de Gerência de Redes da Coordenadoria de Documentação e Informação Jurídica (CDIJ).

à sua disposição ferramentas que identifiquem e reajam às tentativas de invasão e uso não autorizado, minimizando a probabilidade de que o atacante tenha sucesso em suas tentativas.

Esse tipo de ferramenta é conhecida como IDS (do inglês *Intrusion Detection System*) e costuma ser definida como um conjunto pequeno de complexos programas monolíticos, que são executados em máquinas especializadas da rede alvo, coletando dados sobre toda a rede, ou então em cada uma das máquinas da rede, sempre causando um impacto sensível sobre a performance percebida pelo usuário final. Os conhecidos programas "Anti-Vírus" são um bom exemplo desse tipo de programa.

Será proposta aqui uma arquitetura para o desenvolvimento de um IDS que tem como principais características:

- Ser composto de várias entidades autônomas, os agentes, capazes de identificar, cada uma delas, uma pequena parcela das evidências de um ataque;
- Eliminar o ponto único de falha existente no enfoque anterior: como a detecção é feita por vários agentes, é mais difícil que a falha de um ou alguns deles impeça como um todo o funcionamento do sistema;
- Diminuir o impacto na performance dos equipamentos, já que os agentes, muito simplificados, não são tão glutões de recursos;
- Ter reconfiguração muito mais simples: se um novo risco surge, um novo agente (ou um grupo deles) pode ser desenvolvido para responder à ameaça;
- Permitir, graças à sua simplicidade, que um número muito maior de máquinas e sub-redes interligadas sejam monitoradas, aumentando o alcance e a expansibilidade do sistema.

Após esta introdução, passamos, na Seção 2, à descrição de outros trabalhos relacionados, seguida, na Seção 3, da descrição da arquitetura proposta. Encerrando, apresentamos na Seção 4 um resumo final com as conclusões obtidas até o momento para o projeto, e as referências consultadas para a elaboração do trabalho (Seção 5).

2 Trabalhos Relacionados

Um dos primeiros trabalhos a propor o uso de agentes autônomos para o desenvolvimento de Sistemas de Detecção de Intrusos foi Crosbie [2], através da distribuição do funcionamento do IDS em várias pequenas tarefas compostas de atividades simples, e entregues, uma a uma, a agentes autônomos fixos de software. Na proposta original, os agentes seriam desenvolvidos sob medida para a tarefa para que foram designados, com o auxílio de técnicas de Inteligência Artificial (Programação Genética). O desenvolvimento da proposta de [2] resultou no projeto de Zamboni [1], conhecido como arquitetura AAFID. Nessa proposta, que chegou recentemente (outubro/98) à fase de implementação, os agentes foram organizados numa estrutura hierárquica e receberam incumbências diversas, sendo que o projeto não cita o método de desenvolvimento dos agentes.

A proposta AAFID define que os agentes formem uma topologia arbitrária e hierárquica. Todos os agentes localizados dentro de uma máquina reportam seus resultados a um único *transceptor*, responsável pela operação desses agentes, tendo a possibilidade de disparar novos agentes, interrompê-los, ou enviar-lhes comandos de configuração. Os *transceptores* podem também filtrar os dados enviados pelos agentes. Os transceptores reportam-se, por sua vez, a um ou mais *monitores*. Cada monitor controla a operação de um ou mais transceptores, tendo acesso a dados acerca da rede como um todo. Estes, portanto, são capazes de extrair correlações de alto nível e detectar invasões que atinjam várias máquinas. Estes monitores também podem ser organizados hierarquicamente, de forma que um monitor reporte-se a outro monitor, ou trabalhar em paralelo, com redundância (para maior confiabilidade). Por fim, uma *Interface com o Usuário* é definida, possibilitando aos operadores acompanhar o funcionamento do Sistema e controlá-lo.

A comunicação entre os agentes AAFID é separada em dois casos: comunicação intra e inter máquina. Embora a escolha dos mecanismos seja

fundamental, [1] não fixa uma escolha em particular; pelo contrário, apenas define uma série de características básicas que os mecanismos devem apresentar para serem escolhidos. Essas características são:

- Não devem impor um *overhead* às atividades regulares dos hospedeiros;
- Devem fornecer uma expectativa razoável para a recepção das mensagens de maneira rápida e correta;
- Devem ser resistentes a tentativas de ataques (tanto por atacantes externos quanto por entidades internas) com o objetivo de tirá-los de operação por *flooding* ou sobrecarga, e (b) fornecer algum meio de autenticação e confidencialidade.

Existem várias outras pesquisas em curso que usam um enfoque parecido, que seja o de dividir a tarefa de proteger a rede ou as máquinas-alvo de invasão em pequenas tarefas simples, tais como vigiar conexões, vasculhar arquivos de log, etc; consulte [1] para ver uma lista dessas pesquisas.

3 Arquitetura Proposta

Neste trabalho propomos uma nova arquitetura, derivada da proposta original de [2] e assemelhada à arquitetura AAFID, e que será chamada, doravante, de Sistema Micæl. Em nossa proposta, porém, usamos uma divisão de tarefas nova, de forma a melhor aproveitar a característica principal de nossos agentes – a mobilidade –, que não existe nem em [2] nem na arquitetura AAFID.

3.1 Comparação entre Micæl e AAFID

A proposta de [1] é muito interessante, porém apresenta pontos que abrem margem a críticas. O AAFID é um sistema baseado em agentes fixos, desenvolvidos sob medida para as máquinas alvo. A tarefa de distribuição e carga desses agentes pelo sistema é complexa, sendo de difícil automatização. A distribuição atual dos agentes precisa ser informada a cada um deles, uma vez que eles podem precisar trocar informações. Mudar essa distribuição implica, no mínimo, em reconfigurar o sistema; como os mecanismos de comunicação são otimizados em comunicação intra e extra-máquina, pode ser até que seja necessário, no caso da mudança de localização de um agente, recompilar ou reprogramar algum módulo do sistema. O surgimento repentino de uma ameaça num determinado ponto da rede implica na reconfiguração do sistema, mesmo que essa ameaça já seja conhecida em outros pontos.

Já o Sistema Micæl tem na mobilidade dos agentes o seu diferencial; os agentes só precisam saber qual o seu destino, e deslocam-se para lá automaticamente. O sistema também pode adaptar-se a mudanças repentinas na carga de utilização, e deslocar os agentes apropriados para o local onde eles são realmente necessários, no momento da necessidade. É possível reagir rapidamente a novas ameaças, ou a uma mudança no perfil dos ataques. Com o uso da linguagem Java (ver Seção 3.3), qualquer sistema pode ser defendido por Micæl, desde que implemente a VM (*Virtual Machine*).

Outro ponto que chama a atenção no AAFID é a especialização dos mecanismos de comunicação a serem usados intra e extra máquina. Zamboni [1] afirma a necessidade dessa diferenciação com o objetivo de aumentar o desempenho do sistema. No nosso entender, porém, essa diferenciação não dá retorno satisfatório, pois além de inviabilizar o uso da mobilidade nos agentes, torna o código dos agentes mais complexo, o que não é interessante, muito embora tenhamos de reconhecer que o desempenho da comunicação, nessas condições, seja realmente maior.

Um ponto em aberto no trabalho de [1] é como os componentes controladores encontram as informações necessárias para escolher, dentre os programas agentes disponíveis, os necessários para cada tarefa específica, ou como esses controladores sabem quais informações estão em qual agente em execução; fica subentendido, do texto, que essa informação é *hard-wired* no código dos agentes, o que cria uma certa dificuldade na inser-

ção de novas informações na programação. Na nossa proposta, os agentes construirão uma base de informação (MIB) SNMP, que é uma tecnologia bem conhecida.

3.2 Elementos da Arquitetura

Nossa arquitetura divide a tarefa de detecção de intrusos e ameaças entre os principais tipos de agentes: o *quartel general*, os *sentinelas*, os *destacamentos*, os *auditores*, e os *agentes especiais*, conforme veremos a seguir.

Todas as informações reunidas pelos agentes são armazenadas em bases de informações com a mesma estrutura da MIB SNMP [13, 14]. O conteúdo de cada uma das bases é específico de cada agente, fugindo ao escopo deste documento.

3.2.1 Quartel General

O *Quartel General* (*QG*) é um agente que centraliza as funções de controle do sistema. Ele tem também a função de criar os outros agentes, mantendo por isso uma base de dados com os códigos executáveis dos agentes. Ele tem capacidade de mobilidade, sendo que essa mobilidade é usada em duas situações:

- Se a carga de utilização do hospedeiro do *QG* aumenta (por exemplo, quando um usuário se "loga" nele) a um ponto em que a execução das tarefas de controle atrapalharia as tarefas do usuário; nesse caso, o *QG* migra para uma máquina com carga menor;
- Se o hospedeiro do *QG* é invadido ou infectado, caso em que o *QG* migra para evitar que seu código seja subvertido.

Nas situações em que o QG decide por migrar para um novo hospedeiro, os agentes ativos devem ser avisados, uma vez que está previsto no ciclo de vida deles retornar ao hospedeiro do QG para registrar a experiência adquirida. Também é preciso considerar, no momento da migração, que o QG mantém uma série de bases de dados, algu-

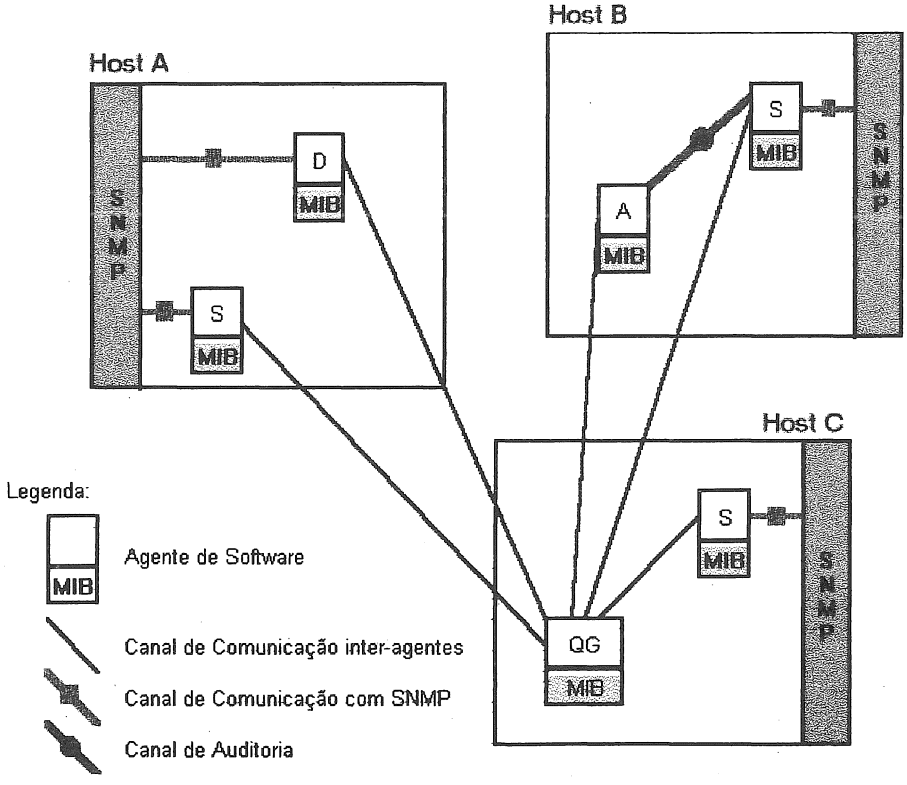

Figura 1 – Exemplo do Sistema Micœl para uma rede com três Hosts. *Cada Host executa um agente Sentinela (S); O Host A executa também um agente Destacamento (D); o Host B executa também um agente Auditor (A); e o Host C executa ainda o agente Quartel General (QG).*

mas delas em disco; essas bases devem continuar acessíveis, independentemente do hospedeiro escolhido. Convém considerar, para tanto, o uso de listas de hospedeiros possíveis e usar mecanismos semelhantes a URLs para identificar os arquivos das bases de dados.

O *QG* reúne as informações recolhidas pelos agentes, emite relatórios e compila estatísticas, sendo o ponto do sistema mais próximo ao usuário, mas não é responsável pela interface com o usuário. Essa interface é fornecida por outros sistemas, por meio de mensagens SNMP. Com essa decisão, tornamos possível controlar o Sistema Micæl por meio de qualquer visualizador de SNMP.

O *QG* não toma decisões a respeito da tarefa de detecção; essas decisões são tomadas apenas pelos *sentinelas* e *destacamentos*. Ele pode, a pedido desses agentes, disparar novos agentes ou enviar alertas para o operador. Periodicamente, o *QG* cria agentes *auditores* para verificar a integridade do sistema, como veremos à frente. Em especial, o QG não controla se o *host* ainda é seguro para execução; essa atribuição se soma às do sentinela da sede atual do QG.

O *QG* não precisa ser um agente inteligente, mas é interessante que ele seja capaz de identificar, entre as várias possibilidades de destacamentos, qual o mais apropriado para tratar da anomalia verificada pelo sentinela, numa requisição.

3.2.2 Sentinelas

Sentinelas são agentes que ficam residentes em cada um dos hospedeiros do sistema, coletando informações relevantes e informando ao *QG* sobre eventuais anomalias para registro.

Quando um *sentinela* detecta um certo nível de anomalia, ele solicita ao *QG* que seja disparado um *destacamento*, para verificar com mais detalhe a anomalia detectada. Dentre os vários códigos executáveis disponíveis, o mais apropriado para lidar com a anomalia verificada é escolhido e disparado. O ciclo de vida do sentinela pode ser descrito como a seguir:

1. O *QG* cria em seu hospedeiro um agente sentinela, que recebe ordem, em seguida, de migrar para o seu hospedeiro destino;
2. O sentinela consulta as bases de informações disponíveis em busca de anomalias ou padrões de invasão;
3. Ao encontrar uma anomalia, o sentinela solicita ao QG o envio de um destacamento, que faz uma detecção mais sofisticada e toma as providências apropriadas.
4. Periodicamente o sentinela salva seu estado no QG, prevenindo quedas repentinas do hospedeiro.
5. O processamento continua até que o sistema seja desligado (*shutdown*).

Quando o sistema recebe ordem de se desligar, o sentinela migra de volta para o hospedeiro do QG, onde grava as informações recolhidas e é terminado.

O sentinela deve possuir uma certa capacidade de aprendizado. É razoável que, no início da operação, vários alarmes falsos sejam gerados. À medida, porém, que o operador expressa sua opinião sobre os alarmes gerados, espera-se que esses alarmes falsos diminuam.

As sentinelas podem exibir um certo nível de especialização, para acomodarem-se melhor a cada ambiente objeto. Assim, espera-se que o sentinela designado para vigiar um sistema Unix seja diferente do designado para vigiar um sistema Novell ou outro, Windows, até porque as ameaças a cada um deles são diferentes. Essa especialização, porém, é restrita aos procedimentos de detecção; como veremos à frente, a independência do código executável é uma característica necessária dos programas deste sistema.

Pode acontecer, também, que se identifique a necessidade de reação rápida, antes de ser possível convocar o agente destacamento para medidas mais elaboradas; nesse caso, o sentinela deveria ser capaz de um certo nível de reação. Um exemplo dessa necessidade é o ataque SynFlood; que bloqueia a comunicação da máquina em pouco tempo (possivelmente, impedindo até mesmo que o destacamento migre para a máquina). Em tal situação o sentinela deve ter a capacidade de, por

si só, tomar atitudes para evitar o bloqueio da máquina.

3.2.3 Destacamentos

Um *destacamento* é um agente especializado que é criado para fazer frente a uma possível ameaça surgida. Quando um sentinela identifica uma anomalia, ou um padrão semelhante a uma invasão, ele solicita ao QG a criação e o envio de um destacamento ao ponto da anomalia. Este agente contém um mecanismo de detecção mais elaborado, e pode tomar medidas de defesa e contra-ataque contra a ameaça, caso ela seja confirmada.

O ciclo de vida dos destacamentos pode ser descrito como a seguir:

1. O sentinela de um hospedeiro identifica uma anomalia e recolhe as informações relevantes. Essas informações são usadas para escolher qual o código mais apropriado para o destacamento.
2. O QG cria, com o código escolhido, um novo agente destacamento em seu hospedeiro, que recebe ordem de migrar para o hospedeiro sob ameaça.
3. Ao ativar-se no hospedeiro ameaçado, o destacamento inicia uma avaliação sobre a real situação, podendo decidir pela confirmação ou negação da ameaça. Esta decisão é repassada ao sentinela local, para futura referência, e ao QG, para registro.
4. Confirmada a ameaça, o destacamento inicia as contramedidas. Estas podem incluir desinfecção de programas, encerramento forçado de sessões de usuário, desligamento da máquina (*shutdown*) ou até mesmo contra-ataques para bloqueio da máquina agressora (no caso de agressões oriundas de fora).
5. Quando a ameaça é controlada, e o nível de alerta retorna ao normal, o destacamento migra de volta ao hospedeiro do QG, grava seu estado para referência futura, e se desliga.

Pode acontecer do destacamento decidir que não é apropriado para lidar com uma ameaça em potencial, e solicitar a criação de outro destacamento. Vários destacamentos podem estar em ação simultaneamente num mesmo hospedeiro. Um nível máximo de carga deve ser estabelecido, de forma que o(s) usuário(s) das máquinas sob ameaça recebam o serviço mínimo definido pelo administrador do sistema.

Os destacamentos devem contar com um nível alto de inteligência, mas o aprendizado pode ser feito *off-line*. Como este tipo de agente tem vida curta (ele só existe enquanto existir a ameaça), não há tempo de aplicar a experiência adquirida numa ativação. Essa experiência pode ser reunida e compilada, gerando novas versões dos destacamentos.

3.2.4 Auditores

Para evitar que agentes cujo código seja subvertido prejudiquem a segurança do sistema, Micæl conta com agentes auditores, que são disparados periodicamente com o objetivo de conferir a perfeita execução dos agentes ativos.

O auditor é o único agente, além do QG, que tem a permissão de criar novos agentes; ele usa essa permissão para recriar o QG, caso este aborte suas funções por qualquer motivo. Caso o auditor perceba, em qualquer hospedeiro, que a sentinela não está ativa, ele solicita ao QG a criação de uma nova sentinela.

Outra peculiaridade do auditor é que ele não usa os meios convencionais de comunicação inter-agente nas suas tarefas; ao contrário, ele usa para isso uma API (*Advanced Program Interface*) de auditoria, que é parte obrigatória de todos os agentes de Micæl.

O auditor não necessita de nenhuma facilidade de inteligência ou de aprendizado; sua tarefa é totalmente automática. Para verificar a integridade, ele se utiliza de tabelas de *check-sums* internos pré-compiladas.

O ciclo de vida do auditor pode ser descrito como a seguir:

1. QG cria o auditor.
2. O auditor conecta-se à API de cada um dos agentes ativos no hospedeiro atual e verifica seus estados.

3. O auditor migra para o próximo hospedeiro no sistema, até que todos os hospedeiros tenham sido auditados, e o auditor esteja de volta ao hospedeiro do QG.

Caso o auditor não encontre o agente que ele deseja auditar, ele conclui que esse agente abortou sua operação, e solicita ao QG que esse agente seja recriado. Caso o agente abortado seja o próprio QG, o auditor toma a iniciativa de recriá-lo, no mesmo hospedeiro em que ele esperava encontrá-lo.

3.2.5 Agentes Especiais

Além dos descritos acima, o Sistema Micæl prevê a existência de agentes especiais, que realizam outras tarefas. Um deles é o agente de controle e monitoração de rede.

Pode ser necessário permitir ao usuário final, ou ao administrador do sistema alvo, desenvolver seus próprios agentes. Para isso, ele deve se basear em modelos de agentes e em bibliotecas de ações predefinidas.

A documentação do SNMP [12,20,8] chama a atenção para o fato de que, apenas com sua infra-estrutura básica, não é possível reunir informações relativas à rede. Por isso, foi criado o RMON, que é uma infra-estrutura para o monitoramento da rede em si. O RMON usa equipamentos especiais, chamados de *probes*, para recolher dados diretamente da rede (em especial, de redes de difusão) e compilá-los numa MIB. Da mesma forma, também o Sistema Micæl é incapaz de acompanhar, de maneira eficiente, o que se passa no barramento da rede. Assim, um agente especial, chamado de monitor de rede, pode ser especificado. Esse agente trabalha em parceria com os *probes* RMON, podendo descobrir ataques de bloqueio da rede (*flooding*, *spoofing*, DoS[1], etc.) e falhas de funcionamento que coloquem em risco a rede (por exemplo, uma placa de rede monopolizando o barramento).

[1] DoS: *Denial of Service*. Nome genérico para ataques que resultam em privação de serviço.

3.3 Mobilidade

A mobilidade dos agentes dá ao sistema Micæl o diferencial de tecnologia que o caracteriza como projeto inovador. É nossa intenção acrescentar mecanismos de mobilidade aos agentes. Essa mobilidade seria útil em várias situações, tais como:

- De tempos em tempos, é necessário auditar os agentes do sistema; alocar um módulo de auditoria neles implicaria em gasto de recursos, além de tornar os agentes mais complexos. Um agente de auditoria móvel pode auditar cada uma das máquinas defendidas, em seqüência, sem sobrecarregar nenhuma delas.
- Ao encontrar um padrão de anormalidade, um agente pode solicitar uma investigação mais detalhada. Essa investigação é conduzida por um novo agente, que se desloca para a máquina atacada.
- Um agente móvel pode rastrear com facilidade ataques do tipo "verme", em que o invasor passa rapidamente de uma máquina para a outra da rede.
- A carga de processamento pode ser dinamicamente distribuída entre as máquinas defendidas.

Em suma: esperamos, com a mobilidade, poder manter o sistema gastando o mínimo de recursos, nos casos normais, e concentrando o máximo de recursos no lugar mais apropriado, nos casos de tentativas de invasão.

Várias infra-estruturas estão disponíveis para o oferecimento de mecanismos de mobilidade para os agentes. Endler [6] cita várias dessas alternativas; escolhemos, preliminarmente, usar a infra-estrutura ASDK [7,19]. Isso implica em implantar essa infra-estrutura em cada uma das máquinas alvo, independentemente do ambiente operacional suportado nelas. Algumas dificuldades podem surgir:

- ASDK é baseado em Java; a máquina alvo deve, então, suportar a execução de programas Java.

- O sistema operacional da máquina alvo deve ser capaz de aceitar novas bibliotecas de rotinas e protocolos, para acomodar as necessidades do ASDK.

O ambiente ASDK (*Aglets Software Development Kit*) foi desenvolvido pela IBM para prover mecanismos de mobilidade a programas. Ele é escrito em Java e inclui primitivas para a criação, movimentação, comunicação e encerramento de programas. Um programa móvel no ASDK é conhecido como um *aglet* (contração de *agent* + *applet*). O *aglet* migra de uma máquina para a outra com o auxílio de um módulo servidor, conhecido como Servidor de Agentes (*aglets server*).

Para se deslocar de uma máquina para outra, o *aglet* entra em contato com o *aglets server* da máquina destino, numa porta TCP predeterminada, e, após autorização, inicia o processo de transferência de seu código e estado (que caracterizam a sua identidade), liberando a máquina origem de seu processamento em seguida. Os *aglets* usam um protocolo próprio para fazer essa movimentação, conhecido como ATP (*Aglet Transfer Protocol*).

Devido ao seu perfil invasivo, o *aglet* é por si só um problema de segurança: um programa capaz de, sem a interferência do usuário, ser executado em várias máquinas, guardando as informações reunidas em suas localizações anteriores. Para minimizar este risco, os *aglets* funcionam conforme a definição da estrutura de segurança do JDK 1.2 [17]. O modelo de segurança do JDK (*Java Developers Kit*) versão 1.2 expande o modelo originalmente usado nas versões iniciais, conhecido como "sandbox" [17]. Nesse modelo, o código Java é dividido em duas classes de segurança: os códigos confiáveis, obtidos da máquina local, e os não confiáveis, que são obtidos da rede de maneira geral. O código confiável tem acesso a toda a máquina, enquanto o código externo vê apenas um sub-conjunto muito restrito de recursos – a chamada "sandbox". A versão 1.1 expandiu esse modelo para permitir que *applets* especiais pudessem ter acesso a partes dos recursos da máquina local, através de assinaturas digitais (certificados).

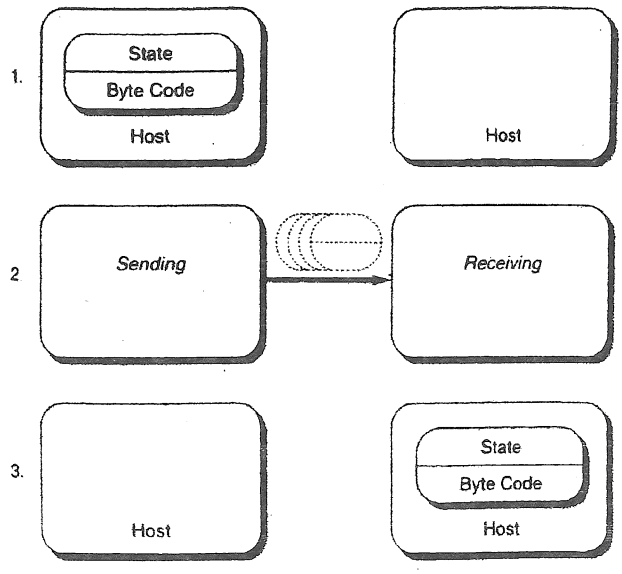

Figura 2 – Processo de Transferência de um agente entre dois Hosts (extraído de [19])

No JDK 1.2, porém, o administrador da maquina alvo tem o poder de determinar, para cada módulo, qual o nível de acesso admitido a cada recurso da máquina local. Essa facilidade, mais do que "um quê a mais", é na verdade essencial para possibilitar o funcionamento dos agentes do sistema Micæl. Como parte da implementação do sistema Micæl, a infra-estrutura básica do ASDK deverá ser instalada em cada máquina alvo. Isso inclui o *aglets server* e toda a biblioteca básica. Uma grande vantagem que se obtém do uso dessa infra-estrutura é que várias das características tidas como necessárias para os módulos componentes dos agentes estão presentes intrinsecamente no ASDK (p.ex., comunicações seguras, *check-sum* interno do código, etc.).

No modelo do ASDK estão definidas as seguintes primitivas de mobilidade: **criação** (create), **movimentação** (dispatch), **retorno** (retract) e **liberação** (dispose). A cada uma delas corresponde um método do objeto Aglet (classe que define o módulo executável do programa ASDK, semelhante à classe *Applet* do JDK). Os *aglets* existem dentro de um **contexto** definido dentro do *host*, sendo que um *host* pode conter vários contextos, e os *aglets* se movimentam de um contexto para o outro. A movimentação dos objetos *Aglet* é feita

pela serialização do seu conteúdo. Em teoria, qualquer objeto Java pode ser utilizado dentro do *aglet*, mas na prática apenas os objetos que aceitem serialização são utilizáveis.

Para possibilitar a comunicação entre os *aglets* de maneira independente da sua localização atual, o ASDK define o conceito do objeto *Proxy*. Um *Proxy* é criado junto com o *aglet*, e mantém referência da sua localização atual, permitindo o contato entre o módulo criador ("pai") e o criado ("filho"), independentemente de sua localização atual. Primitivas de comunicação também estão presentes, possibilitando o envio de mensagens de maneira síncrona ou assíncrona. O objeto *proxy* apresenta algumas dificuldades de uso; para contorná-las, definimos o objeto *Relay*, que será visto mais adiante.

3.4 Comunicação entre Agentes

A comunicação entre os agentes Micæl é feita através de mensagens ATP, que incorporam mecanismos de autenticação e privacidade como é desejável num sistema deste tipo.

Todas as mensagens entre agentes devem ser criptografadas e autenticadas com o auxílio de *time-stamps*. Com isso, esperamos evitar que intrusos interfiram na comunicação entre os agentes. Será usada criptografia de chave assimétrica (chave pública), sendo que cada agente tem seu próprio par de chaves. Essas chaves são atribuídas aos agentes pelo QG, a partir de uma lista de pares de chaves.

A auditoria é um capítulo à parte na comunicação. O auditor e o auditado compartilham uma relação muito estreita, podendo se considerar que, durante a auditoria, o auditado se transforma num módulo de dados do auditor. Assim, as interações entre eles pode ser consideradas como interações de um único programa. Esta decisão também aumenta a robustez do processo de auditoria, ao tornar mais difícil a interferência de entidades externas nessas comunicações. Assim, não é necessário usar esquemas pesados de autenticação (p.ex. criptografia) neste nível; um esquema simples desafio-resposta é suficiente.

Todos os agentes auditáveis devem fornecer uma API (*Advanced Programming Interface*), com as seguintes funções, no mínimo:

- **Identificação:** O auditor identifica-se para o agente auditado, e pede que este identifique-se. A identificação positiva libera as outras funções da auditoria.
- **Verificação de Integridade:** O auditor solicita ao auditado que calcule o check-sum de seu código e retorne. Serve para determinar se o código do agente não foi subvertido.
- **Controle de Execução:** O auditor pode determinar que o auditado aborte sua operação, ou mesmo fazê-lo à força, se concluir que seu código foi subvertido.

3.4.1 Objeto Relay

Um problema enfrentado com os objetos proxy é que eles só são válidos enquanto o agente re-

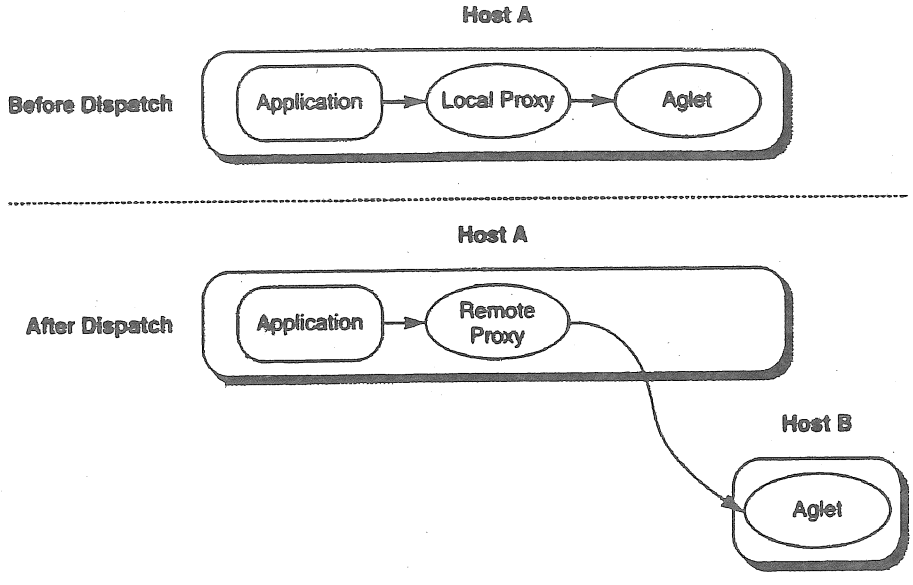

Figura 3 – Troca de mensagens entre dois aglets *por meio do objeto* proxy. *(Extraído de [19])*

ferenciado não se move. Também as referências guardadas por um agente tornam-se inválidas se ele se movem, visto que referências que eram locais podem tornar-se remotas e vice-versa. Em nossa arquitetura podem acontecer casos em que tanto a origem como o destino de uma mensagem podem ter se movido da posição inicial (por exemplo, uma mensagem do auditor para o QG). Para lidar com essa dificuldade, criamos um objeto que chamamos *relay*. Esse objeto conecta todos os *hosts* defendidos pelo sistema por meio de uma comunidade *multicast*, e, com a ajuda da identificação única presente em cada *aglet*, direciona as mensagens para o ponto apropriado. Essa solução também simplifica os agentes, uma vez que eles não necessitam mais saber onde está o agente destino da mensagem mas apenas sua identificação única.

3.5 Ambiente Objeto

É nossa intenção que o Sistema Micæl seja executado num ambiente misto de Unix, MS-DOS/Windows® e Novell Netware®. Estes três ambientes foram escolhidos porque representam a maior parte dos Sistemas Operacionais em uso no mercado brasileiro, sendo extremamente comum encontrá-los convivendo dentro de uma organização. Também é nossa intenção, porém como um prosseguimento futuro do trabalho, estender Micæl para o ambiente Windows NT®.

Outro motivo para a escolha desse ambiente objeto é a pouca integração entre as ferramentas de gerenciamento existente entre eles hoje em dia. Grande parte das Empresas que utilizam essas combinações de sistemas operacionais contam com uma equipe especializada de suporte para cada um dos ambientes, e quase todas interações são feitas manualmente. No produto que objetivamos desenvolver essa integração será incentivada, tendência que é francamente crescente hoje em dia.

O ambiente misto foi uma das razões para a escolha de Java como a linguagem para desenvolvimento. Várias alternativas foram consideradas, cada uma com seus prós e contras:

- **Uso de código objeto de cada máquina alvo:** A pior das soluções, por impedir o uso de mobilidade. Para cada ambiente utilizado, seria necessário desenvolver uma versão de cada agente. Em compensação, seria a que resultaria na maior velocidade de execução.
- **Uso de linguagens interpretadas (perl, tcl, shell scripts):** Uma solução boa do ponto de vista de compatibilidade entre as implementações, porém muito difícil de implementar na prática, uma vez que nem todas as plataformas estão preparadas para cada uma das alternativas. Seria também a de pior desempenho, além de não fornecer (com exceção de **perl**) mecanismos para acesso a informações de baixo nível.
- **Linguagem Java:** O requisito multi-plataforma indica fortemente o uso de Java, que, mais do que uma linguagem, é todo um ambiente de programação multi-plataforma. A disponibilidade de bibliotecas com facilidades de mobilidade foi o *cup-de-grace* para a escolha.

4 Considerações finais

Apresentamos aqui a arquitetura de um Sistema de Detecção e Proteção contra Intrusos e atividades agressivas, que foi apelidado de Sistema Micæl. O Sistema Micæl desempenha suas tarefas por meio de Agentes Autônomos distribuídos de Software, com o auxílio de primitivas de mobilidade, de forma a proteger, com um uso mínimo de recursos, as máquinas localizadas numa determinada área de uma rede de computadores, e, por tabela, a própria rede.

Os agentes de Micæl são classificados conforme as atividades desempenhadas, e se comunicam através de primitivas SNMP, mesma facilidade usada para acessar as informações necessárias para controlar e detectar invasões. Esses agentes são auditados periodicamente, garantindo sua correção e integridade. Os agentes são escritos em Java, o que possibilita o uso de plataformas mistas, o que é um de nossos objetivos básicos. A infra estrutura de mobilidade é fornecida pelo am-

biente ASDK, o que, em conjunto com o JDK 1.2, fornece as facilidades de privacidade e autenticação essenciais a um sistema como este.

O Sistema Micæl será útil não só como produto de segurança em rede, mas também como bancada de pesquisa de várias tecnologias, em especial de agentes móveis, campo tão fértil quanto inexplorado. Uma de suas maiores contribuições é integrar essas tecnologias com uma área de aplicação onde a necessidade de novas ferramentas é constante. Outro ponto que não deve ser desprezado é o caráter modular da arquitetura, que facilita o processo de produção de novos agentes.

Um protótipo muito simplificado do sistema foi desenvolvido numa fase anterior do projeto e obteve resultados muito promissores. Atualmente estamos trabalhando na implantação da infra-estrutura de detecção e comunicação entre os agentes, em especial no objeto *relay*. Serão feitas também algumas medições para avaliar a qualidade desta solução, que supomos extremamente genérica e reaproveitável.

Alguns pontos foram deixados de lado por motivo de simplicidade e adequação ao tempo disponível. Um bom exemplo está nos mecanismos de inteligência e aprendizado. Vários dos agentes do sistema necessitam ter uma certa capacidade de inteligência e/ou de aprendizado. Os motivos dessa necessidade são vários:

- Através do aprendizado, o número de alarmes falsos pode ser minimizado;
- O funcionamento do sistema pode ser ajustado às necessidades do usuário, sem incluir funções altamente elaboradas, mas de pouca aplicação;
- Apenas as funções úteis são carregadas, deixando o sistema mais livre;
- Novas ameaças podem ser identificadas e incluídas, sem recompilação ou reprogramação.

Crosbie [2] defende que os agentes de software sejam dotados de inteligência usando programação genética. Essa proposta, embora interessante, choca com dois dos objetivos do nosso trabalho: o uso de mobilidade com auxílio do ASDK (que pressupõe o uso de Java), e o ambiente Unix/Windows/Novell. Assim, este ponto foi deixado de lado por enquanto, para ser atacado mais tarde.

5 Referências

[1] Zamboni, D., J.S. Balasubramaniyan, J.O. Garcia-Fernandez, D. Isacoff, E.H. Spafford. *An Architecture for Intrusion Detection using Autonomous Agents*. COAST Technical Report. 98/05. COAST Laboratory – Purdue University. Junho/1998. Disponível por FTP anônimo.

[2] Crosbie, M., E.H. Spafford. *Defending a Computer System Using Autonomous Agents*. COAST Technical Report 95/22. COAST Laboratory – Purdue University. Março/1994. Disponível por FTP anônimo.

[3] Staniford-Chen, S. *et al. GrIDS – A Graph Based Intrusion Detection System for Large Networks*. UC/Davis.

[4] Hardarker, W. et al. *CIDF & SNMP*. Apresentação para o Encontro DARPA/CIDF na UC/Davis. Junho de 1998.

[5] Stamatelopoulos, F., G. Koutepas, B. Maglaris. *System Security Management via SNMP*. Universidade Técnica Nacional de Atenas. Apresentado no *HP OpenView University Association Workshop*, Abril de 1997.

[6] Endler, Markus. *Novos Paradigmas de Interação usando Agentes Móveis*. IME/USP. 1998.

[7] Oshima, M., G. Karjoth. *Aglets Specification Version 1.0 (Draft)*. IBM, Abril de 1998.

[8] Rose, Marshal T., *The Simple Book – An Introduction to Internet Management*. 2nd Ed. Prentice-Hall Intl. 1994.

[9] Comer, Douglas E., *Internetworking with TCP/IP – Vol. 1: Principles, Protocols and Architecture*. 3rd Ed. Prentice-Hall Intl. 1995.

[10] Tanenbaum, A. S., _Redes de Computadores. Tradução da 3ª Edição Americana. Ed. Campus, 1997.

[11] Stallings, William, *SNMP, SNMPv2 and RMON – Pratical Network Management*. 2nd Ed. Addison-Wesley Publ. 1996.

[12] Rose, M. T., and K. McCloghrie, *Structure and Identification of Management Information for TCP/IP-based Internets*, STD 16, RFC 1155, Maio de 1990.

[13] McCloghrie, K., and M. Rose, *Management Information Base for Network Management of TCP/IP-based Internets*, RFC 1156, Maio de 1990.

[14] Case, J., M. Fedor, M. Schoffstall, and J. Davin, *The Simple Network Management Protocol*, STD 15, RFC 1157, Maio de 1990.

[15] Case, J. K. McCloghrie, M. Rose, S. Waldbusser, *Protocol Operations for version 2 of the Simple Network Management Protocol (SNMPv2)*, RFC 1905, Janeiro de 1996.

[16] Blumenthal, U., B. Wijnen, *User-based Security Model (USM) for version 3 of the Simple Network Management Protocol (SNMPv3)*, RFC 2274, Janeiro de 1998.

[17] Gong, L., *Java Security Architecture (JDK 1.2)*, Versão 1.0, Outubro de 1998.

[19] Lange, D. B., M. Oshima, O. Mitsuru, *Programming and Deploying Java Mobile Agents With Aglets*, Addison-Wesley Publ. Co., Agosto de 1998.

[20] Waldbusser, S., *Remote Network Monitoring Management Information Base*, RFC 1757, Fevereiro de 1995.

Endereços Eletrônicos para Obtenção de Documentos

AAFID:
ftp://coast.cs.purdue.edu/pub/COAST/diego-zamboni/ zamboni9805.ps

ftp://coast.cs.purdue.edu/pub/doc/intrusion_detection/ mcrosbie-spaf-NISC-paper.ps.Z

Aglets Specification:
http://www.trl.ibm.com/documents.html

Java Security:
http://java.sun.com/products/jdk/1.2/docs/guide/security/spec/ security-spec.doc.html

Página do PGP Internacional na Unicamp
http://www.dca.fee.unicamp.br/pgp