UNIVERSIDADE FEDERAL DO CEARÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

VINÍCIUS DA SILVA THIAGO

ARQUITETURA MULTI-AGENTES PARA DETECÇÃO DISTRIBUÍDA DE INTRUSÃO

FORTALEZA, CEARÁ

2012

VINÍCIUS DA SILVA THIAGO

ARQUITETURA MULTI-AGENTES PARA DETECÇÃO DISTRIBUÍDA DE INTRUSÃO

Dissertação submetida à Coordenação do Programa de Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientador: Prof. Dr. José Neuman de Souza

FORTALEZA, CEARÁ

2012

T422a THIAGO, V. S..

Arquitetura Multi-agentes para Detecção Distribuída de Intrusão / Vinícius da Silva Thiago. 2012.

102p.;il. color. enc.

Orientador: Prof. Dr. José Neuman de Souza

Co-Orientador:

Dissertação(Ciência da Computação) - Universidade Federal do Ceará, Pró-reitoria de Pesquisa e Pós-graduação, Fortaleza, 2012.

1. Detecção 2. Intrusão 3. Agentes I. Prof. Dr. José Neuman de Souza(Orient.) II. Universidade Federal do Ceará– Ciência da Computação(Mestrado) III. Mestre

CDD:004.6

VINÍCIUS DA SILVA THIAGO

ARQUITETURA MULTI-AGENTES PARA DETECÇÃO DISTRIBUÍDA DE INTRUSÃO

Dissertação submetida à Coordenação do Programa de Mestrado e Doutorado em Ciência da Computação, da Universidade Federal do Ceará, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação. Área de concentração: Ciência da Computação

Aprovada em: __/__/____

BANCA EXAMINADORA

Prof. Dr. José Neuman de Souza
Universidade Federal do Ceará - UFC
Orientador

Prof. Dr. Miguel Franklin de Castro
Universidade Federal do Ceará - UFC

Prof. Dr. Joberto S. B. Martins
Universidade Salvador - UNIFACS

Aos meus avós Osmar e José, *in memorian*, que com amor e dedicação ajudaram a formar a pessoa que sou hoje.

AGRADECIMENTOS

Meus sinceros agradecimentos:

- ao professor Dr. José Neuman de Souza, pela orientação e pelos conselhos;
- à minha família, pela minha criação, apoio, amor e incentivo durante toda a minha vida;
- à minha namorada, pelo amor, carinho e incentivo nas horas difíceis;
- ao meu sobrinho Enzo, pela alegria que me contagia e ajuda a superar as dificuldades;
- aos meus amigos do trabalho, pelos conselhos, cumplicidade e apoio, essenciais na realização deste trabalho.

“Inventar é imaginar o que ninguém pensou;
é acreditar no que ninguém jurou;
é arriscar o que ninguém ousou;
é realizar o que ninguém tentou.
Inventar é transcender.”

(Alberto Santos Dumont)

RESUMO

A crescente preocupação com a segurança da informação em redes de computadores é responsável por produzir constantemente novas formas de defender as mesmas. Dentro desse contexto, o desenvolvimento de novas formas de detecção de intrusão assume um papel muito importante na proteção das informações. Os sistemas de detecção de intrusão precisam ser eficientes e ao mesmo tempo não devem sobrecarregar a rede ou a capacidade de processamento dos nós que a compõem. Com o objetivo de ser eficiente, um sistema deve basear as suas decisões em tantas fontes de informação quanto forem possíveis e organizar o conhecimento de forma que permita uma comunicação funcional entre essas fontes. Este trabalho descreve a proposta de uma arquitetura de um Sistema de Detecção de Intrusão Distribuído que utiliza agentes móveis e uma ontologia para o compartilhamento da informação. Os agentes móveis proporcionam uma maneira prática de distribuir o processo de detecção, possibilitando cooperação ponto a ponto entre os nós da rede sem gerar muito tráfego adicional. A ontologia fornece uma maneira organizada de armazenar e compartilhar o conhecimento. A arquitetura proposta foi implementada utilizando a linguagem de programação Java e o *framework* JADE e foi montado um laboratório de testes para verificar o funcionamento do sistema. Os resultados obtidos com os testes confirmaram que uma arquitetura distribuída multi-agentes que faz uso de uma ontologia pode ser eficiente na detecção de ataques a redes e sistemas.

Palavras-chave: Detecção. Intrusão. Agentes.

ABSTRACT

The growing concern about information security in computer networks is responsible for constantly producing new ways to defend them. Within this context, the development of new ways of intrusion detection plays an important role in protecting the information. Detection systems must be efficient and, at the same time, must not overload the network or the processing capabilities of the nodes within it. In order to be effective, a system must base its decisions on as many sources of information as possible and organize knowledge in a way that allows a functional communication between those sources. This dissertation describes the proposal for a Distributed Intrusion Detection System architecture that uses mobile agents and an ontology for information sharing. Mobile agents provide a convenient way to distribute the detection process, enabling peer to peer cooperation between network nodes without generating much additional traffic. The ontology provides an organized way of storing and sharing knowledge. The proposed architecture has been implemented using the Java programming language and JADE framework and a test laboratory has been assembled to verify the operation of the system. The tests results confirmed that a distributed multi-agent architecture that uses an ontology can be effective in detecting attacks on networks and systems.

Keywords: Detection. Intrusion. Agents.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

LISTA DE SIGLAS

ACC	*Agent Communication Channel*
ACL	*Agent Communication Language*
AMS	*Agent Management System*
ARP	*Address Resolution Protocol*
CPU	*Central Processing Unit*
CRM	*Content Reference Model*
DAML	DARPA *Agent Markup Language*
DARPA	*Defense Advanced Research Projects Agency*
DDoS	*Distributed Denial of Service*
DF	*Directory Facilitator*
DoS	*Denial of Service*
FIPA	*Foundation for Intelligent Physical Agents*
HIDS	*Host-based Intrusion Detection System*
HTTP	*Hypertext Transfer Protocol*
ICMP	*Internet Control Message Protocol*
IDS	*Intrusion Detection System*
IEEE	*Institute of Electrical and Electronics Engineers*
IP	*Internet Protocol*
IPS	*Intrusion Prevention System*
IRC	*Internet Relay Chat*
IRE	*Identifying Referential Expressions*
JADE	*Java Agent DEvelopment Framework*
KQML	*Knowledge Query Meta Language*
LOIC	*Low Orbit Ion Cannon*
LPGL	*Lesser General Public License*
MASIF	*Mobile Agent System Interoperability Facility*
MITM	*man in the middle*
NIDS	*Network-based Intrusion Detection System*
OIL	*Ontology Interchange Language*
OWL	*Web Ontology Language*
P2P	*peer-to-peer*
RARP	*Reverse Address Resolution Protocol*
RDF	*Resource Description Framework*
RMI	*Remote Method Invocation*
RPC	*Remote Procedure Calls*
R2U	*remote to user*
TCP	*Transmission Control Protocol*

UDP	*User Datagram Protocol*
U2R	*user to root*
VPN	*Virtual Private Network*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

1 INTRODUÇÃO

Esta dissertação propõe uma arquitetura multi-agentes para detecção de intrusão distribuída. Esta arquitetura, que utilizará agentes no trabalho de detecção e uma ontologia para o compartilhamento de informação entre os nós, visa prover uma maneira eficiente de detectar ataques e elementos maliciosos na rede por meio da cooperação e colaboração entre os nós.

Este capítulo é organizado de forma que na seção 1.1 é discutido o contexto de detecção de intrusão e da utilização de agentes para esse fim. A seção 1.2 apresenta a motivação e problemática do trabalho. A seção 1.3 apresenta os objetivos, a seção 1.4 apresenta trabalhos relacionados, e por fim, a seção 1.5 explica como o restante da dissertação está organizada.

1.1 Contextualização

O crescimento do número de aplicações de redes de computadores faz com que se torne cada dia maior a diversidade e a sofisticação dos ataques e métodos de intrusão, aumentando a preocupação com a segurança das mesmas. Segundo Martins (2006), o conceito de segurança está intimamente ligado a uma intenção maliciosa, como a invasão de redes, e a finalidade da segurança é evitar ataques desse tipo, protegendo as informações e os serviços oferecidos por um sistema. A detecção de intrusão é uma das técnicas chave para proteger as redes contra intrusos e se baseia em coletar e armazenar dados para auditoria de sistemas e redes. De acordo com Puttini et al. (2004), quando detectada, uma intrusão deve ser reportada para o gerente de segurança, ou ainda, uma resposta automática, com o objetivo de eliminar as causas e/ou os efeitos da intrusão, pode ser disparada.

Um Sistema de Detecção de Intrusão (*Intrusion Detection System*, ou IDS) é um sistema que tenta detectar e alertar as tentativas de intrusão em um sistema ou rede, nos quais uma intrusão é considerada como sendo uma atividade não autorizada ou não desejada. Esses sistemas geralmente monitoram atividades dos usuários, dos sistemas e das redes e, normalmente, possuem uma entidade central encarregada de tomar as decisões (KACHIRSKI; GUHA, 2003). Os IDSs centralizados funcionam em um hardware na rede, de modo que possam coletar dados de cada um dos *hosts* e depois analisá-los. Assim, é decidido se está ou não acontecendo uma intrusão e o que deve ser feito. No entanto, a centralização se transforma em um grande ponto fraco, visto que se o *hardware* parar de funcionar, a rede toda fica desprotegida, além do fato de que o analisador central pode facilmente se tornar um gargalo para a rede.

As arquiteturas de detecção distribuídas são mais eficazes e podem remediar os problemas das arquiteturas centralizadas. Quanto mais fontes de informação forem utilizadas para assegurar a detecção de intrusão, mais precisa deve ser essa detecção. O problema da centralização dos dados pode ser superado pelos sistemas *peer-to-peer* (P2P), que são um tipo

de sistema distribuído sem nenhuma autoridade ou gerenciamento central. Nesses sistemas, a busca por um provedor de serviços é reduzida à consulta aos *hosts* por aquele que provê o tipo de serviço requerido (SREENATH; SINGH, 2004). O problema principal enfrentado pelas arquiteturas distribuídas é como coletar e correlacionar informações e, em seguida, avaliar o estado de segurança do sistema monitorado.

1.2 Caracterização do Problema e Motivação

A arquitetura de um IDS deve ser simples e eficaz para prover a segurança contra os diferentes tipos de ataques. De acordo com Ahmed, Samad e Mahmood (2006), uma solução eficiente é se defender contra intrusões de forma cooperativa, ao invés de cada nó realizar análises completas do tráfego passando por ele, e, se possível, possibilitar a punição de entidades que apresentaram comportamento faltoso ou malicioso (exclusão das mesmas da rede, por exemplo) melhorando as condições de segurança. O problema consiste então em como montar uma arquitetura distribuída que seja robusta para suportar ataques à própria estrutura do IDS, possibilitar o compartilhamento de dados entre os nós da rede sem criar um *overhead* exagerado no tráfego e evitar a criação de possíveis gargalos.

Uma solução é utilizar arquiteturas hierarquizadas, por exemplo, dividindo o IDS em nós de controle, nós intermediários e nós sensores, o que causa uma grande quantidade de dados transferidos pela rede e pode causar congestionamento (BRAHMI; YAHIA; PONCELET, 2010). As arquiteturas hierarquizadas também tornam os IDS susceptíveis a ataques, pois um atacante pode interromper o funcionamento do sitema atacando um nó intermediário, ou ainda, atacar os nós de controle, que gerenciam o funcionamento. Existem formas de garantir o funcionamento constante, como a redundância, a mobilidade e a recuperação dinâmica.

No caso de um IDS P2P, cada *host* pode enviar pedidos de detecção para outros *hosts* do sistema para checar se existem atividades suspeitas e estimar se a rede foi invadida, sem o ponto fraco de um controlador central. No entanto, muitos sistemas deste tipo só permitem que os nós da rede obtenham informações de fontes limitadas, como vizinhos diretamente ligados, o que pode levar à tomada de decisões imprecisas, principalmente no caso de ataques a múltiplos *hosts* (YE et al., 2008). Ataques a múltiplos *hosts* são aqueles em que um atacante tenta atacar vários nós de uma rede simultaneamente. A característica mais importante desses tipos de ataques é que o nível de atividade do atacante em cada um dos nós pode não ser suficientemente grande para levantar um alarme para toda a rede. No entanto, se um IDS distribuído puder colher e analisar informações correlacionadas de múltiplos *hosts*, pode ser possível que ele reconheça o ataque.

Uma das formas de se implementar um IDS distribuído é por meio do uso de agentes móveis. Agentes móveis são agentes inteligentes que podem migrar de um *host* para outro. De

acordo com Ye et al. (2008), eles podem executar tarefas de maneira autonôma em ambientes dinâmicos, e assim, podem viajar entre os nós da rede e detectar intrusões no sistema. Um *host* pode enviar agentes móveis para outros com o intuito de coletar informações relevantes e reconhecer um ataque a múltiplos *hosts*. Segundo Brahmi, Yahia e Poncelet (2010) e Jansen (2002), as vantagens da tecnologia de agentes móveis incluem: reduzir o *overhead* da rede, superação do problema de atraso na rede, execução assíncrona e autônoma, enrobustecimento do sistema e tolerância a falhas, escalabilidade do sistema e operação em ambientes heterogênios.

As arquiteturas P2P podem, assim, diminuir a quantidade de processamento em cada *host* utilizando-se de agentes. Vários agentes executando um mecanismo de colaboração podem obter um estado global do sistema protegido, através do compartilhamento de dados relacionados à detecção.

Um dos grandes objetivos no aperfeiçoamento dos IDS envolve a redução do número de falsos positivos e o consequente aumento da confiabilidade. Métodos que envolvem inteligência artificial e representação do conhecimento em ontologias foram sugeridos, tendo a utilização de ontologias em segurança de redes e computadores se tornado uma direção promissora no aperfeiçamento dos sistemas. Segundo Abdoli e Kahani (2009), utilizando ontologias consegue-se uma ferramenta de classificação forte para um número ilimitado de eventos.

1.3 Objetivos

O objetivo deste trabalho é propor uma arquitetura distribuída que seja robusta para suportar ataques à própria estrutura do IDS, possibilitar o compartilhamento de dados entre os nós da rede sem criar um *overhead* no tráfego e evitar a criação de possíveis gargalos. A arquitetura, que utilizará agentes fixos e móveis para realizar o trabalho de detecção, utilizará ainda uma ontologia para o compartilhamento de conhecimento entre os nós da rede. Posteriormente ela será avaliada de acordo com resultados de detecções obtidos em testes em um laboratório.

1.4 Trabalhos Relacionados

O trabalho apresentado por Ye et al. (2008) mostrou uma proposta de um IDS P2P que utiliza agentes móveis com a intenção de conseguir um menor *overhead* de processamento nos *hosts*, reduzir o risco de uma arquitetura centralizada e conseguir detecções mais precisas. A arquitetura proposta, similar com a proposta deste trabalho, é composta por seis tipo de agentes: agente monitor, agente analisador, agente executor, agente gerente, agente de recuperação e agente de resultado. A arquitetura não conta com uma ontologia, sendo o contato direto entre os agentes a única forma de trocar informações sobre as intrusões.

O trabalho apresentado por Abdoli e Kahani (2009) mostrou a proposta de utilização

de técnicas de semântica em IDS, utilizando-se de ontologias para extrair as relações semânticas entre ataques e intrusões em um IDS distribuído. O sistema proposto possui dois tipos de agentes, um tipo representado por vários agentes chamados de *IDSagents* e um tipo representado por um agente especial chamado de *MasterAgent*, sendo esse último o responsável por manter a ontologia de ataques.

Quando os *IDSagents* detectam um ataque ou uma condição suspeita, eles enviam mensagens para o *MasterAgent*, que pode extrair as relações semânticas na rede utilizando a ontologia e decidir se a atividade reportada constitui ou não um ataque. Por ter sido implementado numa arquitetura do tipo hierárquica, o *MasterAgent* se mostrou como um ponto de falha, pois se um intruso pudesse impedir o seu funcionamento, a rede toda ficaria desprotegida. A arquitetura mostrou ser eficiente em reduzir as taxas de falso positivos e falso negativos. Para implementar o sistema multi-agentes foi utilizada a linguagem Java, com o JADE (*Java Agent DEvelopment Framework*).

O trabalho apresentado por Brahmi, Yahia e Poncelet (2010) mostrou a proposta de um IDS distribuído que utiliza agentes móveis e um algoritmo de mineração de dados para classificar o comportamento do tráfego na rede. Ele propõe a criação de agentes de detecção de assinaturas e de agentes de detecção de anomalias, estes últimos utilizando as técnicas de mineração de dados. A proposta é criar um IDS capaz de detectar tanto ataques conhecidos quanto ataques novos, tornando a detecção mais autônoma e eficiente.

O artigo também propõe que sejam criadas diversas classes de agentes, numa estrutura de detecção semelhante a de Ye et al. (2008), com uma técnica diferente, que classifica as conexões de rede de acordo com o nível de anomalia encontrado nas mesmas e ainda propõe que, ao detectar novos ataques, as assinaturas sejam acrescentadas ao agente de detecção de assinaturas. Como o número de conexões anômalas descobertas pelo IDS pode ser muito grande, as técnicas de mineração de dados devem auxiliar na análise e classificação dos dados obtidos. O trabalho não considerou a utilização de uma ontologia para a representação do conhecimento utilizado no processo de detecção. Os autores utilizaram o *framework* JADE para implementar o IDS proposto.

O trabalho apresentado por Zaidi (2011) mostrou a proposta de um IDS distribuído que se utiliza de agentes e de uma ontologia para a detecção de intrusão. No entanto, o autor não definiu o funcionamento interno dos agentes apresentados e sugeriu que a ontologia, que seria uma ontologia global, ficasse hospedada em um servidor, que poderia ser acessado por todos os *hosts* pertencentes ao IDS. Esse servidor, no entanto, se tornaria um ponto fraco no sistema, pois se deixasse de funcionar, o processo de detecção seria comprometido. O presente trabalho apresenta uma proposta de definir os elementos que faltam e, fazendo algumas adaptações e alterações, implementar uma arquitetura bastante semelhante com a proposta pelo autor.

O trabalho apresentado por Undercoffer, Joshi e Pinkston (2003) produziu uma on-

tologia que especifica um modelo de ataques a computadores. Essa ontologia é baseada na análise de mais de quatro mil classes de intrusões em computadores e as suas estratégias de ataques e as classifica de acordo com o componente do sistema que representa o alvo do ataque, os meios utilizados para o ataque, as consequências do ataque e a localização do atacante. O argumento utilizado foi de que as características usadas para definir os ataques deveriam se limitar àquelas que são observáveis e medidas no local alvo do ataque. Dessa forma, o modelo apresentado por Undercoffer et al. (2004) foi de uma ontologia centrada no alvo do ataque.

O trabalho apresentado por Bellifemine, Poggi e Rimassa (2001) mostrou as dificuldades em se trabalhar em um ambiente de agentes, descreveu as especificações da FIPA (*Foundation for Intelligent Physical Agents*) e fez uma descrição do funcionamento do *framework* JADE, de suas características e requisitos necessários para sua utilização. A tabela 1.1 resume os trabalhos relacionados.

Tabela 1.1: Trabalhos relacionados

Traballho	**Semelhanças**	**Diferenças**
(YE et al., 2008)	-Arquitetura P2P -Utiliza agentes móveis	-Não conta com uma ontologia
(ABDOLI; KAHANI, 2009)	-Utiliza agentes -Utiliza ontologia -Utiliza o JADE	-Arquitetura hierárquica, que pode ser um ponto fraco
(BRAHMI; YAHIA; PONCELET, 2010)	-Utiliza agentes móveis -Utiliza o JADE	-Não conta com uma ontologia -Utiliza algoritmo de mineração de dados
(ZAIDI, 2011)	-Utiliza agentes móveis -Utiliza ontologia	-Não define o funcionamento dos agentes e ontologia -Não implementa a arquitetura -Ontologia centralizada em um único servidor
(UNDERCOFFER; JOSHI; PINKSTON, 2003)	-Produziu uma ontologia de ataques a computadores	-Não detalhou a implementação do IDS que utiliza a ontologia
(BELLIFEMINE; POGGI; RIMASSA, 2001)	-Utiliza agentes -Utiliza o JADE	-Não implementou um IDS -Não trata de ontologias

1.5 Estrutura da Dissertação

Este trabalho está organizado da seguinte forma: no Capítulo 2, são abordados os principais conceitos envolvidos em detecção de intrusão, desde a definição do que é considerado como uma intrusão, e os tipos de Sistemas de Detecção de Intrusão, classificados de acordo com a disposição na rede e de acordo com a técnica utilizada no processo de detecção.

O Capítulo 3 trata da definição e descrição do funcionamento de agentes em sistemas. São apresentadas as descrições dos tipos de agentes e de como eles podem ser utilizados na detecção de intrusão. Também são apresentados os conceitos propostos pela FIPA que tratam de definições de como um ambiente de agentes deve funcionar, com o objetivo de criar um padrão para o funcionamento dos mesmos. É apresentada também a descrição do *framework* JADE, escolhido para ser utilizado na implementação da arquitetura para que a mesma possa ser avaliada.

No Capítulo 4 são apresentados os conceitos de ontologias, as definições, seus componentes e vantagens da utilização das mesmas. São apresentadas ainda as formas que as ontologias podem ser utilizadas em sistemas multi-agentes e como elas são tratadas e utilizadas pelo *framework* JADE.

No Capítulo 5 são explanadas as características e funcionalidades da arquitetura de detecção proposta, apresentando a especificação de seus elementos e detalhamento do funcionamento da mesma e as estratégias utilizadas no processo de detecção. A validação da arquitetura realizada através da implementação da arquitetura em um ambiente de laboratório, bem como os resultados obtidos, são apresentados no Capítulo 6.

Por fim, as conclusões do Capítulo 7 trazem uma discussão sobre as contribuições e os resultados obtidos, assim como a oportunidade para trabalhos futuros.

2 DETECÇÃO DE INTRUSÃO

Uma intrusão é definida como qualquer conjunto de ações que tentem comprometer a integridade, confidencialidade ou disponibilidade de um recurso. De acordo com Mukherjee, Heberlein e Levitt (1994), as intrusões geralmente causam perda de confiabilidade, perda de integridade, negação de serviços ou uso não autorizado de recursos. Como exemplos de intrusão tem-se: modificações não autorizadas de sistemas de arquivo permitindo acesso a sistemas ou informações dos usuários; acesso não autorizado e alteração de arquivos e informações dos usuários; modificações não autorizadas em componentes da rede (como as tabelas de roteamento) e o uso de recursos computacionais para fins que não foram os previstos pelo usuário.

Quando uma intrusão acontece, as técnicas de prevenção de intrusão como encriptação e autenticação são geralmente a primeira linha de defesa. No entanto, segundo Zhang, Lee e Huang (2003), a prevenção por si só não é suficiente, porque, como os sistemas ficaram cada vez mais complexos, existem sempre vulnerabilidades a serem exploradas nos mesmos devido a problemas nos projetos e na programação ou a falhas de operação pelos usuários. Além disso, mesmo se esquemas de prevenção de intrusão forem implementados de maneira perfeita, eles ainda não podem eliminar os ataques internos. Se um nó da rede for comprometido, o atacante pode propagar vírus ou *worms* e tentar obter as chaves criptográficas.

Dessa forma, a detecção de intrusão pode ser utilizada como uma segunda linha de defesa para proteger os sistemas em rede porque, quando uma intrusão é detectada, por exemplo, no estágio inicial de um ataque, a resposta pode ser acionada para minimizar os danos, conseguir evidências para processos e até lançar contra-ataques.

De acordo com Huang e Lee (2003), é importante considerar também que um atacante pode não somente tentar comprometer os nós, mas também comprometer o próprio IDS. Dessa forma, é importante que o IDS seja tolerante a falhas e/ou capaz de detectar problemas dentro da sua própria operação. É importante também que ele consiga realizar a sua função sem comprometer o funcionamento normal da rede (FERREIRA et al., 2011), possa ser executado continuamente com mínima supervisão humana, imponha um *overhead* mínimo ao sistema onde estiver sendo executado (de modo a não interferir na operação normal do mesmo), seja configurável para implementar as políticas de segurança do sistema que estiver sendo monitorado, tenha a habilidade de se adaptar a mudanças nos sistemas e comportamentos dos usuários, seja escalável, de modo a monitorar uma grande quantidade de *hosts* provendo resultados de maneira precisa e permita reconfiguração dinâmica, permitindo aos administradores realizarem mudanças sem a necessidade de reiniciar o IDS (SPAFFORD; ZAMBONI, 2000).

O sistema imunológico humano utiliza defesas em vários níveis contra patógenos prejudiciais e desconhecidos por meio de uma imunidade inata e adaptativa. A imunidade inata

protege o corpo contra os invasores conhecidos enquanto a imunidade adaptativa desenvolve uma memória de encontros passados e possui a habilidade de aprender sobre patógenos anteriormente desconhecidos. De acordo com Yu e Dasgupta (2011), essas características do sistema imunológico humano são as inspirações de pesquisadores na área de detecção de intrusão para desenvolver ferramentas de defesa automáticas e adaptáveis.

Quando se trata de detecção de intrusão, deve-se assumir o seguinte: as atividades dos usuários e dos programas são observáveis por mecanismos de auditoria de sistemas e, mais importante ainda, atividades normais e intrusões possuem comportamentos distintos (ZHANG; LEE; HUANG, 2003). Detecção de intrusão envolve então a captura de dados para auditoria e a descoberta de evidências nos mesmos para determinar quando o sistema está sob ataque.

Os IDS geralmente são compostos de quatro componentes (sensores, analisadores, base de dados e unidades de respostas) e são responsáveis pelas seguintes atividades: monitoramento da atividade dos usuários, monitoramento da atividade dos sistemas, auditoria da configuração dos sistemas, acesso aos arquivos de dados, reconhecimento de ataques conhecidos, identificação de atividades anormais, manipulação dos dados de auditoria, marcação de atividades normais, correção de erros de configuração nos sistemas e armazenamento de informação sobre invasores (FARID; RAHMAN, 2010).

Conforme observado por Ahmed et al. (2009), os IDS podem ser classificados de acordo com diversas características operacionais, como a resposta a intrusão, que pode ser passiva ou ativa. Um IDS passivo é aquele que realiza somente o trabalho de detecção de intrusão, deixando o tratamento do problema para outro sistema. Um IDS ativo é aquele que, após identificar uma intrusão, toma medidas como, por exemplo, encerrar todas as conexões de rede para um *host* suspeito. Esses sistemas, também conhecidos como IPS (*Intrusion Prevention System*) podem reagir mais rapidamente a um evento, porém, também reagem exageradamente no caso de alarmes falsos. Os IDS podem ser classificados também de acordo com a coleta de dados e processamento, que podem ser centralizados ou distribuídos.

De acordo com Osman e Elshoush (2011), os IDS totalmente distribuídos precisam estar preparados para enfrentar as dificuldades relacionadas com a precisão na detecção e o balanceamento da carga. A precisão é relativamente simples de ser analisada em IDS centralizados, visto que toda a informação de detecção fica concentrada no mesmo local. No entanto, quando as fontes de informação se encontram espalhadas é importante verificar se os dados disponíveis são suficientes para o processo de decisão sobre intrusão. O balanceamento de carga é necessário para não sobrecarregar sensores que fazem parte do IDS enquanto outros sensores ficam ociosos, o que pode causar atrasos e perda de informação, que podem ser explorados por atacantes.

As técnicas de detecção de intrusão podem ser classificadas como detecção de assinaturas ou detecção de anomalias. Sistemas de detecção de assinaturas procuram por ataques

que já são conhecidos. Sistemas de detecção de anomalias procuram por compartamentos diferentes do que é considerado como normal pelo IDS. Baseado no tipo de dados utilizado nas auditorias, isto é, de acordo com a fonte da informação a ser analisada, os IDS podem ser classificados como baseados em redes (*Network-based* IDS ou NIDS) ou baseados em *hosts* (*Host-based* IDS ou HIDS). Detecção de intrusão é baseada em rede quando o sistema é utilizado para analisar pacotes de rede. Isso a diferencia de detecção de intrusão baseda em *hosts*, que está relacionada com o processamento de dados que se originam no próprio computador, como eventos e logs.

2.1 Detecção de Assinaturas

Sistemas de detecção de assinaturas procuram identificar o uso de padrões conhecidos de ataques ou de comportamento não autorizado (NAKKEERAN; ALBERT; EZUMALAI, 2010). Uma assinatura é como uma pista de um ataque conhecido, como por exemplo, uma série de *bits* do cabeçalho de um pacote IP. A detecção de assinaturas pode ser comparada com o funcionamento de um programa antivírus: o antivírus compara os dados escaneados com os códigos de vírus conhecidos e, se encontra algum vestígio de vírus, o remove (CHEN; CHENG; HSIEH, 2010). Um ponto fraco desses sistemas é que eles não são eficazes contra novos ataques que não possuam assinaturas cadastradas. Dessa forma, os atacantes estão constantemente tentando modificar as características de seus ataques para que os mesmos não correspondam às assinaturas publicadas (FARID; RAHMAN, 2010).

Quando um ataque novo é descoberto, é desenvolvida a assinatura correspondente, no entanto, esse intervalo de desenvolvimento pode não ser pequeno o suficiente e intrusões podem ocorrer sem serem detectadas (BRAHMI; YAHIA; PONCELET, 2010). A principal vantagem da detecção de assinaturas é que ela pode ser precisa e eficiente em detectar instâncias de ataques conhecidos (ZHANG; LEE; HUANG, 2003), apresentando uma baixa taxa de falsos alarmes, além do fato de que proporciona a administradores de rede uma ferramenta de fácil uso para o monitoramento, mesmo que os mesmos não sejam especialistas em segurança (AYDIN; ZAIM; CEYLAN, 2009).

De acordo com NORTHCUTT et al. (2002), embora algumas assinaturas de tráfego de rede sejam bastante simples, existem assinaturas muito complicadas de serem desenvolvidas. Muitas delas são específicas do protocolo e da aplicação. Cada uma especifica as características que ajudam a identificar a vulnerabilidade que o atacante está tentando explorar. O desenvolvimento de assinaturas depende de um grande trabalho de equilíbrio. Uma assinatura específica demais pode ser extremamente precisa na identificação de um ataque, embora possa exigir muitos recursos. Se um atacante modificar ligeiramente o ataque, a assinatura específica pode não conseguir identificá-lo de forma alguma. Por outro lado, uma assinatura geral pode ser muito mais rápida e exigir menos recursos para ser desenvolvida, podendo inclusive ser melhor para

encontrar novos ataques e variantes dos ataques existentes. A desvantagem nesse caso é que ele pode levar à criação de muitos alarmes falsos, quando o IDS identificar uma atividade normal como um ataque. O grande problema com os alarmes falsos é que o analista precisa gastar muito tempo e recurso para determinar a verdadeira natureza deles.

2.2 Detecção de Anomalias

Sistemas de detecção de anomalias apontam como anomalias as atividades observadas que desviam significativamente dos perfis considerados como normais (ZHANG; LEE; HUANG, 2003; NAKKEERAN; ALBERT; EZUMALAI, 2010). Por exemplo, o perfil normal de um usuário pode conter a frequência média de alguns comandos de sistemas que foram usados em suas sessões. Se em uma sessão que estiver sendo monitorada, as frequências estiverem significativamente altas ou baixas, então um alarme de anomalia será criado. Essas detecções baseiam seu funcionamento em modelagem estatística de comportamento. Inicialmente, um modelo estatístico é construído de maneira que possa ser usado pra predizer de maneira precisa o comportamento dos usuários baseado em padrões de utilizações anteriores, ou é guardada a informação referente a um tráfego de rede normal para comparações futuras com o tráfego corrente na rede.

Anomalias em tráfego de rede são definidas então como comportamentos de rede ainda não vistos. Uma grande variedade de eventos incomuns podem ser classificados como anomalias e identificar, diagnosticar e tratar problemas como falhas e ataques é uma parte fundamental da rotina dos operadores de redes. Esses eventos incomuns podem ser ataques de intrusão, *scans*, *worms*, falhas em equipamentos, mudanças nos comportamentos da rede como aumento da demanda e aumento do volume de tráfego, além de novos comportamentos anteriormente desconhecidos (THANGAVEL; THANGARAJ; SARAVANAN, 2010).

De acordo com Garcia-Teodoro et al. (2009), de maneira geral, os IDS baseados em detecção de anomalias funcionam seguindo três estágios: parametrização, no qual a informação observada é representada em um formato pré-estabelecido; treinamento, no qual fica caracterizado o comportamento normal do sistema e é construído o modelo de referência; detecção, no qual a informação coletada é comparada com o modelo e é definido se ocorreu anomalia ou não. A figura 2.1 mostra o modelo proposto pelo autor.

É importante que o conceito de ataque ou intrusão não se confunda com o de anomalia. Um ataque é uma sequência de operações que põe a segurança do sistema em risco, uma anomalia, dentro do contexto da detecção de intrusão, é somente um evento que é suspeito (GARCIA-TEODORO et al., 2009). A maior vantagem de detecção de anomalias é que ela não necessita de conhecimentos anteriores sobre intrusões e pode, assim, detectar novas intrusões.

No entanto, de acordo com Brahmi, Yahia e Poncelet (2010) e Osman e Elshoush

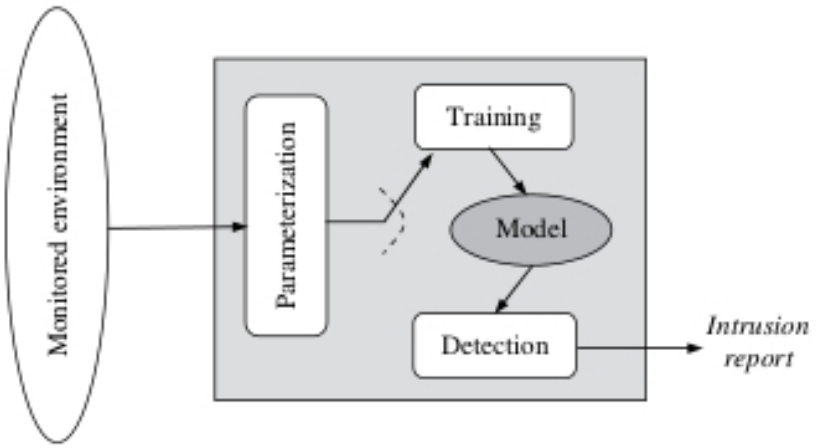

Figura 2.1: Modelo de Detecção de Anomalias (GARCIA-TEODORO et al., 2009)

(2011), essa técnica tem a tendência de gerar maior quantidade de alarmes falsos, já que um novo comportamento na rede, mesmo que legítimo, pode ser considerado como uma anomalia. Para diminuir o número de alarmes falsos, é importante ter uma definição bem precisa do que é um comportamento normal, no entanto, isso é extremamente difícil de definir, além do problema de que, como a utilização da rede e dos sistemas muda constantemente, a definição de comportamento normal também está em constante mutação (OSMAN; ELSHOUSH, 2011).

2.3 IDS baseados em redes

Os NIDS escutam a rede de maneira passiva ou ativa e capturam e examinam pacotes que passam por ela. Eles podem analisar o pacote inteiro, não só os endereços IP ou as portas. Eles podem olhar para o *payload* de um pacote, para ver qual aplicação particular do *host* está sendo acessada e com quais opções e, assim, criar alertas quando um atacante tentar explorar um *bug* no código, detectando assinaturas de ataques conhecidas.

Os NIDS são independentes de *hosts* e podem funcionar como monitores caixas pretas que podem encobrir toda a rede. Na prática, monitoramento ativo diminui a velocidade da rede consideravelmente e somente pode analisar de forma eficiente uma fração do tráfego. NIDS geralmente precisam de *hosts* dedicados ou equipamentos especiais, e assim, se forem alvos de ataques, podem prejudicar a segurança da rede (KACHIRSKI; GUHA, 2003). A utilização deles não torna desnecessária a utilização de um *firewall*. Conforme observado na figura 2.2, as duas ferramentas podem ser utilizadas em conjunto para auxiliar na proteção da rede.

Conforme foi observado por Guillen, Padilla e Colorado (2009), o funcionamento de um NIDS é, de forma resumida, composto por cinco processos. Inicialmente, um sensor atua como *sniffer*, capturando o tráfego de rede a ser analisado. A informação capturada é analisada por um decodificador de pacotes, para identificar cada um dos cabeçalhos de camada. O pre-

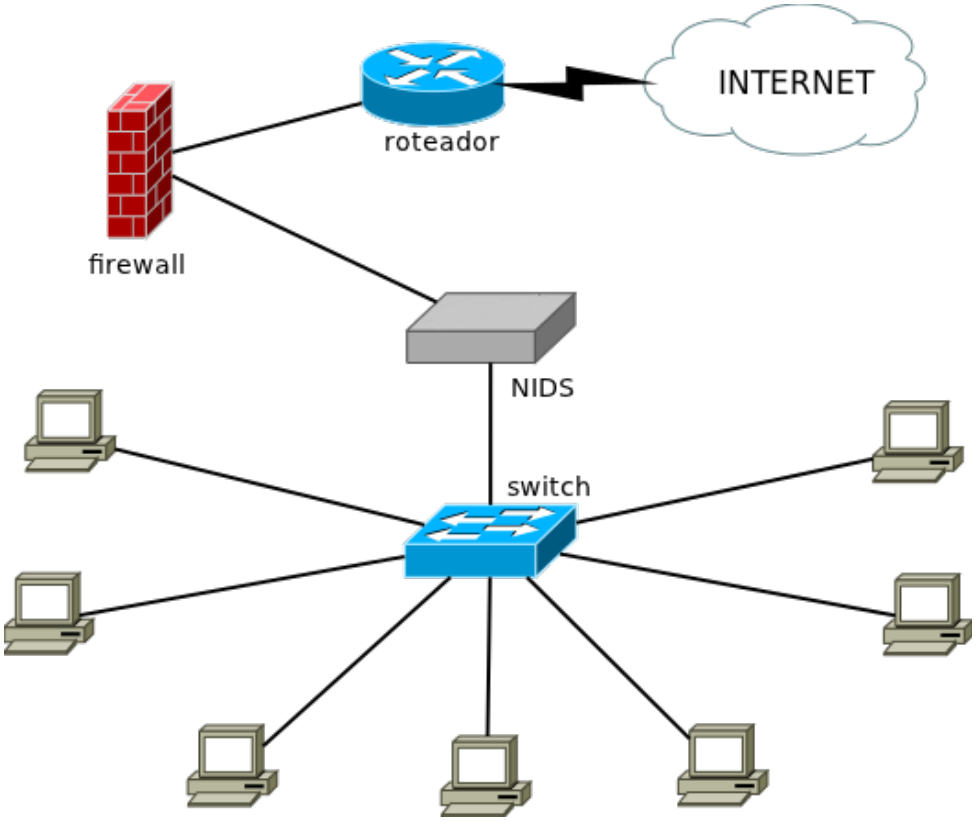

Figura 2.2: Rede com NIDS e *Firewall*

processador procura por anomalias nos cabeçalhos e organiza a informação para ser analisada mais profundamente pelo motor de detecção. O motor de detecção realiza a identificação de intrusão nos pacotes. Quando os pacotes são analisados, os dados são descartados e um alarme é gerado assim como o registro do incidente.

Os protocolos mais comumente analisados são os da pilha TCP/IP. Esses tipos de detectores geralmente são utilizados para tentar detectar ataques originados fora da rede.

2.4 IDS baseados em *hosts*

Do mesmo modo que os NIDS, os IDS baseados em *hosts*, ou HIDS, também são utilizados para detectar e analisar a atividade maliciosa, no entanto, operam monitorando *hosts* individuais. Eles são capazes de detectar ações como repetidas tentativas fracassadas de acesso ou mudanças em arquivos críticos do sistema e, normalmente, operam acessando os arquivos de *log* ou monitorando a utilização do sistema em tempo real.

Para garantir uma operação efetiva, clientes HIDS devem ser instalados em todos os nós da rede e modificados de acordo com as configurações específicas de cada um, no entanto, podem ser configurados para compartilhar suas informações de modo a criar uma visão mais abrangente da rede. Eles não dependem da capacidade do enlace da rede, sendo usados

em redes menores, onde cada nó dedica poder de processamento à tarefa de monitoramento do sistema. Como mencionado, esses sistemas são dependentes de *hosts* e podem diminuir consideravelmente a velocidade daqueles em que estiverem instalados (KACHIRSKI; GUHA, 2003).

Os HIDS podem complementar a operação dos NIDS, visto que podem observar atividades maliciosas que não são percebidas por IDS baseados em redes. De acordo com NORTHCUTT et al. (2002), eles podem ainda monitorar as trocas de informação em tráfego de rede criptografados, como nas VPN (Redes Privadas Virtuais, ou *Virtual Private Networks*), porque executam sua análise na extremidade da conexão, isto é, antes que os pacotes que saem sejam criptografados ou depois que os pacotes que chegam sejam decriptografados.

Ainda segundo NORTHCUTT et al. (2002), os HIDS podem ser utilizados para analisar o resultado de ataques nos *hosts*, quando esses forem detectados pelos NIDS, sendo considerados como ferramentas importantes nos tratamentos de incidentes pois auxiliam na reconstrução de ataques e na determinação da seriedade dos mesmos.

2.5 Organização do Conhecimento em um IDS

O componente central de um IDS é o motor de detecção que dispõe de uma base de conhecimento composta por regras de detecção de ataques, sejam elas baseadas em assinaturas ou em anomalias. Dessa forma, o modelo de dados influencia na eficácia do sistema. O problema de representação de conhecimentos consiste geralmente em determinar o que é um elemento de conhecimento e a forma de organizá-lo para garantir um desempenho satisfatório. Essa problemática consiste também em determinar a linguagem a ser utilizada para representar os conhecimentos do sistema e os mecanismos para explorá-los. A maioria dos trabalhos dessa área se baseiam na utilização de linguagens de especificação de ontologias (ZAIDI, 2011).

O conceito de ontologia é utilizado largamente, notadamente em linguística e no tratamento das linguagens naturais. De acordo com Zaidi (2011), elas definem os termos utilizados pelas pessoas, os bancos de dados e aplicações, que precisam compartilhar informações sobre um domínio bem específico como a medicina, a fabricação de ferramentas, a gestão de finanças, etc. As ontologias associam os conceitos básicos de um domínio específico e as relações entre os conceitos de uma maneira compreensível para as máquinas.

Elas codificam os conhecimentos de um domínio particular e do conhecimento que abrange outras áreas, tornando o conhecimento reutilizável. Em outras palavras, uma ontologia é um modelo de organização dos conhecimentos de um domínio de dados. As ontologias de informática são ferramentas que permitem representar precisamente um corpo de conhecimento em uma forma utilizável por uma máquina. Elas representam um conjunto estruturado de conceitos.

2.6 Ataques Computacionais

Se não for utilizado um IDS na rede, como será possível saber quando a rede foi vítima de um ataque? Alguns ataques são imediatamente perceptíveis, como aqueles que deformam *sites* de Internet, outros nem tanto. Algumas vezes, a única diferença notável é uma queda no desempenho da máquina e muitas vezes, não é possível perceber nada de anormal (NORTHCUTT et al., 2002).

Ainda de acordo com NORTHCUTT et al. (2002), muitos ataques envolvem várias etapas, como, por exemplo, um atacante pode enviar solicitações a cada endereço IP dentro de um determinado intervalo, que deveriam ser respondidas com alguma informação das máquinas alvos. Com base nos resultados do primeiro escaneamento, o atacante poderia tentar enviar uma consulta preparada especialmente para alguns alvos, explorando alguma vulnerabilidade encontrada.

Esse segundo conjunto de consultas poderia ocorrer várias horas ou até mesmo dias depois do primeiro. Se o atacante for bem sucedido, poderá realizar ações não-autorizadas em máquinas alvo, inclusive conseguir acesso como *root*, o que o permitiria ainda utilizar a máquina invadida para disparar novos ataques contra *hosts* dentro da rede invadida e em outras redes. Várias ferramentas de ataques utilizam essa técnica para sondar, atacar e comprometer servidores, sendo necessário que apenas alguns segundos passem entre o escaneamento inicial e o comprometimento total da máquina atacada.

De acordo com Farid e Rahman (2010), os ataques computacionais podem ser classificados como ataques de negação de serviço (DoS), *user to root* (U2R), *remote to user* (R2U) e *probes*. Os ataques U2R constituem uma classe na qual um intruso inicia com o acesso de uma conta normal de usuário e depois se torna um super-usuário explorando várias vulnerabilidade do sistema. Os ataques R2U constituem uma classe na qual um usuário remoto ganha acesso a uma conta local por meio de envio de pacotes para uma máquina por uma rede de comunicação. Os ataques de *probing* são ataques nos quais é realizada um escaneamento da rede com o objetivo de colher informações ou encontrar vulnerabilidade conhecidas. Um intruso com um mapa das máquinas e serviços que estão disponíveis na rede pode utilizar essa informação para criar uma estratégia de intrusão.

Ataques coordenados de várias fontes são extremamente difíceis de detectar já que a evidência fica dividida em vários domínios administrativos de redes. Com o objetivo de detectar esses ataques, é necessário desenvolver a habilidade de combinar evidências de atividades suspeitas em várias redes geograficamente distribuídas. Assim, é necessário analisar evidências em várias redes ao mesmo tempo.

Um ataque típico com essas características pode ser descrito da seguinte forma: inicialmente o atacante usa ataques de larga-escala para procurar por vulnerabilidades em *softwa-*

res ou para gerar uma lista de endereços vulneráveis que pode ser utilizada para propagar um *worm* futuramente. Com o conhecimento obtido anteriormente, ele pode escrever um código *worm* que irá comprometer até milhares de *hosts* simultaneamente. Esses *hosts* comprometidos, que passam a ser chamados de zumbis ou *bots*, podem ser utilizados para lançar um ataque de DDoS (ZHOU; LECKIE; KARUNASEKERA, 2010).

2.6.1 Ataques de Negação de Serviço

Os ataques de negação de serviço (*Denial of Service*, ou DoS) são caracterizados por uma sobrecarga do *host* alvo, que fica incapacitado de prover seus serviços de rede (CHEN; CHENG; HSIEH, 2010), dessa forma, o objetivo de um ataque de DoS é tentar limitar o acesso a uma máquina ou serviço, em vez de tentar atacar o serviço propriamente dito. Uma variante desse tipo de ataque é o tipo distribuído (*Distributed Denial of Service*, ou DDoS) no qual o atacante usa um grande número de computadores para atingir o sistema alvo fazendo com que a prevenção seja mais difícil e com que os impactos sejam mais severos. Não existem características aparentes em tráfegos de rede dos DDoS que possam ser utilizados para sua detecção e filtragem. O ataque alcança seu objetivo utilizando simplesmente um grande volume de dados, podendo variar todos os campos dos pacotes para evitar ser caracterizado ou rastreado.

Normalmente, esses ataques atingem a capacidade disponível dos enlaces ou a conectividade do *host*. Quando o ataque é contra a capacidade do enlace, a rede é inundada com um volume tão grande de tráfego que todos os recursos são consumidos e as requisições legítimas dos usuários não conseguem passar, resultando numa degradação da produtividade. Quando o ataque é à conectividade, o computador alvo é inundado com um grande volume de requisições de conexão, de modo que todos os recursos disponíveis no sistema operacional são consumidos e ele fica incapacitado de processar requisições legítimas dos usuários (DOULIGERIS; MITROKOTSA, 2004).

Os ataques podem, ainda, se aproveitar de vulnerabilidades com pacotes mal formados ou modificados de acordo com as falhas encontradas nas vítimas. Esses pacotes interagem com sistemas instalados causando consumo excessivo de memória e processamento, redução da velocidade do sistema e até reinicialização do mesmo (CARL et al., 2006).

De acordo com Zhou, Leckie e Karunasekera (2010), os ataques de DDoS podem ser divididos em duas fases: o recrutamento e a do ataque propriamente dito. Na primeira fase, o atacante procura *hosts* vulneráveis na internet e instala as ferramentas de ataques nos mesmos. É interessante para o atacante que essas máquinas possuam recursos abundantes que permitirão a geração de sequências de ataques poderosas. Ao instalar as ferramentas, o atacante tenta proteger o código malicioso para evitar que ele seja descoberto e desativado. Os usuários normalmente não percebem que seus sistemas foram comprometidos, pois as ferramentas insta-

ladas utilizam apenas uma proporção pequena dos recursos de memória e rede (DOULIGERIS; MITROKOTSA, 2004). A topologia do ataque nessa fase é geralmente de um atacante para vários alvos, o que é similar à fase de infecção dos *worms*.

Na segunda fase, o atacante envia comandos para os sistemas comprometidos na fase anterior, para que seja iniciado o ataque a outro alvo. A topologia de ataque nessa fase é geralmente de vários atacantes para um único alvo. Esses ataques consomem grande porções de utilização de CPU, memória e largura de banda dos alvos. A figura 2.3 esquematiza um ataque DDoS.

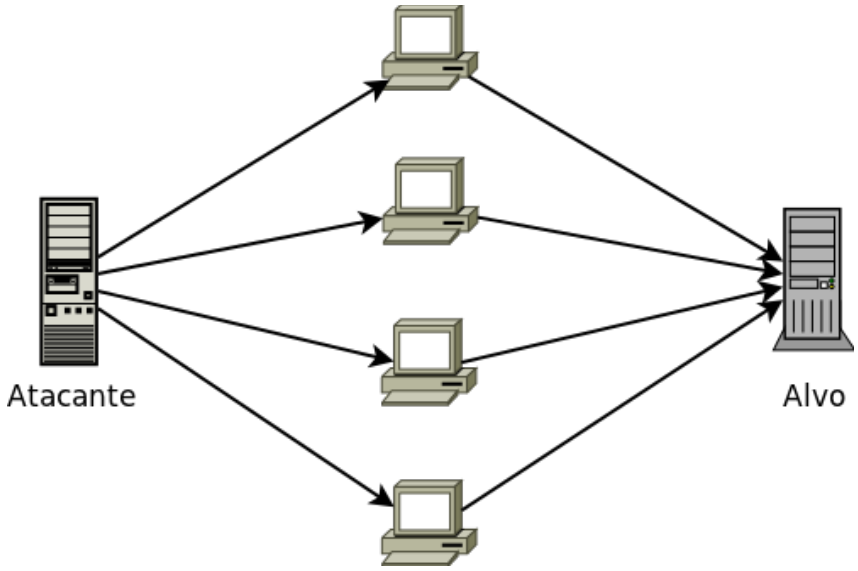

Figura 2.3: Um ataque DDOS

Os ataques SYN *flood* constituem um dos tipos de ataques DDoS mais populares. O atacante inicia um TCP *handshake* enviando um pacote SYN. A vítima, por sua vez, responde com um SYN/ACK normalmente, mas então o atacante simplesmente não completa a negociação e não envia um pacote ACK. Isso não causa efeitos muito significantes, visto que até mesmo roteadores domésticos apresentam proteções contra ataques de SYN *flood*, entretanto, alguns recursos ficam alocados por um período de tempo curto, e enviando pacotes SYN suficientes, abrindo novas conexões, um roteador simples pode ter seus recursos exauridos rapidamente. Os roteadores de alto desempenho são mais difíceis de serem sobrepujados, mas os ataques DDoS acabam sendo bem sucedidos por usarem tentativas de conexão de milhares de máquinas em uma *botnet* (MANSFIELD-DEVINE, 2011).

Botnets são grupos de computadores infectados (*bots*) que operam sem a intenção e conhecimento dos usuários. Elas geralmente são controladas por um ou mais computadores atacantes e formam a estrutura perfeita para a preparação e o lançamento de qualquer tipo de ataque distribuído (SPEROTTO et al., 2010).

De acordo com Roman et al. (2011), os ataques DDoS do tipo *Smurf* ICMP tem o

funcionamento descrito como se segue. Inicialmente o atacante envia mensagens ICMP para um grande número de nós na rede. Essas mensagens verificam se os nós estão ativos e aceitam requisições de serviço, o que, se for verdadeiro, faz com que os mesmos respondam a mensagem com um pacote *echo* enviado para o endereço de origem retirado do cabeçalho do pacote ICMP recebido. Entretanto, o atacante modifica os datagramas enviados inserindo o endereço IP da vítima nesse campo. Assim, ao receber os pacotes ICMP, os nós ativos respondem para o endereço da vítima, o que pode ocasionar da mesma receber centenas de respostas de diversos nós simultaneamente, constituindo o ataque.

Atualmente, os ataques DDoS são considerados como uma das ameaças principais nas esferas comerciais, governamentais e até militares. Esses ataques passaram a chamar a atenção do público em geral devido a reportagens apresentadas na mídia sobre as atividades do grupo *Anonymous*, que utilizou ataques desse tipo para atingir empresas como MasterCard, Visa, Amazon e PayPal. Os ataques do *Anonymous* apresentaram características diferentes de outros ataques, visto que os participantes foram voluntários para contribuírem com as atividades (ROMAN et al., 2011).

Existem alguns modos de combater os efeitos de ataques DDoS, mas para isso é necessário que sejam identificadas as fontes do ataque. Essa identificação pode ser realizada por meio de rastreamento de pacotes IP, mas essa técnica geralmente consegue identificar somente as máquinas zumbis que participaram do ataque e não a real fonte do mesmo. Para reduzir os efeitos de ataques de DDos podem ser utilizadas técnicas de filtragem de pacotes e limitação da taxas (LU et al., 2007). As técnicas de filtragem descartam pacotes que se originam em IPs diferentes dos permitidos pelo gerente de rede e as técnicas de limitação de taxas são utilizadas em roteadores para descartar pacotes em um enlace que ultrapassa algum limite predefinido.

2.6.2 Ataques de ARP *Poisoning*

Em ataques de ARP *Poisoning* o atacante envia respostas ARP com mapeamentos IP e MAC falsos, em uma tentativa de corromper o cache ARP de outros nós da rede. Como exemplo, se um atacante quiser tomar o lugar de um nó X, de modo que um nó Y envie dados destinados a X para o atacante, ele pode enviar uma reposta ARP indicando que o nó com o IP do nó X tem o endereço MAC do nó atacante (ABAD; BONILLA, 2007). Assim, o receptor da mensagem vai atualizar seu cache ARP com a informação recebida e enviar para o atacante todos os pacotes destinados ao IP do nó X (ZDRNJA, 2009; TRIPUNITARA; DUTTA, 1999). Os ataques de ARP *poisoning* podem ser utilizados para fazer com que o atacante se passe por um nó na rede, como parte de ataques de DoS ou ataques do tipo *man in the middle* (MITM) (NAM; KIM; KIM, 2010). A figura 2.4 representa o estabelecimento de um ataque MITM.

Em um ataque de DoS, o atacante pode corromper a tabela ARP de um nó de modo

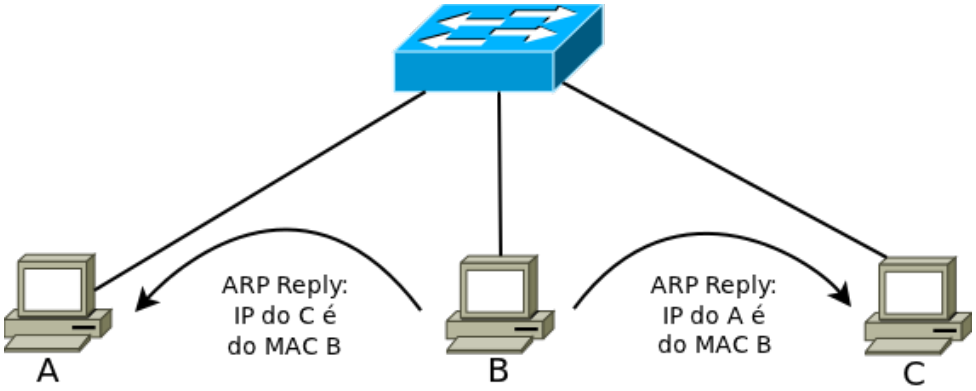

Figura 2.4: Um ataque de MITM sendo preparado

que todo o pacote que esse nó enviar seja direcionado para o atacante, em vez de para seu destino real. Dessa forma, o atacante interrompe a comunicação do nó que é atacado. Em um ataque MITM um atacante pode, ao corromper a tabela ARP de dois nós da rede, interceptar toda a comunicação entre eles sem que eles percebam. A figura 2.5 representa a execução de um ataque MITM. Esse tipo de ataque pode ser direcionado a dois nós dentro de uma rede interna ou ainda entre um nó da rede interna e um externo, bastando para isso que o atacante se posicione entre o nó da rede interna e o *gateway* padrão (ABAD; BONILLA, 2007; ZDRNJA, 2009). Ao realizar o ataque, pode-se ter acesso a qualquer informação sensível que tramite na rede, além de se poder modificar os dados, comprometendo a integridade dos mesmos.

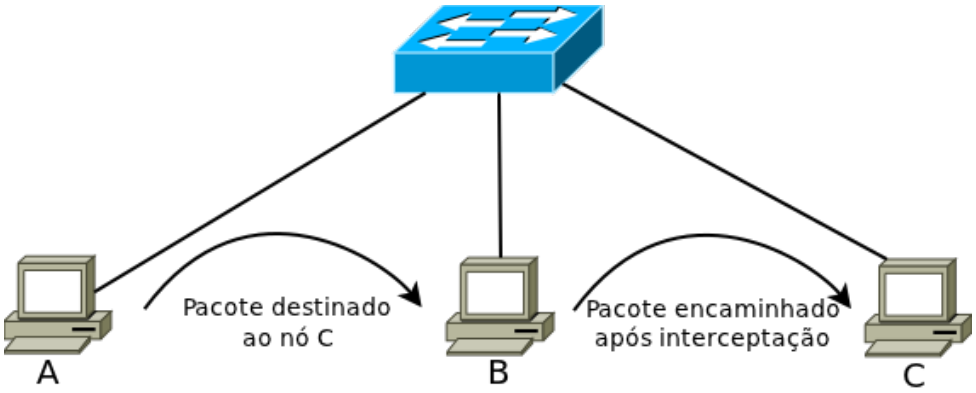

Figura 2.5: Um ataque de MITM sendo executado

2.6.3 *Worms* e Vírus

Um *worm* é um programa auto-replicante que pode enviar cópias de si mesmo para outros computadores na rede sem a intervenção de usuários. Em geral, os ataques que objetivam espalhar um *worm* na rede possuem dois objetivos principais: causar um *overload* no trafégo em redes locais e congestionamentos em enlaces de Internet, o que causa prejuízo para *hosts* afetados, e recrutar *hosts* comprometidos para utilização em ataques futuros. A infestação de *worms* consiste em duas fases. A primeira é aquela na qual é feito o escaneamento à procura

de *hosts* vulneráveis. A segunda fase é aquela na qual o código do *worm* é transmitido para as máquinas alvo. O ataque é, assim, composto por uma máquina atacante atuando sobre várias máquinas alvo (ZHOU; LECKIE; KARUNASEKERA, 2010).

Um vírus pode ser considerado como um *worm* que só se replica no computador onde ele está instalado. Além disso, ele precisa de uma ação do usuário para se propagar para outros *hosts*. Geralmente, os vírus são encontrados anexados a arquivos ou executáveis e são ativados pelo usuário (SPEROTTO et al., 2010).

2.6.4 Ataques de estouro de *buffer*

São ataques nos quais o invasor tenta ganhar o controle ou derrubar um processo em um sistema alvo vulnerável por meio de um estouro no *buffer* desse processo (SPEROTTO et al., 2010), alterando o estado de memória do programa. Independente de onde estiverem localizados, os *buffers* podem ser sobrecarregados com mais dados do que podem comportar se não for realizado nenhum tipo de checagem que garanta que o dado a ser escrito no *buffer* realmente cabe nele. Quando dados de mais são escritos, a quantidade extra vai invadir as estruturas de memória adjacentes e reescrever qualquer coisa que estiver armazenada lá anteriormente.

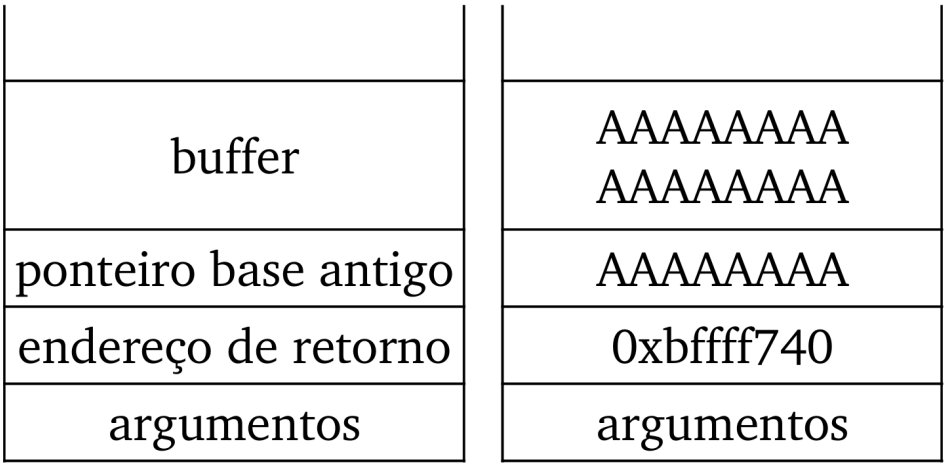

Figura 2.6: Um ataque de estouro de *buffer*

Essa mesma metodologia pode ser utlizada para sobrescrever um ponteiro de código e alterar o fluxo de controle direcionando o ponteiro. Um exemplo desse tipo de ataque, conforme mostrado por Wilander e Kamkar (2003), pode ser visualizado na figura 2.6: um *buffer* representado pela pilha é sobrecarregado com "A"s de modo que o endereço de retorno seja sobrescrito. Se um atacante conseguir fornecer os dados de entrada do buffer, pode confeccionar dados para direcionar o endereço de retorno para um código malicioso que ele escreveu, fazendo com que esse código seja executado (WILANDER; KAMKAR, 2003; RUWASE; LAM, 2004).

3 AGENTES

De acordo com O'Brien e Nicol (1998), Lee e Hwang (2004), Sbodio, Martin e Moulin (2010) e Ören e Yilmaz (2012), um agente é um pequeno software capaz de realizar atividades inteligentes em benefício do usuário, com o objetivo de executar uma tarefa. Eles podem trabalhar em cooperação de modo que uma sociedade de agentes possa combinar suas habilidades para resolver problemas. Os agentes móveis consistem em um tipo especial e são definidos como processos capazes de navegar através de grandes redes interagindo com as máquinas, coletando informações e retornando após terem executado as tarefas definas pelo usuário (NAKKEERAN; ALBERT; EZUMALAI, 2010).

Na figura 3.1, pode ser observada uma representação de agentes móveis coletando informações na rede. Devido à natureza colaborativa dos mesmos, um padrão de criação e comportamento é essencial para o desenvolvimento desse tipo de tecnologia. Com esse objetivo, um padrão deve definir um meio no qual eles possam trocar informações e delegar tarefas, ferramentas por meio das quais eles possam se localizar, um ambiente que seja seguro para os mesmos operarem e trocarem mensagens confidenciais, uma forma de identificação única, uma forma deles interagirem com os usuários e, se necessário, uma forma deles migrarem de uma plataforma para outra.

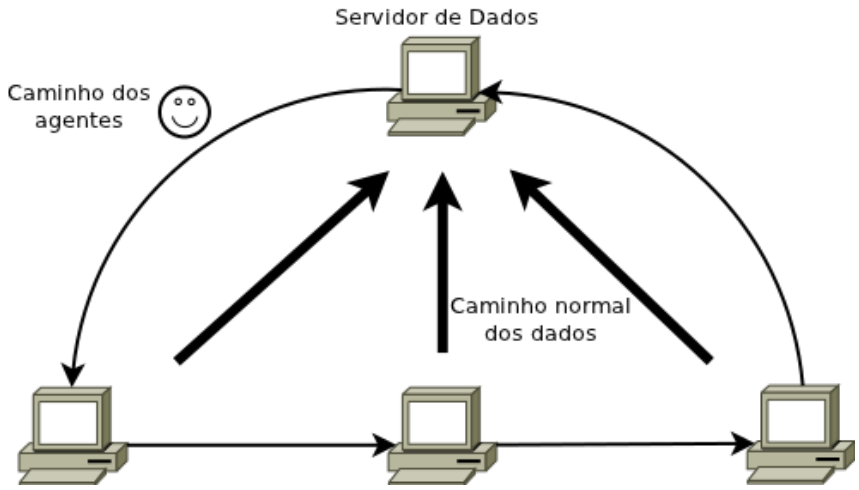

Figura 3.1: Coleta de dados por agentes móveis

A tecnologia baseada em agentes é poderosa para o desenvolvimento de sistemas distribuídos complexos. Segundo Ören e Yilmaz (2012), sistemas de agentes são definidos como sistemas que são compostos de uma coleção de agentes de software que são autônomos e direcionados a um objetivo, situados em um contexto organizacional para cooperarem por meio de interações flexíveis e adaptáveis e mecanismos cognitivos para alcançarem objetivos que não poderiam ser alcançados por um agente isolado. De acordo com Gil (2006), é muito importante que o sistema multi-agentes seja robusto para resolver os problemas de maneira distribuída.

No entanto, criar um sistema robusto é um desafio quando componentes heterogênios precisam ser coordenados em ambientes de execução não centralizados e altamente dinâmicos. O conceito de agentes foi utilizado em diversas áreas de aplicações como em sistemas de controle em tempo real, comércio eletrônico, gerenciamento de redes, desenvolvimento e manutenção de sistemas complexos, gerenciamento de recurso computacional, sistemas de transporte, gerenciamento de informação, computação científica e entretenimento, sempre com o intuito de prover melhor acesso ao conhecimento e aos recursos computacionais (CHMIEL et al., 2004; LEE; HWANG, 2004; LUCK, 2004; CHEN; CHENG, 2010).

A razão para o crescente sucesso da tecnologia baseada em agentes nessas áreas é que a sua distribuição permite uma decomposição natural do sistema em vários agentes que interagem entre si para atingir o objetivo global. Essa tecnologia pode aumentar significamente a análise de problemas que possuam as seguintes características: o domínio do problema seja geograficamente distribuído, os subsistemas existam em um ambiente dinâmico e os subsistemas precisam interagir entre si de maneira mais flexível. Na figura 3.2, pode ser observada uma representação de computação distribuída realizada por meio de agentes móveis.

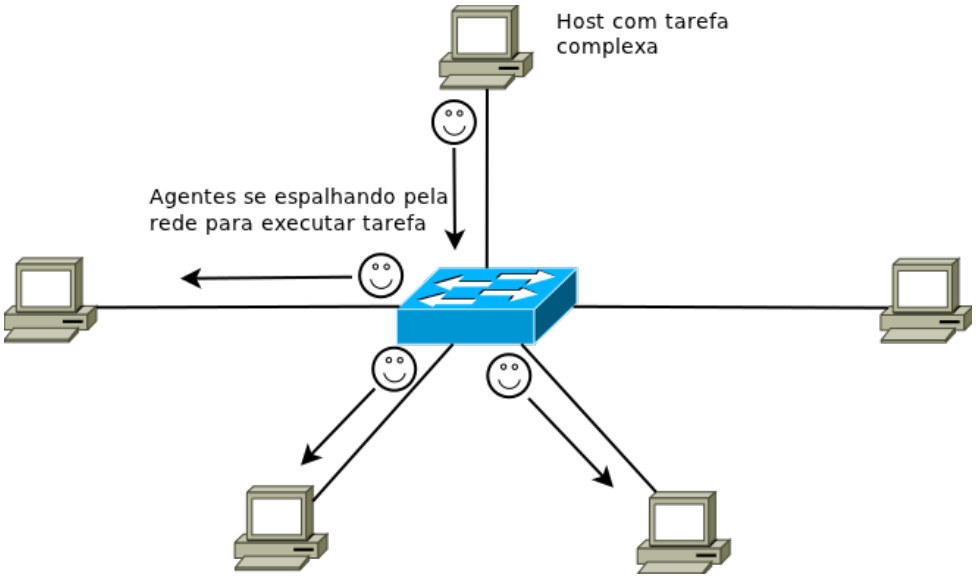

Figura 3.2: Computação distribuída com agentes móveis

De acordo com Hurson et al. (2010), agentes podem utilizar também a habilidade de se clonarem e assim, executar várias cópias de si mesmos em diferentes locais ao mesmo tempo. Essa capacidade de suportar execução concorrente adicionada a sua capacidade de mobilidade os torna ainda mais compatíveis para o uso em aplicações distribuídas. Ressalta-se ainda que os agentes móveis são executados independentes do computador que os originou até que a tarefa que estiverem executando esteja terminada, o que faz com que essa tecnologia seja uma excelente opção para ambientes com recursos limitados ou conectividade intermitente.

A busca de informação distribuída está direcionada para descobrir maneiras de lo-

calizar e reaver a informação pretendida baseada nas especificações do usuário. A migração do código do programa (agente móvel), em vez da migração dos dados, pode aumentar significantemente a eficiência de sistemas que contenham grandes quantidades de dados que não são importantes para a execução da tarefa em questão, o que ocorre bastante em sistemas que procuram reaver informações. Ainda segundo Hurson et al. (2010), o uso dos agentes móveis traz as seguintes vantagens para os sistemas: computação distribuída colaborativa, redução do tráfego na rede e da necessidade de largura de banda por meio da diminuição do número de interações remotas, autonomia e operação contínua em situações de queda de conexão, facilidade de configuração e atualização, e, escalabilidade e dinamicidade.

Segundo Nyulas et al. (2008), qualquer tentativa de criar um modelo de geração de sistemas de agentes deve possuir pelo menos as habilidades de modelar agentes em um sistema e as interfaces dos mesmos, descrever a informação utilizada e gerada por cada agente e pela aplicação como um todo, considerar a possibilidade de relacionamentos entre os agentes e os meios de comunicação entre eles e especificar o conteúdo da informação que pode ser trocada pela comunicação. Algumas características adicionais que seriam benéficas seriam a habilidade de facilmente modificar modelos de sistemas e rapidamente gerar novos sistemas executáveis.

De acordo com Lee e Hwang (2004), um dos problemas mais importantes a ser tratado no desenvolvimento de sistemas baseados em agentes é a maneira como será projetada a arquitetura, visto que esta afeta diretamente o desempenho do sistema. O ponto mais importante a ser trabalhado no desenvolvimento da arquitetura é o de como os agentes irão interagir entre si para atingir o objetivo do sistema. A interação entre os agentes depende da estrutura organizacional e do mecanismo de coordenação. Algumas organizações, como a FIPA, criaram alguns padrões que tratam de gerenciamento de agentes, linguagens de comunicação, ontologias, transmissões de mensagem, etc; especificando assim, referências para os componentes da plataforma, formatos de linguagem e integração de agentes. No entanto, não são especificados métodos para projetos e avaliação de arquiteturas, como quantidade de agentes necessários, como organizá-los e como controlar a interação entre eles.

As arquiteturas de sistemas multi-agentes podem ser classificadas, de maneira resumida, como horizontais e hierárquicas. Nas arquiteturas horizontais, cada agente se comunica diretamente com outro sem nenhuma intervenção entre eles. Essa estrutura, na qual não há nenhum controle centralizado, é chamada de *peer-to-peer* e qualquer nó é equivalente a todos os outros, o que a torna altamente atrativa quando se tem necessidade de uma arquitetura robusta contra falhas, de fácil implementação, escalabilidade e manutenção (MILOJICIC et al., 2002). Nas arquiteturas hierárquicas, existem agentes de controle e a arquitetura fica estruturada no formato de uma árvore, na qual agentes de controle estão localizados nos nós pais e os agentes controlados estão localizados nos nós filhos. Os agentes em nós intermediários fazem o papel de facilitadores para os controladores. A figura 3.3 mostra os dois tipos de arquiteturas de

sistemas multi-agentes.

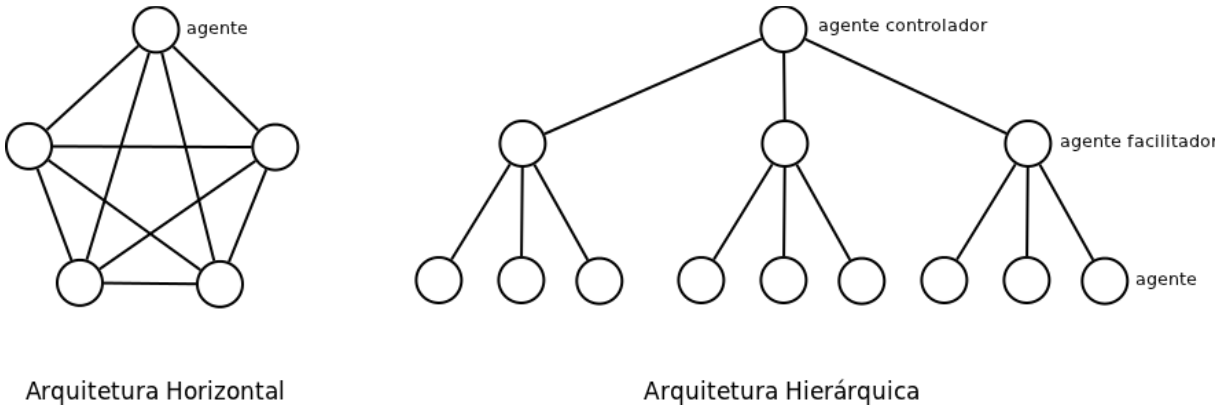

Figura 3.3: As arquiteturas de sistemas multi-agentes

Como mencionado, os IDS, além de protegerem a rede, precisam ser capazes de resistir a ataques contra eles e serem tolerantes a falhas, altamente adaptáveis e configuráveis. Dadas essas características, a tecnologia baseada em agentes parece ser uma alternativa apropriada ao desenvolvimento de IDS (BRAHMI; YAHIA; PONCELET, 2010).Como os agentes possuem execução independentes, eles podem ser adicionados, removidos e reconfigurados sem necessidade de alterar outros componentes.

Se um IDS for composto de agentes autônomos, alguns deles podem ter seu funcionamento parado para manutenção enquanto os outros continuam sendo executados, tornando a detecção um processo contínuo. Agentes podem, ainda, ser ativados e desativados dinamicamente, tornando possível que se otimize o uso de recursos e possibilitando também que se reconfigure o IDS sem necessidade de reiniciar o mesmo, simplesmente reconfigurando agentes que estiverem desativados. Novos agentes podem ser lançados na rede de acordo com o necessário, fazendo com que o sistema possa crescer constantemente (SPAFFORD; ZAMBONI, 2000).

Um IDS distribuído proposto por Kachirski e Guha (2003) tentou prover um mecanismo leve e com pouco *overhead* baseado no conceito de agentes de segurança móveis. Essencialmente, um agente móvel desse IDS é um objeto ativo pequeno e inteligente que viaja pela rede para ser executado em certo *host* e depois retorna com os resultados para o *host* que o originou. Os agentes são dinamicamente atualizáveis, leves, possuem um funcionamento específico e podem ser vistos como componentes de um IDS flexível e dinamicamente configurável.

As arquiteturas baseadas em agentes trazem conceitos de inteligência artificial para o domínio de sistemas distribuídos e, dessa forma, trazem para os IDS características básicas de agentes, como autonomia, reatividade, pró-atividade e sociabilidade. A autonomia implica que os agentes podem realizar tarefas complexas de modo independente e, muitas vezes, por longos períodos de tempo; a reatividade implica que eles podem responder a mudanças no ambiente; a pró-atividade implica que eles podem tomar a iniciativa para executar uma determinada tarefa, mesmo sem um estímulo explícito do usuário; e a sociabilidade implica que eles podem interagir

com outras entidades com a finalidade de alcançar a cooperação e a colaboração, isto é, o seu próprio objetivo ou um objetivo comum.

Entre as tentativas de criar padrões de interoperabilidade para agentes em diferentes tipos de plataformas, pode-se destacar: a comunidade KQML (*Knowledge Query Meta Language*), MASIF (*Mobile Agent System Interoperability Facility*) e a FIPA. Tanto KQML quanto a FIPA definem as interações entre agentes em termos de uma linguagem de comunicação de agentes (*Agent Communication Language* ou ACL), enquanto MASIF define as interações em termos de *Remote Procedure Calls* (RPC) ou *Remote Method Invocation* (RMI).

A ACL definida pela FIPA é uma tentativa de criar uma linguagem de comunicação universal orientada à mensagens que possa permitir abordagens flexíveis para comunicação entre entidades de *software*. Como ainda não se chegou a um consenso sobre suas especificações, existem variações da KQML, assim, diferentes sistemas de agentes que conversam em "dialetos" diferentes podem não ser beneficiados por uma interoperabilidade completa (YANG et al., 2010).

3.1 FIPA

A FIPA é uma associação internacional sem fins lucrativos de companhias e organizações que compartilham esforços a fim de produzir especificações para tecnologias de agentes e sistemas multi-agentes e possibilitar a interoperabilidade dos seus padrões com outras tecnologias. Ela foi aceita como um dos comitês que compõem o IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos ou *Institute of Electrical and Electronics Engineers*) em 2005. Ela promove um conjunto de tecnologias para diferentes áreas de aplicação que os desenvolvedores podem integrar para construir sistemas complexos com um alto grau de interoperabilidade.

As especificações da FIPA, de acordo com O'Brien e Nicol (1998), não têm por objetivo tratar da arquitetura interna dos agentes, nem da forma como eles devem ser implementados, mas tratam de especificar as interfaces necessárias para suportar a interoperabilidade entre sistemas de agentes. A Fundação identificou quatro áreas para trabalhar na definição de padrões: comunicação entre agentes, gerenciamento dos agentes, integração de agentes com softwares e interação dos agentes com os usuários humanos.

Os documentos da FIPA definem normas que permitem a uma sociedade de agente existir, operar e ser gerenciada e definem um modelo de referência para uma plataforma. Esse modelo está estruturado sobre três agentes especiais, necessários para a gerência da plataforma e para a descrição da linguagem de conteúdo para gerenciamento de agentes e ontologia: o AMS (*Agent Management System*), o DF (*Directory Facilitator*) e o ACC (*Agent Communication Channel*) (BELLIFEMINE; POGGI; RIMASSA, 2001).

O AMS é responsável por prover o serviço de nomes, assegurando que cada agente na plataforma possua um nome único. O nome de um agente é composto do nome da plataforma onde o agente foi criado e de um identificador único dentro dessa plataforma. O AMS representa ainda a autoridade na plataforma, possibilitando, por exemplo, que agentes sejam criados ou destruídos, o registro de agentes na plataforma e a migração de agentes de/para plataformas. O DF é responsável por prover o serviço de "páginas amarelas", por meio do qual um agente pode registrar os serviços que ele fornece e encontrar outros agentes que proporcionem o serviço que ele precisa para atingir seus objetivos.

Os membros de um DF em particular definem um domínio, que é uma comunidade de agentes que reflete a organização lógica dos mesmos. Uma plataforma de agentes pode suportar vários domínios, assim como um domínio pode se espalhar por várias plataformas. Os DF podem ainda registrar uns aos outros, criando uma rede e permitindo consultas entre domínios. O ACC é responsável por prover os meios para a comunicação básica entre agentes dentro e fora da plataforma, sendo o método de comunicação padrão, oferecendo um serviço de roteamento de mensagens confiável, ordenado e preciso.

As especificações da FIPA definem ainda uma Linguagem de Comunicação de Agentes (ACL) utilizada pelos agentes para a troca de mensagens. A ACL da FIPA é uma linguagem que descreve a semântica das mensagens e como elas serão codificadas, mas não especifica nenhum mecanismo para o transporte das mensagens. Como agentes diferentes podem rodar em plataformas diferentes em redes distintas, as mensagens são codificadas em forma de texto, existindo, entretanto, uma sêmantica formal definida na ACL, que é responsável por eliminar ambiguidades e confusões que poderiam surgir com o uso da linguagem (BELLIFEMINE; POGGI; RIMASSA, 2001). A figura 3.4, encontrada em (YANG et al., 2010) mostra o modelo de referência de agentes da FIPA.

De acordo com O'Brien e Nicol (1998), a ACL da FIPA pode ser considerada como composta por cinco níveis. O primeiro nível seria o do Protocolo, que define as regras sociais para a estruturação do diálogo entre agentes. O segundo nível seria o do Ato da Comunicação, que define o tipo de comunicação que está sendo realizada, como por exemplo, uma requisição. O terceiro nível seria o do Envio da Mensagem, que define meta-informações sobre a mensagem que poderiam incluir, por exemplo, a identidade do agente que enviou a mensagem e a identidade do agente destinarário. O quarto nível seria o do Conteúdo da Mensagem, que define a gramática utilizada na expressão do conteúdo de uma mensagem e a semântica associada a ela. O quinto nível seria o da Ontologia, que define o vocabulário e o significado dos termos e conceitos utilizados ma expressão do conteúdo das mensagens.

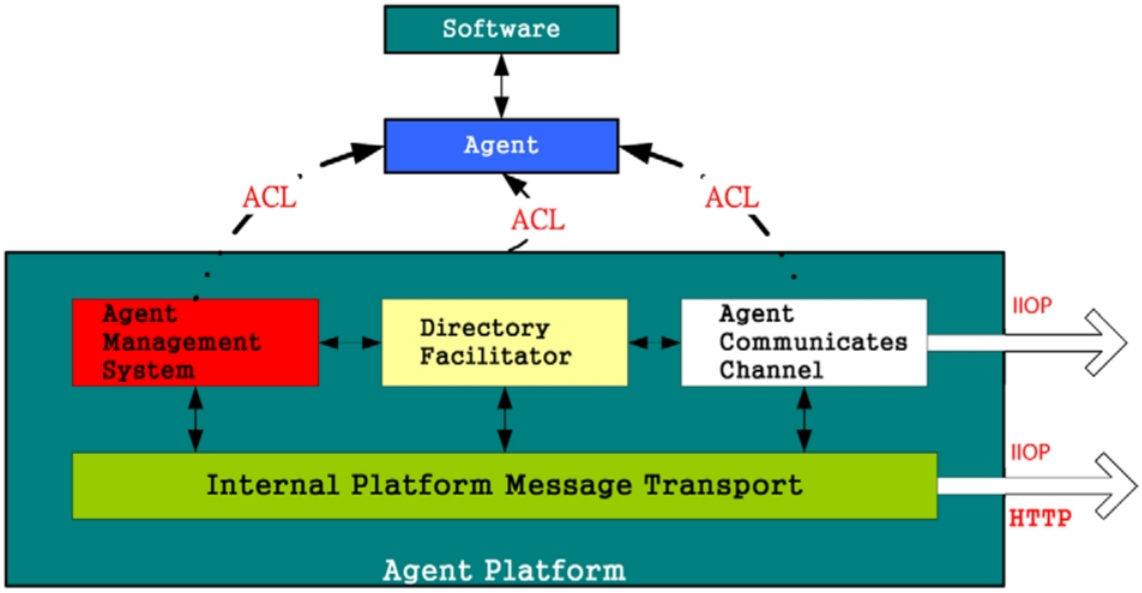

Figura 3.4: Modelo de referência de agentes da FIPA (YANG et al., 2010)

3.2 O *Framework* JADE

O JADE (*Java Agent DEvelopment Framework*) é um *framework* que pode ser utilizado para simplificar a implementação de sistemas multi-agentes de acordo com as especificações definidas pela FIPA (AART et al., 2002; LIU; KÜNGAS; MATSKIN, 2006; SPANOUDAKIS; MORAITIS, 2010). Ele permite que os agentes se movam de máquina em máquina, caracterizando assim, a possibilidade de implementar agentes móveis e, assim, formar uma plataforma P2P. A plataforma foi inicialmente desenvolvida pelo grupo de pesquisa e desenvolvimento da Telecom Itália em parceria com a Universidade de Parma, entretanto, no ano dois mil passou a ser uma tecnologia *open source*, sob as normas da licença LPGL (*Lesser General Public License*).

O *framework* JADE oferece suporte ao ciclo de vida de agentes, além de oferecer uma variedade de ferramentas gráficas que favorecem o desenvolvimento. Ele é escrito em Java, trazendo assim o conjunto de recursos e bibliotecas oferecidas pela linguagem, que por sua vez oferece uma grande quantidade de abstrações de programação e possibilita a construção de sistemas baseados em agentes com conhecimentos anteriores relativamente pequenos sobre a teoria de agentes (BELLIFEMINE et al., 2008). Com isso, o programador pode economizar tempo com o trabalho necessário para construir uma infraestrutura baseada em agentes e partir para a definição e programação do funcionamento interno dos mesmos. A arquitetura de agentes JADE pode ser visualizada na figura 3.5, encontrada em (BELLIFEMINE; POGGI; RIMASSA, 2001).

A estrutura das mensagens trocadas na comunicação entre agentes é baseada na lin-

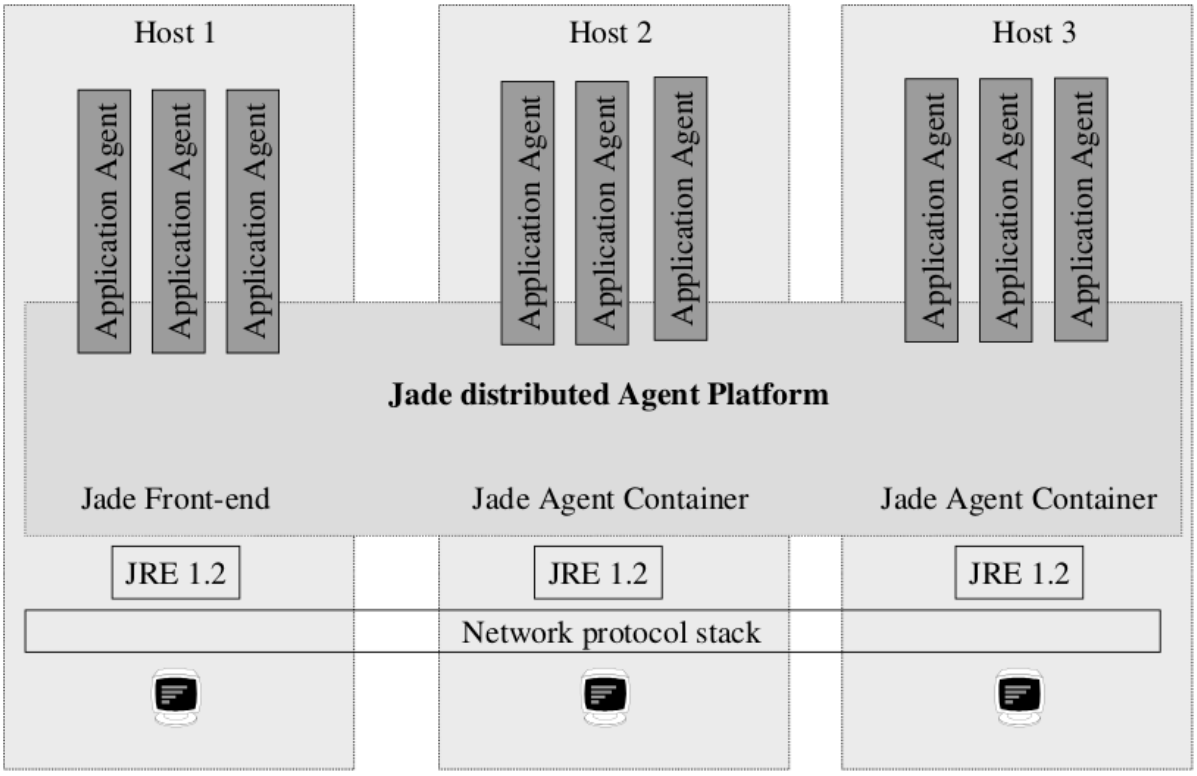

Figura 3.5: A Arquitetura de Agentes JADE (BELLIFEMINE; POGGI; RIMASSA, 2001)

guagem ACL definida pela FIPA e contém variáveis que indicam o contexto ao qual a mensagem se refere e o tempo limite que pode ser aguardado até a resposta ser recebida. A comunicação é, então, realizada por meio de troca de mensagens assíncronas. O sistema de agentes do JADE é baseado no paradigma *peer-to-peer* (BELLIFEMINE et al., 2008). Cada agente é identificado por um nome globalmente único, definido como AID (*Agent Identifier*) pela FIPA, que o JADE constrói concatenando um apelido definido pelo próprio usuário ao nome da plataforma. Ele pode ser associado ou desassociado a qualquer momento e permite descobrir dinamicamente outros agentes a partir dos serviços providos pelo AMS e pelo DF.

Utilizando JADE, a informação, os recursos e o controle podem ser distribuídos em um grupo de *hosts* heterogêneos, em ambientes cabeados ou sem fio (BELLIFEMINE et al., 2008). Um agente pode parar de executar em um *host*, migrar para um *host* remoto diferente sem a necessidade de ter o código do agente já instalado nesse e então reiniciar sua execução a partir do mesmo ponto em que foi interrompido. O JADE fornece ainda suporte a ontologias e a linguagens de conteúdo, sendo a verificação de ontologia e a codificação de conteúdo executadas automaticamente pela plataforma, cabendo ao programador apenas selecionar a ontologia e/ou a linguagem de sua preferência. Entretanto, os programadores podem também optar pela criação de ontologias e linguagens de conteúdo mais adequadas à aplicação específica.

O Guia do Programador JADE (BELLIFEMINE et al., 2002) lista as seguintes funcionalidades que são oferecidas pelo *framework*:

- Plataforma distribuída de agentes. A plataforma é dividida entre vários *hosts*, sendo que somente uma Máquina Virtual Java necessita ser executada em cada um. Os agentes são implementados como *threads* Java e tem seu ciclo de vida dentro de *Agent Containers*.
- Interface gráfica que permite gerenciar vários agentes e *containers* de *hosts* remotos.
- Ferramentas de *debug* para o desenvolvimento de aplicações multi-agentes baseadas em JADE.
- Mobilidade intra-plataforma de agentes, incluindo a transferência do estado e do código (se necessário) do agente.
- Suporte para a execução de várias atividades de agentes paralelas e concorrentes.
- Plataforma em acordo com as especificações da FIPA, que inclui o AMS e o DF, que são ativados automaticamente na ativação da plataforma.
- Registro e cancelamento de registro de agentes automaticamente no AMS.
- Suporte a ontologias.

Conforme observado por Bellifemine et al. (2002) e Giovanni (2007), em uma plataforma de agentes JADE, só pode existir um AMS. Em cada *host* da plataforma, a Máquina Virtual Java age como um *container* de agentes que fornece um ambiente de execução completo para os agentes e permite que vários deles sejam executados simultaneamente no mesmo *host*. A plataforma é formada pelos vários *containers* que se encontram ativos. O *container* onde o AMS e o DF estão fica conhecido como *main-container*. Este *container* deve estar sempre ativo de modo que os outros possam se conectar a ele assim que forem iniciados, provendo, assim, um ambiente completo para a axecução dos agentes JADE.

Devido a essas características, o primeiro *container* que é iniciado na plataforma passa a ser o *main-container*, e qualquer um que for iniciado posteriormente deve ser informado como encontrá-lo. Se outro *main-container* for iniciado em algum lugar da rede, forma-se uma nova plataforma onde novos *containers* poderão ser registrados. A figura 3.6, encontrada em (GIOVANNI, 2007), mostra um cenários composto por duas plataformas, uma composta por três *containers* e uma composta por um único. Seguindo as especificações da FIPA, o DF e o AMS se comunicam utilizando a linguagem de conteúdo da FIPA, a ontologia *fipa-agent-management* e o protocolo de interação *fipa-request*.

De acordo com o ciclo de vida de uma plataforma de agentes da FIPA, um agente JADE pode se encontrar nos seguintes estados:

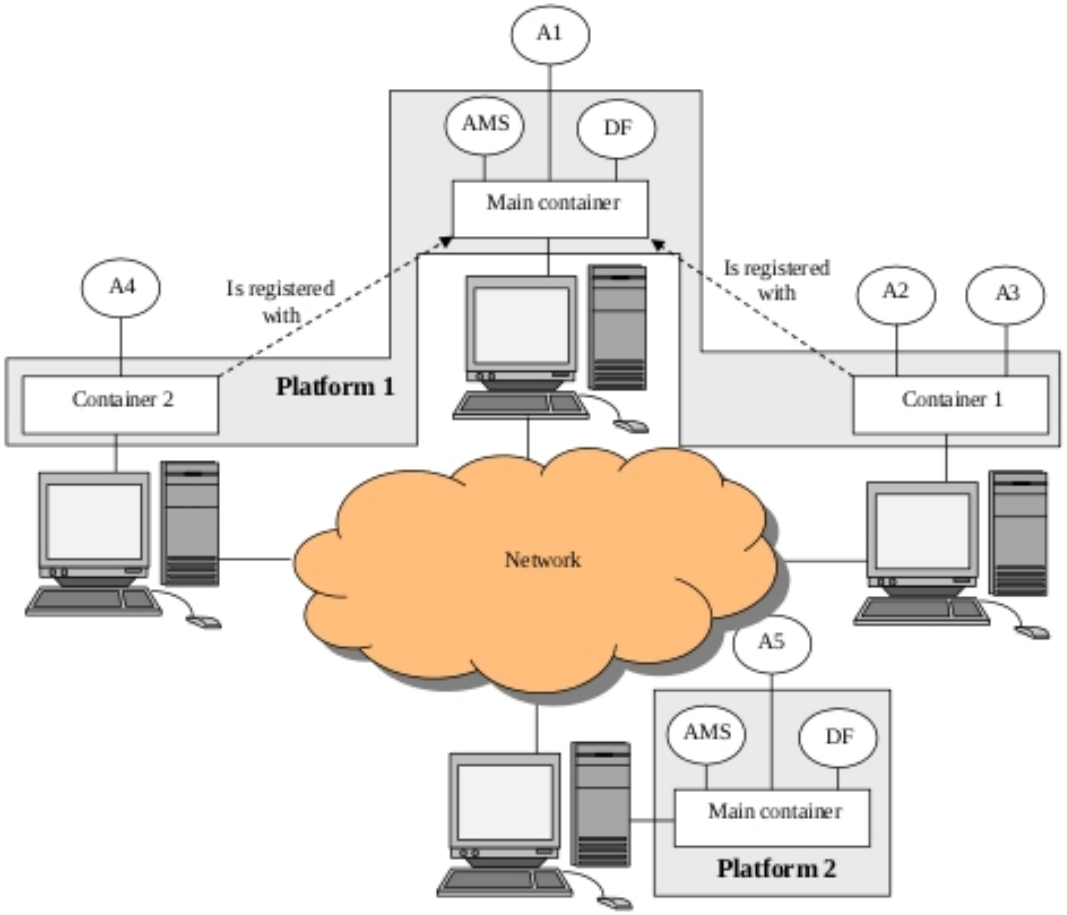

Figura 3.6: Plataformas e *Containers* (GIOVANNI, 2007)

- iniciado: o agente foi construído, mas ainda não se registrou no AMS, não possui nome e não pode se comunicar com outros agentes.
- ativo: o agente está registrado no AMS, possui nome e pode acessar todas as funcionalidades do JADE.
- suspenso: o agente está parado. A *thread* interna está suspensa e nenhum comportamento do agente está sendo executado.
- aguardando: o agente está bloqueado, esperando algum condição ser atingida (como o recebimento de uma mensagem).
- excluído: o agente está morto. A *thread* interna terminou sua execução e o agente não está mais registrado no AMS.
- em trânsito: um agente móvel entra neste estado quando está migrando para uma nova posição. O sistema continua recebendo mensagens para o agente, que serão encaminhadas para a sua nova posição.

A descrição do ciclo de vida dos agentes JADE assim como sua representação, que pode ser observada na figura 3.7, podem ser encontrados em (BELLIFEMINE et al., 2002).

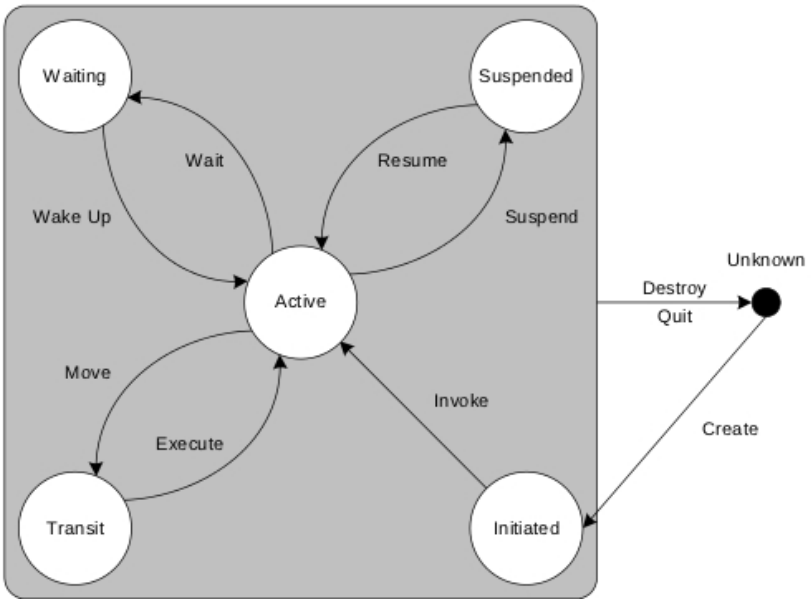

Figura 3.7: O ciclo de vida dos agentes JADE (BELLIFEMINE et al., 2002)

Dadas as propriedades e facilidades fornecidas pelo JADE, a proposta deste trabalho é a implementação da arquitetura de IDS multi-agente descrita em Java, utilizando-se, para isso, o *framework* JADE.

4 ONTOLOGIAS

As ontologias são ferramentas importantes para trabalhar com fontes de informação muito grandes, complexas ou várias fontes, além de serem vantajosas para modelagem de dados em engenharia de *software*. Na área de inteligência artificial, elas se focaram na modelagem de conhecimento, descrevem os conceitos básicos em um domínio e definem relações entre eles (SIRICHAROEN, 2009).

Uma ontologia é uma especificação formal e explícita de uma conceitualização compartilhada. Ela é formal porque é definida de forma declarativa, e dessa forma, entendível para agentes e sistemas; explícita porque os elementos e suas limitações são claramente definidas; conceitualização se refere a um modelo abstrato que não representa nem uma área de conhecimento nem um universo limitado de expressão; compartilhada se refere a um conhecimento consensual, isto é, uma terminologia comum da área modelada, ou acordado pelos desenvolvedores dos agentes de comunicação (AZEVEDO et al., 2010).

Ontologias são artefatos computacionais compostos de um vocabulário de conceitos e suas definições e possíveis propriedades, representando assim, de uma maneira clara, o conhecimento do domínio de interesse. A adoção das mesmas é acompanhada de algumas vantagens. Elas encorajam os desenvolvedores a reusarem o conhecimento, mesmo que com adaptações e extensões. Isso é explicado pelo fato de que a construção de bases de conhecimento é uma das tarefas mais custosas, complexas e vagarosas de um sistema de agentes. Assim, o reuso de ontologias representa um ganho significativo em termos de esforço e investimento.

Outra vantagem está na grande disponibilidade de "ontologias de prateleiras", prontas para uso, reúso e comunicação entre agentes, que podem ser estendidas e complementadas com conceitos de domínios específicos. O acesso *online* a servidores de ontologias, capazes de armazenar milhares de classes e instâncias, dá suporte para companhias ou grupos de pesquisas, já que elas funcionam como ferramentas para manter a integridade do conhecimento compartilhado entre eles, assegurando assim, a existência de um vocabulário uniforme (AZEVEDO et al., 2010).

De acordo com o trabalho de Siricharoen (2009), os conceitos de uma ontologia são grupos, conjuntos ou coleções abstratas de objetos. Conceitos devem fazer referências a objetos (físicos ou lógicos) e às relações em seu domínio de interesse. Eles são representados por substantivos (objetos) e verbos (relações) nas sentenças que descrevem o domínio. Uma classe é definida para um agrupamento conceitual de termos semelhantes. Por exemplo, uma "pessoa" poderia ser representada por uma classe que teria várias subclasses, como "professor".

Os *slots* são utilizados para descrever propriedades de um termo, como as relações entre dois termos. Eles estão ligados aos conceitos e contém meta informação como comen-

tários, restrições e valores padrão. Eles podem representar os atributos das classes, com os possíveis tipos primitivos (inteiro, boolean, String, etc), referências para outros objetos e conjunto de valores desses tipos. Continuando o exemplo anterior, para cada "professor", podemos querer saber seu endereço e salário, ou que curso que ele ministra. Cada "professor", que herda os *slots* do conceito "pessoa" tem um *slot* de endereço, do tipo String. Propriedades das classes, como idade ou endereço podem ser representadas por *slots*, e restrições em propriedades ou relacionamentos entre classes ou *slots*, podem ser representados por *slots* especiais denominados *facets*.

As ontologias podem ser classificadas como leves ou pesadas. As leves incluem conceitos, relações entre eles e as propriedades que os definem. As pesadas adicionam axiomas e restrições às ontologias leves (MALUCELLI, 2006). Um axioma é uma sentença em lógica de primeira ordem que se assume como verdadeira sem uma prova. Na prática, axiomas podem ser usados para fazer referências para sentenças que não podem ser representadas utilizando somente *slots* e valores (SIRICHAROEN, 2009).

As instâncias representam entidades específicas da base de conhecimento do domínio. Um exemplo de base de conhecimento baseada na ontologia "Pessoa" pode conter a base de dados do "Professor" específico "João" e o Curso "T01". Instâncias individuais são os componentes básicos de uma ontologia. Os indivíduos em uma ontologia podem incluir objetos concretos como pessoas, animais, mesas, automóveis, assim como elementos abstratos como números e palavras.

Um relacionamento é utilizado para descrever a relação entre dois ou mais termos. Se são introduzidos relacionamentos na ontologia, descobre-se que essa estrutura hierárquica, inicialmente simples, se torna complexa e significamente mais difícil de interpretar. Uma entidade que é descrita como parte de outra entidade pode ainda ser parte de uma terceira. Consequentemente, entidades podem ter mais de um pai. A estrutura que emerge então, é conhecida como um grafo acíclico dirigido e pode ser observada na figura 4.1.

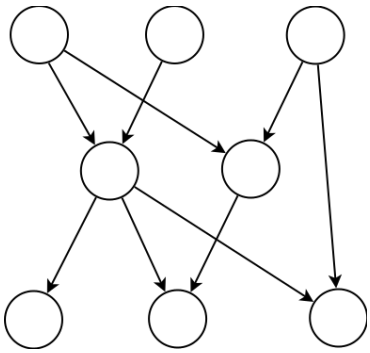

Figura 4.1: Grafo acíclico dirigido

Ontologias ainda incluem tipos adicionais de relações que refinem a semântica que elas modelam. Essas relações geralmente são específicas do domínio e são usadas para respon-

der tipos particulares de perguntas. Como exemplo: se “pessoa” possui o atributo “nome”, então “professor” iria herdar o atributo “nome”. Não é necessário especificar que “professor” possui um “nome”(SIRICHAROEN, 2009). Como exemplo no domínio de “professores”, pode-se definir um relacionamento “vive em” que nos diga onde aquele professor mora. Assim, João mora em Fortaleza. A ontologia pode ainda saber que Fortaleza é no Brasil e que o Brasil é um país da América do Sul.

Ainda de acordo com Siricharoen (2009), a *Web* semântica é simplesmente uma extensão da *Web* atual, na qual a informação é fornecida com significados bem definidos, permitindo uma melhor cooperação entre pessoas e computadores. Como exemplos de tecnologias para a *Web* semântica tem-se XML (*Extensible Markup Language*), RDF (*Resource Description Framework*) e ontologias, cada uma com um papel importante a desempenhar no lançamento e reúso de objetos de conhecimento. Ontologias compartilhadas permitem a diferentes sistemas chegarem a um entendimento comum sobre a semântica de um objeto.

O RDF foi desenvolvido pelo W3C (*World Wide Web Consortium*) para representar informação na *World Wide Web* (ZHANG, 2005). A OWL (*Web Ontology Language*) é uma linguagem de ontologia para a *Web* proposta pelo W3C para a publicação e o compartilhamento de dados e para a automação do entendimento de dados pelos computadores. OWL é planejada e desenvolvida para prover uma linguagem que possa ser usada por aplicações que precisem entender o significado da informação em vez de simplesmente analisar dados com o intuito de exibi-los.

O desenvolvimento de ontologias para a *Web* levou ao aparecimento de serviços que fornecem listas de ontologias, inclusive com a capacidade de buscas. São as chamadas bibliotecas de ontologias. Como exemplo dessas bibliotecas encontramos as seguintes (SIRICHAROEN, 2009):

- A DAML *Ontology Library*, que mantém o legado de ontologias em DAML (DARPA *Agent Markup Language*).
- A Protégé *Ontology Library*. Protégé é um editor de ontologias livre e *open source* e um *framework* para base de conhecimentos. As ontologias do Protégé podem ser exportadas em uma grande variedade de formatos, incluindo RDF, OWL e XML Schema. Protégé é em Java, é estensível e disponibiliza um ambiente *plug-and-play* adequado para a prototipação rápida e para o desenvolvimento de aplicações.
- SchemaWeb é um diretório de esquemas RDF expressado em RDFS, OWL e DAML+OIL.
- Swoogle é um repositório e ferramenta de busca para recursos RDF disponíveis na *Web*, incluindo ontologias. Ele suporta busca por ontologias contendo termos específicos, que

pode ser refinado para encontrar ontologias onde tais termos ocorram como classes ou propriedades.

- A OntoSelect *Ontology Library* fornece um ponto de acesso para ontologias em qualquer tópico ou domínio, que são atualizadas constantemente, com um suporte automático para seleção. Diferente das bibliotecas de ontologia DAML e SchemaWeb, OntoSelect não é baseada principalmente em um registro estático de ontologias publicadas, mas inclui um procedimento de rastreamento que monitora a *web* por qualquer ontologia recém-publicada nos formatos RDF(S), DAML ou OWL.

- Ontaria é um diretório com busca de dados de *web* semântica, com foco em vocabulários RDF com ontologias OWL. Ele foi produzido principalmente para pessoas criando conteúdo RDF que querem entender melhor quais vocabulários estão disponíveis e como eles estão sendo usados. Ontaria pode ser útil para encontrar e explorar conteúdos RDF arbitrários.

Encontrar ontologias já existentes e prontas para o uso é importante para evitar a criação de novas desnecessariamente. Apesar de serem bastante parecidas, existem algumas diferenças entre as linguagens de ontologias e as linguagens orientadas a objetos. Ambas são compostas por classes, propriedades e instâncias e as classes podem ser distribuídas em hierarquias com herança. As diferenças entre as duas linguagens podem ser observadas na tabela 4.1.

Ontologias são comparáveis a esquemas de bancos de dados, mas representam um modelo de informação muito mais rico do que esquemas normais. Elas fornecem esquemas de meta-dados, oferecendo um vocabulário controlado de conceitos. No centro de ambos os modelos de objetos e ontologias estão objetos dentro de um domínio do problema (SIRICHAROEN, 2009).

Para modelar uma ontologia, Malucelli (2006) propôs uma abordagem formada pelos seguintes passos:

- Determinar o domínio e o escopo da ontologia, o tipo de perguntas que a ontologia deverá responder e quem a usará e fará a manutenção da mesma;

- Considerar o reúso de ontologias existentes;

- Enumerar termos importantes na ontologia;

- Definir as classes e a hierarquia de classes;

- Definir as propriedades das classes;

- Definir as *facets* como tipos de valores, valores permitidos e cardinalidade;
- Criar instâncias das classes.

Tabela 4.1: Comparação entre linguagens orientadas a objetos e linguagens de ontologias

	Linguagens orientadas a objeto	**Linguagens de ontologias**
Classes e Instâncias	Classes são consideradas tipos para instâncias	Classes são como conjuntos de indivíduos
	Cada instância tem uma única classe como seu tipo	Cada indivíduo pode pertencer a várias classes
Propriedades, atributos e valores	Propriedades são definidas localmente para uma classe (e para suas subclasses via herança)	Propriedades são entidades auto-suficientes que podem existir sem as classes
	Os valores das instâncias precisam ser do tipo correto e atender às restrições.	As instâncias podem ter valores arbitrários
	As classes descrevem grande parte de seus significados e comportamentos por meio de funções e métodos.	Classes descrevem seus significados por meio de declarações.
	Classes podem encapsular seus conteúdos para ter acesso privado	Todas as partes dos arquivos são livres e podem ser utilizados por qualquer parte do sistema

4.1 Ontologias em Sistema Multi-agentes

Ontologias têm contribuído para facilitar a comunicação e o processamento de informações semânticas em sistemas baseados em agentes, promovendo assim, interoperabilidade entre sistemas quando elas representam dados compartilhados por várias aplicações. Com uma base de conhecimentos tratando de itens relativos a segurança, as corporações serão capazes de desenvolver e implantar mais facilmente mecanismos para proteção, correção e prevenção de acordo com seus requisitos particulares (AZEVEDO et al., 2010).

A comunicação entre qualquer número de agentes em um sistema multi-agentes compreende a troca de pequenas quantidades de informação na forma de mensagens, que foram pre-formatadas para ficar de acordo com as convenções de uma ACL que foi acordada entre os agentes anteriormente, antes que qualquer dado de verdade tenha sido enviado. Esse dado, embutido em uma mensagem ACL, constitui o conteúdo da mensagem e é expressado em uma linguagem de conteúdo apropriada, formando uma expressão de conteúdo. Essa representação,

geralmente uma *string* ou uma sequência de *bytes*, pode não ser considerada como apropriada para os objetivos internos de um agente.

Assim, cada troca de informação que acontecer necessita que as partes interessadas utilizem agentes para converter suas representações internas de informação em uma expressão de conteúdo equivalente que fica encapsulada dentro de uma mensagem ACL antes de ser enviada ao destino, que realiza a mesma operação, só que invertida, com o *overhead* adicional da checagem semântica para ver se o conteúdo da mensagem recebida pode ser interpretado. Esse procedimento ganha importância no desenvolvimento de aplicações abertas, nas quais nem todas as mensagens dos agentes são esperadas que carreguem conteúdo semanticamente correto e consistente (FABRICI, 2006).

A habilidade de compartilhar um entendimento de um assunto, provida pelas ontologias, possibilita que um *software* aumente sua capacidade para analisar e decidir sobre a informação. Ao aplicar ontologias ao problema de detecção de intrusão, o poder e a utilidade das mesmas não são percebidos pela simples representação dos atributos dos ataques, mas sim pelo fato de que podem expressar relacionamentos entre dados coletados e utilizar esses relacionamentos para deduzir que um dado em particular representa um ataque de um determinado tipo. Assim, especificar uma representação de ontologia separa o modelo de dados que define uma intrusão da lógica do IDS.

Essa separação permite que IDS não homogêneos compartilhem informação sem um entendimento anterior sobre a semântica dos dados. Para permitir esse compartilhamento, uma instância da ontologia é compartilhada entre os IDS na forma de um conjunto de declarações. Se o recipiente não entende algum aspecto dos dados, ele acessa a ontologia com o objetivo de interpretar e usar os dados da forma como foi idealizada pelo seu criador. Dessa forma, as ontologias podem ser utilizadas não só para prover ao IDS a habilidade de compartilhar um entendimento comum da informação mas também para fornecer uma melhor capacidade para analisar instâncias e raciocinar sobre dados que representam intrusões (UNDERCOFFER et al., 2004).

Uma ontologia define a estrutura dos elementos de um domínio por meio de esquemas. Especificamente, os esquemas precisam ser definidos para três tipos de elementos, sendo esses predicados, conceitos e ações de agentes, enquanto uma classe precisa ser implementada para cada um desses elementos definidos na ontologia. Como uma coleção de esquemas tipicamente não se desenvolve durante o ciclo de vida de um agente, a ontologia é geralmente definida como um objeto *singleton* que pode ser compartilhado por todos os agentes existentes em uma plataforma executada por uma única máquina virtual java, por exemplo. Ontologias definidas pelos usuários estendem a ontologia básica, que define esquemas para todos os tipos primitivos, os tipo agregado e alguns tipos genéricos específicos do *framework*.

Cada esquema incluído em uma ontologia é associado com uma classe correspon-

dente que é esperada que seja coerente com o esquema via implementação de uma interface adequada, possuindo as relações de herança corretas e possuindo os campos membros corretos com métodos de acesso (CAIRE; CABANILLAS, 2004). Cada slot em um esquema é definido por um nome e um tipo de dados, que é significante para objetivos de validação futuros. *Slots* em um esquema são declarados como opcionais ou obrigatórios, significando que os requerimentos para esse tipo de dados serão preenchidos por uma mensagem a ser declarada como válida. *Slots* podem ainda ter cardinalidade maior que um, o que significa que mais de um elemento é esperado para ser inserido neles (FABRICI, 2006).

4.2 Ontologias e o *Framework* JADE

O *framework* JADE utiliza um modelo de referência de conteúdo (*Content Reference Model* ou CRM) derivado da semântica da ACL da FIPA que requer que expressões de conteúdo inseridas dentro de mensagens ACL tenham características próprias de acordo com as performativas das mensagens (INFORM, REQUEST). No CRM, são encontrados os seguintes elementos distintos (AART et al., 2002):

- Conceitos que expressam entidades que existem no mundo.
- Predicados, expressando o estado de uma parte do mundo, que podem ser falsos ou verdadeiros.
- Ações de agentes expressando aquilo que um agente pode ser requisitado a executar.
- Primitivas são expressões atômicas como Strings e inteiros.
- Agregados, expressando entidades que são grupos de outras entidades.
- Expressões Referenciais de Identificação (*Identifying Referential Expressions* ou IRE) são expressões que identificam entidades para quais um determinado predicado é verdadeiro. Elas são utilizadas geralmente em consultas, isto é, como o conteúdo de uma mensagem QUERY_REF.
- Variáveis expressando um elemento genérico não conhecido anteriormente e usado tipicamente dentro da IRE.

Somente predicados, ações de agentes e IRE são conteúdos significativos para alguma mensagem ACL. Na verdade, um predicado pode ser o conteúdo de uma mensagem INFORM, e uma ação de agente pode ser o conteúdo de uma mensagem REQUEST e um IRE pode ser o conteúdo de uma mensagem QUERY_REF. Somente termos podem ser valores significativos para os *slots* de um conceito (AART et al., 2002).

Os termos podem ser classificados em seis tipos, como observado na figura 4.2.

Um conteúdo com significado está em conformidade com um conjunto de regras predefinidas em uma ontologia. Com uma ontologia propriamente definida, um programador pode alavancar o poder de validação automática do conteúdo de mensagens provido por um *framework* de agentes compatível com a FIPA, como o JADE. No JADE, as operações de conversão e validação são realizadas por um objeto gerenciador de conteúdo dedicado. A classe do objeto gerenciador de conteúdo fornece todos os métodos necessários para conversão de conteúdo, mas na verdade delega todo o trabalho para instâncias especiais de objetos de ontologia e codec de conteúdo.

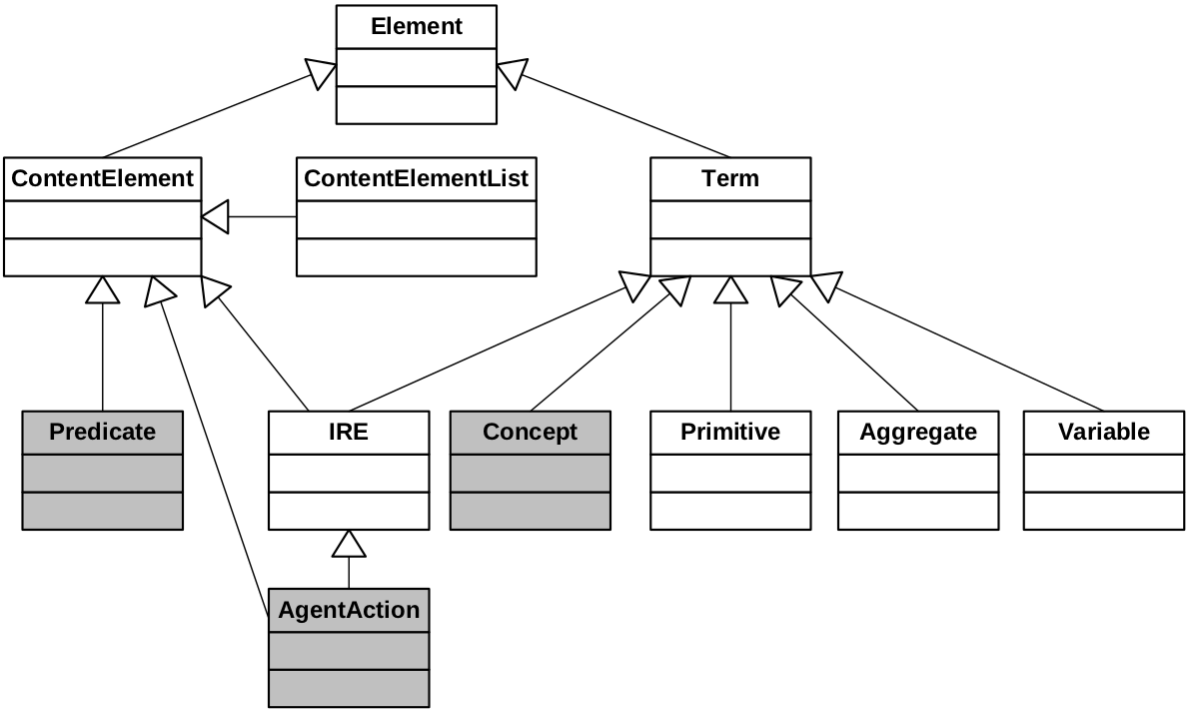

Figura 4.2: Classificação dos termos de uma ontologia (FABRICI, 2006)

Os agentes escritos em Java (como no JADE), podem representar a informação internamente como objetos, dessa forma, dois agentes diferentes possivelmente terão jeitos diferentes de fazer isso. Assim, toda vez que um agente envia dados para outro, o remetente precisa converter da sua representação interna para a expressão de conteúdo ACL correspondendo e o destinatário precisará fazer o inverso ao receber a mensagem, além de realizar checagens semânticas para verificar que a informação possui significado e atende às regras estabelecidas pela ontologia a qual os agentes concordaram em seguir. O suporte a linguagens de conteúdos e ontologias do JADE permite que sejam realizas todas as conversões citadas.

Enquanto o codec realiza as traduções sintáticas com suporte direto a duas linguagens de conteúdo (uma linguagem de conteúdo SL codificada em *strings* que possam ser lidas por humanos; outra é uma linguagem LEAP codificada em *bytes*, que não pode ser lida, defi-

nida somente para interação entre agentes JADE), a ontologia valida toda a informação sob uma perspectiva semântica, que requer que todos os elementos de uma expressão de conteúdo sejam conhecidos e classificados. Com essa informação, uma ontologia serve basicamente como um dicionário ou vocabulário para comunicação entre agentes. A especificação da linguagem de comunicação de agentes da FIPA dita que todas as mensagens apresentem a semântica em acordo com sua performativa (tipo de ação tomada), na qual o modelo de referência de conteúdo discerne entre predicados e termos (FABRICI, 2006).

Como já mencionado, uma ontologia definida pelo usuário em JADE geralmente estende a ontologia básica específica da plataforma, mas não se limita a isso, já que ela pode estender um número de ontologias predefinidas, não estritamente relacionadas com esse domínio em particular. O suporte à linguagem de conteúdo do JADE e suas ontologias também possibilitam ao usuário adicionar restrições adicionais aos predicados, conceitos e ações de agentes. Para adicionar uma restrição (*facet*) a um elemento é preciso somente definir uma nova classe que implementa a interface *Facet*.

5 ARQUITETURA PROPOSTA PARA DETECÇÃO DE INTRUSÃO

Este trabalho propõe uma arquitetura distribuída, baseada na proposta feita por Zaidi (2011) na qual os agentes executam a tarefa de detecção por comunicação e colaboração, utilizando-se para isso de uma ontologia global. O sistema multi-agentes proposto pelo autor utilizaria cinco classes de agentes: agente sensor, agente gerente, agente de ontologia, agente atuador e agente analisador.

O agente sensor capturaria os dados e realizaria a formatação dos mesmos. O agente gerente seria responsável por gerenciar o ciclo de vida dos agentes, verificando regularmente a disponibilidade dos mesmos e dos serviços, mantendo uma visão geral coerente da arquitetura em termos da ausência e presença de serviços e agentes. Ele poderia demandar aos outros agentes de informação locais relacionadas a atividades suspeitas.

O agente da ontologia promoveria a partilha do conhecimento entre os diferentes agentes da arquitetura, oferecendo os serviços de publicação de ontologias e o de verificação semântica dos conhecimentos produzidos por outros agentes. O agente atuador iria registrar seu serviço com um agente gerente e responder invocações de outros agentes.

O agente analisador trataria os dados gerados pelo agente sensor, se baseando em dados salvos localmente em uma base de conhecimento ou utilizando dados disponíveis um uma ontologia global. Ele aplicaria as regras de detecção e, de acordo com o resultado da análise, iniciaria um tratamento externo.

O autor não fez menção ao uso de agentes móveis em sua arquitetura e não detalhou o funcionamento interno de seus agentes. O compartilhamento de informação na arquitetura seria realizado por meio de uma ontologia global que estaria disponível em um servidor web, acessível para os agentes.

A arquitetura proposta neste trabalho é organizada em dois níveis. O primeiro é o sistema multi-agente de detecção composto de seis classes de agentes: agente sensor, agente analisador, agente gerente, agente de ontologia, agente ontologia global e agente atuador. Os agentes gerente e de ontologia são agentes móveis, enquanto o agente sensor, atuador e analisador são agentes fixos dos *hosts* da rede e o agente ontologia global é um agente fixo que fica localizado em um único *host*.

O segundo nível da arquitetura é formado por uma plataforma de agentes composta de agentes de supervisão e agentes de comunicação. O *framework* escolhido para a implementação da arquitetura (JADE) é responsável por este segundo nível, isto é, pela supervisão e comunicação entre os agentes.

Nos processos de detecção, os agentes utilizam dois tipos de conhecimento: os co-

nhecimentos locais e os conhecimentos da ontologia. Os conhecimentos locais são derivados dos eventos ocorridos na rede e dos dados coletados ou aprendidos durante o processo de análise. Os conhecimentos da ontologia, ou conhecimentos permanentes, representam os conhecimentos necessários para a aplicação de uma política de segurança e para assegurar a detecção de intrusão.

Na arquitetura, cada nó que faz parte do sistema de detecção de intrusão é hospedeiro de um agente sensor e um agente analisador. Durante o funcionamento do sistema, de acordo com os dados passados ao agente analisador, este pode criar agentes de ontologia e agentes gerentes nas quantidades que considerar como suficientes para verificar as suspeitas, assim como pode criar agentes atuadores para gerar alarmes. O nó que contém o *main-container* do JADE é hospedeiro do agente ontologia global, assim como da própria ontologia que o agente acessa.

5.1 Responsabilidades dos Agentes

A seguir, são detalhadas as interações entre os agentes e os papéis que cada agente desempenha no processo de detecção. O funcionamento da arquitetura pode ser visualizado na Figura 5.1. Na seção seguinte é detalhada a forma como foi implementada a arquitetura para a realização de testes.

5.1.1 Agente Sensor

O agente sensor é o inicializador do tratamento de detecção. Ele é um agente fixo que está na base da cadeia de funcionamento, capturando o tráfego de rede bruto como um *sniffer*, selecionando nesse tráfego as informaçõs dos pacotes que são interessantes para o agente analisador, transformando essa informação para um formato predefinido que pode ser entendido por esse agente e deixando ela disponível para que ele execute a análise. Ele fica ativo durante todo o funcionamento do sistema de detecção, sempre atualizando as informações fornecidas ao agente analisador.

5.1.2 Agente Analisador

O agente analisador é um agente fixo, responsável por analisar os dados gerados pelo agente sensor e por controlar o processo de detecção em si, aplicando as regras de detecção. A análise realizada por ele pode utilizar somente os dados locais gerados pelo agente sensor, assim como os dados disponíveis na ontologia global, ou ainda, os dados locais de outros *hosts* que fazem parte do sistema. De acordo com o resultado da análise dos dados gerados pelo agente sensor, se houver confirmação ou suspeita de intrusão, o agente analisador irá iniciar um

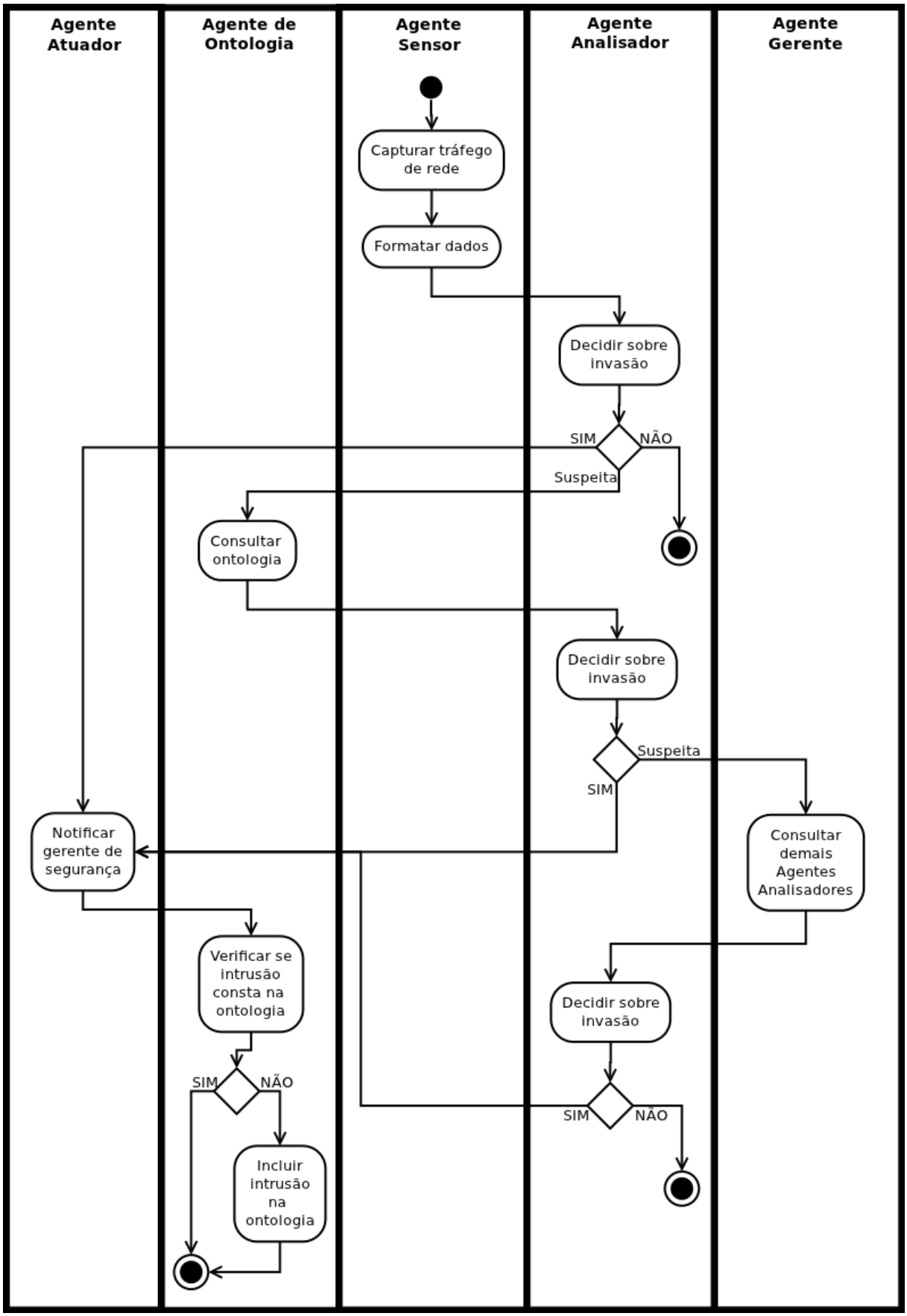

Figura 5.1: Funcionamento do IDS Proposto

tratamento externo.

Dois casos são possíveis: no primeiro caso, uma atividade maliciosa é confirmada e ele comunicará o resultado ao agente atuador para executar as ações necessárias; no segundo caso, a atividade é julgada como suspeita pelas regras de detecção do sistema e ele demandará por informações suplementares para verificar a natureza da atividade. Nesse caso, são acionados primeiramente os agentes de ontologia que irão providenciar o acesso às informações contidas na ontologia para verificar se a suspeita realmente representa uma intrusão.

Caso a consulta à ontologia resulte em informações que confirmem as suspeitas, o agente analisador aciona o agente atuador. Caso a consulta à ontologia não resulte em informações que confirmem as suspeitas, são acionados os agentes gerentes. Caso os agentes gerentes confirmem as suspeitas, o agente analisador aciona o agente atuador. Caso o agente gerente não chegue a nenhuma conclusão sobre as atividades suspeitas, a informação dessa atividade fica armazenada no agente analisador que descobriu a mesma para possíveis consultas por demais agentes gerentes de outros *hosts* do sistema.

Sempre que o sistema conclui que uma suspeita representa realmente uma ameaça, além se ser chamado um agente atuador é chamado também um agente de ontologia com o objetivo de que seja acrecentado o atacante na ontologia global (se ele ainda não constar na mesma). O agente analisador fica ativo durante todo o funcionamento do sistema de detecção, analisando as informações providas pelo agente sensor (que são constantemente atualizadas) e esperando o retorno dos agentes de ontologia e gerentes que foram criados devido à sua análise. Além disso, ele precisa estar sempre disponível para responder às solicitações dos agentes gerentes gerados em outros *hosts* do sistema, devido às solicitações de seus respectivos agentes analisadores.

5.1.3 Agente de Ontologia

Os agentes de ontologia são agentes móveis, responsáveis pelo compartilhamento de dados entre os diferentes agentes do sistema. Ao ser criado para verificar uma atividade suspeita, um agente de ontologia se move para o *container* principal (se já não estiver localizado nesse *container*), onde se localiza o agente ontologia global e a ontologia global e repassa para esse agente as informações relativas à atividade para que possa ser feito a consulta à ontologia global.

Após receber o resultado da consulta, ele se move de volta para seu *host* original, repassa a resposta da consulta ao agente analisador cuja análise foi responsável pela sua criação e encerra a sua execução. Ao ser criado para adicionar uma intrusão na ontologia global, o agente de ontologia se move para o *container* principal e repassa a informação da intrusão para o agente ontologia global incluir na ontologia. Depois ele retorna ao *host* e, como não possui

mais nenhum papel a desempenhar, tem sua execução encerrada.

5.1.4 Agente Gerente

Os agentes gerentes também são agentes móveis, responsáveis por gerenciar o processo de compartilhamento de informação diretamente entre os *hosts* que compoem o sistema, sem a utilização da ontologia global. Ao ser criado para verificar uma atividade suspeita, um agente gerente se move entre os *hosts*, um a um, consultando seus respectivos agentes analisadores até que se confirmem as suspeitas com as informações locais ou que sejam percorridos todos os *hosts* do sistema, sem que se chegue a uma confirmação. Após desempenhar essas atividades, ele se move de volta para seu *host* original, repassa a resposta conseguida com o seu percurso pelo sistema de detecção ao agente analisador cuja análise foi responsável pela sua criação e encerra a sua execução.

5.1.5 Agente Atuador

Os agentes atuadores são agentes fixos, responsáveis pelas ações tomadas quando o sistema detecta que realmente ocorreu um ataque à rede monitorada. Essas ações podem ser simples alertas para os usuários ou para o gerente de segurança, ou ainda, numa abordagem mais ativa, podem ser iniciados processos de encerramento de conexões, por exemplo. Toda a vez que uma intrusão é confirmada, o agente analisador aciona um agente atuador, que executa a ação programada para a intrusão e depois encerra a sua execução.

5.1.6 Agente Ontologia Global

O agente ontologia global é um agente fixo, localizado no *container* principal, responsável por manter e realizar consultas ao repositório de informações da arquitetura sobre intrusões. Todas as suas interações com o sistema de detecção são realizadas por meio dos agentes de ontologia que solicitam que ele realize buscas em seu repositório de intrusões, para as quais ele produz as repostas adequadas, e que ele inclua intrusões detectadas no repositório. Ele fica ativo durante todo o funcionamento do sistema de detecção.

5.2 Implementação da arquitetura

Com o objetivo de verificar o funcionamento da arquitetura proposta, foi realizada a implementação da mesma utilizando a linguagem de programação Java, com o *framework* JADE. Os detalhes da implementação de cada agente e da ontologia são apresentados a seguir. No Apêndice A podem ser observados, ainda, alguns trechos do código fonte dos agentes

implementados.

A disposição dos agentes que formam a arquitetura, tanto fixos quanto móveis, podem ser observados na Figura 5.2, que representa um IDS distribuído formado pelos agentes descritos nesta seção. O IDS, neste caso, é formado por três nós, cada um contendo um *container* de agentes. Cada *container* é representado por uma cor, e cada agente criado em um *host* possui a mesma cor de seu *container* de origem. Podem ser observados agentes gerentes e de ontologia que já migraram e encontram-se em *hosts* diferentes daqueles onde foram criados, assim como agentes que estão se movendo, migrando entre nós.

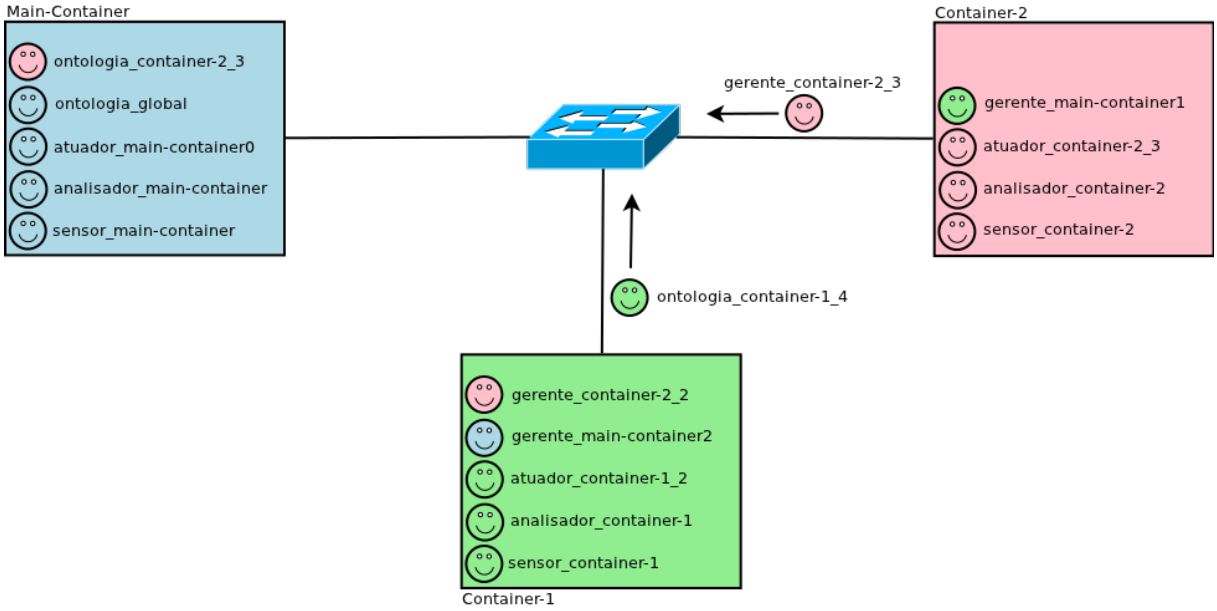

Figura 5.2: Arquitetura proposta em funcionamento

5.2.1 Implementação do Agente Sensor

O agente sensor é criado juntamente com o lançamento do sistema de detecção. Ele salva as informações obtidas na captura do tráfego de rede em um arquivo do tipo ".txt" para deixá-las disponíveis para o agente analisador. A primeira informação que é salva no arquivo é a data e hora do início da captura de pacotes na rede. Após isso, ele utiliza a biblioteca Jpcap para conseguir uma lista das interfaces de rede disponíveis na máquina em que estiver hospedado. Com essa informação, ele inicia a captura dos pacotes na(s) interface(s) escolhidas.

Jpcap é uma biblioteca do Java para a captura e o envio de pacotes de rede que foi testada nos sistemas operacionais Microsoft Windows (98/2000/XP/Vista), Linux (Fedora, Mandriva, Ubuntu), Mac OS X (Darwin), FreeBSD e Solares. Ela permite ao programador capturar pacotes Ethernet, IPv4, IPv6, ARP/RARP, TCP, UDP e ICMPv4. O código da biblioteca é aberto e ela possui uma licensa GNU LGPL (FUJII, 2007).

A seguir, o agente sensor utiliza o comando "createNewAgent" do JADE para criar

um agente analisador, que terá como nome “analisador_” seguido do nome do *container* em que o mesmo estiver sendo criado. É passado como parâmetro na criação do agente analisador o nome do arquivo texto onde estão sendo armazenadas as informações capturadas. Nesse arquivo, as informações são salvas no seguinte formato: é utilizado o método Packet.toString( ) da biblioteca Jpcap e à *string* respondida pelo mesmo é adicionada a data e hora do recebimento do pacote, cada linha do arquivo representando um pacote capturado. O método Packet.toString( ) fornece informaçõs sobre o pacote, como: endereço de origem, endereço de destino, protocolo, prioridade, tempo de vida, número de sequência e, se for o caso, portas de origem e de destino, entre outras. Como exemplo da representação de um pacote TCP no arquivo pode-se observar a Figura 5.3.

```
1337964528:789889 /192.168.0.122->/192.168.0.163 protocol(6) priority(0)  hop(64)  offset(0)
ident(31088) TCP 44519 > 1099 seq(1954952604) win(115) ack 3038657962  25-05-2012 13:48:48
```

Figura 5.3: Pacote TCP representado no arquivo

5.2.2 Implementação do Agente Analisador

O agente analisador inicia o tratamento das informações geradas pelo agente sensor a partir do início do arquivo “.txt” que é passado como argumento na sua criação. A primeira informação que ele retira do arquivo é a data e a hora do início da captura de pacotes pelo agente sensor, considerada, por ele, como o horário no qual o sistema de detecção entrou em funcionamento. Ele trata os pacotes capturados pelo agente sensor um a um e, de acordo com o protocolo do pacote, extrai as informações de endereço de origem e endereço de destino, portas de origem e destino (se for o caso), data e hora que o pacote foi capturado, e calcula e armazena a quantidades de pacotes desse tipo (com as mesmas características excetuando a data e a hora) que já foram analisados, o intervalo de tempo entre a captura do último pacote com essas características, se a origem do pacote é considerada como suspeita de gerar ataques ao sistema e se o pacote é considerado como sendo um pacote de controle do JADE.

Para detectar atividades suspeitas foi utilizado o seguinte método proposto por Zaidi (2011): para cada serviço, o número de pacotes capturados na rede é contado pelo agente analisador e dois limites predefinidos (L_{min} e L_{max}) são utilizados para decidir sobre a natureza do tráfego. Se o número de pacotes observados for inferior a L_{min} então o tráfego é considerado como normal, neste caso, o agente analisador irá guardar o valor da quantidade de pacotes capturados para o caso de mais pacotes desse serviço específico serem capturados futuramente. Caso contrário, se o número excede L_{max} então o tráfego é considerado como anormal (é uma atividade maliciosa). O terceiro caso é quando o número de pacotes é maior do que L_{min} e inferior a L_{max}, caso no qual o tráfego será julgado como suspeito (provavelmente malicioso). Por consequência, análises suplementares deverão ser efetuadas para decidir sobre a natureza

do tráfego, então será necessário colaboração com outros agentes para obter mais informações sobre esse serviço específico.

Nos testes realizados em laboratório, os limiares de detecção L_{min} e L_{max} foram definidos com os valores de 100 e 1000, respectivamente, valores estes fruto de observação do comportamento das máquinas e da rede no laboratório específico. Esses valores são válidos somente para o ambiente preparado para a realização dos testes de detecção e precisariam ser reavaliados no caso de serem utilizados em outras situações.

Foi considerado também como fonte de informação para a decisão sobre ataques, nos agentes analisadores, a quantidade de tempo que determinado serviço ficou sem ser acessado até que se perceba acesso ao mesmo no tráfego de rede (novamente ou pela primeira vez desde que o IDS foi acionado), semelhante à proposta de detecção apresentada por Mahoney (2003), embora um pouco simplificada. O autor considerou o fato de que as taxas de pacotes nos eventos em redes variam de forma imprevisível, utilizando então um modelo dependente do tempo, no qual a probabilidade de um evento ser uma anomalia depende do tempo passado desde que ele ocorreu pela última vez, em vez de depender da taxa média de pacotes.

A ideia é que se um atributo assume um valor não visto recentemente, então o acesso é suspeito. A implementação do autor contou com um período de treinamento, e considerou como anomalia valores diferentes daqueles encontrados nesse período. Para cada atributo novo observado é atribuído um valor que depende do tempo desde que o atributo foi considerado como anomalia, e também do número de vezes que determinada atividade é observada. Ele desconsidera em sua análise pacotes que não sejam do protocolo IP (como ARP) e tráfego de saída, com o objetivo de reduzir a quantidade de falsos positivos e aumentar a velocidade de processamento. No entanto, o próprio autor observou que, devido a essas considerações, alguns ataques deixaram de ser detectados.

Baseado nessa abordagem, na implementação dos agentes analisadores do presente trabalho, se um serviço ficar sem ser acessado por um intervalo de tempo maior que um valor determinado e passar a ser acessado, esse acesso é considerado suspeito e o *host* que está acessando o serviço passa a ser considerado como um possível atacante. Diferente de Mahoney (2003), não foi desconsiderado nenhum tipo de pacote ou de tráfego, com a intenção de deixar o sistema mais genérico e capaz de identificar mais tipos de ataques. Outra diferença é que esta implementação não conta com um período de treinamento, fazendo então uma classificação de anomalias baseada somente no intervalo de tempo. No entanto, esse critério classifica somente como anomalias ou tráfego normal, não identificando intrusões diretamente. Essa identificação é realizada pelo sistema distribuído, pelas interações entre os agentes, o que, como utiliza diversas fontes de informações, espera-se que reduza as taxas de falsos positivos.

Nos testes realizados em laboratório, esse intervalo de tempo foi definido com o valor de 500s, ou oito minutos e vinte segundos, intervalo esse que, também, foi coerente com

o laboratório montado para os testes e teria que ser reavaliado para ser utilizado em outras situações.

No caso de perceber um ataque acontecendo, isto é, de detectar uma quantidade de pacotes superior a L_{max}, o agente analisador cria um agente atuador. Ao criar o agente, são passadas como argumento as informações sobre o protocolo dos pacotes que geraram a detecção, os endereços de origem e destino destes pacotes e, se for o caso, as portas de origem e destino. Após criar o agente atuador, é criado um agente de ontologia, sendo passado como argumento o seu propósito ("inclusao") que é para incluir um ataque na ontologia, assim como as informações passadas ao criar o agente atuador.

No caso de detectar uma quantidade de pacotes superior a L_{min} e inferior a L_{max}, ou detectar um intervalo de tempo de captura de pacotes deste tipo superior ao valor configurado para geração de suspeitas, o agente analisador considera essa detecção como uma suspeita. Dessa forma, ele marca a representação deste tipo de pacotes como suspeita e cria um agente de ontologia, sendo passado como argumento o seu propósito ("consulta") que é para consultar a ontologia global, assim como as informações sobre a suspeita, como protocolo dos pacotes, endereços de origem e destino e, se for o caso, portas de origem e destino.

Cada agente criado recebe um nome de acordo com a seguinte regra: o nome é formado pela concatenação da palavra "atuador_", "ontologia_" ou "gerente_" (dependendo do tipo de agente) com o nome de seu *container* nativo, seguido de um número que identifica esse agente de maneira única (iniciado como zero e incrementado em uma unidade a cada vez que um agente do tipo é chamado).

Após analisar cada pacote, o agente analisador verifica se recebeu alguma mensagem, que de acordo com a performativa, pode ser diferenciada como sendo uma mensagem de um agente de ontologia nativo, informando que não foi possível confirmar uma suspeita; uma mensagem de um agente de ontologia nativo, informando que uma suspeita foi confirmada; uma mensagem de um agente gerente gerado em outro *host* para verificar a lista de suspeitos; uma mensagem de um agente gerente nativo, informando que não foi possível confirmar determinada suspeita com os demais *hosts* do sistema; ou ainda, uma mensagem de um agente gerente nativo, informando que foi confirmada uma suspeita. Caso o agente receba uma mensagem, ele executa o tratamento correspondente, caso não receba nada, ele analisa o próximo pacote de rede descrito no arquivo gerado pelo agente sensor.

Ao receber uma mensagem de um agente de ontologia nativo informando que não foi possível confirmar uma suspeita, ele extrai as informações sobre o protocolo e os endereços e portas de origem e destino e as utiliza como parâmetros para criar um agente gerente.

Ao receber uma mensagem de um agente de ontologia nativo informando que uma suspeita foi confirmada, ele extrai as informações sobre o protocolo e os endereços e portas de

origem e destino e as utiliza como parâmetros para criar um agente atuador.

Ao receber uma mensagem de um agente gerente gerado em outro *host* para verificar a lista de suspeitos, ele extrai a informação do protocolo e do endereço de origem da suspeita e compara com suas informações internas, averiguando se este endereço corresponde ao endereço de origem de algum de seus pacote suspeitos, verificando assim se a informação passada na mensagem corresponde a um pacote oriundo de um *host* que ele também considera como sendo suspeito de executar ataques. Caso seja confirmada a suspeita, ele cria uma mensagem resposta com o conteúdo "Ocorreu intrusao". Caso não seja possível confirmar a suspeita, ele cria uma mensagem resposta com o conteúdo "Nao detectado".

Ao receber uma mensagem de um agente gerente nativo informando que não foi possível confirmar determinada suspeita com os demais *hosts* do sistema, ele considera que não está ocorrendo uma intrusão naquele momento e continua sua execução normal.

Ao receber uma mensagem de um agente gerente nativo informando que foi confirmada uma suspeita, ele extrai as informações de protocolo, endereços e portas da mensagem e cria um agente atuador, passando essas informações como parâmetros. Após criar o agente atuador, é criado um agente de ontologia, sendo passado como argumento o seu propósito ("inclusao") que é para incluir um ataque na ontologia, assim como as informações passadas ao criar o agente atuador.

5.2.3 Implementação do Agente de Ontologia

Ao ser criado, um agente de ontologia verifica se ele se encontra localizado no *container* principal (utilizando o comando "getContainerController( ).getContainerName( )" do JADE) e, se não estiver, se move para lá, pois é nele que se encontra a ontologia global. Essa migração é realizada utilizando o comando "doMove( )" do JADE, passando como parâmetro a identificação do *container* de destino.

A seguir, ele verifica os argumentos passados na sua criação para determinar se o seu propósito é de "consulta" ou de "intrusao". Se o propósito for "consulta", significa que o objetivo do agente é consultar a ontologia global para verificar se uma suspeita corresponde a alguma intrusão salva na ontologia. Se o propósito for "intrusao", significa que o objetivo do agente é incluir na ontologia global uma intrusão que já foi confirmada. É gerada uma mensagem ACL, tendo como destinatário o agente ontologia global, cujo o conteúdo é uma instância da classe "suspeita" que é composta por duas instâncias da classe "*host*" (atacante e alvo) que identificam os protocolos e endereços de rede, e por duas *strings* que identificam números de portas opcionais.

Os dois *hosts* e as portas correspondem aos valores passados como argumentos na criação do agente de ontologia. Essa mensagem é escrita na linguagem de ontologia definida

para o sistema de detecção. As classes “host” e “suspeita” correspondem a declarações da linguagem da ontologia global. Se o propóstio do agente for de consulta, a mensagem gerada apresentará a performativa QUERY_IF, se o propósito for de inclusão, a mensagem gerada apresentará a performativa INFORM.

Se a mensagem enviada for para consulta (performativa QUERY_IF), o agente de ontologia fica aguardando a resposta da mensagem enviada e, ao recebê-la, verifica se ela confirmou a suspeita de intrusão ou não. A seguir, ele migra de volta para seu *host* nativo, utilizando o comando doMove( ) e cria uma mensagem ACL para o agente analisador nativo. O conteúdo da mensagem é preenchido com os mesmos parâmetros que foram utilizados para criar o agente de ontologia, isto é, protocolo, endereços de origem e destino e portas de origem e destino, variando a performativa da mensagem para o tipo REQUEST se tiver confirmado a intrusão (pois requisita que o agente analisador crie um agente atuador para gerar um alarme) e para o tipo PROPOSE se não tiver confirmado a suspeita (pois propõe que o agente analisador crie um agente gerente para verificar a suspeita nos demais *hosts*).

5.2.4 Implementação do Agente Gerente

Ao iniciar sua execução, um agente gerente envia uma requisição ao AMS da plataforma com o objetivo de se informar sobre todos os *containers* que estão ativos, criando uma lista com a identificação dos mesmos. Dessa maneira, ele utiliza o comando doMove( ) do JADE para se mover de *container* em *container*. Ao migrar para um novo *host*, ele envia uma mensagem ACL para o agente analisador local, com performativa QUERY_REF, informando o protocolo, os endereços de origem e destino, assim como as portas de origem e destino, se for o caso, da suspeita que foi passada como argumento em sua criação, perguntando se ela está presente em sua lista de suspeitos. A seguir, ele aguarda receber a resposta da sua mensagem.

Se a resposta à sua mensagem for que não foi confirmada a suspeita, ele migra para o próximo *host* da lista. Se a resposta for que foi confirmada a suspeita, ou já tiver percorrido todos os *hosts* de sua lista, ele migra de volta para o seu *container* nativo. No *container* nativo, ele verifica se a resposta dos demais agentes analisadores confirmou a intrusão ou não. Se a intrusão for confirmada, ele cria uma mensagem ACL com performativa QUERY_IF para o agente analisador, passando como informação o protocolo, os endereços de origem e destino e as portas de origem e destino que foram detectadas como intrusão. Ao receber essa informação, o agente analisador local cria um agente atuador que irá gerar o alarme correspondente. Se a intrusão não for confirmada, ele cria uma mensagem ACL, com performativa INFORM, para informar o agente analisador local de que não foi possível confirmar a suspeita.

5.2.5 Implementação do Agente Atuador

O agente atuador extrai as informações que foram passadas na criação do mesmo, isto é, protocolo, endereço de origem, porta de origem, endereço de destino e porta de destino dos pacotes que geraram a detecção de intrusão e adiciona a informação referente à data e hora do momento em que o alerta é gerado, salvando essas informações em um arquivo "alerta.txt". Dessa maneira, ao final do processo de detecção, o arquivo texto é formado por tantas linhas quantas forem as detecções realizadas e cada linha possui o formato descrito, como pode ser observado na Figura 5.4.

```
22-05-2012 21:07:08 -- Protocolo: UDP End Origem: 192.168.0.135 Porta Origem:
56491 - End Destino: 192.168.0.163 Porta Destino: 80
```

Figura 5.4: Alerta de intrusão representado no arquivo texto

5.2.6 Implementação do Agente Ontologia Global

O agente ontologia global é responsável por manter a informação salva na ontologia e só recebe mensagens escritas na linguagem de ontologia definida para o sistema de detecção. Ao receber uma mensagem, ele verifica a performativa da mesma. Se for uma mensagem do tipo QUERY_IF ele entende que é uma mensagem pedindo para realizar uma consulta às informações presentes na ontologia. Ele extrai a informação da consulta e verifica se o endereço do atacante representado na consulta corresponde a algum endereço de origem das informações presentes na ontologia. Após verificar, ele cria uma mensagem ACL resposta do tipo INFORM contendo a informação confirmando a detecção, se tiver encontrado o atacante na ontologia, ou informando que não detectou, se não tiver encontrado o atacante na ontologia.

Como verifica somente o endereço do atacante nas consultas ao conhecimento da ontologia, o agente ontologia global está utilizando a ontologia global (nos testes realizados em laboratório) como uma maneira de representar o conhecimento dos atacantes, isto é, dos nós responsáveis pelas intrusões na rede monitorada.

Se uma mensagem recebida pelo agente ontologia global for do tipo INFORM, ele entende que o conteúdo da mensagem é uma proposta de informação a ser adicionada na ontologia. Dessa forma, ele verifica se o atacante presente na proposta já consta na ontologia e, se não constar, salva as informações sugeridas na mesma.

5.2.7 Implementação do Ontologia de Detecção

Para definir a ontologia de detecção foi necessário estender a classe *Ontology* do JADE. A construção da ontologia foi iniciada com a definição do vocabulário da mesma. O

vocabulário utilizado é composto por quinze termos, *strings* que representam os elementos da mesma e podem ser utilizadas para representar entidades de conhecimento que a ontologia se propõe a descrever. Os termos que compõem o vocabulário da ontologia são:

- HOST, utilizado para definir *hosts* na rede;
- HOST_END, utilizado para definir o endereço de um determinado *host*;
- HOST_TIPO_END, utilizado para definir o tipo de endereço (protocolo) definido no termo anterior;
- SUSPEITA, utilizado para definir as características de um fluxo de rede considerado como suspeito de ser uma intrusão;
- SUSPEITA_ATACANTE, utilizado para representar o *host* suspeito de ser um atacante;
- SUSPEITA_ATACANTE_PORTA, utilizado para representar a porta que o suspeito está utilizando para originar as suspeitas dos ataques;
- SUSPEITA_ALVO, utilizado para representar o *host* que está sob suspeita de ser o alvo de um ataque;
- SUSPEITA_ALVO_PORTA, utilizado para representar a porta que está sendo utilizada no alvo por uma suspeita de ataque;
- SUSPEITA_NUM_PACOTES, utilizado para representar a quantidade de pacotes que foi detectada acessando determinado serviço e que gerou a suspeita de intrusão correspondente;
- SUSPEITA_INTERVALO, utilizado para representar o intervalo de tempo que determinado serviço ficou sem ser acessado antes de se registrar o acesso que gerou a suspeita de intrusão correspondente;
- INTRUSAO, utilizado para representar uma intrusão;
- INTRUSAO_ALVO, utilizado para representar o *host* correspondente ao alvo do ataque;
- INTRUSAO_ALVO_PORTA, utilizado para representar a porta que foi utilizada pelo ataque no alvo;
- INTRUSAO_ATACANTE, utilizado para representar o *host* correspondente ao atacante;
- INTRUSAO_ATACANTE_PORTA, utilizado para representar a porta que foi utilizada pelo ataque no atacante;

A definição do vocabulário da ontologia de detecção, estendendo a classe *Ontology*, pode ser observada na figura 5.5.

Após definir os termos do vocabulário, foi necessário, ainda, definir os esquemas que representam os conceitos, predicados e ações de agentes. Na ontologia de detecção utilizada, foram definidos os esquemas do conceito HOST, do predicado SUSPEITA e da ação de agente INTRUSAO.

```
public class OntologiaDeteccao extends Ontology {
    // O nome que identifica esta ontologia
    public static final String ONTOLOGY_NAME = "Detection-ontology";
    // VOCABULARIO
    public static final String HOST = "Host";
    public static final String HOST_END = "end";
    public static final String HOST_TIPO_END = "tipoend";

    public static final String SUSPEITA = "Suspeita";
    public static final String SUSPEITA_ATACANTE = "atacante";
    public static final String SUSPEITA_ATACANTE_PORTA = "atacanteporta";
    public static final String SUSPEITA_ALVO = "alvo";
    public static final String SUSPEITA_ALVO_PORTA = "alvoporta";
    public static final String SUSPEITA_NUM_PACOTES = "numpacotes";
    public static final String SUSPEITA_INTERVALO = "intervalo";

    public static final String INTRUSAO = "Intrusao";
    public static final String INTRUSAO_ALVO = "alvo";
    public static final String INTRUSAO_ALVO_PORTA = "alvoporta";
    public static final String INTRUSAO_ATACANTE = "atacante";
    public static final String INTRUSAO_ATACANTE_PORTA = "atacanteporta";

    // A instancia singleton desta ontologia
    private static Ontology theInstance = new OntologiaDeteccao();

    // Metodo para acessar o singleton
    public static Ontology getInstance() {
        return theInstance;
    }
```

Figura 5.5: Definição do vocabulário da ontologia

Como pode ser observado na figura 5.6, cada esquema adicionado está associado a uma classe Java, de modo que, ao usar a ontologia definida, as expressões indicando esses termos precisarão ser instâncias dessas classes Java.

O conceito HOST foi associado à classe Host.class, assim como aos esquemas primitivos HOST_END e HOST_TIPO_END, ambos do tipo BasicOntology.STRING. Essas associações implicam que uma informação passada para a ontologia representar um *host* deve ser uma instância da classe Host.java, que implementa a classe “Concept” e possui como atributos as *Strings* “End” e “TipoEnd” e os métodos necessários para acessar e atribuir valores aos mesmos.

O predicado SUSPEITA foi associado à classe Suspeita.class, assim como aos esquema de conceitos SUSPEITA_ATACANTE e SUSPEITA_ALVO, ambos do tipo HOST; aos esquemas primitivos SUSPEITA_ATACANTE_PORTA e SUSPEITA_ALVO_PORTA, ambos

do tipo BasicOntology.STRING; e aos esquemas primitivos SUSPEITA_NUM_PACOTES e SUSPEITA_INTERVALO, ambos do tipo BasicOntology.INTEGER. Essas associações implicam que uma informação passada para a ontologia representar uma suspeita deve ser uma instância da classe Suspeita.java, que implementa a classe "Predicate" e possui como atributos os *Hosts* "Atacante" e "Alvo", as *Strings* "AtacantePorta" e "AlvoPorta", os inteiros (*long*) "NumPacotes" e "Intervalo", assim como os métodos necessários para acessar e atribuir valores aos mesmos.

```
// Construtor
private OntologiaDeteccao() {

    super(ONTOLOGY_NAME, BasicOntology.getInstance());

    try {
        add(new ConceptSchema(HOST), Host.class);
        add(new PredicateSchema(SUSPEITA), Suspeita.class);
        add(new AgentActionSchema(INTRUSAO), Intrusao.class);

        // Estrutura do esquema para o conceito HOST
        ConceptSchema cs = (ConceptSchema) getSchema(HOST);
        cs.add(HOST_END, (PrimitiveSchema) getSchema(BasicOntology.STRING));
        cs.add(HOST_TIPO_END, (PrimitiveSchema) getSchema(BasicOntology.STRING));

        // Estrutura do esquema para o predicado SUSPEITA
        PredicateSchema ps = (PredicateSchema) getSchema(SUSPEITA);
        ps.add(SUSPEITA_ATACANTE, (ConceptSchema) getSchema(HOST));
        ps.add(SUSPEITA_ATACANTE_PORTA, (PrimitiveSchema) getSchema(BasicOntology.STRING));
        ps.add(SUSPEITA_ALVO, (ConceptSchema) getSchema(HOST));
        ps.add(SUSPEITA_ALVO_PORTA, (PrimitiveSchema) getSchema(BasicOntology.STRING));
        ps.add(SUSPEITA_NUM_PACOTES, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));
        ps.add(SUSPEITA_INTERVALO, (PrimitiveSchema) getSchema(BasicOntology.INTEGER));

        // Estrutura do esquema para a acao de agente INTRUSAO
        AgentActionSchema as = (AgentActionSchema) getSchema(INTRUSAO);
        as.add(INTRUSAO_ALVO, (ConceptSchema) getSchema(HOST));
        as.add(INTRUSAO_ALVO_PORTA, (PrimitiveSchema) getSchema(BasicOntology.STRING));
        as.add(INTRUSAO_ATACANTE, (ConceptSchema) getSchema(HOST));
        as.add(INTRUSAO_ATACANTE_PORTA, (PrimitiveSchema) getSchema(BasicOntology.STRING));
    }
    catch (OntologyException oe) {
        oe.printStackTrace();
    }
}
```

Figura 5.6: Estrutura dos esquemas

A ação de agente INTRUSAO foi associada à classe Intrusao.class, assim como aos esquemas de conceitos INTRUSAO_ALVO e INTRUSAO_ATACANTE, ambos do tipo HOST, e aos esquemas primitivos INTRUSAO_ALVO_PORTA e INTRUSAO_ATACANTE_PORTA, ambos do tipo BasicOntology.STRING. Essas associações implicam que uma informação passada para a ontologia representar uma intrusão deve ser uma instância da classe Intrusao.java, que implementa a classe "AgentAction" e possui como atributos os *Hosts* "Atacante" e "Alvo", as *Strings* "AtacantePorta" e "AlvoPorta" e os métodos necessários para acessar e atribuir valores aos mesmos.

6 AVALIAÇÃO DA ARQUITETURA PROPOSTA

Para avaliar o desempenho de um IDS, geralmente são indicados dois tipos de métricas interessantes: a taxa de detecção e a taxa de falso positivos (BRAHMI; YAHIA; PONCELET, 2010). A taxa de detecção é a medição do número de intrusões corretamente detectadas dividido pelo número total de intrusões. A taxa de falso positivos é o número total de comportamentos normais que foram erroneamente considerados como ataques, dividido pelo número total de comportamentos normais. O valor da taxa de detecção é esperado ser o maior possível enquanto que o valor da taxa de falso positivos é esperado que seja o menor possível. É interessante também analisar a taxa de falso negativos.

Um falso negativo ocorre quando o IDS falha ao gerar um alerta quando ocorre um ataque. Os falsos positivos geralmente são mais comentados que os falsos negativos, no entanto, cada falso negativo é um ataque legítimo que o IDS não conseguiu notar. Se o ataque não gerar alterações perceptíveis no alvo e se nenhum outro IDS o detectar, ele pode nunca ser percebido. Outro problema é que, se o IDS gerar muitos falsos positivos, o gerente de segurança geralmente cancela a regra geradora completamente ou simplesmente ignora os alertas, o que quer dizer que ele não revisa os casos legítimos de alerta e isso acaba se tornando também uma espécie de falso negativo (NORTHCUTT et al., 2002).

Uma dependência ou troca entre as taxas de detecção e de falso positivos está geralmente presente em métodos de aprendizagem de máquinas, embora o ideal seria conseguir que as duas taxas fossem independentes. Comparando essas quantidades entre si pode-se avaliar o desempenho do IDS (BRAHMI; YAHIA; PONCELET, 2010).

6.1 Laboratório de Testes

Após a implementação dos agentes na linguagem Java, utilizando o *framework* JADE, foi montado um laboratório de testes, com máquinas onde foi instalado o Ubuntu 11.04 Natty Narwhal, Sistema Operacional baseado em Linux, e preparadas com o IDS proposto. Para realizar os ataques à rede, foi utilizada uma máquina com o Sistema Operacional Microsoft Windows e o programa LOIC.

A ferramenta LOIC (*Low Orbit Ion Cannon*), hoje um programa de código aberto, foi construída pela empresa *Praetoc Technologies* como uma aplicação para testes de estresse de redes. Ela executa ataques de negação de serviço simples, enviando uma sequência de requisições TCP ou UDP para uma máquina alvo. O usuário pode escolher uma máquina alvo e o método de ataque (TCP ou UDP) e alguns parâmetros para customizar as requisições que serão enviadas, como o número da porta de destino, o *payload* dos pacotes e o número de *threads* concorrentes. Os ataques com essa ferramenta iniciam várias conexões com o mesmo *host* alvo

e enviam continuamente uma *string* predefinida que pode ser definida no parâmetro *message*. Nos ataques TCP e UDP essa *string* é enviada em texto livre, enquanto nos ataques HTTP, a mensagem é incluída no contúdo de uma mensagem HTTP GET. Quando uma grande quantidade de mensagens é enviada, o *host* alvo fica sobrecarregado e incapacitado de responder as requisições dos usuários legítimos.

Atualmente, o programa possui ainda uma opção de modo automático que permite que a mesma seja controlada remotamente por meio do protocolo IRC (*Internet Relay Chat*), que pode ser vista como uma opção para ingressar voluntariamente em uma *botnet*. Uma outra versão da ferramenta, chamada de "JS LOIC" roda em navegadores que suportam Javascript e, diferente da versão para *desktop*, não é necessária nenhuma instalação de pacotes. Essa versão *web* da ferramenta suporta unicamente o modo manual de operação. O grupo *Anonymous* utilizou o LOIC para realizar ataques a diversos sites nos últimos anos (PRAS et al., 2010).

Os *hosts* que fizeram parte do IDS serão denominados de A, B e C. Nos testes realizados, o host C funcionou como "*Main-Container*" (hospedando assim o agente ontologia global, o AMS e o DF), enquanto o host B funcionou como "*Container*-1" e o host A funcionou como "*Container*-2". Foram realizados ataques aos três *hosts* para observar se os mesmos conseguiam identificar os ataques, a forma como os mesmos estavam sendo detectados e os falsos positivos que o sistema geraria. O objetivo dos testes foi determinar se a arquitetura proposta poderia ser utilizada em um Sistema de Detecção de Intrusão e, para isso, foram utilizados critérios de detecção bastantes simples, conforme descrito anteriormente, que não refletem a realidade dos sistemas comerciais utilizados atualmente mas permitem avaliar o funcionamento da arquitetura, observando seus pontos fracos e fortes.

A figura 6.1 representa a estrutura da rede de testes do laboratório realizado. Como tratado anteriormente, o fato do agente ontologia global ficar localizado em um nó específico da rede, faz com que esse nó passe a ser um ponto fraco da arquitetura, no entanto o JADE permite que o *container* principal da arquitetura seja replicado algumas vezes, criando *containers* redundantes que assumem o lugar do principal se o mesmo se tornar indisponível (BELLIFEMINE et al., 2010), fazendo com que a arquitetura seja mais tolerante a falhas.

Ao elaborar a montagem do laboratório, a intenção foi de simular uma rede, de uma empresa por exemplo, que contasse com três roteadores como saídas para uma rede externa (como a Internet). O IDS seria instalado nessas três saídas de modo a identificar intrusões nos *hosts* internos da rede. Fazendo uma comparação com o laboratório montado, cada um dos roteadores que possuiriam o sistema instalado seria um dos *hosts* A, B e C e o atacante estaria na rede externa. A estrutura dessa rede pode ser observada na figura 6.2.

A implementação dos agentes e o método de ataque escolhido para ser usado no laboratório mostram que os testes do IDS foram direcionados para a detecção de ataques de DoS a múltiplos *hosts*. No entanto, a arquitetura proposta é genérica e poderia ser utilizada na

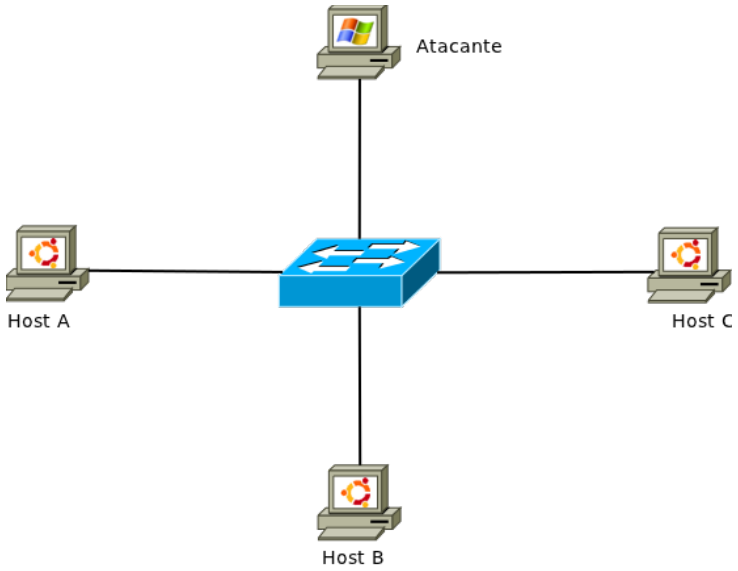

Figura 6.1: Estrutura da rede de testes

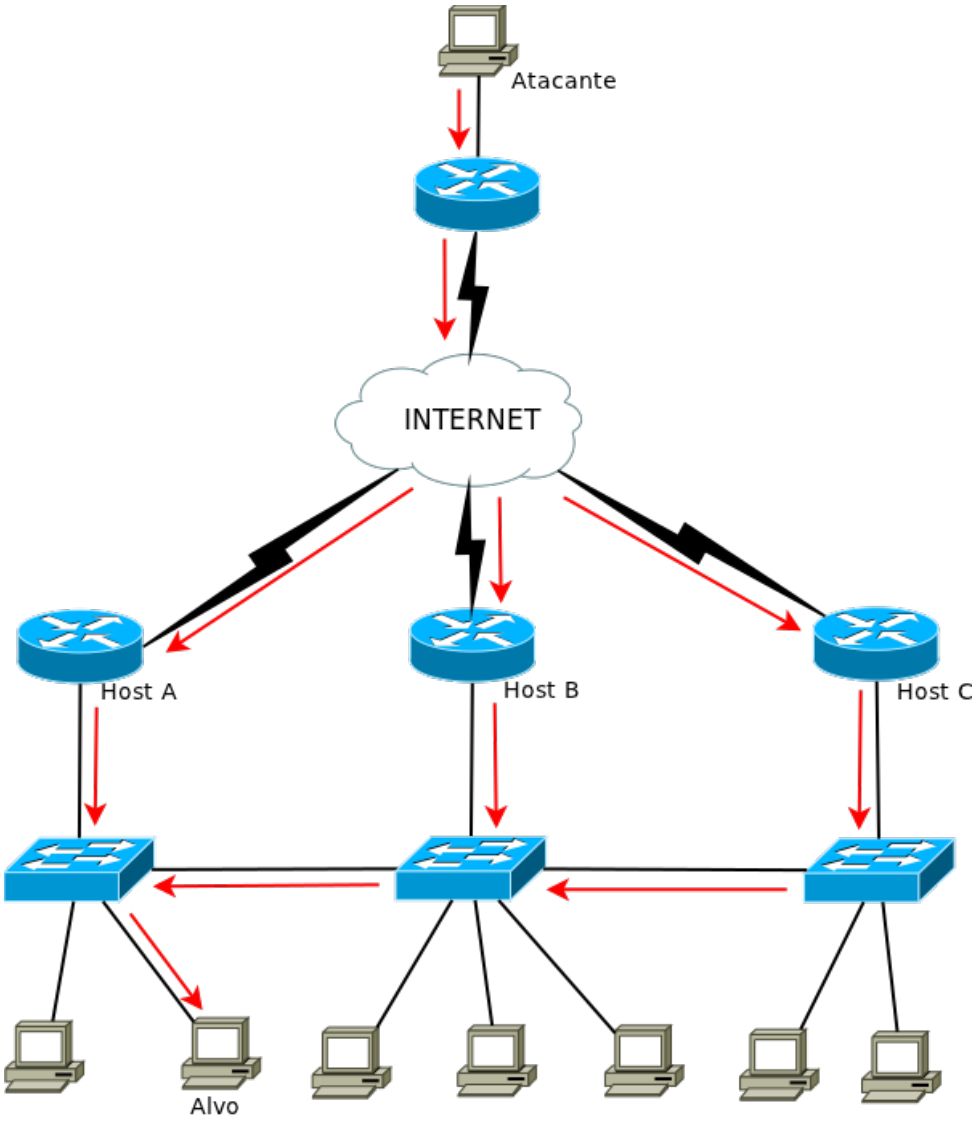

Figura 6.2: Uma rede, com três saídas para a Internet, sendo atacada.

detecção de outros tipos de ataques.

6.2 Ataques realizados em laboratório

Foram realizadas oito sequências de ataques, nas quais o programa LOIC foi configurado para inundar os alvos com pacotes TCP nas quatro primeira sequências e com pacotes UDP nas quatro últimas.

Em cada sequência, os ataques a cada *host* foram executados 10 vezes. Os resultados obtidos foram combinados para calcular a média e os limites superior e inferior com 95% de confiança.

6.2.1 Primeira sequência de ataques

Na primeira sequência de ataques, foram realizados ataques com o programa LOIC, com pacotes TCP, com uma quantidade de pacotes que superasse o limiar L_{min} mas não superasse o limiar L_{max} dos agentes analisadores. Foram atacados os *hosts* A, B e C nessa ordem. Ao final, foram obtidos os seguintes resultados nos 10 ataques executados: o *host* A não detectou nenhum ataque, o *host* B detectou o ataque realizado por meio de seus agentes gerentes e o *host* C detectou o ataque realizado por meio de seus agentes de ontologia. Essas detecções foram de acordo com o esperado, visto que, ao ser atacado primeiro, o agente analisador do *host* A detectou atividade suspeita e acionou um dos seus agentes de ontologia, o que não resultou em nenhuma conclusão pois não havia nenhuma informação sobre esse ataque na ontologia global. Assim, foi acionado um agente gerente, que migrou para os *hosts* B e C que também não tinham nenhuma informação sobre o ataque. Não podendo concluir nada sobre as suspeitas, não foi detectado o ataque.

Quando o *host* B foi atacado, seu agente analisador detectou atividade suspeita, acionou um agente de ontologia (o que também não resultou em nenhuma informação útil) e, em seguida, acionou um agente gerente que, ao migrar para o *host* A, recebeu a informação do agente analisador local que o atacante já era considerado suspeito no mesmo, assim, confirmando um ataque. Ao receber a informação, o agente analisador acionou um agente atuador para gerar um alerta. Após gerar o alerta, o agente analisador acionou um agente de ontologia para adicionar a informação sobre o atacante na ontologia global. Quando o *host* C foi atacado, seu agente analisador, ao detectar atividade suspeita, acionou um agente de ontologia que, ao consultar a ontologia global, confirmou a suspeita de ataque. Ao receber a informação, o agente analisador acionou um agente atuador para gerar um alerta.

6.2.2 Segunda sequência de ataques

Na segunda sequência, foram realizados ataques com o programa LOIC, com pacotes TCP, com uma quantidade de pacotes que superasse o limiar L_{min} mas não superasse o limiar L_{max} dos agentes analisadores. Foram atacados os *hosts* A, B e C nessa ordem. Após esses ataques, foi realizado mais um, no qual o programa inundou o *host* A com uma quantidade de pacotes superior ao limiar L_{max}.

Mais uma vez os resultados obtidos foram exatamente os esperados: nos 10 ataques executados os *hosts* B e C apresentaram comportamentos idênticos aos da primeira sequência de ataques e o *host* A gerou dois alarmes, ambos devido à detecção de pacotes em quantidade superior a L_{max}, um alarme com IP de origem da máquina atacante e um alarme com o IP de origem do *host* A, que corresponde às respostas às requisições feitas pela máquina atacante.

Ao ser atacado primeiro, o agente analisador do *host* A detectou atividade suspeita e acionou um dos seus agentes de ontologia, o que não resultou em nenhuma conclusão pois não havia nenhuma informação sobre esse ataque na ontologia global. Assim, foi acionado um agente gerente, que migrou para os *hosts* B e C que também não tinham nenhuma informação sobre o ataque. Não podendo concluir nada sobre as suspeitas, não foi detectado o ataque.

O agente analisador do *host* B detectou atividade suspeita, acionou um agente de ontologia (o que também não resultou em nenhuma informação útil) e, em seguida, acionou um agente gerente que, ao migrar para o *host* A, recebeu a informação do agente analisador local que o atacante já era considerado suspeito no mesmo, assim, confirmando um ataque. Ao receber a informação, o agente analisador acionou um agente atuador para gerar um alerta. Após gerar o alerta, o agente analisador acionou um agente de ontologia para adicionar a informação sobre o atacante na ontologia global. O agente analisador do *host* C, ao detectar atividade suspeita, acionou um agente de ontologia que, ao consultar a ontologia global, confirmou a suspeita de ataque. Ao receber a informação, o agente analisador acionou um agente atuador para gerar um alerta.

Ao ser atacado novamente, o *host* A percebeu, por meio de seu agente analisador, quantidades de pacotes superiores ao limiar L_{max}, caracterizando assim confirmação de intrusões. Dessa maneira, foram acionados agentes atuadores que geraram os alertas.

6.2.3 Terceira sequência de ataques

Na terceira sequência de ataques, foram realizados mais uma vez ataques consecutivos aos *hosts* A, B e C com pacotes TCP, sendo que o ataque a A foi realizado com uma quantidade de pacotes que superou L_{max}, enquanto os ataques a B e C foram realizados com uma quantidade de pacotes que superou o limite L_{min} mas que ainda era inferior ao valor de

L_{max}. Ao final, nos 10 ataques executados obtiveram-se os seguintes resultados: o *host* A realizou duas detecções pelo seu agente analisador e os *hosts* B e C detectaram os ataques realizados por meio de seus agentes de ontologia.

Ao ser atacado primeiro, o agente analisador do *host* A detectou o ataque, e, como a quantidade de pacotes superava L_{max}, acionou agentes atuadores para gerarem alarmes. Foram gerados dois alarmes, devido à detecção de quantidade de pacotes superior a L_{max} oriundas da máquina atacante e às respostas geradas a essas requisições no próprio *host*. Em seguida foi acionado um agente de ontologia para incluir o atacante na ontologia global. Os agentes analisadores dos *hosts* B e C, ao detectarem atividade suspeita, acionaram agentes de ontologia, que, ao consultarem o agente ontologia global, confirmaram as suspeitas de ataque. Os agentes analisadores então acionaram agentes atuadores que geraram os alarmes.

6.2.4 Quarta sequência de ataques

A quarta sequência de ataques foi realizada de forma semelhante à primeira, sendo que nesta bateria esperou-se, antes de se iniciar os ataques, um intervalo de tempo maior que o intervalo configurado no Sistema de Detecção como aquele a partir do qual os pacotes passam a gerar suspeitas. Esses ataques geraram alguns falsos positivos, como pode ser observado na figura 6.3, sendo que o *host* A gerou uma média de 5,5 falsos positivos, o *host* B gerou uma média de 6,9 falsos positivos e o *host* C gerou uma média de 4 falsos positivos. Todos os ataques foram detectados pelos *hosts*, sendo em alguns casos observados dois alertas relativos aos ataques, um relativo aos pacotes oriundos da máquina atacante e um relativo às respostas geradas pelos próprio *host* ao ataque.

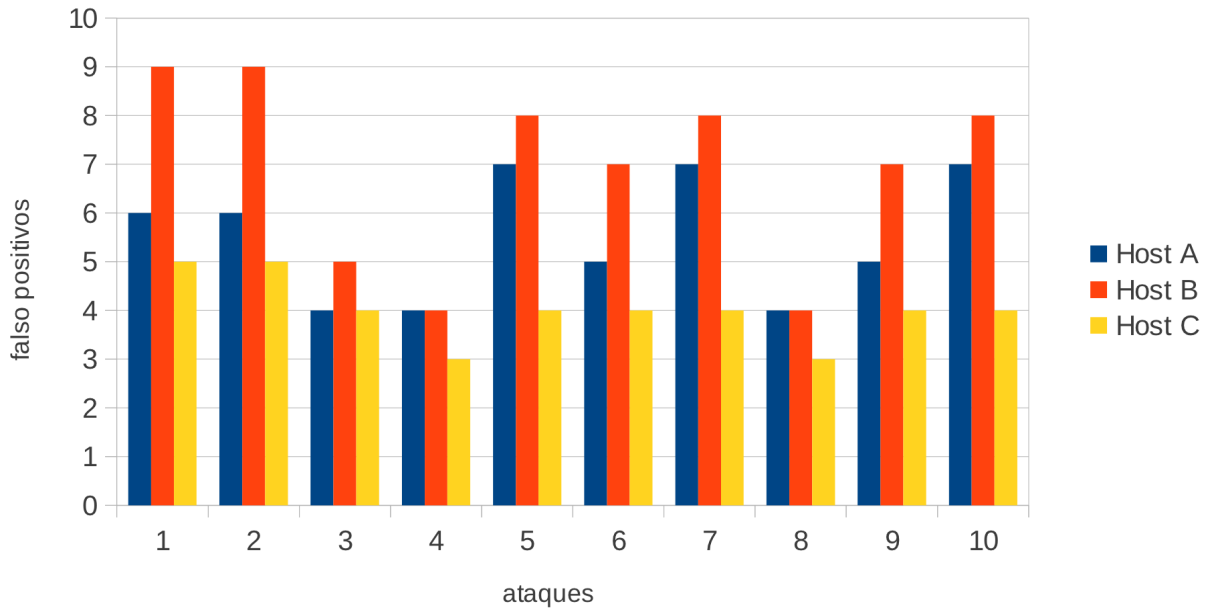

Figura 6.3: Resultados da quarta sequência de ataques

6.2.5 Quinta sequência de ataques

Na quinta sequência de ataques, foram realizados ataques com o programa LOIC, com pacotes UDP, com uma quantidade de pacotes que superasse o limiar L_{min} mas não superasse o limiar L_{max} dos agentes analisadores. Primeiramente foi atacado o *host* A, depois o B e depois o C. Ao final dos ataques, obteu-se os seguintes resultados nos 10 ataques executados: o *host* A não detectou nenhum ataque, o *host* B detectou o ataque realizado por meio de seus agentes gerentes e o *host* C detectou o ataque realizado por meio de seus agentes de ontologia. Ao ser atacado primeiro, o agente analisador do *host* A detectou atividade suspeita e acionou um dos seus agentes de ontologia, o que não resultou em nenhuma conclusão pois não havia nenhuma informação sobre esse ataque na ontologia global. Assim, foi acionado um agente gerente, que migrou para os *hosts* B e C que também não tinham nenhuma informação sobre o ataque. Dessa forma, não foi detectado o ataque pois não foi possível chegar a nenhuma conclusão sobre as suspeitas.

O agente analisador do *host* B detectou atividade suspeita, acionou um agente de ontologia (o que também não resultou em nenhuma informação útil) e, em seguida, acionou um agente gerente que, ao migrar para o *host* A, recebeu a informação do agente analisador local que o atacante já era considerado suspeito no mesmo, assim, confirmando um ataque. Ao receber a informação, o agente analisador acionou um agente atuador para gerar um alerta. Após gerar o alerta, o agente analisador acionou um agente de ontologia para adicionar a informação sobre o atacante na ontologia global. O agente analisador do *host* C, ao detectar atividade suspeita, acionou um agente de ontologia que, ao consultar a ontologia global, confirmou a suspeita de ataque. Ao receber a informação, o agente analisador acionou um agente atuador para gerar um alerta.

6.2.6 Sexta sequência de ataques

Na sequência número seis, foram realizados ataques com o programa LOIC, com pacotes UDP, com uma quantidade de pacotes que superasse o limiar L_{min} mas não superasse o limiar L_{max} dos agentes analisadores, na mesma ordem de ataque que a sequência anterior. Após esses ataques, foi realizado mais um, no qual o programa inundou o *host* A com uma quantidade de pacotes UDP superior ao limiar L_{max}. Nos 10 ataques executados os *hosts* B e C apresentaram comportamentos idênticos aos da quinta sequência de ataques e o *host* A gerou um alarme, devido à detecção de pacotes em quantidade superior a L_{max} oriundos da máquina atacante.

Ao ser atacado primeiro, o agente analisador do *host* A detectou atividade suspeita e acionou um dos seus agentes de ontologia, o que não resultou em nenhuma conclusão pois não havia nenhuma informação sobre esse ataque na ontologia global. Assim, foi acionado um

agente gerente, que migrou para os *hosts* B e C que também não tinham nenhuma informação sobre o ataque. Não foi detectado o ataque porque não foi possível chegar a nenhuma conclusão sobre as suspeitas.

O agente analisador do *host* B detectou atividade suspeita, acionou um agente de ontologia, que não encontrou informação sobre o ataque na ontologia global, e, em seguida, acionou um agente gerente que, ao migrar para o *host* A, recebeu a informação do agente analisador local que o atacante já era considerado suspeito no mesmo, assim, confirmando um ataque. Ao receber essa informação, o agente analisador acionou um agente atuador para gerar um alerta. Após gerar o alerta, o agente analisador acionou um agente de ontologia para adicionar a informação sobre o atacante na ontologia global. O agente analisador do *host* C, ao detectar atividade suspeita, acionou um agente de ontologia que, ao consultar a ontologia global, confirmou a suspeita de ataque. Ao receber a informação, o agente analisador acionou um agente atuador para gerar um alerta.

Ao ser atacado novamente, o *host* A percebeu, por meio de seu agente analisador, quantidades de pacotes superiores ao limiar L_{max}, ou seja, que estava ocorrendo um ataque. Dessa maneira, acionou um agente atuador para que gerar um alerta.

6.2.7 Sétima sequência de ataques

Na sequência número sete, foram realizados mais uma vez ataques consecutivos aos *hosts* A, B e C com pacotes UDP, sendo que o ataque a A foi realizado com uma quantidade de pacotes que superou L_{max}, enquanto os ataques a B e C foram realizados com uma quantidade de pacotes que superou o limite L_{min} mas que ainda era inferior ao valor de L_{max}. Em cada um dos 10 ataques executados o *host* A realizou uma detecção por meio de seu agente analisador e os *hosts* B e C detectaram os ataques realizados por meio de seus agentes de ontologia.

Ao ser atacado primeiro, o agente analisador do *host* A detectou o ataque, e, como a quantidade de pacotes orindos da máquina atacante superava L_{max}, acionou um agente atuador para gerar um alarme. Em seguida foi acionado um agente de ontologia para incluir o atacante na ontologia global. Os agentes analisadores dos *hosts* B e C, ao detectarem atividade suspeita, acionaram agentes de ontologia, que, ao consultarem o agente ontologia global, confirmaram as suspeitas de ataque. Os agentes analisadores então acionaram agentes atuadores que geraram os alarmes.

6.2.8 Oitava sequência de ataques

A oitava sequência foi realizada de forma semelhante à quarta, esperando-se, antes de se iniciar os ataques, um intervalo de tempo maior que o intervalo configurado no Sistema

de Detecção como aquele a partir do qual os pacotes passam a gerar suspeitas, no entanto, nessa sequência os alvos foram inundados com pacotes UDP. Esses ataques geraram falsos positivos, como pode ser observado na figura 6.4, sendo que o *host* A gerou uma média de 5,3 falsos positivos, o *host* B gerou uma média de 6,9 falsos positivos e o *host* C gerou uma média de 4,3 falsos positivos. Todos os ataques geraram um alerta correto em cada *host*.

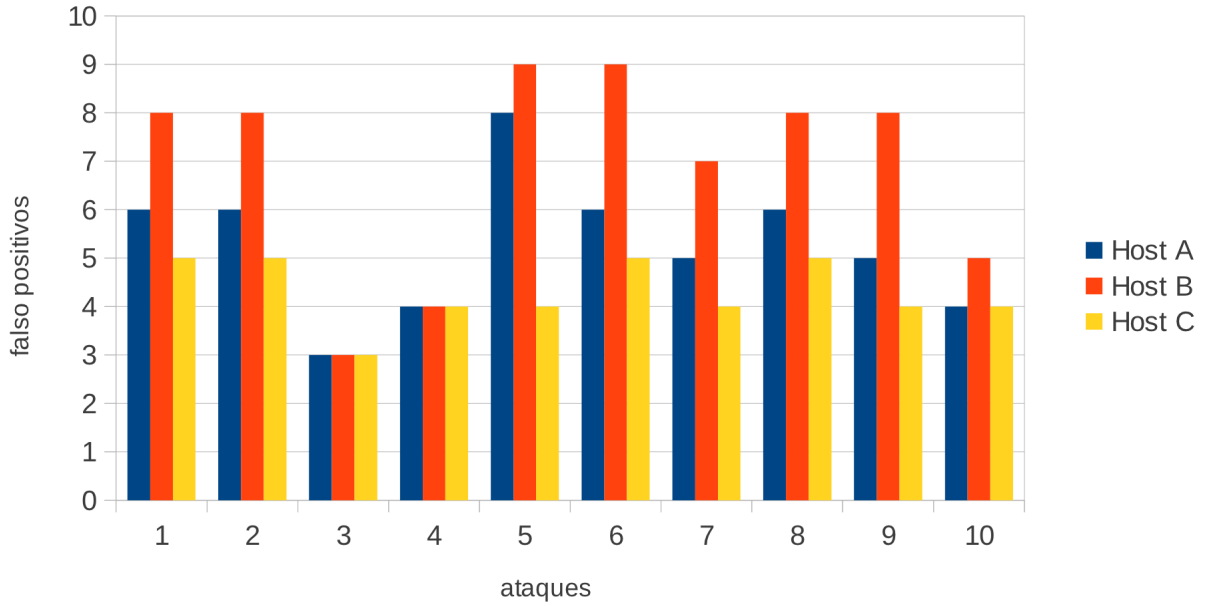

Figura 6.4: Resultados da oitava sequência de ataques

6.3 Resultados obtidos

A figura 6.5 mostra as quantidades médias de falsos positivos obtidos, a figura 6.6 mostra as taxas de detecção obtidas e a tabela 6.1 resume os resultados.

Na primeira sequência de ataques, dos três ataques realizados, foram detectados dois: os ataques aos *hosts* B e C. Dessa forma, a taxa de detecção foi de aproximadamente 0,67. Não foi gerado nenhum alarme falso.

Na segunda sequência, dos quatro ataques realizados, foram detectados três, pois apesar do último ataque ao *host* A ter gerado dois alarmes, o primeiro ataque a esse *host* não foi detectado. Assim, a taxa de detecção foi de 0,75. Não foi gerado nenhum alarme falso.

Na terceira sequência de ataques, todos os três ataques realizados foram detectados, com a observação de que o ataque ao *host* A gerou dois alertas. Dessa maneira, a taxa de detecção foi de 1,0. Não foi gerado nenhum alarme falso.

Na quarta sequência, todos os três ataques foram detectados, com a observação de que os ataques aos *hosts* A e B algumas vezes geraram dois alertas cada um, gerando uma média de 1,7 alertas corretos para cada um, o *host* A gerou em média 5,5 alarmes falsos, B gerou em

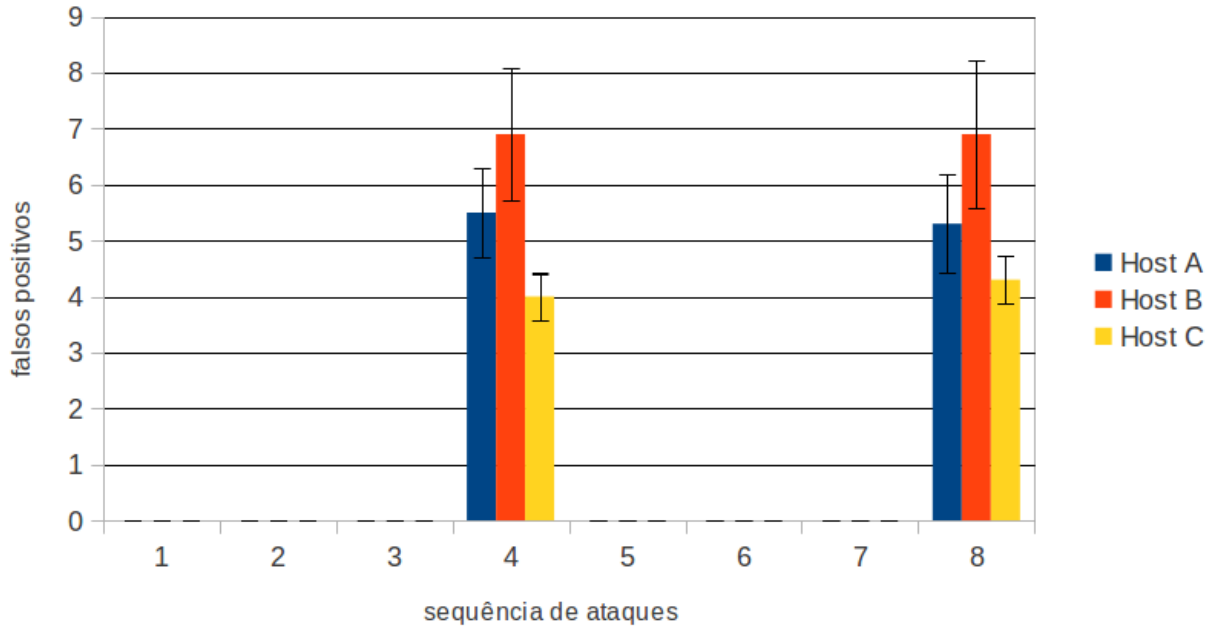

Figura 6.5: Quantidades de falsos positivos observados

média 6,9 e C gerou em média 4. Dessa maneira, a taxa de detecção foi de 1,0. Nessa sequência, no entanto, foi observada a geração de uma média de 16.4 alarmes falsos pelo IDS como um todo.

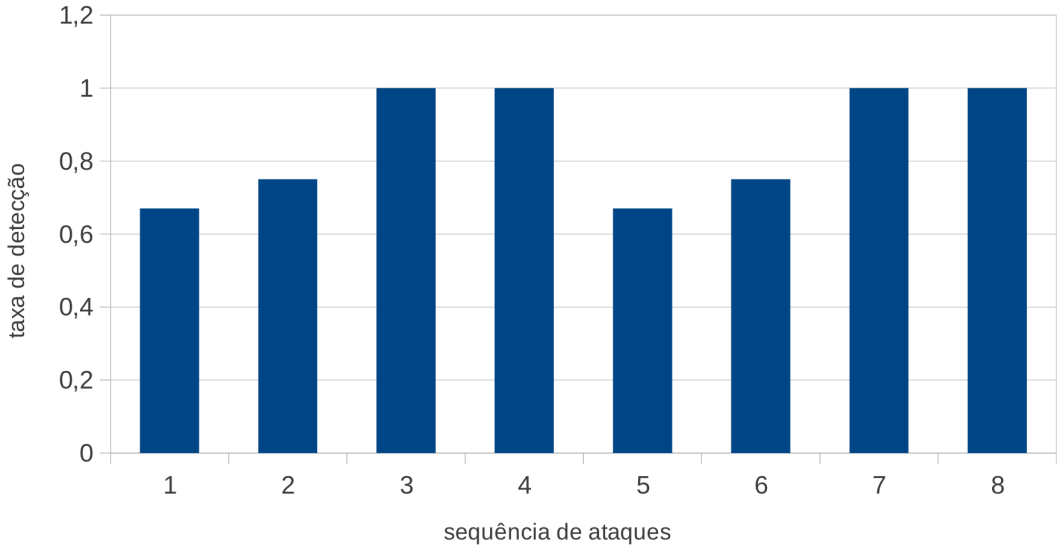

Figura 6.6: Taxas de detecção observadas

Na quinta sequência, dos três ataques realizados, foram detectados dois: os ataques aos *hosts* B e C. Dessa forma, a taxa de detecção foi de aproximadamente 0,67. Não foi gerado nenhum alarme falso.

Na sexta sequência, dos quatro ataques realizados, foram detectados três, tendo o primeiro ataque ao *host* A ocorrido sem ser detectado. Assim, a taxa de detecção foi de 0,75.

Não foi gerado nenhum alarme falso.

Na sétima sequência, todos os três ataques realizados foram detectados. Dessa maneira, a taxa de detecção foi de 1,0. Não foi gerado nenhum alarme falso.

Tabela 6.1: Resultados das sequências de ataques

		Host A	**Host B**	**Host C**
Sequência 1	detecções corretas	0	1	1
	falsos positivos	0	0	0
Sequência 2	detecções corretas	2	1	1
	falsos positivos	0	0	0
Sequência 3	detecções corretas	2	1	1
	falsos positivos	0	0	0
Sequência 4	detecções corretas	1,7	1,7	1
	falsos positivos	5,5	6,9	4
Sequência 5	detecções corretas	0	1	1
	falsos positivos	0	0	0
Sequência 6	detecções corretas	1	1	1
	falsos positivos	0	0	0
Sequência 7	detecções corretas	1	1	1
	falsos positivos	0	0	0
Sequência 8	detecções corretas	1	1	1
	falsos positivos	5,3	6,9	4,3

Na oitava sequência de ataques, todos os três ataques foram detectados, com a observação de o *host* A gerou em média 5,3 alarmes falsos, B gerou em média 6,9 e C gerou em média 4,3. Dessa maneira, a taxa de detecção foi de 1,0. Nessa sequência, no entanto, foi observada a geração de uma média de 16.5 alarmes falsos pelo IDS como um todo.

A quantidade de falsos positivos encontrada nas sequências quatro e oito não foi inesperada. O critério que considera como parâmetro o intervalo de tempo de acesso ao serviço é muito aberto, considerando como suspeitas pacotes normais da rede que não foram direcionados a determinados serviços dentro do intervalo. Era esperado que, como o critério gera somente suspeitas e não gera alarmes de imediato, o processo de detecção distribuído controlasse as detecções desses tipos de pacotes. Embora muitos pacotes desse tipo tenham realmente sido considerado suspeitos sem se confirmarem, muitos acabaram sendo confirmados pelo sistema distribuído e os alarmes falsos foram gerados.

No total, foram realizados vinte e seis ataques à rede do laboratório (repetidos 10 vezes cada), sendo detectados vinte e dois desses ataques. Dessa maneira, a taxa de detecção total foi de aproximadamente 0,846. Foi observado, na média, um total de 32,9 alarmes falsos.

7 CONCLUSÃO

A grande diversidade de serviços e facilidades que vieram junto com o crescimento da Internet aumentou a importância dos dados armazenados em computadores e trafegando em redes, assim como a criticidade na proteção dos mesmos. Nesse contexto, os Sistemas de Detecção de Intrusão assumem um papel importantíssimo para garantir a disponibilidade dos serviços e proteger as plataformas de negócios e informações sigilosas de empresas, organizações ou até de governos.

A necessidade de executar um processo de detecção eficiente, que seja capaz de perceber as invasões nas redes e computadores sem ocasionar crescimento excessivo de tráfego e de processamento, que podem prejudicar o funcionamento normal dos nós envolvidos, levou ao desenvolvimento de maneiras diferentes para organizar o funcionamento dos IDS.

No presente trabalho, foi proposta uma arquitetura de detecção distribuída que se vale da utilização de agentes móveis para facilitar a comunicação entre os nós que executam o processo de detecção e de uma ontologia para organizar o conhecimento do sistema sobre as intrusões.

A arquitetura foi descrita com o detalhamento do funcionamento de cada um dos tipos de agentes que a compõem, os métodos considerados para o reconhecimento dos ataques, a descrição da ontologia utilizada e os detalhes de implementação do sistema. Para avaliar a proposta, foi montado um laboratório para a execução de testes e a visualização do sistema criado em execução.

Os resultados obtidos com os testes em laboratório confirmaram que a arquitetura proposta pode ser utilizada em processos de detecção de intrusão. Todas as ataques realizados foram identificados por pelo menos um dos *hosts* do sistema, e muitos foram identificados por todos os *hosts* atacados. O metódo de detecção que considera como parâmetro a quantidade de pacotes capturados para um serviço específico se mostrou capaz de detectar ataques, mesmo que esses não sejam detectados por alguns dos componentes do sistema.

O método de detecção que considera o intervalo de tempo que o serviço ficou sem ser acessado também se mostrou como capaz de detectar os ataques, no entanto, levou à geração de uma grande quantidade de falsos positivos. Muitos falsos positivos foram causados por pacotes que poderiam ser desconsiderados na execução dos agentes analisadores (como pacotes ARP e RARP) o que não iria prejudicar a detecção dos ataques realizados nos testes, no entanto optou-se por configurar os agentes que participam do processo de detecção de uma forma mais genérica, permitindo que a arquitetura proposta pudesse ser testada futuramente com outros tipos de ataques.

Pode-se identificar como contribuições do trabalho:

- a proposta de uma arquitetura para detecção distribuída de intrusão;
- a utilização de uma ontologia para o gerenciamento do conhecimento no processo de detecção;
- a comprovação, com testes realizados em máquinas reais em laboratório, de que a arquitetura proposta pode ser eficaz.

7.1 Trabalhos Futuros

Ao longo do desenvolvimento do trabalho, foram identificadas diversas oportunidades para trabalhos futuros. Dentre elas, destacam-se:

1. a implantação de uma forma mais complexa de detecção, com agentes mais inteligentes, utilizando detecções de anomalias estatísticas que poderiam ser identificadas pelos agentes analisadores e gerentes, possibilitando a criação de assinaturas de ataques, que poderiam ser armazenadas no agente ontologia global juntamente com assinaturas já conhecidas;
2. o desenvolvimento de uma ontologia mais complexa, com maior quantidade de parâmetros para caracterizar os ataques, reduzindo a quantidade de falso positivos.
3. estudar mais profundamente o impacto da utilização da arquitetura proposta no tráfego de rede;
4. a implantação e teste do funcionamento da arquitetura com um *container* principal redundante e tolerante a falhas, impedindo que a ontologia global se torne um ponto fraco do sistema;
5. implementar a arquitetura proposta utilizando outro *framework*, com o objetivo de comparar as duas implementações, auxiliando na identificação de pontos fracos e fortes.

REFERÊNCIAS BIBLIOGRÁFICAS

AART, C. van et al. Creating and using ontologies in agent communication. In: *Proceedings of the Workshop on Ontologies in Agent Systems*. [S.l.: s.n.], 2002.

ABAD, C.; BONILLA, R. An analysis on the schemes for detecting and preventing ARP cache poisoning attacks. In: IEEE. *Distributed Computing Systems Workshops, 2007. ICDCSW'07. 27th International Conference on*. [S.l.], 2007. p. 60–60.

ABDOLI, F.; KAHANI, M. Ontology-based distributed intrusion detection system. In: IEEE. *Computer Conference, 2009. CSICC 2009. 14th International CSI*. [S.l.], 2009. p. 65–70.

AHMED, E.; SAMAD, K.; MAHMOOD, W. Cluster-based intrusion detection (CBID) architecture for mobile ad hoc networks. In: *5th Conference, AusCERT2006 Gold Coast, Australia, May 2006 Proceedings*. [S.l.: s.n.], 2006.

AHMED, M. et al. NIDS: A network based approach to intrusion detection and prevention. In: IEEE. *Computer Science and Information Technology-Spring Conference, 2009. IACSITSC'09. International Association of*. [S.l.], 2009. p. 141–144.

AYDIN, M.; ZAIM, A.; CEYLAN, K. A hybrid intrusion detection system design for computer network security. *Computers & Electrical Engineering*, Elsevier, v. 35, n. 3, p. 517–526, 2009.

AZEVEDO, R. de et al. An autonomic ontology-based multiagent system for intrusion detection in computing environments. *International Journal for Infonomics (IJI)*, v. 3, n. 1, 2010.

BELLIFEMINE, F. et al. JADE: A software framework for developing multi-agent applications. lessons learned. *Information and Software Technology*, Elsevier, v. 50, n. 1-2, p. 10–21, 2008.

BELLIFEMINE, F. et al. JADE programmer's guide. *Italy: CSELT SpA*, 2002. Disponível em <http://jade.tilab.com/doc/programmersguide.pdf>. Acesso em: 15 jan. 2012.

BELLIFEMINE, F. et al. *JADE Administrator's Guide*. 2010. Disponível em <http://jade.tilab.com/doc/index.html>. Acesso em: 06 jul. 2011.

BELLIFEMINE, F.; POGGI, A.; RIMASSA, G. Developing multi-agent systems with JADE. *Intelligent Agents VII Agent Theories Architectures and Languages*, Springer, p. 42–47, 2001.

BRAHMI, I.; YAHIA, S.; PONCELET, P. MAD-IDS: Novel intrusion detection system using mobile agents and data mining approaches. *Intelligence and Security Informatics*, Springer, p. 73–76, 2010.

CAIRE, G.; CABANILLAS, D. JADE tutorial: creating and using application specific ontologies. 2004. Disponível em <http://jade.tilab.com/doc/tutorials/CLOntoSupport.pdf>. Acesso em: 14 fev. 2012.

CARL, G. et al. Denial-of-service attack-detection techniques. *Internet Computing, IEEE*, IEEE, v. 10, n. 1, p. 82–89, 2006.

CHEN, B.; CHENG, H. A review of the applications of agent technology in traffic and transportation systems. *Intelligent Transportation Systems, IEEE Transactions on*, IEEE, v. 11, n. 2, p. 485–497, 2010.

CHEN, R.; CHENG, K.; HSIEH, C. Using rough set and support vector machine for network intrusion detection. *International Journal of Network Security & Its Applications (IJNSA)*, v. 1, n. 1, p. 1–13, 2010.

CHMIEL, K. et al. Testing the efficiency of JADE agent platform. In: IEEE. *Parallel and Distributed Computing, 2004. Third International Symposium on/Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks, 2004. Third International Workshop on*. [S.l.], 2004. p. 49–56.

DOULIGERIS, C.; MITROKOTSA, A. DDoS attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks*, Elsevier, v. 44, n. 5, p. 643–666, 2004.

FABRICI, D. *Defining Ontologies in a Multi Agent Scenario Using the* JADE *Framework*. [S.l.]: 4th Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence, 2006. Disponível em <http://old.bmf.hu/conferences/sami2006/>. Acesso em: 01 mai. 2012.

FARID, D.; RAHMAN, M. Anomaly network intrusion detection based on improved self adaptive bayesian algorithm. *Journal of computers*, v. 5, n. 1, p. 23–31, 2010.

FERREIRA, E. T. et al. Intrusion detection system with wavelet and neural artifical network approach for networks computers. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, IEEE, v. 9, n. 5, p. 832–837, 2011.

FUJII, K. *Jpcap*. 2007. Disponível em <http://netresearch.ics.uci.edu/kfujii/Jpcap/doc/>. Acesso em: 10 abr. 2012.

GARCIA-TEODORO, P. et al. Anomaly-based network intrusion detection: Techniques, systems and challenges. *computers & security*, Elsevier, v. 28, n. 1-2, p. 18–28, 2009.

GIL, Y. On agents and grids: Creating the fabric for a new generation of distributed intelligent systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, Elsevier, v. 4, n. 2, p. 116–123, 2006.

GIOVANNI, C. JADE tutorial: JADE programming for beginners. *TILAB CSELT, Setiembre*, 2007. Disponível em <http://jade.tilab.com/doc/index.html>. Acesso em: 06 jul. 2011.

GUILLEN, E.; PADILLA, D.; COLORADO, Y. Weaknesses and strengths analysis over network-based intrusion detection and prevention systems. In: IEEE. *Communications, 2009. LATINCOM'09. IEEE Latin-American Conference on*. [S.l.], 2009. p. 1–5.

HUANG, Y.; LEE, W. A cooperative intrusion detection system for ad hoc networks. In: ACM. *Proceedings of the 1st ACM workshop on Security of ad hoc and sensor networks*. [S.l.], 2003. p. 135–147.

HURSON, A. et al. Recent advances in mobile agent-oriented applications. *Mobile intelligence*, Wiley Online Library, p. 106–139, 2010.

JANSEN, W. Intrusion detection with mobile agents. *Computer Communications*, Elsevier, v. 25, n. 15, p. 1392–1401, 2002.

KACHIRSKI, O.; GUHA, R. Effective intrusion detection using multiple sensors in wireless ad hoc networks. In: IEEE. *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*. [S.l.], 2003. p. 8–pp.

LEE, S.; HWANG, C. Architecture modeling and evaluation for design of agent-based system. *Journal of Systems and Software*, Elsevier, v. 72, n. 2, p. 195–208, 2004.

LIU, S.; KÜNGAS, P.; MATSKIN, M. Agent-based web service composition with JADE and JXTA. In: *Proceedings of the 2006 International Conference on Semantic Web and Web Services, SWWS*. [S.l.: s.n.], 2006. p. 26–29.

LU, K. et al. Robust and efficient detection of DDoS attacks for large-scale internet. *Computer Networks*, Elsevier, v. 51, n. 18, p. 5036–5056, 2007.

LUCK, M. Guest editorial: Challenges for agent-based computing. *Autonomous Agents and Multi-Agent Systems*, Springer, v. 9, n. 3, p. 199–201, 2004.

MAHONEY, M. Network traffic anomaly detection based on packet bytes. In: ACM. *Proceedings of the 2003 ACM symposium on Applied computing*. [S.l.], 2003. p. 346–350.

MALUCELLI, A. *Ontology-based services for agents interoperability*. Tese (Tese) — Universidade do Porto, Porto, 2006. 235 f.

MANSFIELD-DEVINE, S. DDoS: threats and mitigation. *Network Security*, Elsevier, v. 2011, n. 12, p. 5–12, 2011.

MARTINS, F. S. *Um Modelo de Diagnóstico Distribuído e Hierárquico para Tolerância a Ataques de Manipulação de Resultados em Grades Computacionais*. Dissertação (Mestrado) — Universidade Federal do Ceará, Fortaleza, 2006.

MILOJICIC, D. et al. *Peer-to-peer computing*. [S.l.]: Technical Report HPL-2002-57, HP Labs, 2002.

MUKHERJEE, B.; HEBERLEIN, L.; LEVITT, K. Network intrusion detection. *Network, IEEE*, IEEE, v. 8, n. 3, p. 26–41, 1994.

NAKKEERAN, R.; ALBERT, T.; EZUMALAI, R. Agent based efficient anomaly intrusion detection system in adhoc networks. *IACSIT International Journal of Engineering and Technology*, v. 2, n. 1, 2010.

NAM, S.; KIM, D.; KIM, J. Enhanced ARP: preventing ARP poisoning-based man-in-the-middle attacks. *Communications Letters, IEEE*, IEEE, v. 14, n. 2, p. 187–189, 2010.

NORTHCUTT, S. et al. Desvendando segurança em redes: o guia definitivo para fortificação de perímetros de rede usando firewalls, VPNs, roteadores e sistemas de detecção de intrusão. *Tradução Daniel Vieira. Rio de Janeiro: Campus*, 2002.

NYULAS, C. et al. An ontology-driven framework for deploying JADE agent systems. In: IEEE. *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*. [S.l.], 2008. v. 2, p. 573–577.

O'BRIEN, P.; NICOL, R. FIPA—towards a standard for software agents. *BT Technology Journal*, Springer, v. 16, n. 3, p. 51–59, 1998.

ÖREN, T.; YILMAZ, L. Synergies of simulation, agents, and systems engineering. *Expert Systems with Applications: An International Journal*, Pergamon Press, Inc., v. 39, n. 1, p. 81–88, 2012.

OSMAN, I.; ELSHOUSH, H. Alert correlation in collaborative intelligent intrusion detection systems-a survey. *Applied Soft Computing*, Elsevier, 2011.

PRAS, A. et al. Attacks by "anonymous" wikileaks proponents not anonymous. University of Twente, Centre for Telematics and Information Technology (CTIT), 2010. Disponível em < http://doc.utwente.nl/75331/>. Acesso em: 10 fev. 2012.

PUTTINI, R. et al. A fully distributed IDS for MANET. In: IEEE. *Computers and Communications, 2004. Proceedings. ISCC 2004. Ninth International Symposium on*. [S.l.], 2004. v. 1, p. 331–338.

ROMAN, J. et al. Launching distributed denial of service attacks by network protocol exploitation. In: WORLD SCIENTIFIC AND ENGINEERING ACADEMY AND SOCIETY (WSEAS). *Proceedings of the 2nd international conference on Applied informatics and computing theory*. [S.l.], 2011. p. 210–216.

RUWASE, O.; LAM, M. A practical dynamic buffer overflow detector. In: *Proceedings of the 11th Annual Network and Distributed System Security Symposium*. [S.l.: s.n.], 2004.

SBODIO, M.; MARTIN, D.; MOULIN, C. Discovering semantic web services using SPARQL and intelligent agents. *Web Semantics: Science, Services and Agents on the World Wide Web*, Elsevier, v. 8, n. 4, p. 310–328, 2010.

SIRICHAROEN, W. Ontology modeling and object modeling in software engineering. *International Journal of Software Engineering and Its Applications*, v. 3, n. 1, p. 43–59, 2009.

SPAFFORD, E.; ZAMBONI, D. Intrusion detection using autonomous agents. *Computer Networks*, Elsevier, v. 34, n. 4, p. 547–570, 2000.

SPANOUDAKIS, N.; MORAITIS, P. Modular JADE agents design and implementation using ASEME. In: IEEE. *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2010 IEEE/WIC/ACM International Conference on*. [S.l.], 2010. v. 2, p. 221–228.

SPEROTTO, A. et al. An overview of IP flow-based intrusion detection. *Communications Surveys & Tutorials, IEEE*, IEEE, v. 12, n. 3, p. 343–356, 2010.

SREENATH, R.; SINGH, M. Agent-based service selection. *Web Semantics: Science, Services and Agents on the World Wide Web*, Elsevier, v. 1, n. 3, p. 261–279, 2004.

THANGAVEL, M.; THANGARAJ, D.; SARAVANAN, K. *Defend against Anomaly Intrusion Detection using SWT Mechanism*. [S.l.]: IACSIT, 2010.

TRIPUNITARA, M.; DUTTA, P. A middleware approach to asynchronous and backward compatible detection and prevention of ARP cache poisoning. In: IEEE. *Computer Security Applications Conference, 1999.(ACSAC'99) Proceedings. 15th Annual*. [S.l.], 1999. p. 303–309.

UNDERCOFFER, J.; JOSHI, A.; PINKSTON, J. Modeling computer attacks: An ontology for intrusion detection. In: SPRINGER. *Recent Advances in Intrusion Detection*. [S.l.], 2003. p. 113–135.

UNDERCOFFER, J. et al. A target-centric ontology for intrusion detection. In: *18th International Joint Conference on Artificial Intelligence*. [S.l.: s.n.], 2004. p. 9–15.

WILANDER, J.; KAMKAR, M. A comparison of publicly available tools for dynamic buffer overflow prevention. In: *10th Network and Distributed System Security Symposium*. [S.l.: s.n.], 2003.

YANG, Y. et al. An agent-based workflow system for enterprise based on FIPA-OS framework. *Expert Systems with Applications*, Elsevier, v. 37, n. 1, p. 393–400, 2010.

YE, D. et al. P2P distributed intrusion detections by using mobile agents. In: IEEE. *Computer and Information Science, 2008. ICIS 08. Seventh IEEE/ACIS International Conference on*. [S.l.], 2008. p. 259–265.

YU, S.; DASGUPTA, D. An effective network-based intrusion detection using conserved self pattern recognition algorithm augmented with near-deterministic detector generation. In: IEEE. *Computational Intelligence in Cyber Security (CICS), 2011 IEEE Symposium on*. [S.l.], 2011. p. 17–24.

ZAIDI, A. *Recherche et détection des patterns d'attaques dans les réseaux IP à haut débits*. Tese (Doutorado) — Université d'Évry Val d'Essonne, Évry, 2011. 109 f.

ZDRNJA, B. Malicious javascript insertion through ARP poisoning attacks. *Security & Privacy, IEEE*, IEEE, v. 7, n. 3, p. 72–74, 2009.

ZHANG, Y.; LEE, W.; HUANG, Y. Intrusion detection techniques for mobile wireless networks. *Wireless Networks*, Kluwer Academic Publishers, v. 9, n. 5, p. 545–556, 2003.

ZHANG, Z. *Ontology query languages for the semantic Web*. Dissertação (Mestrado) — University of Georgia, Atenas, 2005.

ZHOU, C.; LECKIE, C.; KARUNASEKERA, S. A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security*, Elsevier, v. 29, n. 1, p. 124–140, 2010.

APÊNDICE A – CÓDIGOS DOS AGENTES

Com o objetivo de demonstrar o funcionamento interno dos agentos do Sistema de Detecção implementado, foram selecionados alguns trechos do código fonte dos mesmos.

A.1 Agente Sensor

A seguir, podem ser observados alguns trechos do código fonte dos agentes sensores.

```
public class SensorAgent extends Agent implements PacketReceiver{

       public static final String ArquivoCaptura = "captura2.txt";  //arquivo para salvar os pacotes
       SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
       String local = null;                                          //container local


    @Override
        protected void setup() {
            System.out.println("Hello! Sensor Agent "+getAID().getName()+" is ready.");
            try {                                                    //gravando dados no arquivo texto
                File arquivo;
                arquivo = new File(ArquivoCaptura);
                FileOutputStream fos = new FileOutputStream(arquivo,true);
                Date data = new Date();
                String texto = "Tempo_inicial: "+sdf.format(data)+"\n";    //hora do inicio da captura
                fos.write(texto.getBytes());
                fos.close();
            }
            catch (Exception ee) {
            }

            try {

                Object [] args = new Object[1];
                args[0] = ArquivoCaptura;
                ContainerController cc = getContainerController();
                try {
                    local = cc.getContainerName();
                } catch (ControllerException ex) {
                    Logger.getLogger(GerenteAgent.class.getName()).log(Level.SEVERE, null, ex);
                }
                //Chamando o agente analisador
                AgentController ac;
                try {
                    ac = cc.createNewAgent("analisador_"+local, "javajade.AnalisadorAgent", args);
                    ac.start();
                } catch (StaleProxyException ex) {
                    Logger.getLogger(AnalisadorAgent.class.getName()).log(Level.SEVERE, null, ex);
                }
                //terminou de chamar o agente analisador

                jpcap.loopPacket(-1,new SensorAgent());

            } catch (IOException ex) {
                Logger.getLogger(SensorAgent.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

    @Override
        protected void takeDown() {
                System.out.println("Sensor Agent "+getAID().getName()+" terminating.");
        }

    }

}
```

Figura A.1: Extrato do código do agente sensor

A.2 Agente Analisador

A seguir, podem ser observados alguns trechos do código fonte dos agentes analisadores.

```
public class AnalisadorAgent extends Agent{

    @Override
    protected void setup() {
        ...
        Object[] arg = getArguments();     //recebe os argumentos de quando o agente foi chamado
        if(arg != null && arg.length > 0)
        {
            mensagem = (String) arg[0];
        }
        getContentManager().registerLanguage(new SLCodec());
        getContentManager().registerOntology(MobilityOntology.getInstance());
        // Pede locais disponiveis ao AMS e recebe resposta
        sendRequest(new Action(getAMS(), new QueryPlatformLocationsAction()));
        MessageTemplate mt = MessageTemplate.and(
        MessageTemplate.MatchSender(getAMS()),
        MessageTemplate.MatchPerformative(ACLMessage.INFORM));
        ACLMessage resp = blockingReceive(mt);
        ContentElement ce = null;
        ce = getContentManager().extractContent(resp);
        Result result = (Result) ce;
        jade.util.leap.Iterator it = result.getItems().iterator();
        while (it.hasNext()) {
            Location loc = (Location)it.next();
            if (loc.getName().equals("Main-Container"))
            {
                mainIP=loc.getAddress();main
            }
        }
        ...
        ContainerID cid = (ContainerID) here();
        ...
        AnalisaPacotes analise = new AnalisaPacotes();
        analise.analisa(mensagem);              //analisar arquivo de texto indicado na mensagem
    }


    public class AnalisaPacotes{        //classe que analisa os dados do agente sensor

        public String[] palavra;
        public List<Pacote> lista=new ArrayList<Pacote>();   //lista de pacotes analisados
        ContainerController cc = getContainerController();
        AgentController ac;

        public void analisa(String entrada) throws IOException{
        ...
            String s = br.readLine();     // primeira linha do arquivo
            palavra = s.split(" ",2);     // divide a primeira linha do arquivo em duas strings para
determinar o tempo inicial
            ...
            tempoInicial = sdf.parse(palavra[1]);
            ...
            while (true){
                String[] aux = new String[10];
                aux[1]=null;
                s=br.readLine();
                if (s!=null)
                {
                bw.write(s);
                bw.write("\n");
                palavra = s.split(" ");
                Pacote temp = new Pacote();
                temp.Suspeito= false;
                temp.Controle=false;
                ...
                aux=palavra[1].split("/");
                  if((aux[0].equals("")))
                  {
                    temp.IpDestino=aux[2];
                    aux=aux[1].split("-");
                    temp.IpOrigem=aux[0];
```

Figura A.2: Extrato do código do agente analisador. Parte 1

```
                ...
                    if (palavra[2].equals("protocol(17)"))  //pacote UDP
                    {
                       aux=palavra[4].split("\\(");
                       if (aux[0].equals("flowlabel"))    //quando o pacote aparace sem IP
                       {
                          temp.Protocolo=palavra[6];
                          temp.PortaOrigem=palavra[7];
                          temp.PortaDestino=palavra[9];
                          palavra[10]=palavra[10].concat(" "+palavra[11]);
                          try {
                          temp.Data = sdf.parse(palavra[10]);
                          } catch (ParseException ex) {
                              Logger.getLogger(AnalisadorAgent.class.getName()).log(Level.SEVERE,
null, ex);
                          }
                       }else                              //quando o pacote aparece normalmente
                       {
                          temp.Protocolo=palavra[9];
                          temp.PortaOrigem=palavra[10];
                          temp.PortaDestino=palavra[12];
                          palavra[13]=palavra[13].concat(" "+palavra[14]);
                          try {
                             temp.Data = sdf.parse(palavra[13]);
                          ...

                       int j = 0;
                       boolean naoachou = true;
                       while ((j < lista.size())&& naoachou) {
                          if (lista.get(j).Protocolo.equals(temp.Protocolo)&&lista.get
(j).IpOrigem.equals(temp.IpOrigem)&&lista.get(j).IpDestino.equals(temp.IpDestino))
                          {
                             naoachou = false;
                             lista.get(j).Intervalo=temp.Data.getTime()-lista.get(j).Data.getTime();
                             lista.get(j).Data=temp.Data;
                             lista.get(j).Contador++;
                             Detecta(j, lista);              //implementa as regras de detecção
                          }
                          j++;
                       }
                       if (naoachou)
                       {
                          temp.Intervalo = temp.Data.getTime()-tempoInicial.getTime();
                          temp.Contador = 1;
                          lista.add(temp);
                          Detecta(lista.indexOf(temp), lista);
                       }
                ...
                RecebeMsg(lista);
                ...

     public void RecebeMsg(List<Pacote> listabusca){

        MessageTemplate mtInform = MessageTemplate.MatchPerformative(ACLMessage.INFORM);  //template
para msgs tipo INFORM
        MessageTemplate mtRequest = MessageTemplate.MatchPerformative(ACLMessage.REQUEST);  //template
para msgs tipo REQUEST
        MessageTemplate mtPropose = MessageTemplate.MatchPerformative(ACLMessage.PROPOSE);  //template
para msgs tipo PROPOSE
        MessageTemplate mtQuery = MessageTemplate.MatchPerformative(ACLMessage.QUERY_IF );  //template
para msgs tipo QUERY_IF
        MessageTemplate mtQuery2 = MessageTemplate.MatchPerformative(ACLMessage.QUERY_REF );  //
template para msgs tipo QUERY_REF
        String[] info;          //array de string para trabalhar com as informações contidas nas
mensagens

        ContainerController cc = getContainerController();
        try {
            local = cc.getContainerName();
        } catch (ControllerException ex) {
            Logger.getLogger(GerenteAgent.class.getName()).log(Level.SEVERE, null, ex);
        }
```

Figura A.3: Extrato do código do agente analisador. Parte 2

```
        ACLMessage msg = receive(mtInform);  //recebe mensagens dos agentes gerentes NEGATIVA
        ...
        msg = receive(mtQuery2);        //recebe msg de agente gerente externo, pedindo pra verificar
listas de suspeitos
        if (msg!=null){
            String mensagem = msg.getContent();         //copia o conteudo da mensagem
            info = mensagem.split(" ",5);               //divide o conteudo em quatro partes,
separando nos espaços
            Object [] args = new Object[5];
            args = info;
            int k = 0;
            boolean naoachou = true;
            while ((k < listabusca.size())&& naoachou) {        //verifica se a informacao recebida e
de um suspeito
                if (listabusca.get(k).Protocolo.equals(args[0])&&listabusca.get(k).IpOrigem.equals(args
[1])&&listabusca.get(k).Suspeito)
                {
                    naoachou = false;
                }
                k++;
            }
            ACLMessage reply = msg.createReply();
            reply.setPerformative( ACLMessage.INFORM );
            if(!naoachou){              //se for suspeito, cria reposta dizendo que e uma intrusao
                reply.setContent("Ocorreu intrusao");
            }else
            {
                reply.setContent("Nao detectado");
            }
            send(reply);
        }

        msg = receive(mtQuery);            //recebe msg dos agentes gerentes POSITIVA
        if (msg != null){
            //tratar a mensagem recebida, chamando um agente atuador
            String mensagem = msg.getContent();         //copia o conteudo da mensagem
            info = mensagem.split(" ",5);               //divide o conteudo em quatro partes,
separando nos espaços
            Object [] args = new Object[5]; args = info;
            //Chamando o agente atuador
            ...
            AgentController ac;
            try {
                ac = cc.createNewAgent("atuador_"+local+(numAtuador++), "javajade.AtuadorAgent", args);
                ac.start();
            } catch (StaleProxyException ex) {
                Logger.getLogger(AnalisadorAgent.class.getName()).log(Level.SEVERE, null, ex);
            }
            //Chama o agente de ontologia para acrescentar a invasao na ontologia
            Object [] args2 = new Object[6];
            args2[0]="inclusao"; args2[1]=args[0]; args2[2]=args[1];
            args2[3]=args[2];    args2[4]=args[3]; args2[5]=args[4];
            try {
                ac = cc.createNewAgent("ontologia_"+local+(numOntologia++), "javajade.OntologiaAgent",
args2);
                ac.start();
         ...
```

Figura A.4: Extrato do código do agente analisador. Parte 3

A.3 Agente de Ontologia

A seguir, podem ser observados alguns trechos do código fonte dos agentes de ontologia.

```
public class OntologiaAgent extends Agent{
    ...
    private Ontology ontology = OntologiaDeteccao.getInstance();
    ...
    protected void setup() {
       ...
       Object[] args = getArguments();     //recebe os argumentos de quando o agente foi chamado
       if(args != null && args.length > 0){
          proposito = (String) args[0]; protocolo = (String) args[1];
          endOrigem = (String) args[2]; ptOrigem = (String) args[3];
          endDest = (String) args[4];   ptDest = (String) args[5];
       }
       try {
          ContainerController cc = getContainerController();
          local = cc.getContainerName();
          ...
       //se nao estiver no main-container, se move para lá
       if (!(local.equals("Main-Container"))){
          ...                  // Registrar linguagem e ontologia
          getContentManager().registerLanguage(new SLCodec());
          getContentManager().registerOntology(MobilityOntology.getInstance());
          ContainerID dest1 = new ContainerID("Main-Container", null);
          doMove(dest1);
       }else
       {
          ACLMessage msg = null;
          // Registrando linguagem e ontologia
          getContentManager().registerLanguage(codec);
          getContentManager().registerOntology(ontology);
          if (proposito.equals("consulta")){
             msg = new ACLMessage(ACLMessage.QUERY_IF);
          }
          if (proposito.equals("inclusao")){
             msg = new ACLMessage(ACLMessage.INFORM);
          }
          msg.addReceiver(new AID("ontologiaglobal", AID.ISLOCALNAME));
          msg.setLanguage(codec.getName());
          msg.setOntology(ontology.getName());
          // Prepara o conteúdo
          Host attack = new Host(); attack.setEnd(endOrigem);
          attack.setTipoEnd(protocolo);
          Host target = new Host(); target.setEnd(endDest);
          target.setTipoEnd(protocolo); Suspeita susp = new Suspeita();
          susp.setAtacante(attack);
          ...
          try {
              // Converte de objetos JAVA para string
              getContentManager().fillContent(msg, susp);
              send(msg);
          }
          ...
          if (proposito.equals("consulta")){
              msg = blockingReceive();        //recebe resposta da msg enviada à ontologia global;
              resposta = msg.getContent();
              ...
          if (resposta.equals("Ocorreu intrusao")){
              ...
              msg = new ACLMessage(ACLMessage.REQUEST);
          }
          if (resposta.equals("Nao detectada")){
              ...
              msg = new ACLMessage(ACLMessage.PROPOSE);
          }
          msg.addReceiver(new AID("analisador_"+local, AID.ISLOCALNAME));
          msg.setContent(protocolo+" "+endOrigem+" "+ptOrigem+" "+endDest+" "+ptDest);
          send(msg);
       }
       doDelete();
       ...
```

Figura A.5: Extrato do código do agente de ontologia. Parte 1

```
    protected void afterMove() {
        ...
        ACLMessage msg = null;
        if (!moveu){    //agente está no Main-Container
            moveu=true;
            ...
            // Registrando linguagem e ontologia de novo
            getContentManager().registerLanguage(codec);
            getContentManager().registerOntology(ontology);
            if (proposito.equals("consulta")){
                msg = new ACLMessage(ACLMessage.QUERY_IF);
            }
            if (proposito.equals("inclusao")){
                msg = new ACLMessage(ACLMessage.INFORM);
            }
            msg.addReceiver(new AID("ontologiaglobal", AID.ISLOCALNAME));
            msg.setLanguage(codec.getName());
            msg.setOntology(ontology.getName());
            // Prepara os conteúdos
            Host attack = new Host(); attack.setEnd(endOrigem); attack.setTipoEnd(protocolo);
            Host target = new Host(); target.setEnd(endDest); target.setTipoEnd(protocolo);
            Suspeita susp = new Suspeita(); susp.setAtacante(attack);
            ...
            susp.setAtacantePorta(ptOrigem); susp.setAlvo(target);
            ...
            susp.setAlvoPorta(ptDest);
            ...
            try {
               // Converte objetos java para strings
               getContentManager().fillContent(msg, susp);
               send(msg);
            }
            ...
            if (proposito.equals("consulta")){
               msg = blockingReceive();               //recebe resposta da msg enviada à ontologia
global;
               resposta = msg.getContent();
            }
            //registra ontologia de mobilidade de novo, para se mover
            getContentManager().registerOntology(MobilityOntology.getInstance());
            ContainerID dest1 = new ContainerID(local, null);
            doMove(dest1);
        }else           //agente voltou para seu container de origem
        {
            if(proposito.equals("consulta")){
               if (resposta.equals("Ocorreu intrusao")){
                  msg = new ACLMessage(ACLMessage.REQUEST);
               }
               if (resposta.equals("Nao detectada")){
                  msg = new ACLMessage(ACLMessage.PROPOSE);
               }
               msg.addReceiver(new AID("analisador_"+local, AID.ISLOCALNAME));
               msg.setContent(protocolo+" "+endOrigem+" "+ptOrigem+" "+endDest+" "+ptDest);
               send(msg);
            }
            doDelete();
        }
    }
```

Figura A.6: Extrato do código do agente de ontologia. Parte 2

A.4 Agente Gerente

A seguir, podem ser observados alguns trechos do código fonte dos agentes gerentes.

```
public class GerenteAgent extends Agent{
    ...
    protected void setup() {
        ...
        Object[] args = getArguments();     //recebe os argumentos
        if(args != null && args.length > 0)
        {
            protocolo = (String) args[0];  endOrigem = (String) args[1];
            ptOrigem = (String) args[2];   endDest = (String) args[3];
            ptDest = (String) args[4];
        }
        ContainerController cc = getContainerController();
        ...
            local = cc.getContainerName();
        ...
        // Registra linguagem e ontologia
        getContentManager().registerLanguage(new SLCodec());
        getContentManager().registerOntology(MobilityOntology.getInstance());
        // Descobrir locais disponíveis com o AMS
        sendRequest(new Action(getAMS(), new QueryPlatformLocationsAction()));
        //Recebe resposta do AMS
        MessageTemplate mt = MessageTemplate.and(
        MessageTemplate.MatchSender(getAMS()),
        MessageTemplate.MatchPerformative(ACLMessage.INFORM));
        ACLMessage resp = blockingReceive(mt);
        ContentElement ce = null;
        ...
        ce = getContentManager().extractContent(resp);
        ...
        Result result = (Result) ce;
        jade.util.leap.Iterator it = result.getItems().iterator();
        while (it.hasNext()) {
            Location loc = (Location)it.next();
            ...
            containers.add(loc.getName());
            locations.put(loc.getName(), loc);
        }
        ...
        containerDestino = containers.get(containerCont);
        ContainerID dest1 = new ContainerID(containerDestino, null);
        doMove(dest1);
    }
    ...
    protected void afterMove() {
        ...
        if(!moveu){
            ...
            ACLMessage msg = new ACLMessage(ACLMessage.QUERY_REF);
            msg.addReceiver(new AID("analisador_"+containerDestino,
```

Figura A.7: Extrato do código do agente gerente. Parte 1

```
AID.ISLOCALNAME));
            msg.setContent(protocolo+" "+endOrigem+" "+ptOrigem+" "+endDest+"
"+ptDest);
            send(msg);
            ...
            ACLMessage msg2 = blockingReceive(); //resposta da msg enviada
analisador local;
            resposta = msg2.getContent();
            ...
            // vai para outro container ou volta para container de origem
            containerCont++;
            ...
            if (containerCont<containers.size())
            {
                containerDestino=containers.get(containerCont);
                dest1 = new ContainerID(containerDestino, null);
                if (resposta.equals("Ocorreu intrusao"))
                {
                    dest1 = new ContainerID(local, null);
                    //vai voltar para container origem
                    moveu=true;
                }
            }else{
                dest1 = new ContainerID(local, null);
                //vai voltar para container origem
                moveu=true;
            }
            doMove(dest1);
       }else
       {
            if (resposta.equals("Nao detectado"))
            {
                ACLMessage msg3 = new ACLMessage(ACLMessage.INFORM);
                msg3.addReceiver(new AID("analisador_"+local, AID.ISLOCALNAME));
                msg3.setContent(protocolo+" "+endOrigem+" "+ptOrigem+" "+endDest+"
"+ptDest);
                send(msg3);
            }
            if (resposta.equals("Ocorreu intrusao"))
            {
                ACLMessage msg4 = new ACLMessage(ACLMessage.QUERY_IF);
                msg4.addReceiver(new AID("analisador_"+local, AID.ISLOCALNAME));
                msg4.setContent(protocolo+" "+endOrigem+" "+ptOrigem+" "+endDest+"
"+ptDest);
                send(msg4);
            }
            doDelete();
        }
    }
    ...
}
```

Figura A.8: Extrato do código do agente gerente. Parte 2

A.5 Agente Ontologia Global

A seguir, podem ser observados alguns trechos do código fonte do agente ontologia global.

```
public class OntologiaGlobalAgent extends Agent {
    public class Pacote{
        Host Origem;
        Host Destino;
        String PortaOrigem;
        String PortaDestino;
    }
    public List<Pacote> lista=new ArrayList<Pacote>();
    private Codec codec = new SLCodec();
    private Ontology ontology = OntologiaDeteccao.getInstance();

    protected void setup(){
        getContentManager().registerLanguage(codec);    //registrando codec
        getContentManager().registerOntology(ontology); //registrando ontologia
        ...
        MessageTemplate mt = MessageTemplate.and(
            MessageTemplate.MatchLanguage(codec.getName()),
            MessageTemplate.MatchOntology(ontology.getName()) );
        while (true)
        {
            ACLMessage msg = blockingReceive(mt);
            try {
                ContentElement ce = null;
                if (msg.getPerformative() == ACLMessage.QUERY_IF)  //foi feita uma
consulta à ontologia
                {
                  // conversão de String para objetos Java
                    ce = getContentManager().extractContent(msg);
                    if (ce instanceof Suspeita) {
                        Suspeita suspeita = (Suspeita) ce;
                        int k = 0;
                        boolean naoachou = true;
                        if (!lista.isEmpty())
                        {
                            while ((k < lista.size())&& naoachou) {   //verifica se
a informacao recebida é de um suspeito
                                if (lista.get(k).Origem.End.equals
(suspeita.getAtacante().End))
                                {
                                    naoachou = false;
                                }
                                k++;
                            }
                        }
                        ACLMessage reply = msg.createReply();
                        reply.setPerformative( ACLMessage.INFORM );
                        if(naoachou){
                            reply.setContent("Nao detectada");    //resposta para a
consulta
```

Figura A.9: Extrato do código do agente ontologia global. Parte 1

```
                        }else
                        {
                            reply.setContent("Ocorreu intrusao"); //resposta para a
consulta
                        }
                        send(reply);
                    }
                }
                if (msg.getPerformative() == ACLMessage.INFORM) //foi sugerida uma
adicao à ontologia
                {
                        // conversão de String para objetos Java
                        ce = getContentManager().extractContent(msg);
                        if (ce instanceof Suspeita)
                        {
                            Suspeita suspeita = (Suspeita) ce;
                            int k = 0;
                            boolean naoachou = true;
                            if (!lista.isEmpty())
                            {
                                while ((k < lista.size())&& naoachou) {  //verifica
se a informacao recebida é de um suspeito
                                    if (lista.get(k).Origem.End.equals
(suspeita.getAtacante().End))
                                    {
                                        naoachou = false;
                                    }
                                    k++;
                                }
                            }
                            if(naoachou)
                            {
                                Pacote temp = new Pacote();
                                temp.Origem=suspeita.getAtacante();
                                temp.PortaOrigem=suspeita.getAtacantePorta();
                                temp.Destino=suspeita.getAlvo();
                                temp.PortaDestino=suspeita.getAlvoPorta();
                                lista.add(temp);
                            }
                        }
                    }
                }
                ...
        }
    }
    ...
}
```

Figura A.10: Extrato do código do agente ontologia global. Parte 2

A.6 Agente Atuador

A seguir, podem ser observados alguns trechos do código fonte do agente atuador.

```
public class AtuadorAgent extends Agent {

    public static final String ArquivoResposta = "Alerta.txt";
    SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss"); //indica o
formato das variaveis do tipo Date

    protected void setup(){
        ...
        File arquivo;
        String protocolo = null;
        String endOrigem = null;
        String endDest = null;
        String ptOrigem = null;
        String ptDest = null;

        arquivo = new File(ArquivoResposta);
        FileOutputStream fos;
        fos = new FileOutputStream(arquivo, true);
        Object[] args = getArguments();     //recebe os argumentos de quando o
agente foi chamado
        if(args != null && args.length > 0)
        {
            protocolo = (String) args[0];
            endOrigem = (String) args[1];
            ptOrigem = (String) args[2];
            endDest = (String) args[3];
            ptDest = (String) args[4];
        }
        Date data = new Date();
        String texto = sdf.format(data)+" -- Protocolo: "+protocolo+" End Origem:
"+endOrigem+" Porta Origem: "+ptOrigem+" - End Destino: "+endDest+" Porta Destino:
"+ptDest+"\n";
        fos.write(texto.getBytes());
        fos.close();
        ...
        doDelete();
    }
}
```

Figura A.11: Extrato do código do agente atuador