UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

Rômulo Alves Dias

UM MODELO DE ATUALIZAÇÃO AUTOMÁTICA DO MECANISMO DE DETECÇÃO DE ATAQUES DE REDE PARA SISTEMAS DE DETECÇÃO DE INTRUSÃO

São Luís
2003

Rômulo Alves Dias

UM MODELO DE ATUALIZAÇÃO AUTOMÁTICA DO MECANISMO DE DETECÇÃO DE ATAQUES DE REDE PARA SISTEMAS DE DETECÇÃO DE INTRUSÃO

Dissertação de Mestrado submetida à Coordenação do Curso de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão para obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Edson Nascimento

São Luís
2003

UM MODELO DE ATUALIZAÇÃO AUTOMÁTICA DO MECANISMO DE DETECÇÃO DE ATAQUES DE REDE PARA SISTEMAS DE DETECÇÃO DE INTRUSÃO

Rômulo Alves Dias

Dissertação aprovada em 21 de novembro de 2003.

_____

Prof. Dr. Edson Nascimento
(Orientador)

_____

Prof. Dr. Edson Costa de Barros Carvalho Filho
(Membro da Banca Examinadora)

_____

Prof. Dr. Zair Abdelouahab
(Membro da Banca Examinadora)

A Deus.

A meus pais.

Às mais novas belas Dias de minha vida, Giselle e Amanda.

AGRADECIMENTOS

A Deus, pela oportunidade que me foi concedida de realizar o mestrado.

Ao Prof. Edson Nascimento, pela orientação, confiança e incentivo que foram de fundamental importância para a conclusão desta dissertação.

A minha esposa Giselle, pela compreensão, carinho e apoio incondicionais.

Aos amigos Bruno Feres, Christiane Lima e Nilson Santos, pelo companheirismo e auxílio técnico indispensáveis.

Aos meus pais, irmãos, sogros, cunhados, sobrinhos que se mantiveram firmes ao meu lado.

A Rogério Dias, que novamente auxiliou na finalização de minhas pesquisas.

A todos os outros que contribuíram e que certamente não serão esquecidos.

“Não sabendo que era impossível, foi lá e fez.”

Paulo Leminsk

RESUMO

A obsolescência dos mecanismos de identificação de ataques dos SDI´s tem comprometido de forma crítica o nível de segurança das redes de computadores. Esta dissertação apresenta uma proposta de um modelo de atualização automática do mecanismo de detecção de ataques de rede para sistemas de detecção de intrusão baseados na noção de sociedade de agentes inteligentes. A Agência Central de Segurança integrante deste modelo distribui uma mini-sociedade de agentes de detecção de ataque, denominada de SAARA, que utiliza uma rede neural treinada a partir de dados coletados de diversas fontes de tráfego. Uma implementação computacional da SAARA, abordando detecção de ataques orientados a dados, é apresentada dentro do contexto do SDI multiagentes NIDIA.

Palavras Chaves: detecção de intrusos, segurança de redes de computadores, ataques baseados em conteúdo, redes neurais.

ABSTRACT

The obsolescence of the IDS's attack identification mechanisms critically compromises the security level of the networks. This research work presents a proposal of a automatic updating model of the network attack detection mechanism for intrusion detection systems based on a society of intelligent agents. The Security Central Agency, a component of the model, distributes a mini-society of attack detection agents, called SAARA, that uses a neural network trained with data captured from several network traffic sources. A computational implementation of the SAARA model, focusing data driven attack detection, is presented for the multiagent IDS NIDIA.

Keywords: intrusion detection, network security, data driven attacks, neural networks.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ACS	Agência Central de Segurança
ANS	*Agent Name Server*
BD	Banco de Dados
CIDF	*Common Intrusion Detection Framework*
DARPA	*Defense Advanced Research Projects Agency*
DFDB	Incidentes de Intrusão e Informação Forense
DOS	*Denial of Service*
IDS	*Intrusion Detection System*
IIDB	Padrão de Intrusos e Intrusões
IP	*Internet Protocol*
NIDES	*Next-Generation Intrusion Detection Expert System*
NIDIA	*Network Intrusion Detection System based on Intelligent Agents*
NNID	*Neural Network Intrusion Detector*
NSM	*Network Security Monitor*
PCA	*Principal Component Analysis*
PHF	*Phone Book Script*
RADB	Base de Dados de Ações
SAA	Agente de Atualização do Sistema
SAARA	Sociedade Atualizada de Agentes de Reconhecimento de Assinaturas
SCA	Agente Controlador de Ações
SCIA	*Security Central Inteligence Agency*
SDI	Sistema de Detecção de Intrusão
SEA	Agente de Avaliação de Segurança
SIA	Agente de Integridade do Sistema

SMA	Agente de Monitoramento do Sistema
SNNS	*Stuttgart Neural Network Simulator*
STDB	Base de Dados de Estratégias
TCP	*Transmission Control Protocol*
VPN	*Virtual Private Network*

SUMÁRIO

1 INTRODUÇÃO

A presença direta ou indireta do uso de rede de computadores na manipulação de informações tornou-se definitivamente essencial na vida moderna. O advento desta tecnologia contribuiu de forma significativa para a troca de informação nos dias atuais, devido a facilidades como rapidez no intercâmbio e gerenciamento de conhecimento, diminuição de custos, acesso simplificado à informação etc. Tais vantagens fizeram com que, tanto no âmbito corporativo quanto doméstico, as redes de computadores, em particular a Internet, fossem amplamente utilizadas para transmissão e compartilhamento de dados.

É também fato notório que atualmente os maiores ativos das companhias são dados próprios e/ou de seus clientes, cuja confidencialidade lhes garante poder competitivo no mercado e credibilidade perante a seus usuários. Devido à natureza e à importância do tratamento eletrônico do conhecimento, a possibilidade da ocorrência de incidentes de segurança tem sido uma preocupação constante dos administradores de redes. Segundo [28], são considerados incidentes de segurança quaisquer eventos que interrompam os procedimentos normais causando algum nível de crise, tais como invasões de computador, ataques de negação de serviço, furto de informações por pessoal interno etc.

Outro fato relevante é o crescente número de invasões e tentativas de ataque a rede de computadores [10] visando à obtenção, adulteração e destruição de dados privados, motivados por interesses econômicos e até mesmo por atos de vandalismo. Este crescimento de incidentes intrusivos está diretamente vinculado ao

surgimento de ferramentas que permitem aos usuários com pouco conhecimento se aventurar em invasões a ambientes corporativos, muitas vezes com registro de sucesso. Segundo [9], as fraudes em informática têm representado prejuízo para as empresas na ordem de bilhões de dólares, quando analisados em escala mundial.

Em face deste cenário, torna-se evidente a necessidade de ferramentas automatizadas que garantam a segurança dos dados das empresas no mundo digital. Recursos como *firewall* [38], Sistemas de Detecção de Intrusão, VPN´s (*Virtual Private Networks*) e antivírus são usados com cada vez mais freqüência pelas corporações no intuito de compor um ambiente seguro. No entanto, a atualização incorreta destes "kits de segurança" pode comprometer de forma crítica todo o investimento feito pelas empresas na área de proteção de suas informações.

1.1 Definição do Problema

O complexo ambiente informatizado em vigor é constituído de softwares e hardwares heterogêneos com capacidade de interoperabilidade para alcançar o objetivo maior de controlar e gerenciar de forma eficiente o conhecimento. Verifica-se que neste panorama a segurança perfeita é um mito. A onipresença da conectividade de rede e erros inerentes ao desenvolvimento de software [28] ajudam a explicar esse fato. Segundo [28], a única certeza que se pode ter é que incidentes ocorrerão.

Verifica-se que devido às dezenas de vulnerabilidades de software descobertas por semana e/ou devido a má configuração de sistemas (quase 100% dos ataques ocorrem sobre vulnerabilidades já conhecidas) faz-se emergir a

urgência de ferramentas de segurança que sejam facilmente adaptáveis à singularidade deste ambiente instável.

Atualmente os SDI's[1] têm aumentado a sua participação no rol das ferramentas de segurança em conjunto com os *firewalls*. Basicamente um SDI é um software que coleta informações de tráfego de rede e/ou logs de servidores e posteriormente analisa esses dados no intuito de verificar se alguma atividade suspeita está ocorrendo. Torna-se útil quando algum invasor ultrapassa o bloqueio do *firewall* ou quando o atacante já está presente na rede interna. Em relação a esta última situação, existe uma pesquisa de âmbito nacional [32] analisando o grau de incidência de problemas de segurança causados pelos próprios funcionários das empresas.

Um modelo que didaticamente representa o funcionamento de um SDI, mostrando o fluxo de informações e as suas funcionalidades básicas é o *Common Intrusion Detection Framework* – CIDF [37].

[1] SDI (Sistema de Detecção de Intrusão) ou IDS (Intrusion Detection System).

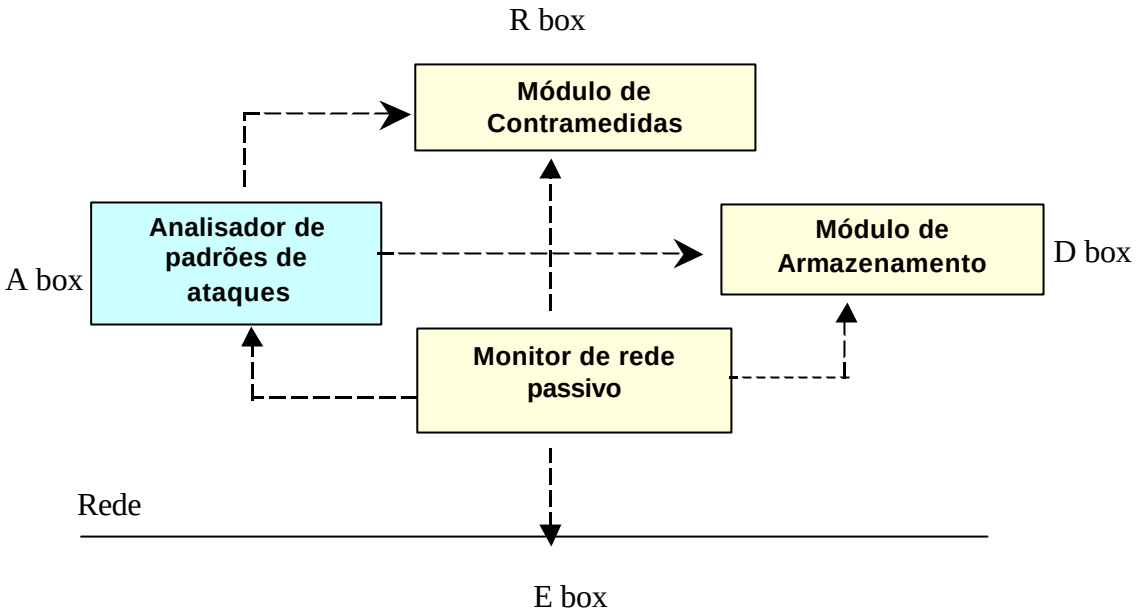

Figura 1.1 *Common Intrusion Detection Framework* com capturador de pacotes na rede [37].

O CIDF é composto basicamente de 4 tipos básicos:

E boxes (*event generators* - geradores de eventos) – são os sensores responsáveis por captar a ocorrência de eventos e fornecer informações sobre os mesmos para o resto do sistema. Podem ser considerados eventos a presença de um pacote transitando na rede, o registro de alguma atividade no log, etc.;

A boxes (analysers - analisadores) – as informações sobre os eventos coletados pelos *E boxes* são repassadas a esses mecanismos de análise para que se verifique se algo suspeito está em andamento;

D boxes (*database components* – mecanismos de armazenamento) – responsáveis por manter informações provenientes dos geradores de eventos e dos

mecanismos de análise para que possam ser apreciadas pelos operadores do sistema no futuro, como por exemplo dados de um atacante reincidente;

R boxes (response boxes – módulos de resposta) - caso um problema de segurança seja detectado, os módulos de resposta são responsáveis por tomar alguma contramedida. Por exemplo, interromper uma conexão TCP (*Transmission Control Protocol)*.

Tomando por base esta arquitetura, percebe-se que um grande desafio a ser enfrentado pelos administradores de segurança de rede é manter os *"A boxes"* de seus SDI´s atualizados, isto é, possuir uma ferramenta detectora de intrusão capaz de identificar as variações de técnicas de ataque que surgem com freqüência. [27] ratifica que "para o IDS proteger um sistema com maior segurança, deve estar sempre em atualização, pois constantemente são encontradas novas vulnerabilidades".

Em um contexto mais abrangente, pode-se afirmar que os conjuntos de ferramentas automatizadas para integrar as defesas das redes corporativas devem acompanhar o ritmo de inovação e diversificação das técnicas de ataque, sob risco dos mesmos não serem mais efetivos. Em se tratando particularmente dos sistemas de detecção de intrusão, deve-se ter alguma forma de manter atualizados os seus módulos identificadores de ataques correspondentes aos *analysers* do CIDF (Figura 1.1). Isto é devido a fatores como:

- O estrito relacionamento entre as classes de ataques de rede e as vulnerabilidades derivadas de erros de implementação ou abuso de alguma ação legítima;
- O surgimento constante de softwares com elevado grau de complexidade;
- O pouco comprometimento de programadores em desenvolver softwares seguros;
- O crescente número de indivíduos interessados em se aventurar em investidas a redes de computadores por motivos diversos;
- A popularização da internet como um grande acervo de ferramentas prontas de ataque que podem ser usadas por usuários que não necessariamente possuem um grande conhecimento técnico.

1.2 Referencial Teórico

Devido à diversidade da natureza dos ataques, o domínio das técnicas de detecção de intrusão é dividido em duas classes [8]:

- Detecção por anomalia – trata-se da definição de perfis que representem um comportamento normal no uso de recursos do sistema por parte de uma categoria de usuários, hosts ou conexões de redes. Assim durante um período considerado normal de atividade do sistema, procura-se mensurar o que seria um comportamento aceitável. Desta forma, um possível ataque é identificado quando as

ações executadas por determinado usuário diferem do seu perfil previamente estabelecido.

- Detecção por abuso - comparam-se as ações do usuário com padrões de ataques já conhecidos. A detecção por abuso ou análise de assinaturas de ataques, consiste na verificação de eventos no intuito de encontrar tentativas de exploração de vulnerabilidades de softwares notoriamente reconhecidas como abusivas e que visam causar algum dano ao sistema.

Dentro desta perspectiva, pesquisas foram desenvolvidas para criar formas automatizadas de detecção de intrusão. Em [20] é apresentado o NIDES (*Next-Generation Intrusion Detection Expert System*) como proposta de um SDI capaz de reunir ambas categorias de detecção de intrusão através da análise de dados auditados em logs do sistema operacional. O detector por anomalia do NIDES utiliza métodos estatísticos para comparar os modelos de perfis “normais” com atividades correntes dos usuários. Quando determinada atividade ultrapassa os limites da normalidade, é possível que esteja ocorrendo alguma atividade maliciosa. O detector por abuso utiliza um sistema especialista para analisar os dados auditados e verificar se alguma das regras que definem ataques conhecidos tem todas as suas condições satisfeitas.

Aparece também como tecnologia alternativa para detecção por anomalia o uso de redes neurais [18]. O NNID (*Neural Network Intrusion Detector*) utiliza uma rede neural para modelar o comportamento de um usuário legítimo, para poder reconhecer as atividades de um atacante que está usando a conta deste usuário [40].

Outras fontes de dados amplamente utilizadas para verificar a ocorrência de incidentes de segurança são pacotes captados do tráfego de rede através de monitores de rede passivos[2]. Diferentes partes da estrutura destes pacotes podem ser úteis para realizar incursões de ataques, como será visto no Capítulo 2. O NSM (*Network Security Monitor*) [19] realiza a contagem de palavras-chaves na porção de dados dos pacotes TCP e posteriormente utiliza um sistema baseado em regras para identificar ataques conhecidos através da análise dos contadores das palavras-chaves.

Em [7] são destacadas vantagens do uso de redes neurais para detecção de ataques por abuso em relação ao uso de sistemas baseados em regras. Dentre elas, pode-se citar a capacidade de generalização de uma rede neural que permite a emissão de um parecer sobre determinada porção de dados para a qual a rede nunca foi apresentada. Um sistema especialista por sua vez só indicará a presença de um problema caso haja todas as condições necessárias para que uma de suas regras dispare. Sendo assim, mesmo que ocorra um ataque que seja uma variação de um outro presente na base de regras do sistema, o sistema especialista não apontará nenhum incidente. Citam-se como desvantagens do uso de redes neurais a necessidade de uma massa de treinamento para o domínio de aprendizado, assim como uma metodologia de treinamento apropriada.

Com o intuito de usufruir as vantagens apresentadas pelas redes neurais, a pesquisa de [24] apresenta uma abordagem utilizando esta tecnologia para a identificação de ataques através da verificação de palavras-chaves encontradas nos

[2] Monitores de rede passivos utilizam interfaces de rede no modo promíscuo, isto é, capturam cópias dos pacotes que trafegam na rede independente do endereço de destino. Também são denominados de *sniffers*.

dados dos pacotes TCP. Esta referida pesquisa almejou suprir deficiências intrínsecas do *Baseline System* [49], arquitetura derivada do NSM, devido ao uso de sistemas baseados em regras. Maiores detalhes sobre a proposta de [24] são mostrados no Capítulo 2.

Em [23] é apresentada uma proposta de arquitetura para um SDI multiagentes, denominado NIDIA, com o intuito de analisar tanto dados oriundos de tráfego de rede quanto logs de hosts, através de técnicas de detecção por abuso e por anomalia. A referida pesquisa focou o modelo e a implementação dos agentes responsáveis pela coleta dos dados a serem inspecionados e relacionou dentre as sugestões para trabalhos futuros a verificação destes dados em busca de atividade maliciosa.

A Tabela 1.1 mostra um resumo comparativo dos SDI's mencionados anteriormente:

SDI / Características		NIDES	NNID	NSM	NIDIA
Métodos de detecção	Anomalia	X	X		X
	Abuso	X		X	X
Fontes de dados	Logs	X	X		X
	Tráfego de rede			X	X
Tecnologia	Sistemas Baseados em Regras	X		X	**?**
	Redes Neurais		X		
	Análise Estatística	X			

Tabela 1.1 Comparativo entre sistemas de detecção de intrusão.

Por fim, observou-se durante o estudo sobre o estado da arte do desenvolvimento de SDI's a escassez de literatura no que se refere à atualização dos módulos detectores de ataques.

1.3 Objetivo Geral e Específico da Dissertação

Esta dissertação visa propor um modelo de atualização do mecanismo de detecção para SDI´s baseados na tecnologia de sociedade de agentes inteligentes. O mecanismo de detecção apresentado é genérico e flexível, permitindo que diferentes sistemas de detecção multiagentes possam utilizá-lo, desde que em conformidade com as especificações requeridas. Desta forma, pretende-se contribuir para a disseminação de pesquisas nesta área, almejando maiores avanços nas técnicas de atualização automática do conjunto de ferramentas de segurança das corporações. Dentro desta arquitetura serão enfocadas técnicas de detecção por abuso de ataques de rede. Este tipo de ataque está sendo fortemente praticado devido à facilidade de obtenção de programas prontos para este fim e pelo crescente uso de redes de computadores.

Esta dissertação apresenta como objetivos específicos:

a. apresentar um modelo de criação de uma sociedade de agentes específica para detecção de ataques (SAARA);
b. apresentar uma metodologia de treinamento para uma rede neural utilizada na identificação de atividades intrusivas;

c. apresentar a aplicabilidade do mecanismo de detecção proposto em uma arquitetura de SDI multiagentes, o NIDIA;
d. apresentar a implementação de um protótipo do mecanismo de identificação de ataques para o NIDIA.

1.4 Organização da Dissertação

Esta dissertação está dividida em quatro capítulos. No Primeiro Capítulo é apresentado um panorama sobre a dependência digital em que atualmente se encontram as atividades humanas e a necessidade de se proteger os sistemas computadorizados e suas informações com ferramentas de atualização automática.

O Capítulo 2 apresenta conceitos sobre a detecção de ataques de rede e propõe um modelo de atualização automática para o mecanismo de detecção de intrusão, enfatizando a metodologia de treinamento da rede neural utilizada no modelo. É também apresentada uma implementação parcial da metodologia de treinamento da rede neural.

O Capítulo 3 apresenta a utilização da sociedade de agentes de detecção (SAARA) no SDI multiagentes NIDIA. É também mostrada a construção de um protótipo do mecanismo de detecção de intrusão para o NIDIA baseado na SAARA.

O Capítulo 4 apresenta as considerações finais da dissertação, ressaltando as contribuições das pesquisas realizadas. São apresentadas também sugestões para trabalhos futuros.

2 PROPOSTA DE UM MODELO DE ATUALIZAÇÃO AUTOMÁTICA PARA O MECANISMO DE DETECÇÃO DE ATAQUES DE REDES

Este capítulo aborda conceitos sobre a identificação de ataques a redes de computadores e propõe a utilização de um modelo de atualização automática de detecção de intrusão para um SDI. Apresenta-se, também, uma implementação parcial do modelo proposto com foco em detecção de ataques de rede orientados a dados.

2.1 Mecanismos de Detecção de Ataques de Rede

O processo de detecção de atividades agressoras ocupa um papel fundamental nas funções de um SDI. Segundo [28], “a detecção é a primeira etapa da resposta”. Portanto, para que o sistema possa tomar as medidas necessárias para proteger a rede e começar o processo de rastrear o possível invasor, é necessário que o ataque seja identificado de forma rápida e precisa.

O ponto central é procurar por peculiaridades que permitam de alguma forma automatizar a detecção da ocorrência deste tipo de evento. Segundo [34], “uma assinatura define ou descreve um padrão de tráfego de interesse”. As seções seguintes apresentam abordagens para reconhecer assinaturas de ataques baseados em cabeçalho e em conteúdo. A sistemática de funcionamento de tais ataques, assim como conceitos importantes a respeito deste assunto, são explanados no Apêndice A.

2.1.1 Detecção de ataques de rede baseados em cabeçalho

Como esta categoria de ataque possui padrões com características bem claras, um filtro[3] de pacotes poderia manter a rede livre de muitos tipos de tráfego externo indesejável, a partir da análise dos campos do cabeçalho de cada pacote de rede [13]. Esse princípio de filtragem de pacotes é a essência do funcionamento dos *firewalls*. No entanto, os SDI´s também devem estar preparados para esse tipo de ataque, para o caso do *hacker* já estar na rede interna.

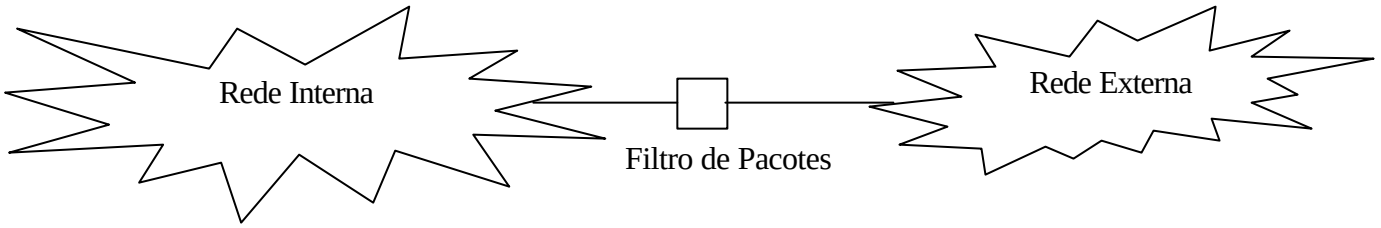

Figura 2.1 Filtro de pacotes.

No caso dos ataques de rede que exploram as vulnerabilidades da pilha TCP/IP de alguns sistemas operacionais utilizando apenas um pacote para causar um travamento no sistema, um filtro simples seria capaz de detectar e registrar tal ocorrência. Como por exemplo, no ataque **land** pode-se adotar o módulo genérico de detecção mostrado na Figura 2.2 em um SDI.

[3] Filtro de pacotes – termo comumente utilizado para o software que é configurado para permitir ou bloquear a passagem de pacotes para dentro de uma rede.

```
filtro land () {
# Caso o endereço IP origem seja igual ao endereço IP destino e a
#porta  TCP origem seja igual a porta TCP destino

    if  ((ip.src = = ip.dest) and  (tcp.src = =  tcp.dest))
     {
    #registrar quando o evento ocorreu e dados do possível  atacante
    #como o endereço MAC (Media Access Controller), o endereço IP
    #do computador origem  e também o endereço MAC e o endereço
    #IP do computador destino

    record system.time,
                eth.src, ip.src, eth.dst, ip.dst
        to land_recrdr;
    }
}
```

Figura 2.2 Filtro para o ataque *land*. Adaptado de [34].

Outro tipo de ataque de rede baseado no cabeçalho é aquele que utiliza muitos pacotes para causar algum tipo de indisponibilidade de serviço, através do consumo excessivo de recursos do sistema alvo. Neste caso, também, é possível obter um filtro baseado em procedimentos que identifique este tipo de ataque. [29] ilustra bem essa situação com um filtro para o ataque clássico **multipacotes synflood.**

```
count = 0; dest = 0; source = 0; ethsrc=0; maxcount = 90;  maxtine = 1; time = 0;

filter synflood ip () {
    if ( tcp.is) )   {
        #Se mais do que 90 pacotes SYN forem enviados a um único destino em 1 segundo,
        #uma inundação SYN é provável.
        if ( byte(ip.blob, 13) = = 2) {  #verifica se somente flag SYN ativo
            if (dest = =  ip.dest)
                count = count +1;
            else  {
                dest  =  ip.dest;
                count = 0;
                }
          }
     }
     ethsrc = eth.src;
     source = ip .source;
     time = system.time;
}
filter dishesdone timeout (sec:1, repeat) {
    if (count >>= maxcount) {
        echo ("Inundaçao SYN encontrada! Horario: ", time, "\n");
        #grava system.time, source, dest, ethsrc
        dest  = 0;
        count = 0;
    }
}
```

Figura 2.3 Filtro para o ataque DOS *synflood*. Adaptado de [29].

O objetivo deste filtro é acompanhar a freqüência de requisições de estabelecimento de conexões destinada a um determinado *host*. Isto pode ser feito através da verificação de quantos pacotes com esta característica (pacotes com o flag SYN ligado) estão endereçados para a máquina vítima em uma determinada unidade de tempo. Este excesso de solicitações é que pode ocasionar um estado de negação no *host* atacado.

2.1.2 Detecção de ataques de rede baseados no conteúdo

O cabeçalho TCP/IP dos pacotes possui uma estrutura bem definida que permite expressar os filtros responsáveis por reconhecer as assinaturas de ataques baseados em cabeçalhos. Também neste caso existe uma quase totalidade de ocorrência de verdadeiros positivos[4] na identificação dos abusos.

No caso das violações de segurança baseadas no conteúdo dos pacotes, a análise torna-se mais complexa, pois se lida com uma verdadeira tentativa de invasão, ou seja, o objetivo da incursão do atacante não é simplesmente suspender a operação de uma rede ou sistema e sim ganhar acesso para obter/adulterar informações e/ou utilizar a máquina invadida como base para a execução de outros ataques. Neste caso, não há necessidade de manipular o cabeçalho de pacotes ou disparar uma sobrecarga de tráfego para a vítima, pois a atividade agressora ocorre no nível da camada de aplicação. Sendo assim, a análise de uma cadeia variável de caracteres em busca de traços que possam denunciar a presença de uma tentativa de intrusão, e ainda possuir um baixo índice de falsos positivos[5], não é tarefa fácil.

A orientação utilizada em [24] para detecção de ataques de rede baseados no conteúdo é uma alternativa para identificar esse tipo de incidente (Figura 2.4).

---

[4] Verdadeiros positivos representam indicativos de ataques emitidos que realmente correspondem ao surgimento de alguma atividade maliciosa.

[5] Falsos positivos são alarmes falsos que confundem o administrador do sistema indicando que existe atividade suspeita acontecendo, quando na verdade tratam-se de ocorrências normais.

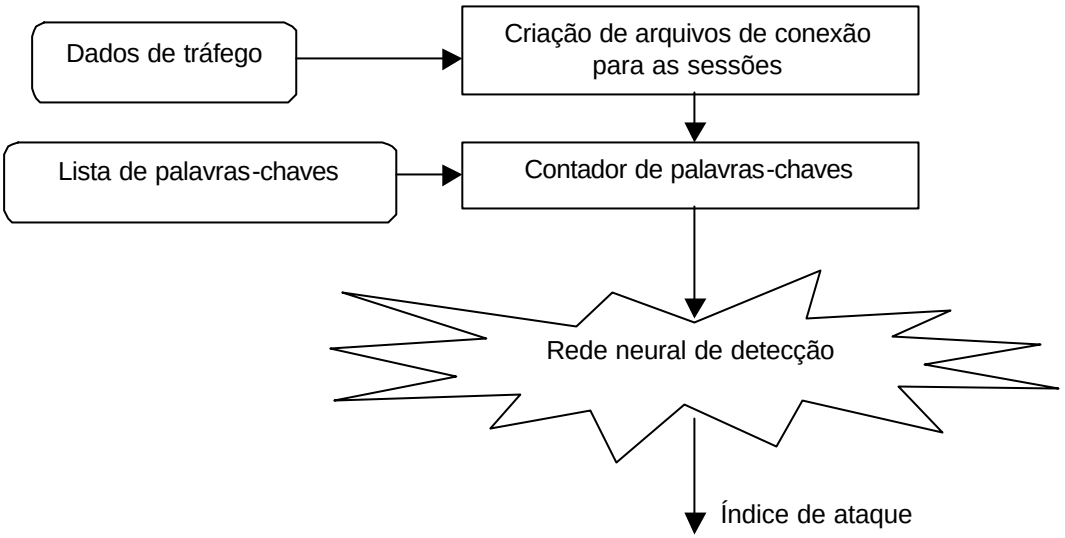

Figura 2.4 Diagrama de bloco de um mecanismo de identificação de intrusões.

Nesta abordagem, os conteúdos dos pacotes coletados no tráfego da rede são agrupados em suas respectivas conexões TCP. Posteriormente as conexões são processadas para produzir contadores do número de ocorrências de palavras-chaves, presentes em uma lista previamente estabelecida. Então, os contadores destas palavras são analisados por uma rede neural na determinação de um índice de ataque.

Um aspecto decisivo para o bom desempenho deste mecanismo é a escolha destas palavras especiais. Faz-se necessário um grupo de palavras que envolvam não só a execução do ataque, como também a sua preparação e atividades desenvolvidas pelo invasor na fase pós-ataque. Desta forma, tenta-se englobar todo o cenário do processo de invasão e analisar o problema dentro de um contexto mais abrangente.

Outro fator importante a ser considerado é a natureza de uma sessão TCP. Como a mesma é constituída na realidade de duas conexões (cliente-servidor

e servidor-cliente), conforme se observa na Figura 2.5, prioriza-se as respostas dadas pelos softwares dos servidores em detrimento aos estímulos emitidos pelos clientes, com o objetivo de obter um conjunto de termos mais genéricos, capaz de ser utilizado também para verificar ataques novos. Isto se baseia no fato de que diferentes estímulos podem provocar a mesma resposta do servidor.

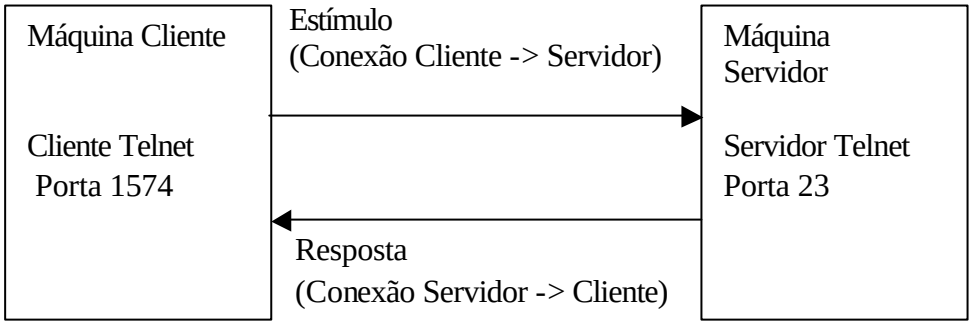

Figura 2.5 Estímulo-Resposta em uma conexão TCP.

Quanto ao treinamento da rede neural deve-se considerar que uma mesma *keyword* pode aparecer tanto em uma atividade normal quanto em uma suspeita. Logo, a fase de treinamento deve englobar não só conexões de ataque, como também conexões dentro da normalidade para que a rede neural possa avaliar o peso que determinada palavra tem em cada um dos tipos de conexão (Figura 2.6).

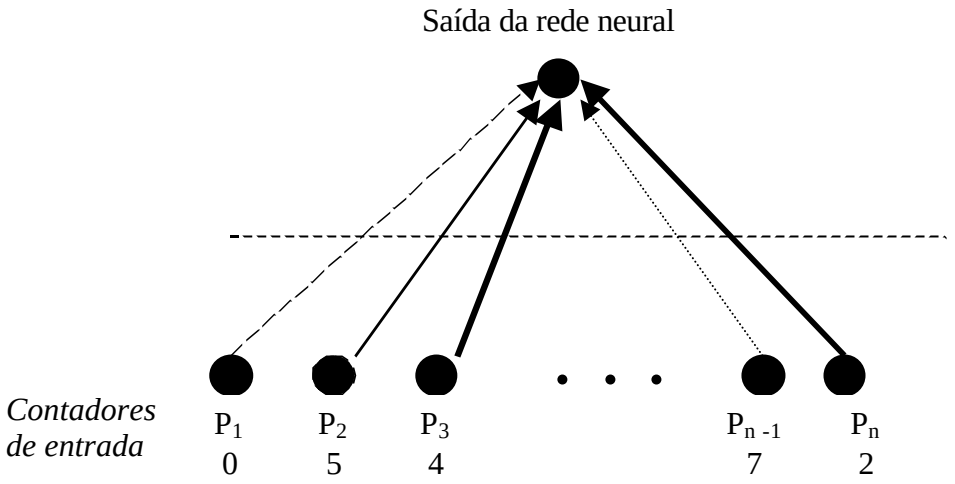

Figura 2.6 Influência de cada contador de ocorrência de palavras-chave (Pi) na indicação de presença de um ataque.

2.2 SAARA - Um modelo de atualização automática para o mecanismo de detecção de ataques

Este trabalho propõe um modelo de atualização automática do mecanismo de detecção de assinaturas para SDI´s baseados na tecnologia de sociedades de agentes inteligentes.

Conforme mostra a Figura 2.7, uma Agência Central de Segurança[6] será responsável por coletar informações contidas em diversas bases de dados na Internet, periodicamente atualizadas, que disponibilizarão em algum formato padronizado informações sobre ataques de rede. Esta captura de dados poderá ser feita através de agentes móveis responsáveis por identificar as alterações (por exemplo, o surgimento de um novo ataque e/ou mutações de um ataque já conhecido) que ocorressem nestas bases e repassá-las para a Agência Central.

[6] ACS (Agência Central de Segurança) ou SCIA (Security Central Inteligence Agency).

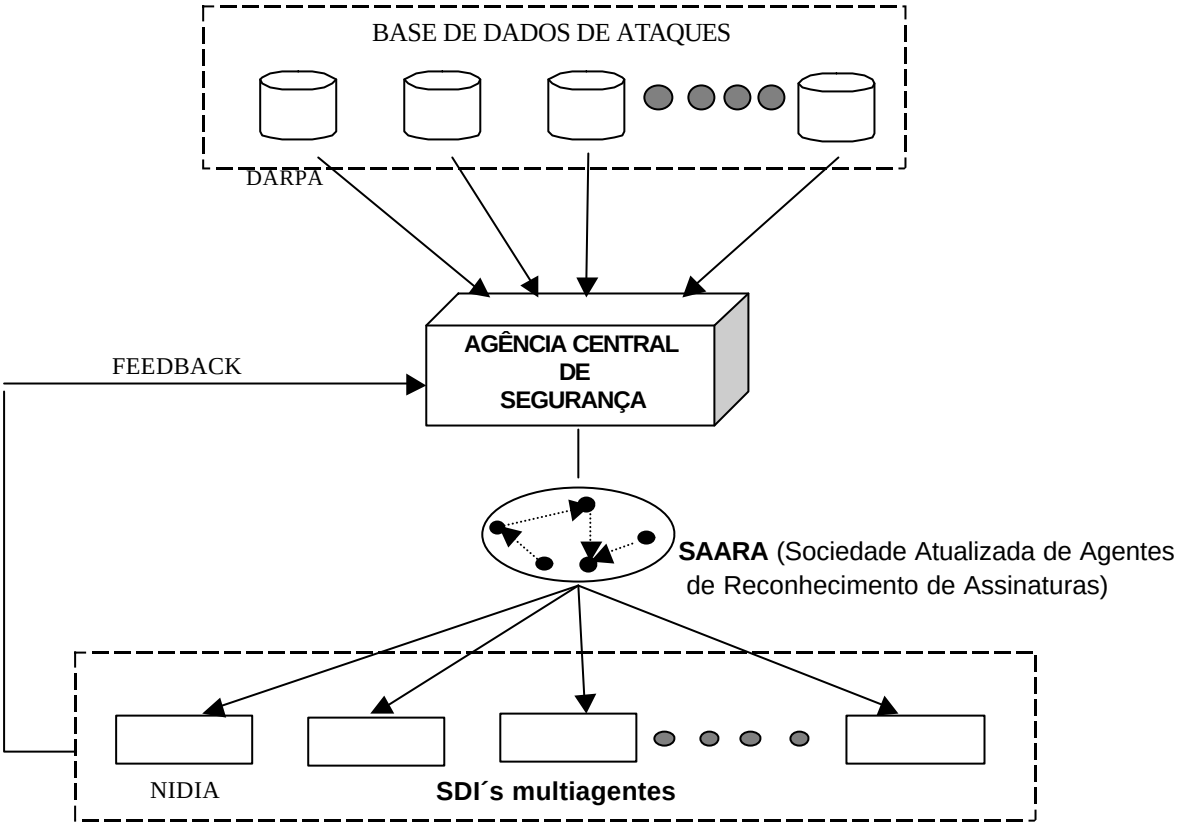

Figura 2.7 Modelo de atualização automática do mecanismo de identificação de assinaturas.

O consórcio de instituições responsáveis por manter estas diferentes bases de dados deverá reportar o surgimento de assinaturas por meio do estabelecimento de interfaces, para que os agentes móveis possam efetuar a coleta das informações necessárias.

Com base nas informações recebidas, a ACS produzirá uma mini-sociedade de agentes responsável especificamente em desempenhar o papel do mecanismo de reconhecimento de ataques de rede para SDI´s multiagentes. Os SDI´s devem ser compatíveis com as interfaces de comunicação das sociedades de identificação de assinaturas criadas. Sempre que uma nova SAARA (Sociedade Atualizada de Agentes de Reconhecimento de Assinaturas) for produzida, a ACS

informará aos SDI´s parceiros sobre a sua disponibilidade. Além disso, informações reportadas dos SDI's usuários sobre o desempenho da SAARA (*feedback*) servem de subsídio para que a ACS refine o mecanismo de detecção da mini-sociedade de agentes. Observa-se na Figura 2.7 que aparecem, como potenciais entidades participantes deste modelo, a base de dados do DARPA[7] e o sistema de detecção NIDIA. Maiores detalhes sobre os mesmos são mostrados nas Seções 2.3 e 3.1, respectivamente.

O cerne de uma SAARA são agentes especializados na identificação de ataques de rede baseados em cabeçalho, possivelmente através de filtros, e agentes especializados na identificação de assinaturas no conteúdo das conexões TCP através do uso da tecnologia de redes neurais.

No caso de ataques de rede por conteúdo, por possuírem natureza não-determinística, forma não-estruturada, tamanho variável de dados e informações que podem estar distorcidas e/ou incompletas, é difícil estabelecer de forma clara quais seriam as particularidades que pudessem denunciar que se trata de um padrão suspeito, de forma a estabelecer as regras de identificação. Neste caso, a tecnologia das redes neurais apresenta características que poderiam suprir as deficiências de outros métodos de detecção, destacando-se a propriedade de alta flexibilidade/generalização que possibilitaria que uma rede neural alertasse quão próximo um ataque (não apresentado à rede anteriormente) está de um padrão suspeito para o qual ela tenha sido devidamente treinada. Assim esta ferramenta não trabalha restritamente gerando saída sim ou não; ela possui a capacidade de

[7] *Defense Advanced Research Projects Agency.*

inferir a natureza de dados novos a partir do conhecimento que adquiriu anteriormente, através de uma fase de treinamento.

Segundo [7], “os vários comandos que são inclusos nos dados constituem o mais crítico elemento no processo de determinar se um ataque está ocorrendo contra uma rede”. Logo, para que uma rede neural possa identificar as atividades de intrusão de forma eficaz e eficiente, é necessário que haja um treinamento adequado com dados e métodos apropriados. Com o objetivo de obter resultados mais precisos e para que a rede neural não apresente um comportamento “viciado”, é condição necessária haver uma grande e diversificada massa de dados, com inúmeras seqüências de ataques imersas em atividades normais. Uma base de treinamento com estas características não é simples de ser obtida e consiste no grande entrave na utilização desta ferramenta para contemplar atributos desejáveis no mecanismo de identificação de ataques de um SDI.

Abaixo segue a descrição da metodologia proposta para o treinamento e atualização automática para uma rede neural destinada a auxiliar a inspeção do conteúdo de conexões TCP em busca de atividades maliciosas. Tal abordagem trata-se de uma extensão da sistemática apresentada por [24].

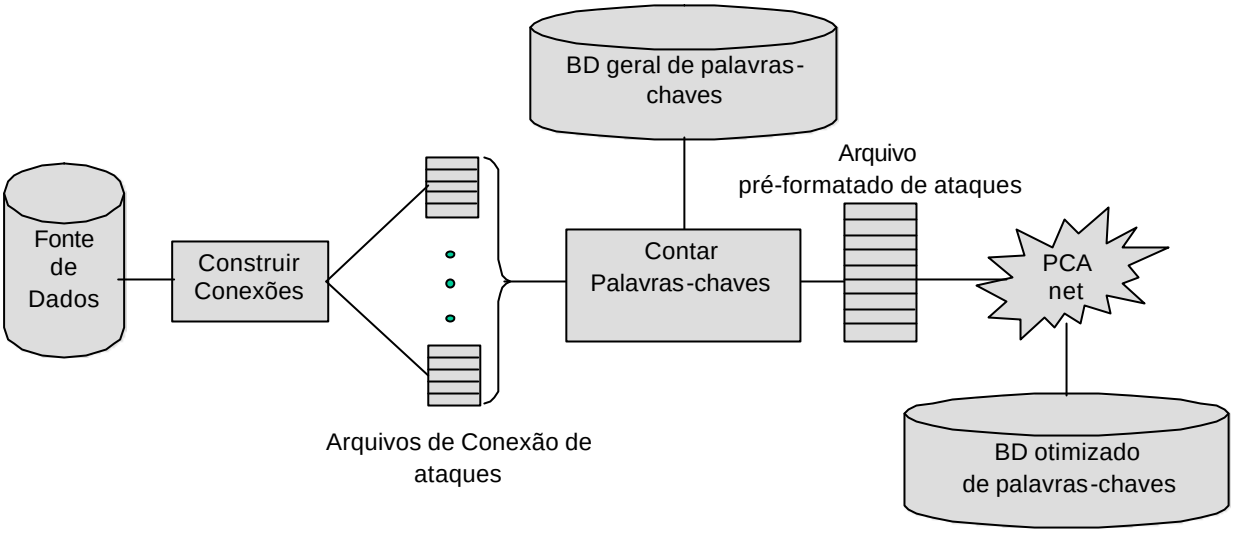

Figura 2.8 Processo de otimização do banco de palavras-chaves.

Na Figura 2.8 presume-se a existência de uma fonte de dados de tráfego de rede que contenha diversas modalidades de ataques, distribuídos e organizados como uma rede de computadores real. Esta fonte de dados, regularmente alimentada pelas atividades dos agentes móveis, serve de entrada para um processo que reconstrói apenas conexões referentes a atividades maliciosas que atuam na camada de aplicação (ataques orientados a dados). Para que isto seja possível, a fonte de tráfego deve fornecer de alguma maneira todos os detalhes necessários, tais como: IP origem, IP destino, porta origem, porta destino, presença e nome do ataque e tempo de ocorrência. Desta forma, a conexão problemática, que declaradamente consista em um incidente de segurança, pode ser identificada.

Este processo de reconstrução de conexões gera como saída um arquivo para cada conexão de ataque observada, sendo que apenas as conexões TCP que representem respostas dos servidores aos requisitos dos clientes são consideradas,

com o intuito de buscar por situações que sejam mais genéricas em torno de categorias de ataques e não de ataques em particular.

Um dos pontos críticos para utilizar uma rede neural que processe contadores de ocorrências de determinados termos especiais, contidos em uma cadeia de caracteres não organizada e variável, é justamente enumerar qual seria este conjunto de palavras-chaves. A escolha de determinada palavra-chave deve ser baseada no contexto geral de uma ameaça, ou seja, as fases de preparação, execução e aproveitamento do resultado do ataque (fase pós-ataque)[8]. Utilizando-se destes critérios, pode-se por empirismo, pela análise de outros trabalhos científicos [8] e pela observância de SDIs já existentes, como o SNORT [39], obter um banco de dados deste tipo de palavras. A Figura 2.9 mostra um exemplo de regra do SNORT que emite uma mensagem de alerta quando qualquer ip/porta origem envia um pacote cujo campo de dados contenha a string “cgi-bin/phf” para a subrede 192.168.1.0/24 na porta 80.

```
alert tcp any any -> 192.168.1.0/24 80 (content: "cgi-bin/phf";  msg: "CGI-PHF access";)
```

Figura 2.9 Exemplo de regra do SNORT.

Tem-se, então, um processo que lê as conexões de ataque e conta as ocorrências das palavras-chaves presentes num banco de dados formado como descrito anteriormente. Este processo gera como saída um arquivo de contadores onde cada linha do mesmo corresponde ao resultado da contagem referente a uma conexão intrusiva (Figura 2.11).

[8] Maiores detalhes da fase pós-ataque são apresentados no Apêndice A.

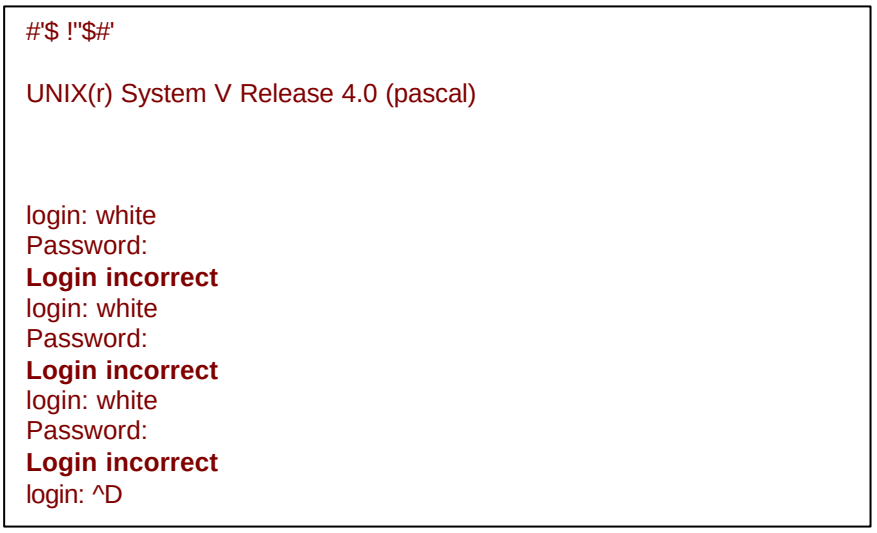

Figura 2.10 Exemplo de conexão de ataque para o ataque *guess* (adivinhação de senha). Destaque para a potencial palavra-chave "*Login incorrect*".

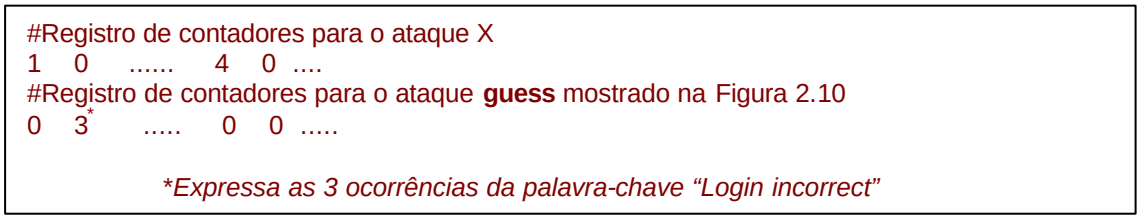

Figura 2.11 Formato do arquivo de contadores.

Decorre, entretanto, que tal arquivo de contadores foi criado a partir de palavras específicas para determinados ataques, mas que provavelmente não é adequado para generalizar no caso de ataques novos. Assim sendo, é possível que um ataque inesperado, que mesmo sendo uma variação de um ataque conhecido, acabe não sendo adequadamente identificado como um problema na fase de produção do SDI, devido à excessiva especificidade dada às palavras-chaves em relação a ataques individualizados.

Então, faz-se necessário, além dos critérios para escolhas das conexões e das fases da intrusão onde devem ser identificadas potenciais palavras-chaves, um mecanismo para obter, a partir do conjunto inicial de palavras, um subconjunto genérico que identifique quais são os termos mais úteis para revelar que tal conexão possua atividade hostil e consiste em uma infração. Uma rede neural PCA (*Principal Component Analysis*) [18] teria a capacidade de desempenhar este papel, devido à sua característica inerente de extrair quais são os principais elementos que representam as peculiaridades em um determinado domínio. Sendo assim, poder-se-ia obter uma redução de dimensionalidade no conjunto de palavras originais, resultando em um subconjunto de palavras mais abrangentes que não seriam vinculadas apenas a determinados ataques em particular.

Após o processo de obtenção do banco de dados otimizado de palavras-chaves, passa-se para a fase de treinamento da rede neural MLP [18] que será responsável pela emissão dos alertas indicando o grau de suspeita de determinada conexão (Figura 2.12). Neste esquema, o processo de reconstrução das conexões servidor-cliente cria tanto conexões normais, quanto conexões de ataque. Posteriormente é feita a contagem de ocorrências das palavras contidas no banco previamente otimizado. O processo que realiza a contagem gera um arquivo pré-formatado de treinamento supervisionado para a rede MLP, indicando quando há presença de ataque.

É importante ressaltar que o treinamento da rede neural é feito tanto com conexões suspeitas quanto conexões normais. Isto decorre do fato de que as palavras-chaves assumidas como peculiares a problemas com segurança, podem também estar presentes em conexões normais. Então cabe à fase de treinamento

prover à rede o discernimento para saber qual a influência que determinada palavra tem em conexões intrusivas ou não.

Verifica-se, portanto, que a utilização da rede neural com a metodologia de treinamento descrita faz-se necessária devido ao fato de que a presença de palavras-chaves seja condição necessária, mas não suficiente para a emissão de um índice de severidade que denuncie algum problema de segurança relacionado a ataques orientados a dados (Apêndice A). A essência da análise é o contexto em que o aparecimento destas palavras acontece e isto pode ser interpretado por uma rede neural adequadamente treinada, evitando-se assim o aparecimento de altas taxas de falsos positivos.

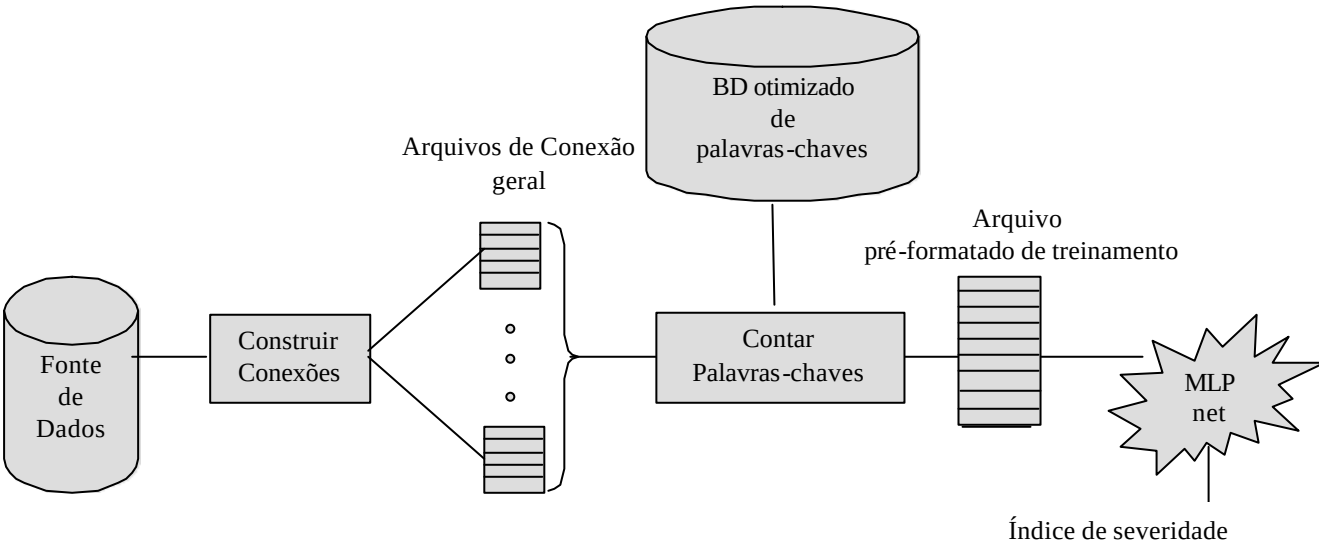

Figura 2.12 Esquema de Treinamento da rede MLP.

Por fim, a capacidade de generalização da rede neural permite que a mesma emita um índice de severidade elevado a respeito de ataques semelhantes àqueles para os quais a mesma foi treinada. Mas a natureza dos ataques também

evolui com o tempo para tornar-se mais furtiva, isto é, os infratores poderão utilizar técnicas para dificultar a detecção de suas ações [25]. Entretanto esta seria uma preocupação da equipe de pesquisadores da Agência Central de Segurança, que seria responsável pela alimentação do banco geral de palavras-chaves, pelo acompanhamento do refinamento deste banco e pela geração de redes neurais[9] que comporiam a SAARA juntamente com os filtros. Cabe também a ACS verificar se as atualizações da sua fonte de dados estão refletindo o surgimento de novas técnicas presentes nas bases de dados mantidas por outras organizações e que foram recolhidas pelos agentes móveis.

2.3 Implementação parcial do modelo apresentado

Para demonstrar a viabilidade da metodologia de detecção de intrusos com base na análise das conexões TCP, o presente trabalho apresenta uma implementação parcial da sistemática de treinamento descrita anteriormente.

O ponto de partida, para um projeto desta natureza, é encontrar uma fonte de dados que de alguma forma contemple o esquema proposto. Visando a avaliação e construção de novos SDI's, o Projeto DARPA [26] criou uma rede de computadores simulando tráfego real recolhido de redes de órgãos públicos americanos, adicionando ao mesmo determinadas categorias de ataques. Tais arquivos são disponibilizados em formatos bem conhecidos, como o *Tcpdump* [22],

[9] Devido à provável dificuldade para que apenas uma rede neural consiga convergir tanto para ataques novos quanto velhos, poder-se-ia trabalhar com redes especializadas em diferentes domínios de intrusão.

de forma a permitir avaliações e desenvolvimento de novos métodos de detecção de invasão.

Devido à facilidade de obtenção e relativa consistência das informações, este protótipo utilizará, como massa de dados para treinamento da rede neural, arquivos do Projeto DARPA. Conforme mencionado anteriormente, as bases de dados de tráfego que servem de subsídio para a construção da fonte de dados própria da ACS deve possuir meios de poder mapear as conexões hostis dentro da gama de informações disponibilizadas. No caso específico do DARPA, cada arquivo no formato *tcpdump* possui um arquivo de índice correspondente que identifica exatamente onde podem ser encontrados os ataques que estão inseridos entre as conexões consideradas normais (Figura 2.13). Cada linha deste arquivo possui informações na ordem descrita a seguir:

1. Identificador único da conexão;
2. A data de início da sessão no formato mm/dd/yyyy;
3. O horário de início da sessão no formato hh:mm:ss;
4. Duração da sessão;
5. O nome do serviço usado na sessão (fornecido por conveniência);
6. Porta origem da sessão;
7. Porta destino da sessão;
8. Endereço IP origem da sessão;
9. Endereço IP destino da sessão;
10. Índice com valor 1 quando houver presença de ataque e valor 0 em caso contrário;

11. Nome do ataque quando o índice anterior for igual a 1. Para sessões normais apresenta “-“.

```
    Start      Start                      Src   Dest Src          Dest          Attack
    Date       Time     Duration  Serv   Port  Port IP           IP           Score Name
 1  01/27/1998 00:00:01 00:00:23  ftp    1755  21   192.168.1.30 192.168.0.20 0     -
 2  01/27/1998 05:04:43 67:59:01  telnet 1042  23   192.168.1.30 192.168.0.20 0     -
 3  01/27/1998 06:04:36 00:00:59  smtp   43590 25   192.168.1.30 192.168.0.40 0     -
 4  01/27/1998 08:45:01 00:00:01  finger 1050  79   192.168.0.40 192.168.1.30 1     guess
 5  01/27/1998 09:23:45 00:01:34  http   1031  80   192.168.1.30 192.168.0.40 0     -
 7  01/27/1998 15:11:32 00:00:12  sunrpc 2025  111  192.168.1.30 192.168.0.20 1     rpc
 8  01/27/1998 21:53:17 00:00:45  exec   2032  512  192.168.1.30 192.168.0.40 1     exec
 9  01/27/1998 21:58:21 00:00:04  http   1031  80   192.168.1.30 192.168.0.20 0     -
10  01/27/1998 22:57:53 26:59:00  login  2031  513  192.168.0.40 192.168.1.20 0     -
11  01/27/1998 23:57:28 130:23:08 shell  1022  514  192.168.1.30 192.168.0.20 1     guess
```

Figura 2.13 Arquivo *tcpdump.list* que identifica as conexões de ataque dentro do arquivo *tcpdump*[10].

De posse das informações fornecidas por este arquivo, geralmente nomeado de *tcpdump.list*, torna-se fácil identificar quais são as conexões com problemas de segurança que podem ser relacionadas tanto às ameaças baseadas nos cabeçalhos dos pacotes TCP/IP, quanto às ameaças baseadas nos conteúdos das conexões. O módulo responsável por gerar os arquivos texto que são o espelho das conexões formadas por diversos pacotes foi chamado de “**GeraArqConexões**”.

Para limitar o escopo da implementação, presumiu-se que o processo de otimização do conjunto de palavras-chaves já foi realizado (Figura 2.8). O subconjunto de palavras que realmente foi utilizado durante a implementação é derivado da orientação de outros trabalhos científicos e da observação de conexões com problemas de segurança, facilmente identificadas por um arquivo do tipo “*tcpdump.list*”. Em face destas considerações, segue a explicação dos detalhes de implementação dos módulos envolvidos.

[10] Detalhes sobre os ataques da Figura 2.13 podem ser encontrados em [31].

2.3.1 GeraArqConexões

Este módulo feito em Java[11] [44] é inspirado no conhecido *sniffer Tcpdump*. O suporte à leitura dos pacotes na rede é feito através da biblioteca jpcap [17] que fornece uma interface mais amigável para manipular outra biblioteca de baixo nível, a libpcap [22], que é vinculada à plataforma onde será realizada a captura de pacotes da rede[12]. A jpcap permite ler os pacotes da interface de rede ou através de um arquivo no formato *tcpdump* correspondente a um tráfego previamente coletado.

O programa **GeraArqConexões** recebe como parâmetros o arquivo *tcpdump*, a indicação do tipo de conexão a ser gerada (ataque ou normal), o caminho do arquivo *tcpdump* e opcionalmente o número de conexões que se deseja gerar, conforme mostra a Figura 2.14:

```
C:\ids> java GeraArqConexoes

C:\ids>Use: java GeraArqConexoes [dumpfile] [A-ataque N-normal] [path]
        Ou use: java GeraArqConexoes [dumpfile] [A-ataque N-normal] [path] [Numero de conexoes]
```

Figura 2.14 Parâmetros do programa GeraArqConexões.

Um dos objetivos de se obter, em um determinado momento, apenas um conjunto de ataques é fazer a verificação dos arquivos gerados e porventura estabelecer quais termos que vinculam estas conexões a atividades maliciosas. Para executar esta função, o programa lê o arquivo de índice, verifica quais são as

[11] A linguagem Java foi adotada para o desenvolvimento deste e dos outros módulos funcionais, devido principalmente às suas características de portabilidade entre diferentes plataformas e suporte à programação concorrente.
[12] Na plataforma Windows a versão libpcap é instalada com o winpcap.

conexões de ataque baseadas em conteúdo e cria automaticamente um filtro que lê apenas conexões que satisfaçam as regras de filtragem estabelecidas (Figura 2.15).

A estrutura de cada operando da operação lógica "**ou**" determina os endereços IPs dos *hosts* origem e destino e os números TCP das portas origem e destino, com base no arquivo "*tcpdump.list*" fornecido junto com o arquivo no formato *tcpdump*. Desta forma, apenas os pacotes que satisfizerem as condições estabelecidas pelo filtro passado para o método *setFilter* da jpcap serão capturados.

```
Filtro :(ip src host 192.168.0.40 and ip dst host 192.168.1.30 and tcp src port 80 and tcp dst port 1784) or
(ip src host 192.168.1.30 and ip dst host 192.168.0.40 and tcp src port 1784 and tcp dst port 80) or (ip src
host 192.168.0.20 and ip dst host 192.168.1.30 and tcp src port 23 and tcp dst port 1867) or (ip src host
192.168.1.30 and ip dst host 192.168.0.20 and tcp src port 1867 and tcp dst port 23) or (ip src host
192.168.0.20 and ip dst host 192.168.1.30 and tcp src port 23 and tcp dst port 1884)
```

Figura 2.15 Parte de um filtro gerado automaticamente.

Após determinar o filtro, o programa faz chamada ao método *processPacket*[13] da jpcap para a leitura de pacotes IP, como se pode observar na Figura 2.16.

```
jpcap.processPacket(-1,new GeraArqConexoes())
```

Figura 2.16 Chamada ao método processPacket.

Para remontar diversos pacotes que compõem uma conexão específica, deve-se conseguir reconhecer de forma única aquela comunicação. Naturalmente a escolha para essa espécie de identificador unívoco, em um determinado instante no tempo, seria derivado da composição:

[13] O método processPacket captura pacotes e possibilita a posterior manipulação dos mesmos. O parâmetro –1 indica para coletar pacotes indefinidamente.

IP origem + IP destino + Porta Origem + Porta Destino.

Além disso, a fim de atender ao requisito da metodologia de trabalhar somente com as conexões no fluxo servidor-cliente, restringiu-se o aproveitamento de pacotes apenas para conexões oriundas da segunda etapa do estabelecimento de comunicação TCP (*handshake*). Em outras palavras, as entradas na *hashtable*[14] [16] utilizada são criadas somente quando este programa lê um pacote que possua os *flags* SYN e ACK ligados (Apêndice A). A partir de então, os pacotes subseqüentes de determinada comunicação são redirecionados para o seu arquivo correspondente, onde são escritos os caracteres visíveis e também os caracteres de retorno-de-carro (*carriage return*) e de alimentação de linha (*line feed*).

A Figura 2.17 apresenta o processo de remontagem de conexões com a *hashtable* (tabela *hash*), onde cada "*slot*" corresponde a um objeto que representa a conexão a ser recriada em um arquivo texto. Para simplificar o entendimento, foi omitido o processo de mapeamento inerente à *hashtable*.

[14] A *hashtable* é uma estrutura de dados mais flexível que um vetor comum, pois permite que objetos possam ser utilizados como índice de busca e como dados de armazenamento. O índice de um vetor é um inteiro primitivo.

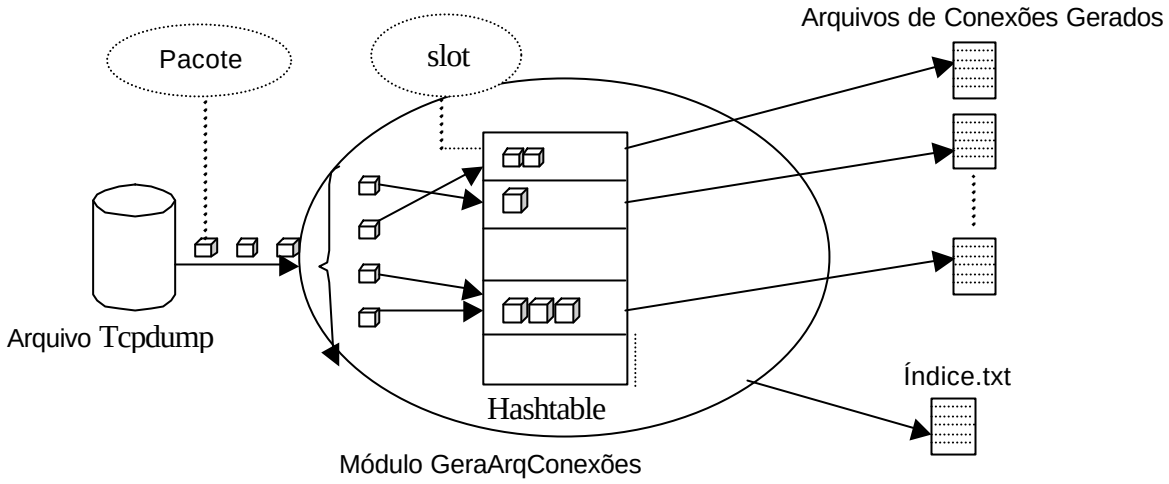

Figura 2.17 Remontagem de conexões.

Adicionalmente, o módulo **GeraArqConexões** cria um arquivo auxiliar "índice.txt" que contém todos os arquivos gerados, de forma a facilitar a leitura dos mesmos pelo módulo funcional subseqüente. Neste arquivo tem-se o nome do arquivo de conexão gerado, um indicador de presença de problema de segurança e por fim o nome do ataque, se for o caso (Figura 2.18).

```
192.168.0.40_192.168.1.30_80_1784_1.txt 1 phf
192.168.0.20_192.168.1.30_23_1867_1.txt 1 guess
192.168.0.20_192.168.1.30_23_1884_1.txt 1 guess
192.168.0.20_192.168.1.30_514_1023_1.txt 1 rcp
192.168.0.20_192.168.1.30_23_1906_1.txt 1 guess
192.168.0.20_192.168.1.30_513_1022_1.txt 1 rlogin
192.168.0.20_192.168.1.30_514_1022_1.txt 1 rsh
192.168.0.20_192.168.1.30_23_1914_1.txt 1 guess
```

Figura 2.18 Arquivo de índice auxiliar gerado por GeraArqConexões.

2.3.2 GeraArqContadores

Após a criação dos arquivos de conexão, passa-se para a fase de formatação da massa de treinamento para a rede MLP. Isto é feito através da

contagem de ocorrências das palavras-chaves, que neste trabalho foram direcionadas para um sistema “Unix like”. Numa primeira análise, pode-se imaginar que se trata apenas de buscar “substrings” inseridas em um texto, controlando a quantidade de vezes que cada uma delas é encontrada.

Decorre que devido a peculiaridades de plataformas e aplicações, um mesmo termo pode sofrer pequenas variações, como, por exemplo, alternância entre letras maiúsculas e minúsculas, quantidade variável de espaços dentro de uma mesma palavra-chave, etc. Além disso, determinados termos apenas carregam consigo características de periculosidade se presentes em determinada posição da linha de comando. Sendo assim, os termos especiais devem ser representados de uma maneira bem flexível para atender de forma adequada o objetivo para o qual foram propostos. A utilização de expressões regulares[15] para este fim contempla a necessidade mencionada e foi adotada nesta implementação.

Basicamente os ataques orientados a dados são caracterizados por chamadas a aplicativos que notoriamente têm problemas com a sua validação de entrada ou pela utilização de *exploits*[16] baseados nas técnicas de estouro de buffer (Apêndice A). Comumente este último caso envolve a transferência de fontes de programas de ataques para a máquina alvo seguida de uma etapa de compilação no próprio ambiente da vítima. Com isso, obtém-se executáveis binários compatíveis com o sistema operacional e com o processador da máquina que se deseja invadir.

[15] A versão da linguagem Java utilizada neste trabalho foi a 1.4 que já possui um pacote para tratamento de expressões regulares (java.util.regex).
[16] *Exploits* são ataques projetados para tirar vantagem de algum ponto fraco em um software de aplicação ou de sistema que resulte em comprometimento do sistema [33].

Um princípio básico e intuitivo para se levantar suspeitas sobre determinada conexão é constatar a presença de situações não corriqueiras para um usuário comum. Segue abaixo um descritivo de algumas das palavras utilizadas neste trabalho, representadas pelas suas respectivas expressões regulares [35]:

- **^\#\s**

  Quando logado em um sistema Linux, um usuário comum recebe um *shell* interativo [14] onde ele pode, de acordo com os seus direitos, executar as ações que necessita. O *prompt* desta linha de comando geralmente é o símbolo **$**. O fato deste usuário conseguir realizar uma escalação de privilégios e obter um *shell* interativo com direitos de superusuário, sem dúvida, é um evento no mínimo suspeito. Para identificar a criação de uma linha de comando de administrador, pode-se buscar pelo sinal de *prompt* #. A expressão regular ainda possui os caracteres especiais **^** e **\s** para reger que o símbolo **#** procurado deve aparecer no início da linha e ser seguido por um caracter de espaço em branco.

- **gcc\s**

  O **gcc** é um popular compilador da linguagem C e que é costumeiramente utilizado para compilar scripts de ataques.

- **(?i)login incorrect**

  Ao tentar efetuar uma conexão remota com um servidor, o mesmo provavelmente vai solicitar alguma forma de autenticação do tipo usuário/senha. No caso do atacante ainda não possuir uma autenticação

válida neste sistema, ele pode se aventurar em alguma técnica de adivinhação de senhas. A palavra-chave **login incorrect** é uma resposta enviada pelo servidor quando há algum erro no par usuário/senha informado. Constitui-se em um indicativo de ameaça a presença de diversas ocorrências deste termo em uma conexão, revelando que pode não se tratar apenas de um usuário que digitou erroneamente a sua senha, mas sim do início de um processo de invasão. A expressão **(?i)** significa que a busca por este termo é *case insensitive*, ou seja, não deve haver distinção entre caracteres maiúsculos e minúsculos. É interessante utilizar este artifício por causa de pequenas variações que podem existir dependendo do sistema operacional que estiver sendo utilizado.

- **/cgi-bin/phf**

Aplicativos que notoriamente têm problemas com vulnerabilidades relativas às suas validações de entradas de dados ou relativas a scripts de estouro de *buffer*, também podem figurar entre as palavras a serem buscadas nas conexões.

- **cat\s*>**

O comando **cat** quando utilizado no formato **cat > arquivoqualquer** cria um arquivo e grava no mesmo todos os caracteres digitados após a emissão do comando, até que a combinação de teclas <Ctrl + d> seja pressionada. A classe predefinida de caracteres **\s** seguida do símbolo ***** significa que entre o comando **cat** e o sinal de **>** pode haver um ou mais espaços em branco. Especificamente no campo das invasões, esta

característica do **cat** pode ser utilizada para criar os arquivos fontes em linguagem C, por exemplo, que serão posteriormente compilados e executados para as ações de ataque.

- **passwd**

Toda pretensão de manipular o arquivo **/etc/passwd** deve ser vista com cautela. Este arquivo contém as contas dos usuários do sistema e pode possuir também as respectivas senhas. Considera-se um esquema de segurança melhor a separação das senhas em um outro arquivo denominado de **/etc/shadow**, que pode também originar uma palavra-chave em potencial. A ameaça latente que existe na manipulação não autorizada destes arquivos é derivada da possibilidade de descoberta da senha de usuários legítimos e até mesmo da criação de uma conta especificamente para fins ilegais.

- **Last login**

Depois que um usuário consegue a sua autenticação, o computador servidor geralmente emite uma resposta do tipo **Last login** indicando quando foi a última vez que este usuário acessou o sistema. Esta palavra-chave é bastante comum em conexões normais e sua presença após alguns termos **Login incorrect** pode servir para indicar que houve apenas descuido do usuário ao digitar a senha e não uma situação de risco para o sistema. Desta forma, a análise deste contexto diminui a taxa de ocorrência de falsos positivos.

Tem-se a seguir alguns arquivos de conexão extraídos do DARPA pelo módulo **GeraArqConexões**:

```
#'$ !"$#'

UNIX(r) System V Release 4.0 (pascal)

login: tristank
Password:
Last login: Mon Jun  1 08:05:31 from beta.banana.edu
Sun Microsystems Inc.   SunOS 5.5       Generic November 1995

/* Trecho retirado para facilitar entendimento */

tcsh: using dumb terminal settings.
pascal> which gcc

/bin/gcc
pascal> cat > formatexploit.c
   /*
   Solaris 2.5.1 - this exploited was compiled on Solaris2.4 and tested on
   2.5.1
   */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#define BUF_LENGTH 364
#define EXTRA 400
#define STACK_OFFSET 704
#define SPARC_NOP 0xa61cc013

   u_char sparc_shellcode[] =
   "\x2d\x0b\xd8\x9a\xac\x15\xa1\x6e\x2f\x0b\xda\xdc\xae\x15\xe3\x68"
   "\x90\x0b\x80\x0e\x92\x03\xa0\x0c\x94\x1a\x80\x0a\x9c\x03\xa0\x14"
   "\xec\x3b\xbf\xec\xc0\x23\xbf\xf4\xdc\x23\xbf\xf8\xc0\x23\xbf\xfc"
   "\x82\x10\x20\x3b\x91\xd0\x20\x08\x90\x1b\xc0\x0f\x82\x10\x20\x01"
   "\x91\xd0\x20\x08";

   u_long get_sp(void)
   (
   __asm__("mov %sp,%i0");
    ");
```

```
void main(int argc, char *argv[])

    {
    char buf[BUF_LENGTH + EXTRA + 8];
    long targ_addr;
    u_long *long_p;
    u_char *char_p;
    int i, code_length = strlen(sparc_shellcode),dso=0;


 if(argc > 1) dso=atoi(argv[1]);

    long_p =(u_long *) buf ;
    targ_addr = get_sp() - STACK_OFFSET - dso;
    for (i = 0; i < (BUF_LENGTH - code_length)/sizeof(u_long); i++)
    *long_p++ = SPARC_NOP;

    char_p = (u_char *) long_p;

    for (i = 0; i < code_length; i++)
    *char_p++ = sparc_shellcode[i];

    long_p = (u_long *) char_p;

    for (i = 0; i < EXTRA / sizeof(u_long); i++)
    *long_p++ =targ_addr;

    printf("Jumping to address 0x%lx B[%d] E[%d] SO[%d]\n",
    targ_addr,BUF_LENGTH,EXTRA,STACK_OFFSET);
    execl("/bin/fdformat", "fdformat", & buf[1],(char *) 0);
    perror("execl failed");

^Dpascal> /bin/gcc -o formatexploit formatexploit.c

pascal> ./formatexploit

Jumping to address 0xeffff698 B[364] E[400] SO[704]
#
^D#
pascal>
pascal> ^Dlogout
```

Figura 2.19 Estouro de *buffer* usando o comando *fdformat* do Unix para obter um *shell* privilegiado.

Na Figura 2.19 tem-se um ataque cuja sistemática consiste em criar um programa na vítima, compilá-lo e posteriormente executá-lo para obter acesso privilegiado através de um estouro de *buffer* (Apêndice A). As potenciais palavras-chaves para identificar o contexto desta invasão estão em negrito. Como é comum o fato do invasor não ter sido o criador do *script* de ataque, então é interessante

utilizar também como palavras-chaves mensagens de *debug* que usualmente são deixadas pelos programadores na fase de teste do programa. É possível incorporá-las ao mecanismo de detecção pela análise de *exploits* maliciosos que estão disponíveis em diversos *sites* da Internet. Neste caso específico, seria a mensagem de depuração deixada **Jumping to address** que possivelmente foi utilizada para verificar se realmente houve mudança no fluxo de execução normal do programa. Outras palavras-chaves presentes neste arquivo e que contribuem para a identificação deste alerta são: **gcc**, **cat >** e *#.*

```
<HTML><HEAD>
<TITLE>404 File Not Found</TITLE>
</HEAD><BODY>
<H1>File Not Found</H1>
The requested URL /cgi-bin/phf was not found on this server.<P>
</BODY></HTML>
```

Figura 2.20 Ataque PHF.

A Figura 2.20 mostra uma conexão suspeita onde possivelmente houve a tentativa do cliente em explorar o script CGI PHF para permitir que um usuário mal intencionado execute comandos arbitrários em uma máquina com um servidor *web* mal configurado [31]. Em negrito a palavra-chave candidata.

Outro exemplo de investida com sucesso aparece na Figura 2.21. Neste caso, o atacante cria um *script* na linguagem *perl* na máquina alvo para gerar um *shell* através da chamada *exec("/bin/bash")*. Depois do *script* criado, o comando *chmod* é chamado para setar os bits de SUID e GUID, que permitem que este script seja executado por qualquer usuário com as permissões do proprietário e do grupo

proprietário. Sendo assim, quando o atacante emite o comando **perl magic** para interpretar o *script magic,* ele cria uma linha de comando de administrador.

Dentre as palavras-chaves candidatas em destaque aparece o comando **perl** que é suspeito por permitir a um usuário interpretar um script que pode ter intenções danosas ao sistema. Maiores detalhes a respeito do funcionamento desta técnica de invasão podem ser encontrados na pesquisa de [21].

```
login: tristank
Password:

Last login: Tue Jun  2 02:28:37 from calvin.world.net
..
Trecho Retirado para facilitar o entendimento
…
marx> which perl

/usr/local/bin/perl
marx> cat > magic

#!/usr/local/bin/perl
$ENV{PATH="/bin:/usr/bin";
$>=0;$<=0;
exec("/bin/bash");

marx> chmod 4744 magic

marx> perl magic

bash#
bash# exit
marx>

marx> ^Dlogout
```

Figura 2.21 Ataque *Perl magic* para escalação de privilégios.

Como mencionado anteriormente, o treinamento da rede neural utilizada para identificar as conexões suspeitas não deve ser feito estritamente com conexões reconhecidamente de ataques, pois fatalmente a rede terá um comportamento tendencioso. Isto ocorre porque palavras-chaves que contribuem para delatar um

contexto de intrusão, também podem aparecer em conexões que nada têm de danoso para o sistema. A Figura 2.22 ilustra este tipo de situação em que uma conexão, onde um usuário legítimo simplesmente se equivocou ao digitar a sua senha, pode ser erroneamente confundida com o ataque de adivinhação de senhas **guess** (Figura 2.10).

```
#'$% !"$#'

UNIX(r) System V Release 4.0 (pascal)

login: bullard
Password:
Login incorrect
login: bullard
Password:
Last login: Fri Jan 23 14:34:50 from attacker
Sun Microsystems Inc.   SunOS 5.5       Generic November 1995
You have new mail.
…
```

Figura 2.22 Conexão normal parecida com o ataque de adivinhação de senhas.

Conexões remontadas pelo módulo **GeraArqConexões**, tais como as já apresentadas, servem de subsídio, juntamente com o conjunto de *keywords*, para que o módulo de contagem das palavras-chaves gere a massa de dados de treinamento para rede neural.

2.3.3 Treinamento da rede neural

Neste trabalho foi utilizado o simulador SNNS (*Stuttgart Neural Network Simulator*) [48] para verificar o comportamento de uma rede neural quando submetida à um treinamento com os dados produzidos pelos módulos em java implementados. Foi criada uma rede com 22 entradas, correspondente às 22

palavras-chaves adotadas, e com um neurônio de saída (Figura 2.23). O algoritmo de aprendizado utilizado foi o *BackPropagation.*

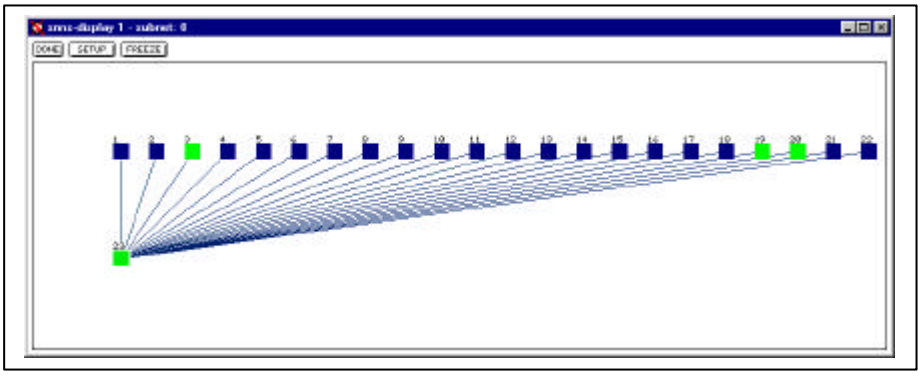

Figura 2.23 Arquitetura da rede neural.

Posteriormente foi feito um treinamento supervisionado com um arquivo de ocorrências, sendo que cada linha do mesmo foi composta por 22 contadores e mais uma saída. Esta saída teve dois valores possíveis: 0 (zero) se a conexão fosse considerada normal e 1 (um) se a conexão fosse reconhecidamente de ataque[17]. Foram apresentadas randomicamente à rede neural 1065 conexões (65 de ataque e 1000 de normais) durante 4.000 ciclos de treinamento (Figura 2.24).

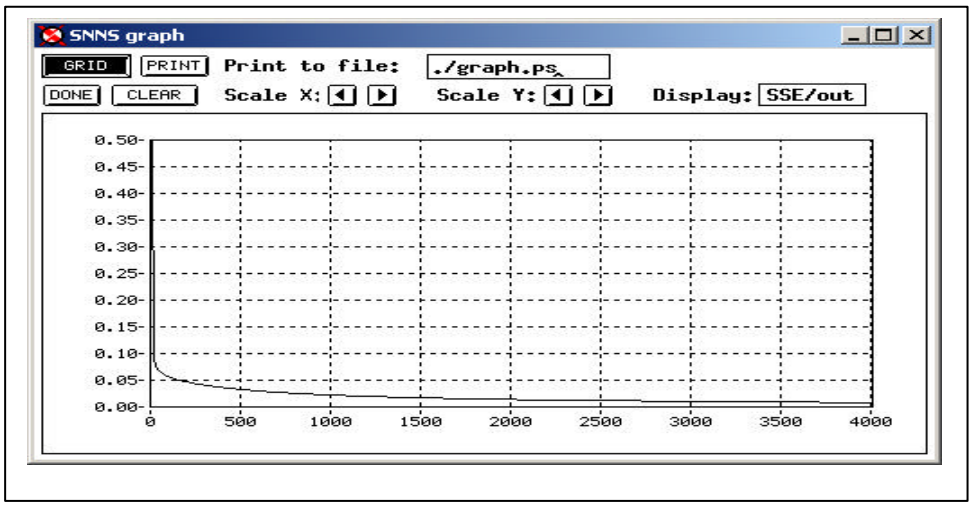

Figura 2.24 Gráfico de aprendizagem da rede neural.

[17] As conexões de ataque foram assumidas como tal de acordo com a documentação do DARPA.

Após a fase de treinamento a rede apresentou o comportamento mostrado na Figura 2.25, onde as conexões de ataque estão representadas pelos picos no gráfico, ou seja, aquelas cujo grau de severidade apresentado pelo neurônio de saída está próximo do valor 1 (eixo vertical). O eixo horizontal do gráfico representa o instante no tempo em que determinada conexão foi apresentada à rede neural.

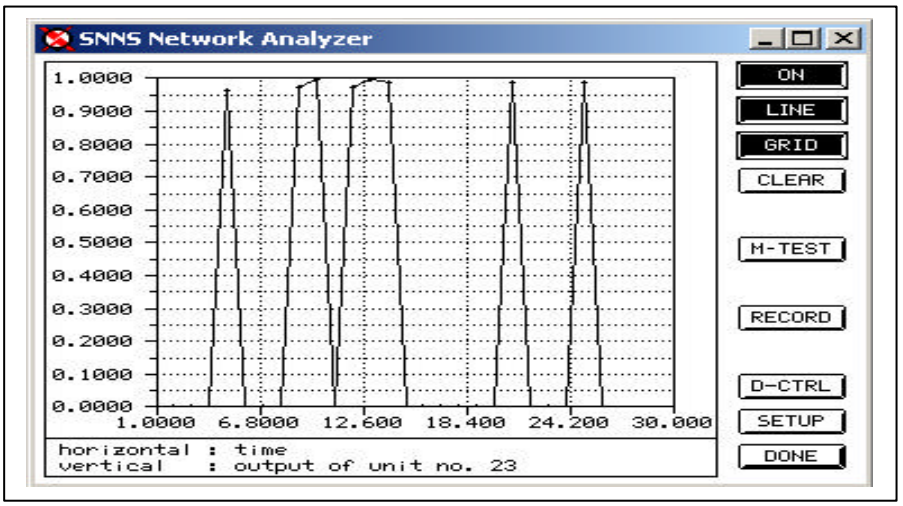

Figura 2.25 Comportamento da rede neural após a fase de treinamento.

A Tabela 2.1 mostra as naturezas das conexões utilizadas neste teste com o respectivo instante em que foram submetidas à apreciação da rede neural, conforme a representação da Figura 2.25.

Instante de apresentação	Natureza da Conexão	Grau de severidade
1 a 5	Normal	*
6	Ataque PHF	0,962
7 a 9	Normal	*
10	Ataque rcp	0,973
11	Ataque guess	1,000
12	Normal	*
13	Ataque ssh	0,973
14	Ataque guess	1,000
15	Ataque format clear	0,989
16 a 20	Normal	*
21	Ataque perl clear	0,987
22 a 24	Normal	*
25	Ataque guest	0,990

* representa valores próximos de 0 (zero).

Tabela 2.1 Grau de severidade emitido pela rede neural.

2.4 Conclusões

Os resultados indicam que a utilização de redes neurais para o reconhecimento de ataques por abuso, através da análise do conteúdo de conexões TCP, pode ser viável. Cabe ressaltar a importância das expressões regulares para representar as palavras-chaves utilizadas, pois se verificou durante a fase de treinamento a presença de falsos positivos relativos a conexões normais que continham palavras potencialmente suspeitas. Por exemplo, uma conversa entre dois usuários onde um ensinava ao outro como compilar um programa, ou como criar um novo usuário. Com o uso de expressões regulares foi possível restringir quando determinados termos realmente carregam consigo tendências maliciosas.

3 APLICABILIDADE DO MODELO SAARA NO NIDIA

A mini-sociedade de detecção a ser gerada pelo modelo de atualização proposto neste trabalho deve ter características básicas para ser capaz de atender aos requisitos de um mecanismo genérico de detecção de intrusão baseado na tecnologia de sociedade de agentes inteligentes. Esta seção descreve como seria aplicada a estrutura de uma SAARA no modelo de SDI apresentado por [23], o NIDIA[18].

3.1 O Sistema NIDIA

O NIDIA constitui-se na proposta de um SDI multiagentes [50], capaz de gerar um índice de suspeita de ataque a partir da análise de dados coletados de logs de *hosts* e de pacotes de tráfego de rede. A escolha da arquitetura multiagentes para um SDI visa obter as seguintes vantagens enumeradas por [3, 15]:

- facilidade de manutenção e atualização do sistema com adição e remoção de agentes, mantendo o máximo de disponibilidade do mesmo;
- possibilidade de atualização dos agentes responsáveis pelo mecanismo de identificação de ataques;
- capacidade de se obter maior adequação a determinado ambiente, através da utilização de agentes especializados para um computador ou rede em particular;

[18] *Network Intrusion Detection System based on Intelligent Agents.*

- maior tolerância a falhas que sistemas monolíticos que possuem um único ponto crítico de falhas;
- alto potencial de escalabilidade através da adição de novos agentes para manter a performance exigida em sistemas em expansão.

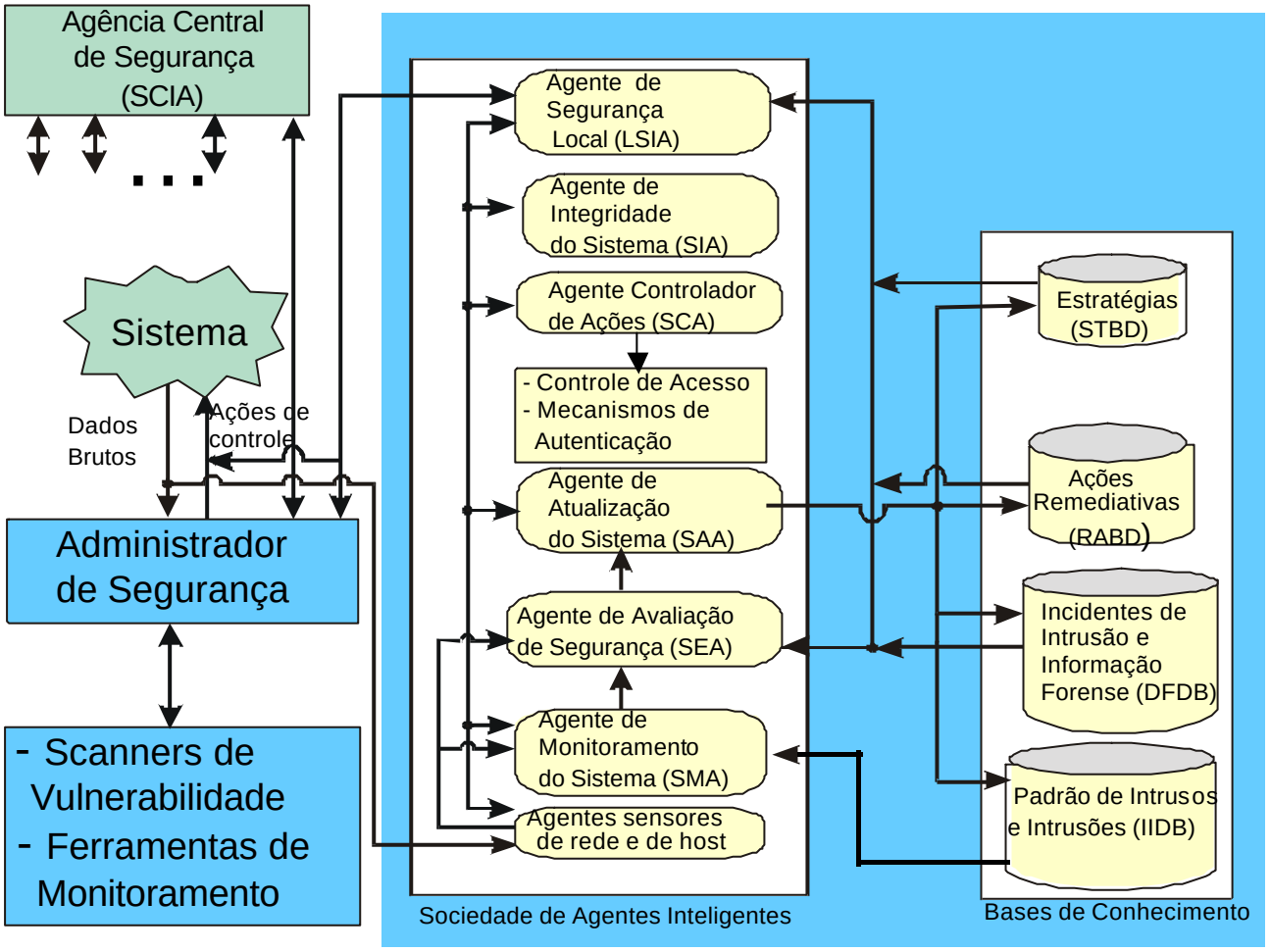

Figura 3.1 Arquitetura do NIDIA

Arquitetura apresentada para o NIDIA é inspirada no modelo lógico do CIDF, possuindo para este fim agentes com função de geradores de eventos (agentes sensores), mecanismos de análise dos dados (agentes de monitoramento e de avaliação de segurança), mecanismos de armazenamento de histórico e um módulo para realização de contramedidas (agente controlador de ações). Além

disso, existem agentes responsáveis pela integridade do sistema e pela coordenação das atividades do SDI como um todo.

Segue abaixo um pequeno descritivo das funcionalidades dos agentes e demais elementos que compõem o NIDIA:

Agentes Sensores: funcionam como os “sentidos receptores” do sistema, com a função de capturar o que está ocorrendo no meio exterior. Dividem-se em duas categorias: agentes sensores de rede e agentes sensores de host. Aqueles atuam em pontos estratégicos e comportam-se como monitores de rede passivos, capturando pacotes e visando não interferir na performance nem sequer corromper o tráfego. Já os agentes de host coletam informações dos logs de servidores específicos.

Agente de Monitoramento de Sistema (System Monitoring Agent – SMA): recebe os dados dos agentes sensores, realiza uma pré-formatação e repassa para o agente de avaliação de segurança.

Agente de Avaliação de Segurança do Sistema (Security Assessment Agent ou Security Evaluation Agent – SEA): responsável por emitir um grau de suspeita sobre os eventos que foram previamente formatados. Para isso utiliza bases de conhecimento, como a base de dados de padrões de intrusões (IIDB), a base de dados de incidentes de intrusão (DFDB) e a base de estratégias (STDB).

Agente de Atualização do Sistema (System Updating Agent ou System Atualization Agent – SAA): responsável pela atualização das bases de conhecimento (DFDB, IIDB, RADB, STDB).

Agente Controlador de Ações (System Controller Agent – SCA): com base no parecer do SEA, este agente deve tomar uma contramedida de acordo com as bases de dados de estratégia (STBD) e de ações (RADB).

Agente de Integridade do Sistema (**Self-Integrity Agent – SIA)**: responsável por manter o sistema resistente à subversão, evitando que agentes do próprio SDI possam fazer parte de algum esquema de ataque.

Agente de Segurança Local (**Local Security Intelligent Agent – LSIA**): funciona como o gerente de toda a sociedade de agentes e também como interface com o administrador do sistema.

Agente Administrador do Sistema: é o próprio administrador do sistema (agente humano) que realiza as atribuições de configurar as bases de dados do sistema, ativar/desativar agentes e analisar os alertas dados pelo sistema a respeito da cadeia de dados coletados.

Bases de conhecimento: o NIDIA possui bases de dados com função de armazenar as estratégias para a sua política de segurança (STBD), as contramedidas a serem tomadas em caso de atividade suspeita (RADB) e os padrões de ataque utilizados para a detecção (IIDB). Além disso, possui uma base de dados responsável por armazenar os incidentes de intrusão (DFDB), que pode servir de fundamento para investigar e provar a culpa de um determinado atacante.

3.2 Mecanismo de Detecção de Ataques no NIDIA baseado no SAARA

O NIDIA apresenta-se como um SDI capaz de analisar diversos tipos de informações, coletadas a partir de eventos gerados no meio exterior, a fim de verificar a ocorrência de diferentes tipos de ataques. Uma questão importante a ser tratada neste momento é relativa a sobrecarga de trabalho que uma ferramenta com esse escopo de tarefas tem que lidar.

Enquanto o exame dos cabeçalhos em busca de assinaturas de ataques é mais simples de ser feito, consome poucos recursos computacionais e possui baixas taxas de falsos positivos, o mesmo não se pode dizer da análise de conteúdo. Não é tarefa trivial estabelecer qual seria a “impressão digital” de uma determinada invasão. Além disso, quando um SDI se propõe a abranger ambas as técnicas, pode ocorrer que durante a análise de palavras-chaves imersas nas conexões, a estrutura de coleta/identificação perca um pacote maliciosamente forjado que contenha um ataque de cabeçalho.

Segundo [34], “devido ao fato de detecções precoces serem as melhores, qualquer pacote que você possa identificar por causa de uma assinatura no cabeçalho é uma grande vitória”. Em outras palavras, um SDI que utilize diversas técnicas de identificação não deve permitir que análises concorrentes interfiram na eficiência uma da outra.

Faz-se necessário, portanto, um aprofundamento da arquitetura multiagente proposta em [23] para que o NIDIA possa cumprir o papel de um

identificador de ataques bem abrangente. Devido às características dos dados capturados e da diversidade dos ataques a serem analisados, propõe-se um detalhamento do SMA e do SEA. Ambos agora são expressos como sociedades de agentes especializados que cooperam entre si e executam de forma independente, com o objetivo de manter um nível de desempenho aceitável, mesmo com a carga de tarefas simultâneas a serem realizadas. Agora o mecanismo de detecção no NIDIA é uma sociedade de agentes independentes com o potencial de ser atualizada com grande flexibilidade. Em outras palavras, este mecanismo de detecção pode ser integrado ao modelo SAARA[19] (Figura 3.2).

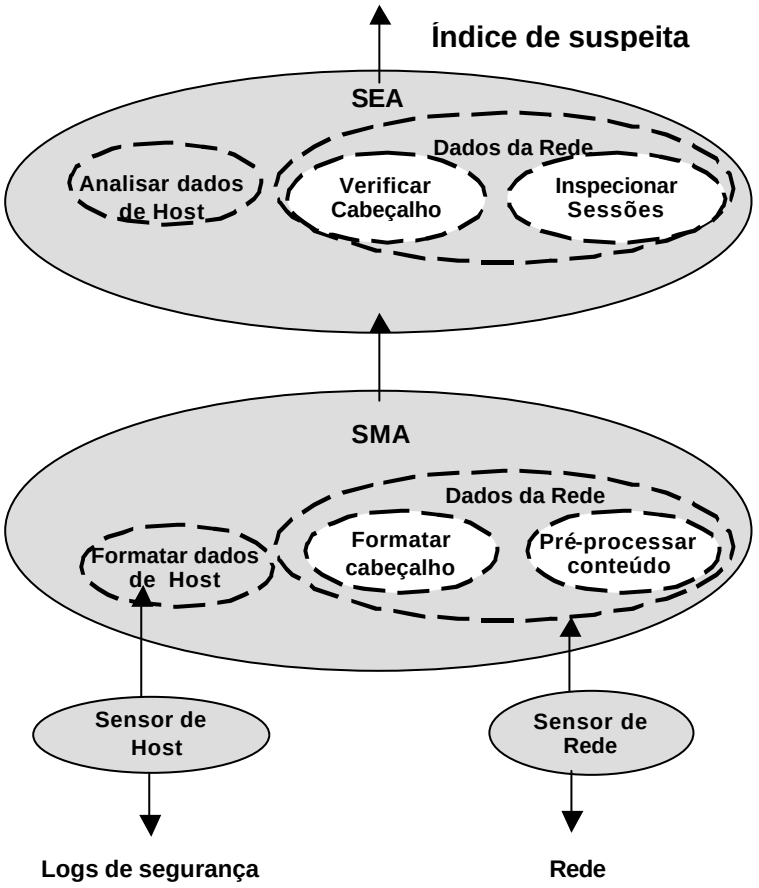

Figura 3.2 Arquitetura de reconhecimento de ataques para o NIDIA, utilizando uma SAARA.

[19] Apesar do foco deste trabalho ter sido direcionado para a detecção de ataques de rede, técnicas para detecção de ataques de host também poderiam ser incluídas na SAARA.

O fluxo de informações no mecanismo de identificação é descrito nos tópicos a seguir:

Tratamento de informações de host

Os agentes sensores de host fazem a leitura dos logs de segurança de servidores específicos e repassam para o agente SMA de host para que o mesmo formate os dados. Posteriormente tais informações são enviadas para o agente SEA de host para que seja feita a avaliação dos dados coletados, que foram previamente formatados.

Tratamento de informações de tráfego de rede

Os agentes sensores de rede capturam pacotes do tráfego e realizam duas tarefas: a) enviam o cabeçalho dos pacotes para os agentes SMA de rede específicos para formatar esse tipo de informação e b) constroem as conexões TCP entre servidor-cliente a partir do conteúdo dos pacotes recolhidos.

Os agentes SMA realizam a tarefa de formatar os dados recebidos. No caso específico do tratamento das conexões montadas pelo agente sensor de rede o agente SMA especializado consulta um banco de dados de palavras-chaves para a contagem de ocorrência das mesmas.

Posteriormente os agentes SMA enviam os dados pré-processados para os agentes SEA avaliarem. Desta forma, existem agentes SEA especializados em tratar o cabeçalho dos pacotes através de filtros, ou possivelmente algum sistema baseado em regras, e existem agentes SEA especializados em inspecionar o conteúdo das conexões TCP. O produto gerado pelos agentes SEA é o grau de

suspeita dos dados verificados que é enviado para o Agente Controlador de Ações (SCA), para caso necessário, tomar alguma contramedida baseada em um banco de ações.

Contudo, para manter maior facilidade na adequação por parte do usuário do SDI, ainda existe a alternativa de dividir o SEA de detecção de conteúdo em duas categorias. Uma utilizaria a rede da SAARA disponibilizada pela ACS e a outra teria uma rede neural paralela que poderia ter seu treinamento disparado pelo próprio administrador do sistema, de acordo com a política de segurança da empresa.

3.3 Criação da Sociedade de Agentes no NIDIA baseado no SAARA

Com o objetivo de criar a sociedade de agentes de forma mais amigável, utilizou-se a ferramenta **Zeus** [6] que consiste em um *toolkit* capaz de dar suporte ao desenvolvimento de sistemas com agentes cooperativos. O **Zeus** suporta o gerenciamento e a comunicação de agentes autônomos estáticos em java, de forma que o programador tem que se preocupar apenas com o problema do negócio específico, pois a infraestrutura para criação e comunicação de agentes já se encontra pronta no **Zeus**.

O processo prático de criação deste protótipo no Zeus foi embasado nas etapas de desenvolvimento de uma sociedade de agentes descritas em [11]. As seções 3.3.1 a 3.3.5 detalham cada uma destas etapas.

3.3.1 Criação da ontologia

Para que os agentes possam se comunicar entre si, necessita-se da criação de um vocabulário de termos que sejam comuns aos agentes daquele domínio. Esse vocabulário é denominado de ontologia[20].

A Figura 3.3 mostra o editor de ontologias do Zeus onde foi criada a ontologia *teste.ont* que contém os termos comuns usados nesta implementação. O agente sensor de rede será responsável por produzir o fato[21] **ArqConexão,** que servirá de insumo para o agente de monitoramento do sistema produzir o fato **ArqRede[22].** Este representa os dados da conexão formatados para serem processados pela rede neural e servirá de entrada para o agente de avaliação de segurança. Então, o SEA repassa os dados formatados para a rede MLP e emite um índice de severidade com base na resposta da rede neural representado na ontologia pelo fato **ÍndiceAtaque.**

[20] Segundo [11], ontologia da aplicação é "o conhecimento declarativo que representa os conceitos relevantes dentro do domínio da aplicação".
[21] Por conveniência, o termo fato é utilizado no Zeus para descrever um conceito individual no domínio.
[22] O fato ArqRede não se refere à rede de tráfego de dados, e sim à entrada formatada para a rede neural.

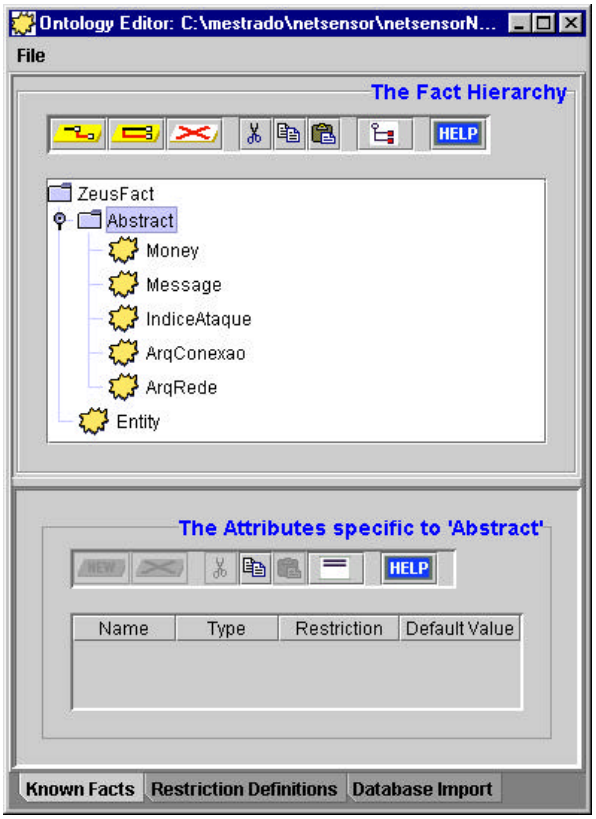

Figura 3.3 Editor de ontologias da ferramenta Zeus.

3.3.2 Criação dos agentes

Posteriormente à fase de criação da ontologia, inicia-se a criação dos agentes e a atribuição de suas tarefas. Foram definidos os agentes NetSensor (sensor de rede), SMA (agente de monitoramento) e o SEA (agente de avaliação). A Figura 3.4 também mostra, respectivamente, as tarefas associadas a cada um dos agentes: **GerarArqConexão, GerarArqRede** e **GerarÍndiceAtaque.**

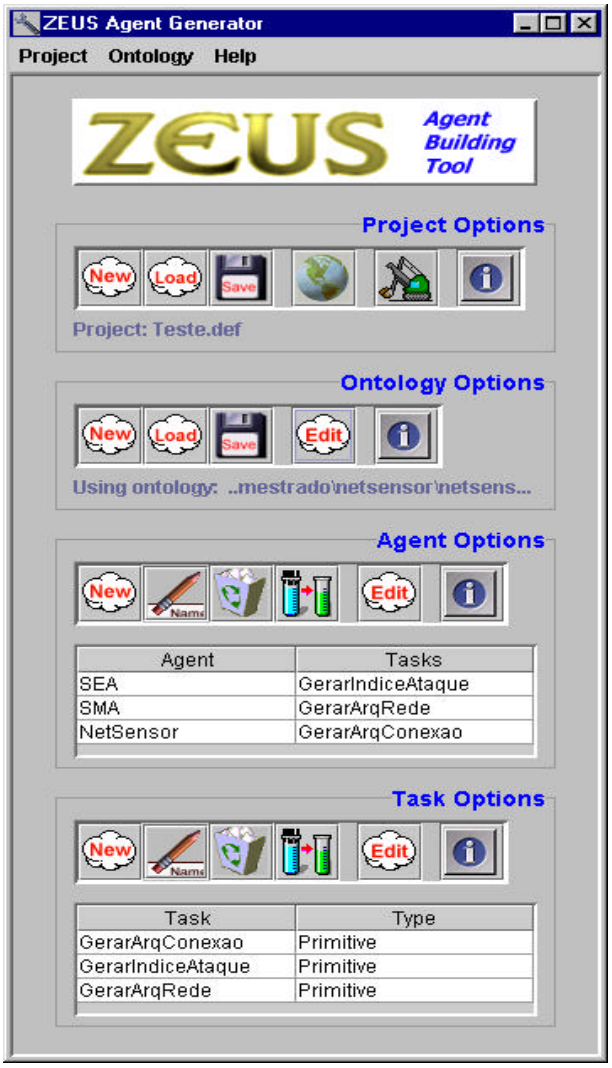

Figura 3.4 Gerador de agentes do Zeus.

Para que uma tarefa seja executada pode ser necessário que determinadas condições estejam presentes (pré-condições). Por exemplo, o agente SEA para realizar a tarefa de **GerarÍndiceAtaque** necessita ter o arquivo de entrada já formatado para rede neural e assim produzir como resultado **ÍndiceAtaque**, conforme apresentado na Figura 3.5.

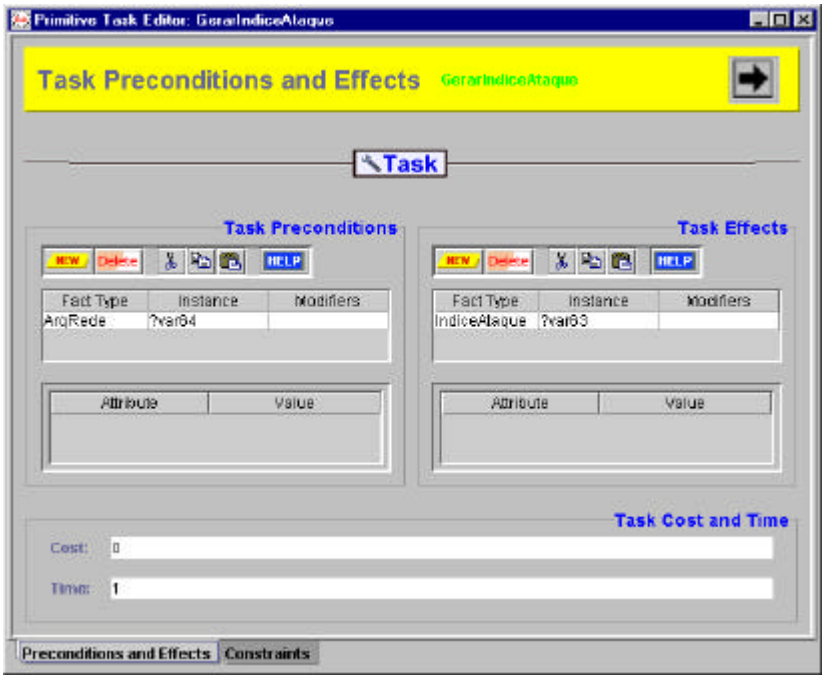

Figura 3.5 Configuração da tarefa GerarÍndiceAtaque do agente SEA.

Para o que o agente SMA realize a tarefa **GerarArqRede** ele precisa de um fato do tipo **ArqConexão** como entrada a fim de produzir **ArqRede** (Figura 3.6). Este último servirá de insumo para a tarefa do agente SEA.

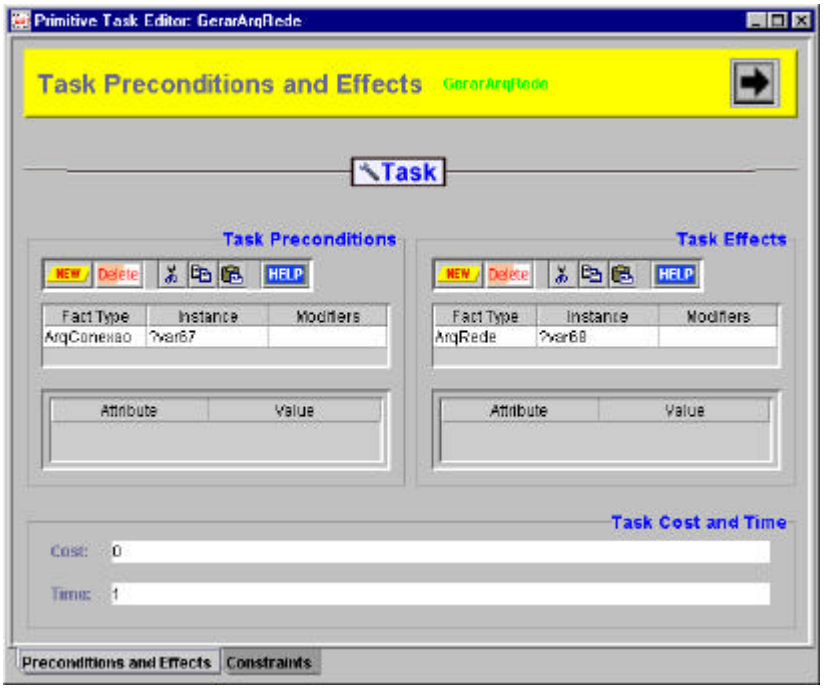

Figura 3.6 Configuração da tarefa GerarArqRede do agente SMA.

No início desta cadeia está o **NetSensor** que é responsável por gerar **ArqConexão,** que representa os dados presentes em uma conexão TCP. Como neste protótipo estabeleceu-se que este mesmo agente capturaria os pacotes e os montaria em conexões, então o **NetSensor** não necessita de pré-condições para a realização desta tarefa (Figura 3.7).

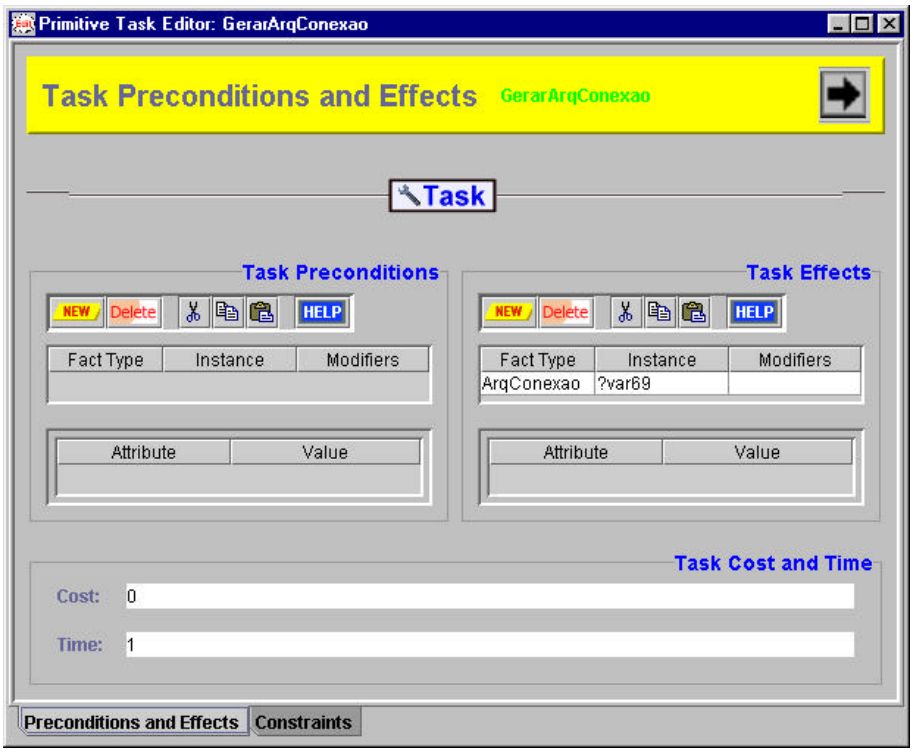

Figura 3.7 Configuração da tarefa GerarArqConexão do NetSensor.

Percebe-se nas configurações acima que os termos comuns utilizados nas tarefas dos agentes, em suas pré-condições (*Task Preconditions*) e resultados (*Task Effects)*, são os que foram definidos anteriormente na ontologia da sociedade. Nota-se também que os três agentes formam uma cadeia de fornecimento de recursos onde o efeito esperado somente é encontrado quando cada agente realiza a sua tarefa (Figura 3.8). Neste caso, o agente SEA só produzirá **ÍndiceAtaque**, se o

SMA produzir **ArqRede**. Por sua vez o SMA precisa que o agente NetSensor produza **ArqConexão** para cumprir a sua tarefa.

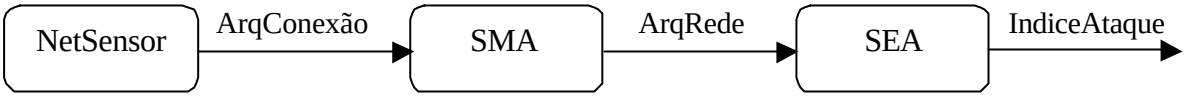

Figura 3.8 Cadeia de recursos mostrando quais os recursos produzidos e consumidos.

Nesta cadeia de recursos, os agentes que suprem outros com insumos para executar determinado trabalho estão organizados em uma hierarquia. Estabeleceu-se para o caso em questão que o NetSensor seria subordinado[23] ao SMA, enquanto este seria subordinado ao SEA, conforme ilustra a Figura 3.9.

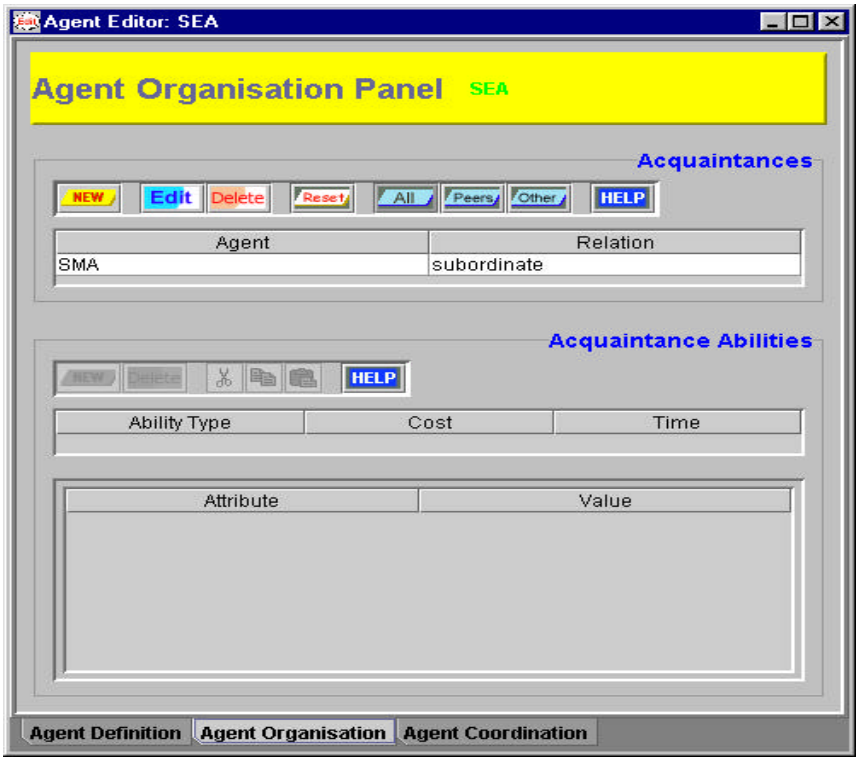

Figura 3.9 Configuração da hierarquia entre os agentes SEA e SMA.

[23] O agente subordinado deve atender as solicitações emitidas pelo agente que é o superior na relação.

3.3.3 Configuração dos agentes utilitários

Agentes utilitários são necessários para que se estabeleça uma comunicação de alto nível na sociedade e também se alcance um elevado grau de flexibilidade na adição/modificação de agentes. São eles que provêem a infraestrutura de suporte para a sociedade de agentes [11]. Estes agentes são:

- Agente Servidor de Nomes (*Agent Name Server*-ANS) – responsável por mapear o nome do agente com a sua localização na rede;
- Facilitador (*Facilitator*) – mantém um banco de dados de habilidades referentes a cada agente;
- Visualisador (*Visualiser*) – utilizado quando se deseja monitorar o funcionamento da sociedade de agentes.

Quando um determinado agente tarefa precisa de um recurso presente na pré-condição de sua tarefa, ele consulta o Facilitador para descobrir qual agente na sociedade possui esta habilidade. De posse do nome deste agente, o agente iniciador da comunicação solicita ao ANS a localização na rede (endereço IP) do agente possuidor da habilidade requerida. A partir deste ponto inicia-se o processo de contratação entre os dois agentes. A Figura 3.13 mostra a tela de configuração dos agentes utilitários no Zeus, através da qual é possível informar em que *host* da rede os agentes utilitários estão localizados. Detalhes sobre outras informações contidas nesta tela podem ser encontrados em [11].

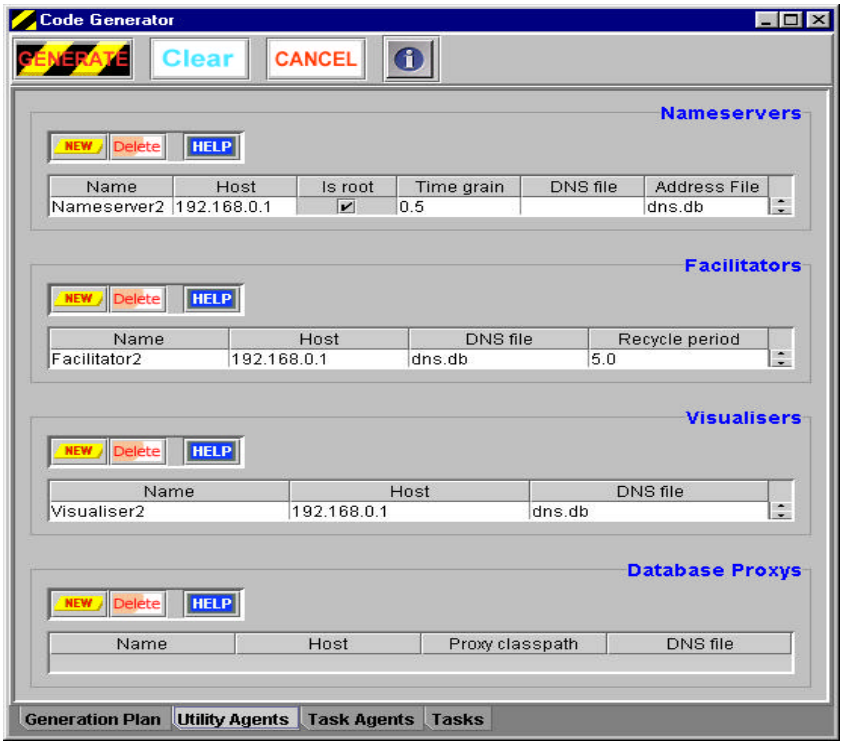

Figura 3.10 Configuração dos agentes utilitários, mostrando o endereço IP onde cada um deles reside.

3.3.4 Configuração dos agentes tarefas

Um agente criado na Seção 3.3.2 corresponde a um modelo genérico da arquitetura **Zeus**, que é capaz de interagir com os agentes utilitários para descobrir como acionar o(s) agente(s) que irão satisfazer às suas pré-condições requeridas para que ele próprio possa cumprir o seu objetivo. Mas a lógica de manipulação destas entradas (insumos) faz parte do domínio específico da aplicação. O Zeus permite então a associação dos agentes tarefas com programas externos que são os que realmente irão tratar as informações recebidas. Além disso, na configuração dos agentes tarefas é possível indicar em que *host* o agente irá executar e se ele terá uma interface gráfica (GUI) que permita verificar características do agente durante a sua execução (Figura 3.11).

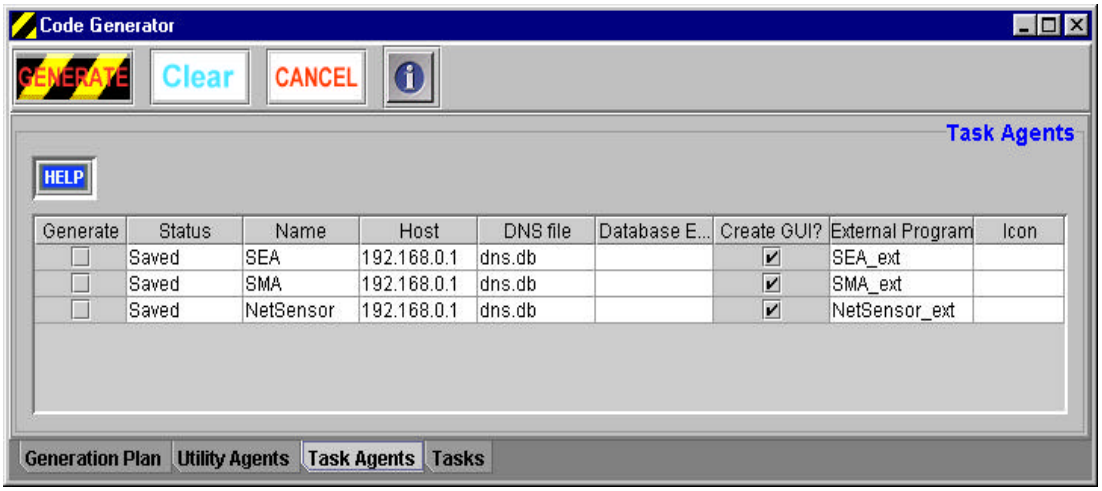

Figura 3.11 Configuração dos agentes tarefas.

Antes de explicar o modo de funcionamento dos programas externos propriamente ditos, cabe enfatizar que outra grande facilidade de utilizar uma ferramenta como o **Zeus** para a criação de uma sociedade multiagentes é justamente a geração automática de código tanto dos agentes definidos quanto de suas tarefas especificadas (Figura 3.12). Os produtos obtidos por esta geração são os fontes em java dos agentes e de suas tarefas prontos para serem compilados.

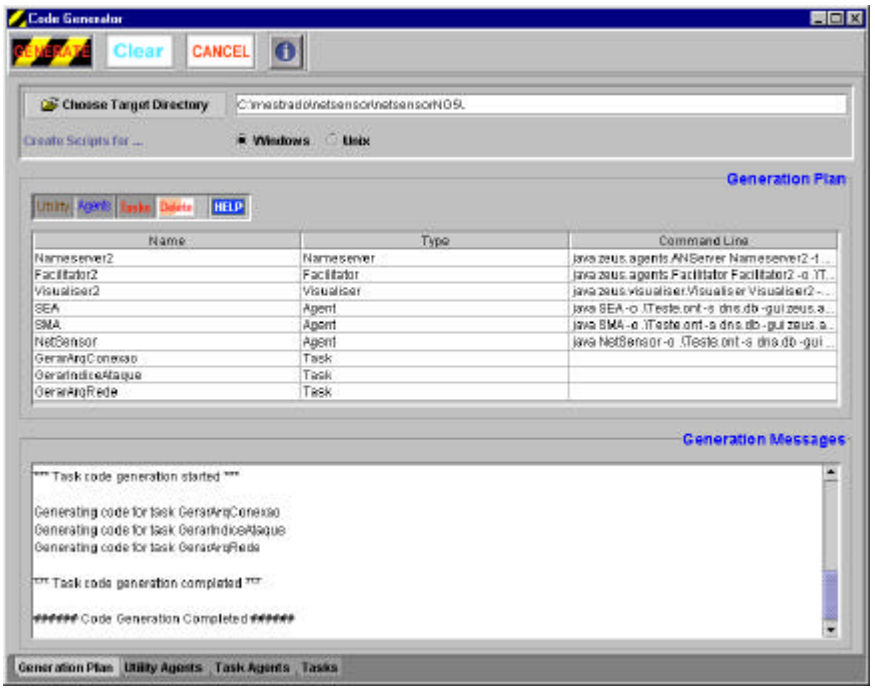

Figura 3.12 Geração de agentes e suas tarefas.

3.3.5 Implementação dos programas externos dos agentes

Até este momento, os agentes gerados pela ferramenta estão preparados para se comunicar entre si. Faz-se necessário agora a implementação do tratamento das informações recebidas por cada agente de acordo com a lógica do negócio específico que esta sociedade está tratando. Neste caso, utilizou-se o esquema adotado nos módulos em java desenvolvidos (Figura 2.12), considerando-se que a rede neural de detecção já estava devidamente treinada. Assim, este protótipo representaria uma implementação parcial de uma SAARA em ambiente de produção, sendo que as atividades específicas de cada agente foram atribuídas aos seus respectivos programas externos.

Primeiramente deve-se considerar que a estrutura apresentada na metodologia de treinamento na Seção 2.2 trata basicamente com dados *offline*, isto é, sua sistemática não precisa estar preparada para analisar informações em tempo real e emitir pareceres sobre a periculosidade de trechos de tráfego de rede a tempo de algum sistema de contramedidas tomar alguma ação para evitar maiores danos ao sistema.

Este protótipo no **Zeus**, entretanto, constitui-se de uma sociedade de agentes representando uma SAARA já em funcionamento em um SDI, ou seja, capaz de fazer o tratamento de dados *online*. Sendo assim, algumas extensões devem ser feitas na estrutura para que a SAARA analise os eventos à medida que eles vão ocorrendo.

A implementação do programa externo do NetSensor foi feita incorporando técnicas utilizadas no módulo java GeraArqConexões, dentro da perspectiva mencionada nos dois parágrafos anteriores. Para tal foram utilizadas as classes ***NetAgent_ext***, ***CapturaPacotes*** e ***CriaConexões***.

O ***NetAgent_ext*** instancia um objeto da classe ***CapturaPacotes*** para que o mesmo inicie a coleta de pacotes na rede, podendo fazer a utilização de algum filtro para selecionar determinado protocolo, subrede, host, etc. Os pacotes então são enviados para a classe ***CriaConexões*** que é responsável por montar as conexões com auxílio de uma *hashtable* (Figura 3.13).

Logo que ocorre o término da conexão TCP, indicado pelo recebimento de um pacote com flag FIN ou RST setados, esta classe envia o identificador da conexão[24] e seu respectivo conteúdo para o agente de monitoramento (SMA). Paralelamente são criados os arquivos referentes a cada conexão montada, para posterior checagem do correto funcionamento do sistema. Não foi foco desta implementação o tratamento dos ataques de cabeçalho.

[24]O identificador usado foi uma string no seguinte formato: IpOrigem_IpDestino_PortaOrigem_PortaDestino.

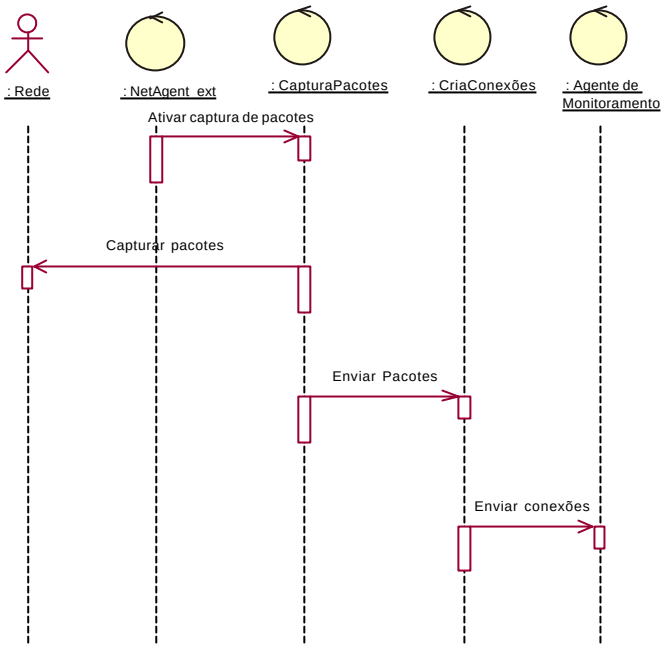

Figura 3.13 Diagrama de seqüência do programa externo do *NetSensor*.

Para cada linha de conexão recebida do NetSensor, o SMA_ext[25] conta as ocorrências das palavras-chaves lidas previamente de um arquivo. Quando toda a conexão for recebida, ele então envia para o agente de avaliação de segurança (SEA) uma mensagem contendo o identificador desta conexão seguido dos valores acumulados de cada palavra-chave separados por espaço. O programa externo do SMA também vai criando um arquivo contendo as mensagens enviadas para o SEA para fins de histórico e verificação.

Para o SEA atender às suas funções de emitir graus de risco das cadeias de contadores recebidas do SMA, o seu respectivo agente externo faz uso de uma rede neural em Java que foi baseada na rede MLP, com algoritmo de aprendizagem *BackPropagation*, encontrada em [4].

[25] Programa externo associado ao Agente de Monitoramento do Sistema.

Refletindo a filosofia da arquitetura de atualização, esta rede sofreu um processo de treinamento em separado e foi então serializada[26] para compor a SAARA que seria enviada a um cliente.

3.4 Resultados Parciais

Apresenta-se neste capítulo alguns resultados dos testes de aplicabilidade do SAARA no NIDIA feitos com os agentes criados na ferramenta **Zeus**. Primeiramente percebe-se que, independentemente da atividade específica de cada agente, a sociedade passa por uma fase de reconhecimento comum que constitui o cerne da infraestrutura de comunicação inter-agentes no **Zeus**.

3.4.1 Estabelecimento da comunicação entre os agentes

Logo que um novo agente é executado ele é registrado no agente utilitário servidor de nomes (ANS) informando seu nome e endereço IP. Em seguida o Facilitador solicita as habilidades deste novo agente para manter em seu banco de habilidades.

[26] O processo de serialização de um objeto permite gravá-lo e assim salvar os seus atributos. No caso da rede neural é possível salvá-la após a fase de treinamento para então utilizá-la posteriormente.

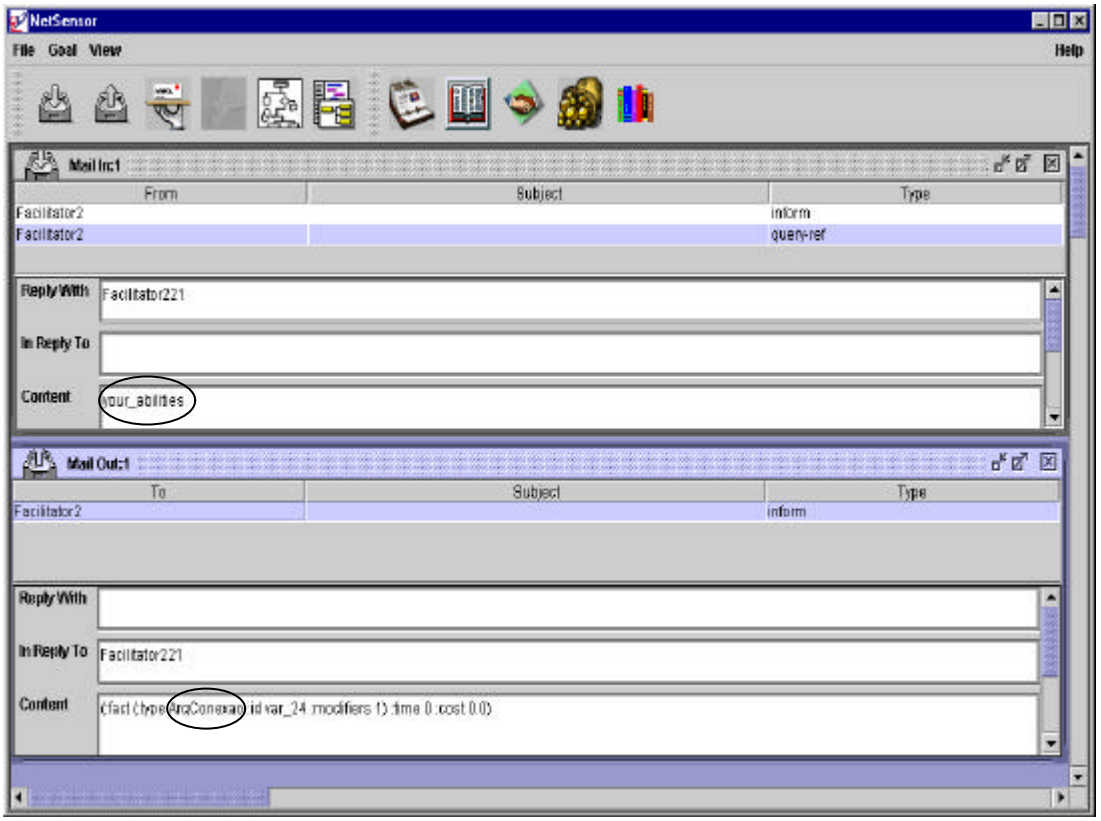

Figura 3.14 Registro de habilidades no Facilitador.

A Figura 3.14 mostra a *mailbox* associada ao agente NetSensor, onde é possível acompanhar as mensagens que chegam para este agente (*Mail In* – caixa de entrada) e as mensagens que são enviadas pelo mesmo (*Mail Out* – caixa de saída). Encontram-se destacadas a solicitação recebida do Facilitador e a resposta enviada pelo NetSensor informando que sua habilidade é produzir **ArqConexão**. Cada um dos agentes tarefas passa por uma fase similar de registro de habilidades no Facilitador.

Uma vez realizada esta etapa de preparação, o agente SEA dispara o processo para conseguir atingir o seu objetivo de produzir **ÍndiceAtaque.** Como a sua tarefa associada, **GerarÍndiceAtaque**, tem como pré-condição **ArqRede** (dados formatados para a rede neural) o SEA precisa descobrir qual agente teria a

habilidade de produzir este recurso exigido. Para isso, ele consulta o Facilitador e este retorna o nome do agente capaz de gerar a pré-condição requerida.

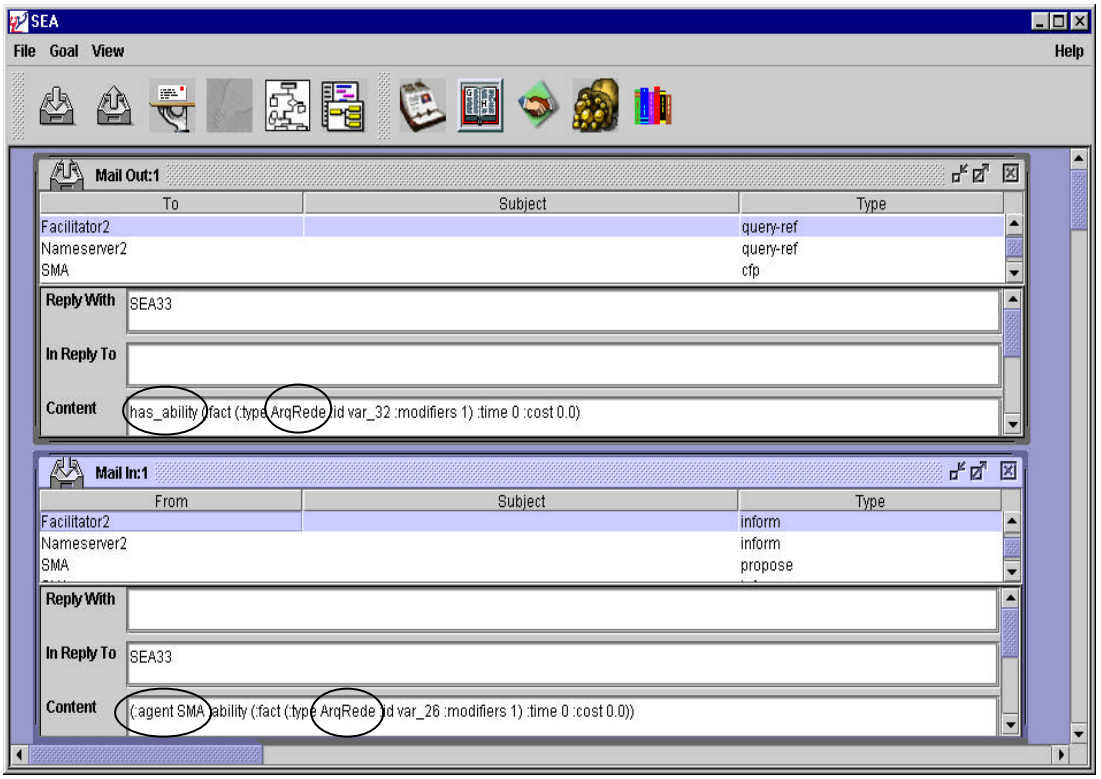

Figura 3.15 Consulta habilidade ao Facilitador.

Neste ponto, o SEA já sabe o nome do agente que tem a habilidade de produzir o recurso exigido, mas para se comunicar com ele é necessário saber em qual endereço IP o mesmo está localizado. Isto é feito através de uma consulta ao Agente Servidor de Nomes. Outros agentes que necessitem satisfazer pré-condições para realizar as suas tarefas podem passar por etapas semelhantes às mostradas nas Figura 3.15 e Figura 3.16.

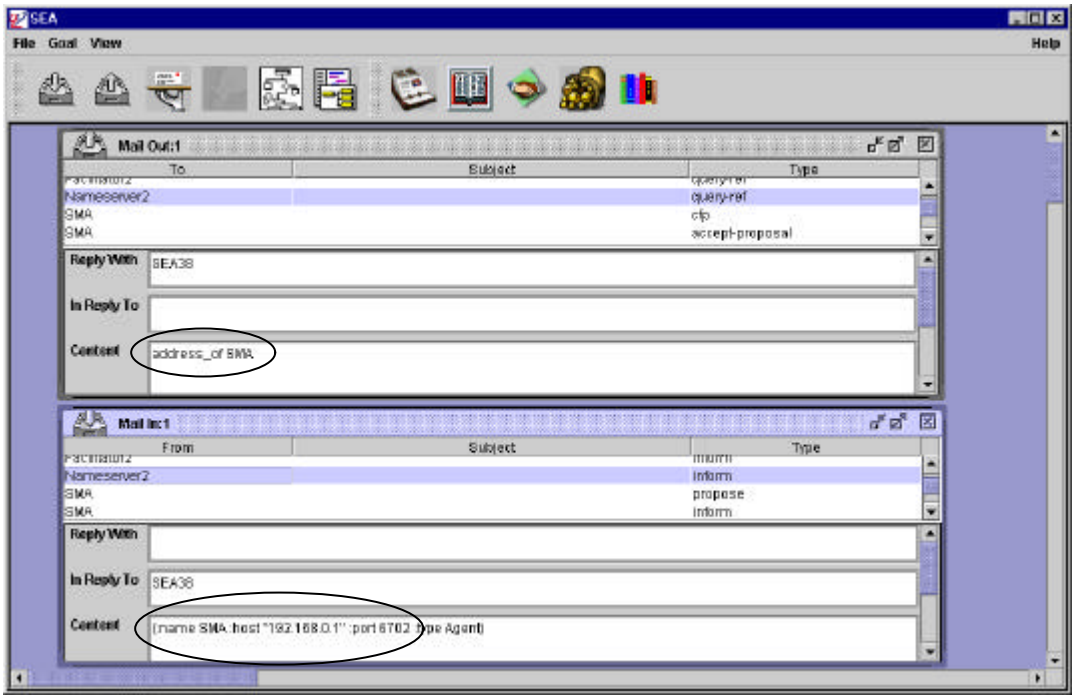

Figura 3.16 Consulta ao ANS.

O último passo necessário para que os agentes de avaliação e de monitoramento efetivamente se comuniquem é um processo de contratação dos serviços do agente fornecedor do recurso. Assim o SEA emite uma mensagem *cfp* solicitando uma proposta para os serviços do SMA. Este envia uma mensagem com a proposta e, caso aceite a mesma, o SEA envia uma mensagem de *accept-proposal* (Figura 3.17). Maiores detalhes sobre a linguagem de comunicação de agentes (ACL) utilizada pela plataforma **Zeus** podem sem encontrados em [12, 47].

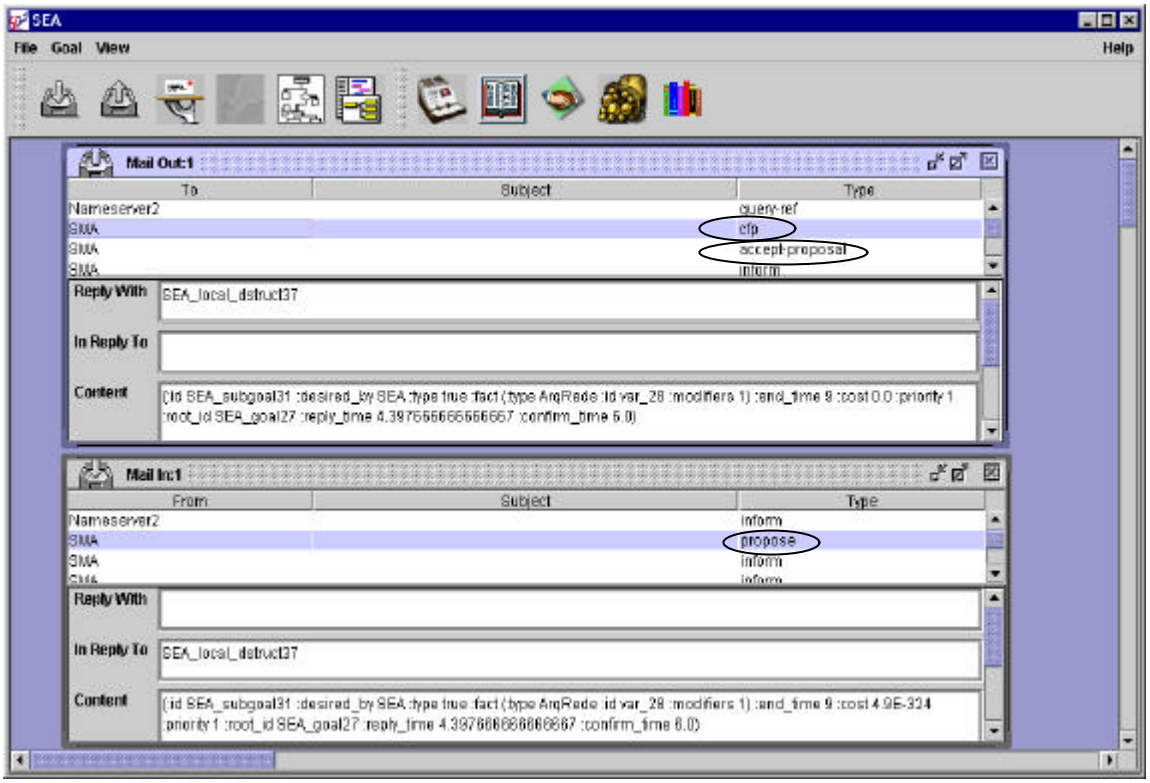

Figura 3.17 Processo de negociação entre os agentes.

3.4.2 Troca de dados entre os agentes

A fase anteriormente descrita permite que os agentes encontrem quem são aqueles capazes de suprirem as suas necessidades na cadeia de recursos. Dá-se início, então, à comunicação direta entre os agentes.

Depois de disparada a busca pelo objetivo do agente de mais alto nível (que na verdade representa a finalidade maior da mini-sociedade de detecção), os outros agentes começam a realizar as suas tarefas para que os seus produtos sejam utilizados.

A Figura 3.18 mostra a caixa de entrada da *mailbox* do SMA, onde pode-se observar as mensagens enviadas pelo NetSensor. Nesta implementação parcial, sempre que este último termina a montagem de uma conexão, ele envia cada linha da mesma para o SMA com um indicador para mostrar que tal mensagem corresponde ao conteúdo de uma conexão.

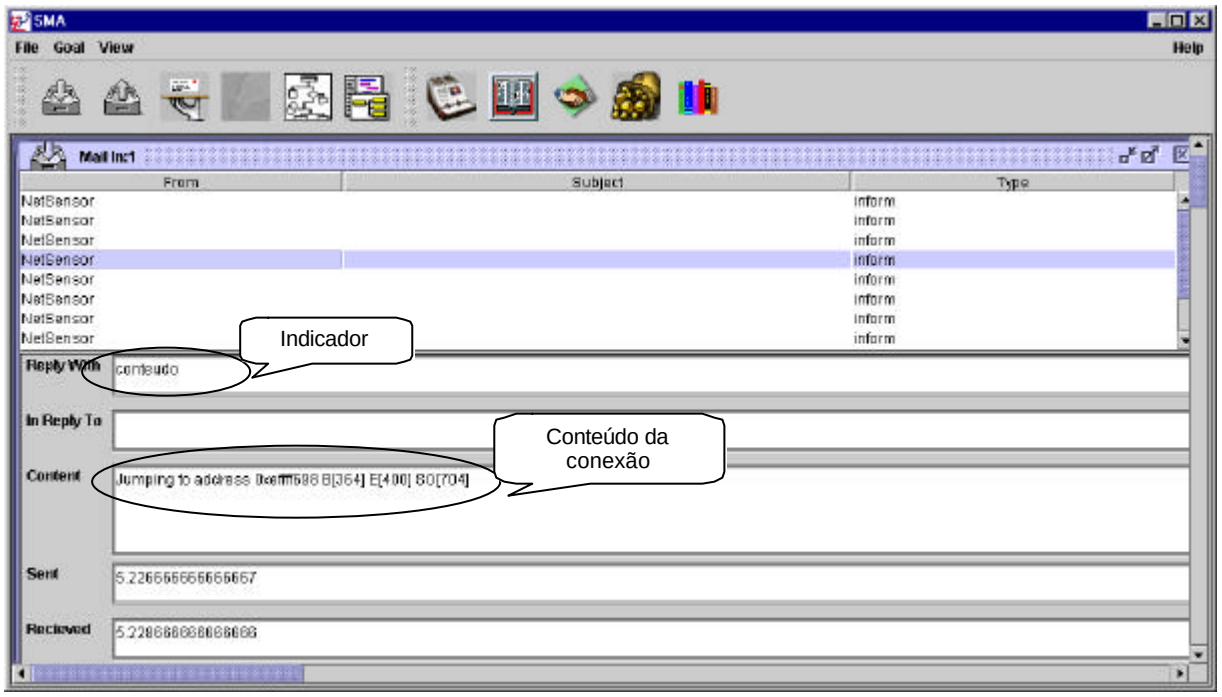

Figura 3.18 Recebendo o conteúdo das conexões.

Quando o envio do conteúdo chega ao fim, o NetSensor informa ao SMA sobre o término desta conexão através de uma mensagem que contém o indicador “fim_conexão” e também o identificador de conexão (Figura 3.19).

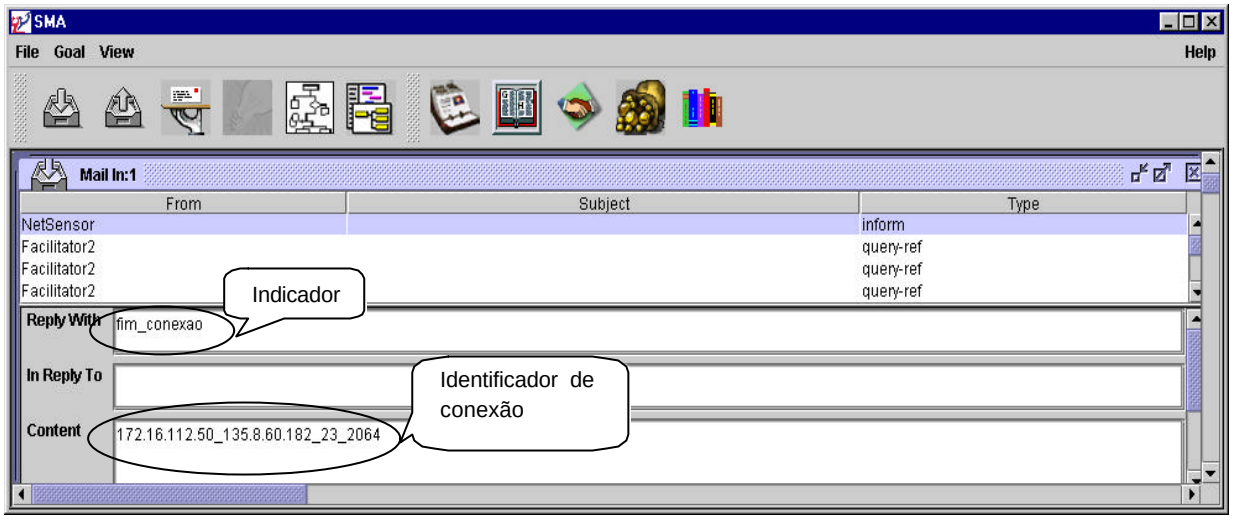

Figura 3.19 Informação sobre o término da conexão.

Para cada linha de conexão recebida, o SMA já faz a contagem de ocorrência das palavras-chaves e ao final ele envia os valores acumulados da conexão específica para o SEA, com o respectivo identificador da mesma (Figura 3.20).

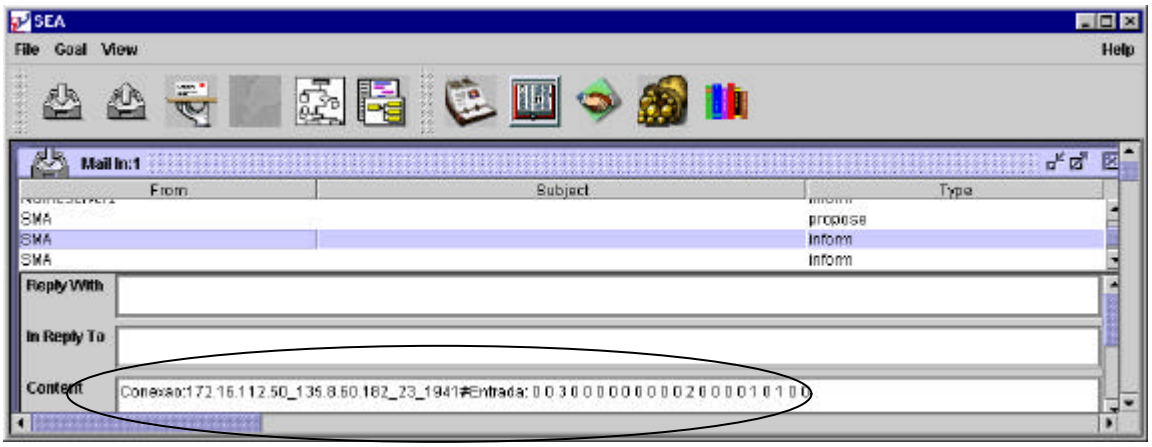

Figura 3.20 Recebimento dos contadores de ocorrência de palavras-chaves.

Por fim, o SEA repassa os dados para uma rede neural previamente treinada e recebe como resultado o grau de severidade apresentado pela conexão,

sendo que este valor encontra-se na faixa de 0 (zero) a 1 (um), representando atividades normal e suspeita, respectivamente. A seguir são mostrados alguns resultados obtidos durante a fase de testes (Figura 3.21 e Figura 3.22):

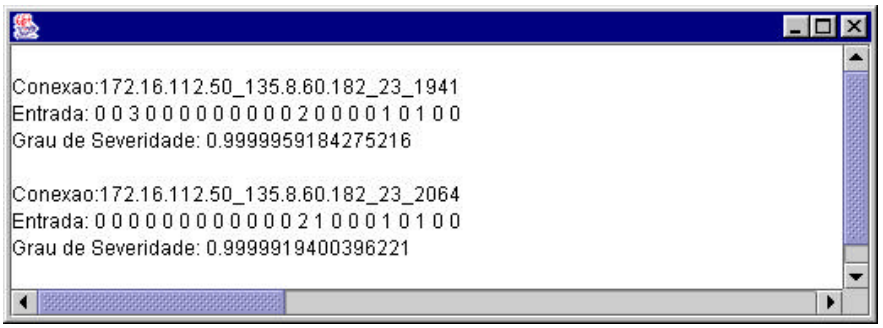

Figura 3.21 Obtenção do grau de severidade de conexões de ataque.

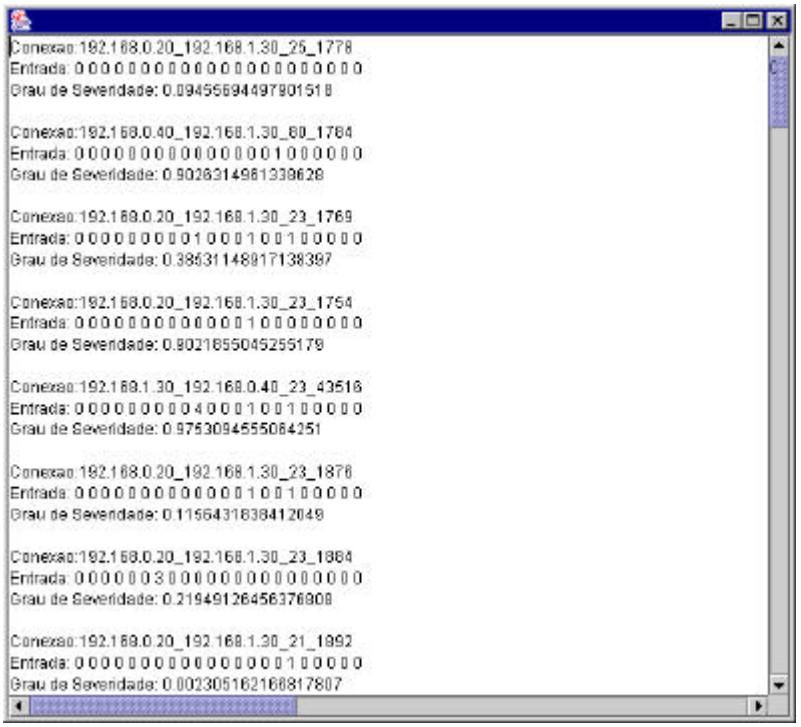

Figura 3.22 Grau de severidade de conexões diversas.

As informações geradas pela SAARA serão utilizadas, dentre outras, pelo Agente Controlador de Ações (SCA) do NIDIA que definirá quais as contramedidas

que deverão ser tomadas de acordo com o grau de risco apresentado pela conexão. Este assunto será tratado com mais detalhes na pesquisa encontrada em [41].

3.5 Conclusões

Os resultados demonstram o potencial de uma sociedade de agentes, utilizando a tecnologia de redes neurais, para a identificação de ataques de rede orientados a dados de conexões TCP. Percebe-se também que é possível, através de um sistema multiagentes, criar um mecanismo de detecção em tempo real capaz de avaliar os eventos à medida que eles vão ocorrendo para então prover tempestivamente contramedidas a fim de evitar que danos maiores possam ser causados ao sistema.

Nesta implementação foi contemplada apenas a obtenção do grau de severidade da conexão. No entanto, a análise deste dado isoladamente não pode ser considerada conclusiva para que sejam tomadas as atitudes defensivas adequadas. Faz-se necessário identificar em conjunto qual o tipo de ataque que está em andamento, se o host ou a rede origem são reincidentes, se o dia ou horário em que está ocorrendo o incidente são incomuns para os administradores de rede e outros elementos que poderiam não estar cumprindo com o plano de segurança da empresa. Isto também é objeto da pesquisa que está sendo desenvolvida em [41].

4 CONCLUSÕES

4.1 Contribuições do trabalho

Os benefícios da era digital na manipulação de informações fizeram com que uma gama extremamente diversificada de usuários se tornasse dependente dos dados armazenados no mundo cibernético. Sendo assim, transtornos que vão desde perda de algumas horas de trabalho até prejuízos de milhões de reais podem assolar aqueles que negligenciam a segurança das suas informações depositadas nesse ambiente.

Visando garantir proteção contra ataques dirigidos às suas redes de computadores, muitas empresas estão adotando a utilização de conjuntos de ferramentas de proteção, tais como *firewalls*, sistemas de detecção de intrusão e VPN´s. Entretanto, incorrem em erro as corporações que simplesmente implementam tais tecnologias em seus parques computacionais e acreditam que com isso estarão imunes aos incidentes de segurança. A realidade é que o surgimento de novas soluções informatizadas acarreta também o aparecimento de novas vulnerabilidades. Logo, a questão da segurança de informações deve ser uma preocupação constante, pois ameaças emergem freqüentemente constituindo-se em potenciais ataques sobre as debilidades dos sistemas.

Devido ao fato notório de que ferramentas de segurança estanque não defendem efetivamente as informações contra a diversidade de novas ameaças que surgem constantemente, foi proposto nesta dissertação um modelo de atualização

automática do mecanismo de detecção (SAARA) de um SDI multiagentes. Concomitantemente, apresentou-se uma sistemática para o reconhecimento de assinaturas de ataque através dos dados de conexões TCP, utilizando-se a tecnologia de redes neurais.

Além disso, outra contribuição do trabalho foi a implementação parcial do modelo apresentado, através da elaboração de módulos para a montagem das conexões TCP e análise do seu grau de periculosidade por meio da verificação da presença de palavras-chaves nos dados destas conexões.

Para demonstrar a aplicabilidade do modelo SAARA, foi apresentada a sua utilização na arquitetura multiagentes de sistema de detecção de intrusos NIDIA, através da construção de um protótipo feito na plataforma de desenvolvimento **Zeus**.

4.2 Considerações Finais

Esta dissertação desperta o interesse de pesquisas na área de sistemas de detecção de intrusão, particularmente na atualização dos mecanismos de identificação de ataques em si. Entretanto, não contempla em detalhes a questão do desempenho do mecanismo de detecção de intrusão implementado. Além disso, a fonte de dados DARPA utilizada na implementação parcial do modelo, não apresentou uma grande variedade de ataques de conteúdo. Arquivos de centenas de megabytes de tráfego de rede apresentaram em geral menos de uma dezena de ataques baseados em conteúdo; a grande maioria dos ataques encontrados baseava-se em cabeçalho. Este fato restringiu a realização de maiores análises

sobre a natureza das ações práticas de tentativas de invasão, assim como limitou os resultados obtidos.

Outras pesquisas que estão sendo atualmente realizadas para compor o sistema de contramedidas do NIDIA [36, 41] podem fazer uso e estender o mecanismo de detecção proposto por esta dissertação, a fim de avaliar o grau de risco fornecido e, então, aperfeiçoar ações preventivas e/ou corretivas contra os possíveis atacantes.

4.3 Trabalhos Futuros

Apresenta-se como proposta para trabalhos futuros as seguintes sugestões:

- Elaborar o detalhamento do modelo de atualização automática, contemplando as interfaces necessárias entre a SAARA e SDI´s multiagentes, bem como as interfaces de comunicação entre a Agência Central de Segurança e as fontes de dados;

- Desenvolver um conjunto de palavras-chaves relacionadas a ataques em outros ambientes operacionais, como Windows, e aprofundar o estudo de palavras-chaves no ambiente Unix;

- Implementar o módulo de otimização do banco de palavras-chaves por meio do uso de uma rede neural PCA;

- Implementar os módulos de identificação de ataques baseados em cabeçalho;

- Implementar os módulos de identificação de ataques de anomalia;

- Utilizar outra plataforma de desenvolvimento de sociedades de agentes, bem como outras arquiteturas de redes neurais, para verificar se há incremento no desempenho do sistema;

- Utilizar outras bases de dados, para fazer um comparativo com as fontes de dados do Projeto DARPA.

APÊNDICE A — Ataques de rede

Um dos princípios fundamentais de segurança em um ambiente computacional refere-se à segurança física. Se o atacante tiver acesso direto à máquina que contém informações que deveriam ser protegidas, torna-se questão de tempo ter dados roubados, adulterados ou destruídos. Caso a segurança física esteja garantida, a rede de computadores figura como ponto central de preocupação. Ela será alvo de muitas investidas em busca de vulnerabilidades que várias vezes passam desapercebidas pelos administradores de rede.

Existem diversos tipos de ataques de rede envolvendo diferentes protocolos de comunicação. Mas a consolidação do TCP/IP como padrão de fato na comunicação entre redes [42], como é o caso da Internet, torna o estudo dos incidentes de segurança que utilizam esta arquitetura de extrema importância. Neste contexto, a comunicação entre as máquinas utiliza troca de pacotes através da rede. Logo, as técnicas de ataque também usam, de alguma maneira, essa unidade de intercâmbio de informações.

A composição de um pacote TCP/IP é derivada dos protocolos da camada de rede e de transporte que compõem esta arquitetura. Restringindo-se apenas ao protocolo de rede IP e ao de transporte TCP[27] observa-se o seguinte esquema:

- Um **datagrama** IP é composto de um cabeçalho contendo informações de controle e um campo de dados para encapsular o protocolo de transporte (TCP);

[27] O protocolo não orientado à conexão UDP também serve de suporte a uma gama de ataques.

- Um pacote TCP, de forma similar, possui duas partes: o cabeçalho que possui informações de controle e o campo de dados (*PAYLOAD*) que transporta as informações propriamente ditas que devem ser trocadas entre dois elementos da rede.

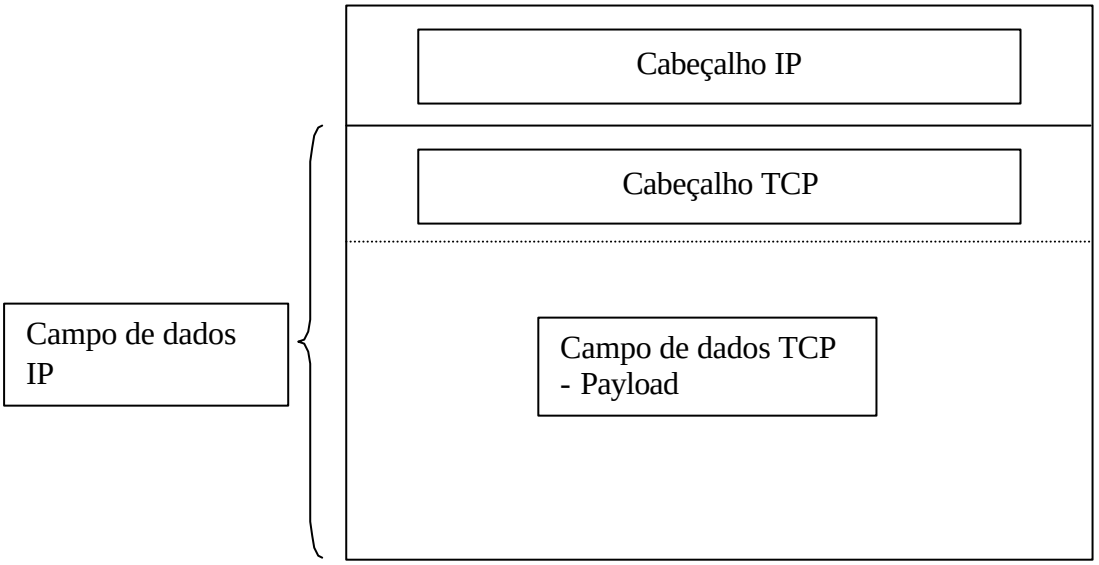

Figura A.1 Simplificação de um datagrama IP contendo um pacote TCP no seu campo de dados.

Com base na estrutura do pacote TCP/IP mostrado na Figura A.1 é possível classificar os ataques de rede em duas grandes categorias: ataques baseados em cabeçalho e ataques baseados em conteúdo.

A.1 Ataques baseados em cabeçalho

Os cabeçalhos de rede e de transporte podem ser utilizados basicamente para as seguintes ameaças:

- Mapeamento da Rede (*Probe*) - antes de iniciar ataques, *hackers* experientes realizam uma fase de reconhecimento para identificar potenciais vulnerabilidades na possível rede da vítima. Através desta sondagem o atacante pode identificar quais são as máquinas ativas e seus respectivos serviços disponíveis e desta forma realizar uma investida qualitativa em cima de alvos realmente existentes. Este mapeamento é composto de duas fases: a varredura da faixa de endereços IP para descobrir quais máquinas são respondentes e a varredura de portas para identificar quais serviços estão ativos nestas máquinas [33].

- Negação de Serviço (*Denial of Service* – DOS) - segundo [29], “um ataque DOS interrompe ou nega completamente serviço a usuários legítimos, redes, sistemas ou outros recursos”. São variedades de DOS ataques que abusam de ações legítimas (ex: envio de pacotes ao extremo para ocasionar alguma falha no sistema alvo) e ataques que utilizam pacotes com cabeçalhos mal formados que causam perturbações no funcionamento da pilha TCP/IP da máquina que está recebendo o pacote [21]. Estas duas classes de DOS são explicadas com mais detalhes nas seções seguintes.

A.1.1 Negação de serviço através de pacotes mal-formados

Na arquitetura TCP/IP há regras quanto às informações que trafegam nos cabeçalhos IP e TCP. Como existem diferentes implementações da pilha TCP/IP em diferentes sistemas operacionais, determinados pacotes, cujos cabeçalhos não seguem as regras definidas para a realização de um tráfego considerado normal, poderiam explorar alguma vulnerabilidade decorrente de um erro de programação na pilha. Isto se deve ao fato de algumas implementações da pilha TCP/IP não preverem o que fazer com um pacote com cabeçalho inesperado e/ou malformado. São exemplos de ameaças com esta característica ataques de recusa de serviço (DOS) que utilizam apenas um pacote.

Seguem abaixo alguns ataques de recusa de serviço que possuem esta particularidade:

Ataque Land – ocorre quando o mesmo par IP/Porta aparece tanto no endereço da máquina origem quanto da máquina destino no cabeçalho do pacote. Segundo [43], “algumas versões de sistemas operacionais, como o *Windows* 95 e *Windows* NT *Workstation* com *Service Pack* 3, têm suas máquinas travadas quando sofrem esse tipo de ataque pela porta 139 e as distribuições mais antigas do Linux apresentam queda de desempenho quando várias cópias deste ataque são disparadas contra ele”.

Ataque Xmas Tree – o cabeçalho TCP possui os seguintes bits de controle (flags) para gerenciar o fluxo das conexões:

FIN – finaliza a conexão enviando dados

SYN – Inicia uma conexão

PSH – envia dados para aplicação

RST – reseta a conexão

ACK – flag de confirmação de recebimento

URG – urgent pointer

Um tráfego malicioso pode ser obtido forjando pacotes que possuem determinadas combinações de flags que não deveriam estar presentes em um tráfego considerado normal. Por exemplo, antes de iniciar realmente a troca de informações, duas máquinas devem estabelecer primeiramente uma conexão. Para isso devem passar por um processo de negociação denominado *handshake* (Figura A.2).

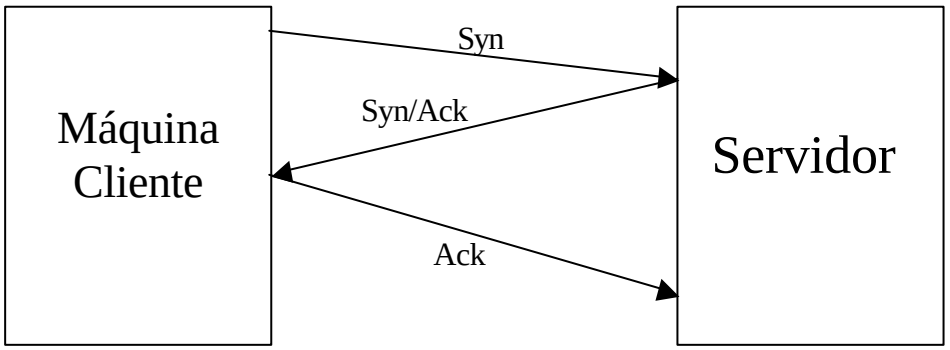

Figura A.2 Processo de *handshake*.

A máquina cliente envia um pacote com o flag SYN ligado para o computador servidor indicando que tem interesse de iniciar uma conexão. O servidor responde através de um pacote com os flags SYN e ACK ativos, indicando a sua

disponibilidade de prosseguir com o processo. Por fim, o cliente envia um pacote com o flag ACK para então estabelecer a conexão. Maiores detalhes sobre o funcionamento do processo de *handshake* e sobre a utilização dos demais flags podem ser encontrados em [45].

Deve ser considerada suspeita qualquer combinação dos bits de controle que não sejam coerentes com as regras do protocolo TCP. Assim, por exemplo, um pacote forjado que tenha ao mesmo tempo os flags SYN e FYN ligados pode causar uma negação de serviço no computador para o qual foi enviado. Esta combinação não deveria normalmente acontecer, porque seria uma tentativa de simultaneamente iniciar e terminar uma conexão.

O ataque **XMAS Tree** faz uso destes pacotes fora de especificação[28] para causar instabilidade nas máquinas, forjando cabeçalhos que tenham todos os bits de flags ligados, podendo desta forma englobar várias combinações não esperadas dos bits de controle.

A.1.2 Negação de serviço através do abuso de ações legítimas

Outros ataques de cabeçalho utilizam um grande número de pacotes considerados normais para gerar uma situação de DOS em uma máquina alvo. Em linhas gerais, submete-se à vítima uma enorme quantidade de pacotes em um pequeno espaço de tempo para que a mesma fique tão ocupada reservando recursos para estas conexões que acabe recusando requisições de usuários legítimos.

[28] Segundo [33], pacotes fora de especificação são aqueles que têm grupos inválidos ou incomuns de flags TCP ativos.

Como exemplo deste tipo de DOS tem-se o ataque **SynFlood**. Este ataque utiliza uma técnica denominada *IP Spoofing* que pode ser entendida como uma espécie de falsidade ideológica em relação ao endereço IP do host do atacante, ou seja, o *hacker* manipula o cabeçalho do pacote alterando o IP origem para outro diferente do verdadeiro.

No ataque **SynFlood** o atacante envia inúmeros pacotes com o flag SYN ativado. Tais pacotes têm o seu endereço de origem alterado para números IPs não respondentes (*ip spoofing*). Desta forma, o processo de *handshake* não é completado e o computador da vítima manterá recursos alocados para estas conexões até que todos os recursos da máquina alvo sejam consumidos e a mesma recuse requisições de usuários reais (Figura A.3).

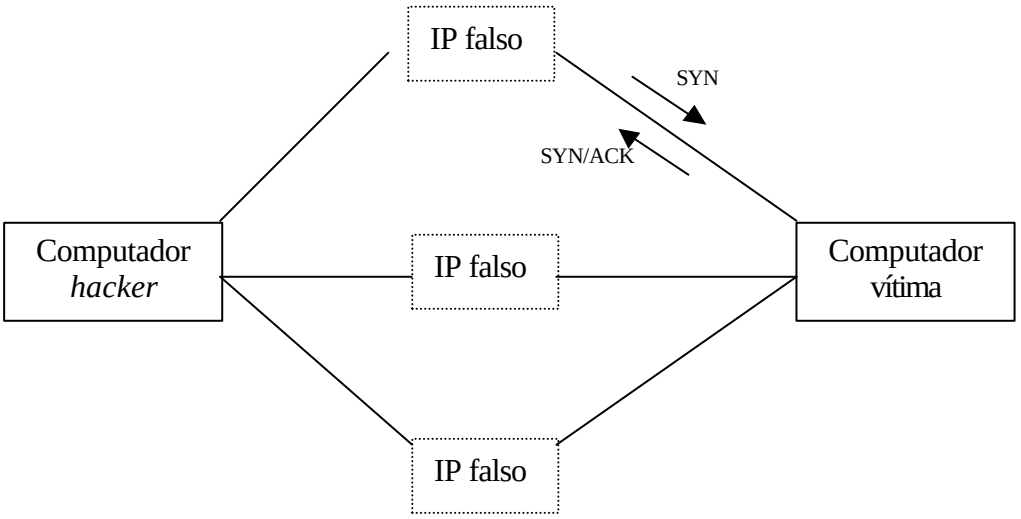

Figura A.3 DOS SynFlood - computador atacante envia requisições de início de conexão através de pacotes com endereços IPs origem falsos.

A.2 Ataques baseados em conteúdo

No ataque baseado em cabeçalho não se obtém acesso à máquina, porém pode se conseguir torná-la inoperante ou prejudicar algum serviço remoto. Por outro lado, no ataque baseado em conteúdo o atacante visa conseguir acesso ao computador da vítima como superusuário. Logo, para efetuar uma invasão um conjunto de ações mais elaboradas deve ser realizado e a parte do pacote TCP a ser usada para este fim é o conteúdo do campo de dados TCP (*payload*).

O campo de dados é utilizado para enviar as informações úteis entre os computadores cliente e servidor depois que a sessão de comunicação tem início (pós-handshake). Estes dados podem ser comandos de estímulo do cliente, respostas do servidor, etc. Diferentemente dos ataques de cabeçalho, o objetivo principal neste caso não é a interrupção dos serviços do servidor, mas conseguir acesso privilegiado para capturar/destruir/adulterar informações importantes, implantar um software para atacar outras máquinas utilizando a vítima como um "zumbi", utilizar o poder de processamento deste servidor invadido para atividades não autorizadas etc.

Pressupondo-se que o atacante não possui acesso físico à vítima, então o acesso através da rede é a maneira pela qual os ataques terão início. Caso essa fase tenha sucesso e o *hacker* adquira um acesso de usuário normal à máquina alvo, comumente ele necessita fazer agora uma escalação de privilégios para obter acesso de superusuário (como o usuário administrador do Windows 2000/NT ou o root do Linux). Segundo [29], "acesso remoto é definido como ganhar acesso via

rede (por exemplo, um serviço ouvindo) ou outro canal de comunicação". Consideram-se ataques de acesso local o conjunto de ações para se obter a escalação de privilégios, se necessário.

Dentre as formas de se efetivar esses ataques utilizando o *payload* destacam-se:

Técnicas de adivinhação de senha – a forma clássica de impor segurança a determinado serviço ou sistema é através da autenticação utilizando o par usuário/senha. Senhas fracas definidas por usuários podem ser descobertas com métodos automatizados e podem garantir pelo menos o acesso remoto inicial.

Ataques orientados a dados (*data driven attack*) – conforme [1] "o desenvolvimento de sistemas é uma das áreas mais afetadas pelos aspectos de segurança. Muitos dos problemas de segurança existentes hoje não são físicos, nem de procedimentos, mas sim devido a erros de programação ou arquiteturas falhas". Erros na implementação de programas podem ocasionar resultados imprevistos quando tais programas são submetidos a dados especialmente preparados para explorar estas vulnerabilidades. Para comprometer a segurança de um sistema utilizando ataques dirigidos por dados, comumente são utilizadas as técnicas de validação de entradas (*input validation*) ou de estouro de buffer (*buffer overflow*) que serão explicadas nas Seções A.2.1 e A.2.2.

A.2.1 Ataques de Validação de Entradas

Conforme orientação obtida em [29], este tipo de ataque pode ocorrer quando um dos seguintes descuidos em relação a uma programação considerada segura ocorrem:

1. Um programa não consegue reconhecer uma entrada sintaticamente incorreta;
2. Um módulo aceita uma entrada estranha;
3. Um módulo não consegue tratar campos de entrada ausentes;
4. Ocorre um erro de correlação campo-valor.

A lista acima representa nada mais do que “bugs” de implementação em determinados aplicativos que podem ser usados para explorar alguma vulnerabilidade no sistema. Um exemplo clássico deste tipo de problema de segurança é o script PHF[29]. Em algumas versões, este *script* não validava seus dados de entrada e desta forma permitia que, através do uso de metacaracteres[30], um usuário malicioso pudesse executar comandos com os mesmos privilégios da conta de usuário que estivesse executando o servidor *web*.

```
http://www.vitima.com/cgi-bin/phf?Qalias=x%0a/bin/cat/%20/etc/passwd
```

Figura A.4 Exploração de vulnerabilidade do script PHF. Os valores expressos em hexa %0a e %20 significam nova linha e espaço em branco respectivamente.

[29] PHF (*Phone Book Script* – Script de Lista Telefônica) é um script CGI de exemplo que acompanhava versões de alguns servidores *web* e servia para exemplificar o uso de formulários em uma pequena aplicação de lista telefônica.

[30] Metacaracteres são reservados para uso especial, como na representação de caracteres não visíveis tipo “nova linha”, “tabulação”, “fim de linha” etc.

Este script CGI quando submetido ao caractere de nova linha (\n ou %0a em hexa) permitia que uma lista de comandos pudesse ser executada. Na Figura A.4 o comando *cat* é chamado para mostrar o conteúdo do arquivo *passwd* que contém os usuários do sistema. Outras explorações de ataques de validação de entrada podem até conceder acesso de superusuário como descrito em [29].

Deduz-se deste exemplo que como este tipo de vulnerabilidade decorre da ação de programadores pouco zelosos pelas boas práticas de programação segura, não é surpreendente que esta categoria de incidentes apareça com freqüência.

A.2.2 Estouro de buffer

Estouro de buffer (*buffer overflow*) também é um problema de segurança oriundo de erros de programação. No entanto, trata-se de uma técnica mais refinada de invasão onde o atacante manipula o conteúdo do buffer de um programa para obter acesso privilegiado. Seguem abaixo alguns detalhes e conceitos básicos de como esta vulnerabilidade pode ser explorada.

Primeiramente um buffer, neste contexto, é uma estrutura muito comum utilizada em linguagens de programação e que consiste em um bloco contíguo na memória para armazenar dados homogêneos, ou seja, refere-se a um vetor[31]. Um buffer pode ser estático (alocado no segmento de dados quando o programa é carregado) ou dinâmico (alocado em tempo de execução na pilha).

[31] Um vetor serve para armazenar múltiplas instâncias de dados do mesmo tipo.

Quando um processo está na memória ele obedece a uma estrutura dividida em 3 regiões (Figura A.5), conforme explicado a seguir:

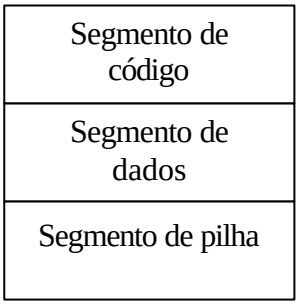

Figura A.5 Estrutura da memória. Adaptado de [2].

- Segmento de código – área de tamanho fixo e que comporta as instruções do programa e dados somente de leitura;

- Segmento de dados – espaço onde são armazenadas as variáveis estáticas;

- Segmento de Pilha – área de memória contígua que dá suporte às técnicas de modularização de programas através de subrotinas[32] (procedimentos ou funções). A pilha é usada para passar parâmetros, alocar variáveis dinâmicas e retornar valores das funções.

[32] Com o uso da pilha viabiliza-se o desvio do fluxo de execução de um programa através de subrotinas e não através de instruções tipo “jump”. Após o término da subrotina o controle do fluxo de execução retorna ao ponto seguinte à chamada da subrotina. O endereço de retorno também é armazenado na pilha.

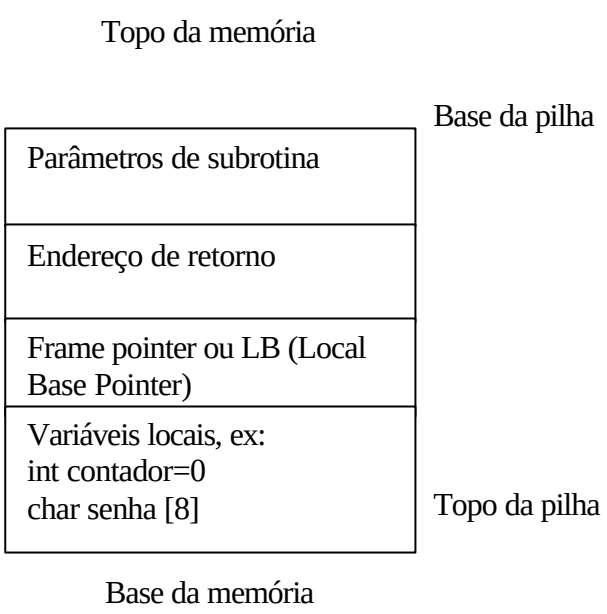

Figura A.6 Estrutura da pilha. Adaptada de [33].

A Figura A.6 mostra uma estrutura de pilha (*stack frame*) que é colocada na pilha cada vez que uma subrotina é chamada. Adota-se neste modelo a implementação Intel, onde a pilha cresce em direção aos endereços mais baixos de memória [46].

O ataque de estouro de *buffer* explora vulnerabilidades das estruturas apresentadas anteriormente. Este tipo de ataque é passível de ocorrer quando um programa é escrito sem a preocupação da checagem dos limites dos vetores e algum dos *buffers* recebe uma quantidade de dados superior àquela para qual ele foi definido. Neste caso, o excedente de dados vai ser escrito em uma área de memória adjacente àquela ocupada pelo *buffer*. Quando o vetor está alocado no segmento de pilha, esta sobra de dados pode ser especialmente preparada para conter código malicioso e também para sobrescrever o valor de retorno da subrotina para algum endereço dentro do próprio *buffer*. Desta forma, ao invés do programa seguir o seu fluxo normal de execução após o término da subrotina, o novo código implantado

pelo atacante é que realmente é executado. Maiores detalhes sobre os ataques de estouro de buffer podem ser encontrados em [2].

A.3 Fase pós-ataque

Uma vez que uma fragilidade na segurança de um ambiente seja descoberta, permitindo algum dos tipos de ataques baseados em conteúdo mostrados, o atacante pode se atrever na execução de algum código como o apresentado na Figura A.4. Mas sem dúvida a opção que garante maior flexibilidade é conseguir ganhar acesso privilegiado de linha de comando (*shell*) na máquina da vítima. A partir de então quaisquer comandos poderiam ser realizados. Isto acontece de forma recorrente quando da exploração de vulnerabilidades de aplicativos que possuem o bit SUID[33] ativo e são pertencentes ao *root* (superusuário). Assim, neste caso, se a exploração resultar na criação de um *shell* privilegiado para o atacante, este terá a partir de então total domínio sobre o ambiente invadido.

Depois que o invasor entra no sistema e já obtém, de alguma forma, direitos de superusuário, ele, em geral, toma atitudes como [30]:

1) Toque da morte – o atacante simplesmente destrói todas as informações presentes na máquina da vítima.

```
[ / ] #  rm   –fr   / *
```

Figura A.7 Destruição total de um sistema Linux através da remoção recursiva de todos os arquivos e diretórios a partir da raiz do sistema de arquivos.

[33] O bit SUID quando setado permite que outros usuários executem um aplicativo com as permissões do seu proprietário.

2) Garantir alguma forma de acesso ao computador alvo, independente da vulnerabilidade que foi inicialmente explorada. Com isso, o intruso ainda poderá realizar atividades hostis na vítima, mesmo que as devidas correções tenham sido feitas nas debilidades originais do sistema. O atacante pode, por exemplo, adicionar um novo superusuário ao sistema. Na Figura A.8 o novo usuário "*hacker*" não possui senha e tem *UserId* igual a zero. Isso equivale a adicionar um usuário sem senha no grupo administradores na família de Sistemas Windows NT/2000. Outra forma de instalar um novo meio de acesso ao computador que tenha sido invadido é criar uma *backdoor*, conforme explicado na Seção A.3.1.

```
root:swNq99dWPTM9U:0:0:root:/root:/bin/bash
bin:*:1:1:bin/bin:
.
.
.
nobody:*:99:99:Nobody:/:
hacker::0:0:hacker:/:/bin/bash
```

Figura A.8 Criação de um usuário privilegiado no arquivo /etc/passwd.

A.3.1 Backdoor (Porta dos fundos)

Uma *backdoor* pode ser compreendida como uma "passagem secreta" implantada em algum software ou sistema que permita que o atacante tenha uma facilidade para ter acesso privilegiado à máquina alvo sem que seja alardeada a sua presença ao administrador do sistema.

Segundo [30], se um usuário malicioso tiver acesso a um terminal logado pelo *root*, ele pode fazer uma cópia do *shell* do superusuário para o seu diretório e modificar os direitos de execução desta cópia com o comando *chmod* para que qualquer um possa executá-la (atributo “a”) como se fosse o próprio administrador (atributo “s”) (Figura A.9).

```
[ /root ] #  cp /bin/bash      /home/hacker/meuSuperShell
[ /root ] #  chmod a+s         /home/hacker/meuSuperShell
```

Figura A.9 Criação de um shell privilegiado. Adaptado de [30].

Considerando que esse usuário já conseguiu acesso através de uma conta comum[34] (evidenciado pelo *prompt* $), pode agora executar o *shell* obtido para fazer a sua escalação de privilégios (evidenciado pelo *prompt* #) e ter então todo o sistema sob o seu comando (Figura A.10).

```
[ hacker ]$ /home/hacker/meuSuperShell
[ hacker ]# _
```

Figura A.10 Escalação de privilégios. Adaptado de [30].

Também é possível que o *hacker* se conecte remotamente ao sistema da vítima sem precisar de uma conta de usuário legítima e sem fazer uso da vulnerabilidade inicial que foi explorada [33].

Conforme orientação contida em [30] uma forma de criar uma fraqueza na segurança de sistemas Linux é implantar uma *backdoor* através da manipulação dos

[34] Obtida através da descoberta da senha através de algum método de decifração, através de um ataque de acesso de remoto etc.

arquivos */etc/services* e */etc/inetd.conf*. Aquele tem como função mapear os serviços existentes em uma máquina a um número de porta. Já o arquivo */etc/inetd.conf* tem como responsabilidade determinar qual *daemon* (programa) deve ser executado quando um determinado serviço for solicitado.

Assim quando um atacante conseguir acesso de *root,* ele pode adicionar um novo serviço ao arquivo */etc/services* (Figura A.11) e associá-lo a um *shell* no arquivo *inetd.conf* (Figura A.12).

```
ServicodeInvasao          7010/tcp     # Porta para acesso hacker
```

Figura A.11 Novo serviço adicionado ao arquivo */etc/services*. Adaptado de [30].

```
ServicodeInvasao  stream  tcp  nowait  root  /bin/sh  /bin/bash –i
```

Figura A.12 Linha adicionada ao arquivo */etc/inetd.conf* que indica que o novo serviço deve solicitar a execução de um shell interativo.

Feitas estas modificações, executa-se o comando *kill –HUP inetd* para que as alterações realizadas tenham efeito e o novo serviço esteja disponível. Na Figura A.13, tem-se uma chamada ao serviço da porta 7010 através de um telnet e, como resultado, obtém-se um shell privilegiado[35], sem necessitar informar nenhuma senha.

[35] Considerando-se que os serviços contidos no /etc/services são executados com direitos de root.

```
[  hacker ]$ telnet maquina_vitima 7010
Trying <IP_vitima] ...
Connected to maquina_vitima.
Escape character is `^]`.
[ / ] #
```

Figura A.13 Ilustração da invasão em um computador alvo através de um *backdoor* anteriormente implantado. Adaptado de [30].

REFERÊNCIAS

[1] ALBUQUERQUE, R.; RIBEIRO, Bruno. **Segurança no desenvolvimento de software: como garantir a segurança do sistema para o seu cliente usando a ISO/IEC.** Rio de Janeiro: Campus, 2002.

[2] ALEPHONE. **Smashing the Stack for Fun and Profit.** Disponível em: <http://www.insecure.org/stf/smashstack.txt>. Acesso em 23 nov. 2001.

[3] BALASUBRAMANIYAN, Jai; GARCIA-FERNANDEZ, Jose; ISACOFF, David; SPAFFORD, E. H.; ZABONI, Diego. **An Architecture for Intrusion Detection using Autonomous Agents.** Department of Computer Sciences, Purdue University, Coast TR 98-05, 1998. Disponível em: <http://www.cd.purdue.edu/coast/coast/coast-library.html>. Acesso em 06 jun. 2002.

[4] BIGUS, Joseph P.; BIGUS, Jennifer. **Constructing Intelligent Agents using Java.** 2. ed. Wiley Computer Publishing, 2001.

[5] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The unified modeling language user guide**. Addison-Wesley, Longman. 1999.

[6] BT INTELLIGENT AGENT RESEARCH. **Zeus Toolkit**. Disponível em: http://www.btexact.com/projects/agents/zeus/. Acesso em 11 maio 2002.

[7] CANNADY, James. **Artificial Neural Networks for Misuse Detection.** Disponível em: http://csrc.nist.gov/nissc/1998/proceedings/paperF13.pdf. Acesso em: 18 out. 2001.

[8] CANSIAN, Adriano Mauro. **Desenvolvimento de um Sistema Adaptativo de Detecção de Intrusos em Redes de Computadores.** São Carlos, 1997. Tese (Doutorado em Física Aplicada, sub-área Física Computacional), USP, 1997.

[9] CARUSO, C.; STEFFEN, F. **Segurança em informática e de informações.** 2. ed. São Paulo: Editora SENAC São Paulo, 1999.

[10] CERT – COMPUTER EMERGENCY RESPONSE TEAM. Disponível em <http://www.cert.org/stats/cert_stats.html>. Acesso em 10 de agosto 2002.

[11] COLLIS, Jaron; NDMU, Divine. **The Zeus Agent Building Toolkit, Zeus Realisation Guide,** Intelligent Systems Research Group, BT Labs, v 1.0, maio 1999.

[12] COLLIS, Jaron; NDMU, Divine. **The Zeus Agent Building Toolkit, Zeus Technical Manual,** Intelligent Systems Research Group, BT Labs, v 1.0, set. 1999.

[13] COMER, D. E. **Computer Networks and Internet.** 2. ed. Prentice Hall, 1999.

[14] CONECTIVA. **Fundamentos de Administração de Sistemas.** 1 ed. 2001.

[15] CROSBIE, M.; SPAFFORD, E. H. **Defending a Computer System using Autonomous Agents.** Department of Computer Sciences, Purdue University, 1995. (Relatório Técnico CSD-TR-95-0022;Coast TR 95-02). Disponível em: http://www.cs.purdue.edu/homes/spaf/tech-reps/9522.os. Acesso em 05 jan. 2002

[16] DEITEL, H. M.; DEITEL, P, J. **Java, como programar.** Porto Alegre: Bookman, 2001.

[17] FUJII, Keita. **Java package for libpcap.** Disponível em: http://www.goto.info.wasesa.ac.jp/~fujii/jpcap/. Acesso em 02 abr. 2002

[18] HAYKIN, Simon ; **Neural Networks - A Comprehensive Foundation**. Prentice Hall - Upper Saddle River, New Jersey, 2nd ,1999.

[19] HEIRBERLEIN, L. T. **A network security monitor.** In Proceedings of the 1990 Symposyum on Research in Security and Privacy, pages 296-304, Oakland, CA, May 1990, IEEE.

[20] JAGANNATHAN, R. et al. **Next-generation Intrusion Detection Expert System(NIDES).** Technical report. Computer Science Laboratory, SRI International, Menlo Park, California 94025, March 1993.

[21] KENDALL, K. **A database of Computer Attacks for the Evaluation of Intrusion Detection Systems.** Massachusetts, 1999. Dissertação (Mestrado em Computação), Massachusetts Institute of Technology, 1999.

[22] LBNL´S NETWORK RESEARCH GROUP**. Tcpdump.** Disponível em: <http://ee.lbl.gov/>. Acesso em 15 jun. 2001.

[23] LIMA, Christiane F. L. **Agentes Inteligentes para Detecção de Intrusos em Redes de Computadores.** São Luís, 2002. Dissertação (Mestrado em Engenharia de Eletricidade), Universidade Federal do Maranhão - UFMA, 2002.

[24] LIPPMANN, Richard. P.; CUNNINGHAM, Robert K. **Improving Intrusion Detection Performance Using Keyword Selection and Neural Networks.** Disponível em: http://www.raid-symposium.org/raid99/PAPERS/Lippmann1.pdf.>. Acesso em: 18 jun. 2002.

[25] ——. **Guide to Creating Stealthy Attacks for the 1999 DARPA Off-Line Intrusion Detection Evaluation**. MIT Lincoln Laboratory Project Report IDDE-1, June 1999.

[26] LIPPMANN, Richard. P. et al. **Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation.** Disponível em: <http://www.ll.mit.edu/IST/ideval/pubs/1999/Evaluating_IDs_DARPA_1998.pdf>. Acesso em 10 set. 2001.

[27] LOPES, E. N.; SILVA FILHO, M. V.; PEREIRA, R. S. **Hacker Curso Completo.** Rio de Janeiro: Book Expressss, 2002.

[28] MANDIA, K.; PROSISE, C. **Hackers Resposta e ContraAtaque.** Rio de Janeiro: Campus 2001.

[29] MCCLURE, Stuart; SCAMBRAY, Joel; KURTZ, George. **Hackers Expostos: Segredos e soluções para a segurança de redes.** 2. ed. São Paulo: Makron Books, 2000.

[30] MENDES, W. R. **Linux e os hackers - proteja o seu sistema: ataques e defesas.** Rio de Janeiro: Editora Ciência Moderna Ltda., 1999.

[31] MIT LINCOLN LABORATORY. **1998 Trainning Data Attack Schedule**. Disponível em: <http://www.ll.mit.edu/IST/ideval/docs/1998/attacks.html>. Acesso em: 19 abr. 2002.

[32] MÓDULO SECURITY SOLUTIONS. **7ª Pesquisa Nacional sobre Segurança da Informação.** Disponível em : <http://www.modulo.com.br.>. Acesso em: 19 mar. de 2002

[33] NORTHCUTT, S.; et al. **Intrusion Signatures and Analysis.** New Riders, 2001.

[34] NORTHCUTT, S.; NOVAK J. **Network Intrusion Detection - An Analyst´s Handbook.** 2. ed. New Riders, 2000.

[35] NOURIE, Dana; MCCLOSKEY, Mike. **Regular Expressions and the Java™ Programming Language**. Disponível em: <http://developer.java.sun.com/developer/technicalArticles/releases/1.4regex/>. Acesso em: 20 maio 2002.

[36] OLIVEIRA, Antonio Alfredo Pires. **Proposta de um servidor armadilha, baseado em agentes inteligentes, para investigação de atos suspeitos em sistemas de computadores.** Dissertação (Mestrado em Engenharia da Eletricidade) – UFMA, Maranhão (em fase de elaboração).

[37] PTACEK, T. P.; NEWSHAM, T. N. **Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection.** Disponível em: http://www.robertgraham.com/mirror/Ptacek-Newsham-Evasion-98.html. Acesso em: 05 fev. 2002.

[38] RANUM, M. J. **An Internet Firewall**, proceedings of World Conference on Systems Management and Security, 1992. Disponível em: <ftp://decuac.dec.com/pub/docs/firewallfirewall.ps>. Acesso em: 12 jun. 2000.

[39] ROESCH, Martin; GREEN, Chris. **SNORT Users Manual – SNORT Release: 1.9.1**. Disponível em: <http://www.snort.org/docs/writing_rules/>**.** Acesso em 19 mar. 2002.

[40] RYAN, Jake et al. **Intrusion detection with neural networks.** In Advances in Neural Information Processing Systems 10, May 1998.

[41] SANTOS, Glenda de Lourdes Ferreira. **Um Agente Inteligente Controlador de Ações do Sistema.** Dissertação (Mestrado em Engenharia da Eletricidade) – UFMA, Maranhão (em fase de elaboração).

[42] SOARES, L. F. G.; LEMOS, G.; COLCHER, S. **Redes de computadores: das LANS, MANS e WANS às redes ATM.** 2. ed. Rio de Janeiro: 1997.

[43] SOUSA, Bruno. F. **Agentes Inteligentes para a Detecção e Tratamento de Ataques a Redes de Computadores.** São Luís, 2001. Monografia (Graduação em Ciência da Computação), Universidade Federal do Maranhão – UFMA, 2001.

[44] SUN MICROSYSTEMS, **JAVA™ 2 SDK, Standard Edition Documentation.** Disponível em: <http://java.sun.com/j2se/1.4.0/docs/index.html >. Acesso em 05 ago. 2002.

[45] TANENBAUM, Andrew S. **Redes de Computadores.** 5. ed. Rio de Janeiro: Campus, 1997.

[46] ——. **Organização estruturada de computadores**. 3 ed. Rio de Janeiro: Editora Prentice Hall do Brasil Ltda, 1992.

[47] THE FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS. **FIPA Interaction Protocol Library Specification.** Disponível em: <http://www.fipa.org/specs/fipa00025/XC00025E.html>. Acesso em: 10 abr. 2002.

[48] UNIVERSITY OF STUTTGART. **Stuttgart Neural Network Simulation (SNNS) 4.2 for MS-Windows.** Disponível em: <http://www-ra.informatik.uni-tuebingen.de/SNNS/>. Acesso em 18 jul. 2002.

[49] WEBSTER, Seth E. **The Development and Analysis of Intrusion Detection Algorithms.** Massachusetts, 1998. Dissertação (Mestrado em Computação), Massachusetts Institute of Technology, 1998.

[50] WEISS, Gerhard; **Multiagent Systems- A Modern Approach to Distributed Artificial Intelligence**, The MIT Press - Cambridge, Massachusetts, London, England, 1999