UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

Uma Proposta de Uso da Arquitetura Trace como um Sistema de Detecção de Intrusão

por

EDGAR ATHAYDE MENEGHETTI

Dissertação submetida à avaliação,
como requisito parcial para a obtenção do grau de
Mestre em Ciência da Computação

Prof. Dr. Liane Margarida Rockenbach Tarouco
Orientadora

Porto Alegre, julho de 2002.

CIP — CATALOGAÇÃO NA PUBLICAÇÃO

Meneghetti, Edgar Athayde

Uma Proposta de Uso da Arquitetura Trace como um Sistema de Detecção de Intrusão / por Edgar Athayde Meneghetti. — Porto Alegre: PPGC da UFRGS, 2002.

105 f.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR–RS, 2002. Orientadora: Tarouco, Liane Margarida Rockenbach.

1. Detecção de intrusão. 2. Redes de computadores. 3. Segurança. 4. Gerência de redes de computadores. I. Tarouco, Liane Margarida Rockenbach. II. Título.

"The combination of space, time and strength that must be considered as the basic elements of this theory of defense, makes this a fairly complicated matter. Consequently, it is not easy to find a fixed point of departure"

— On War

Carl Von Clausewitz

Agradecimentos

Este trabalho é resultado de um esforço pessoal, mas que deve muito a várias pessoas. Alguns pelas palavras de incentivo, quando o mestrado era apenas uma idéia. Outros pelas várias horas de discussão em torno de PTSL, Traços de protocolos, IDS, etc. Outros ainda pelo apoio nas horas mais difíceis. Obrigado a todos.

À minha orientadora, Liane Tarouco, obrigado pelas dicas e sugestões, tanto para esta dissertação como para minha vida profissional.

Um agradecimento especial à minha companheira de todos os momentos, Márcia Notare. O mestrado passou a ter um sabor especial depois de ti.

Ao Luciano Gaspary, obrigado por todo o apoio e incentivo nesta jornada. É bom constatar que, após este período, passei a ter um grande amigo.

Ao grupo que se criou ao redor da plataforma Trace (e que continua crescendo), obrigado! Que este projeto continue sendo levado adiante!

Sumário

Lista de Abreviaturas

APM	Application Performance Measurement
ASCII	American Standard Code for Information Interchange
ASN.1	Abstract Syntax Notation One
BNF	Bacchus-Naur Form
BPF	BSD Packet Filter
CCITT	Comité Consultatif International Téléphonique et Télégraphique
CGI	Common Gateway Interface
DISMAN	Distributed Network Management
DNS	Domain Name System
FCAPS	Fault, Configuration, Accounting, Performance and Security
FTP	File Transfer Protocol
G-PTSL	Graphical PTSL
HTTP	Hypertext Transfer Protocol
HTTPS	Secure HTTP
ICMP	Internet Control Message Protocol
IEEE	Institute of Electrical and Electronic Engineers
IETF	Internet Engineering Task Force
IIS	Internet Information Server
IP	Internet Protocol
ISO	International Standards Organization
ITU	International Telecommunications Union
MAC	Medium Access Control
MIB	Management Information Base
NFS	Network File System
OID	Object Identifier
PHP	PHP: Hypertext Preprocessor
POP	Post Office Protocol
POSIX	Portable Open System in Unix
PTSL	Protocol Trace Specification Language
RFC	Request for Comments
RMON	Remote Network Monitoring
SMTP	Simple Mail Transport Protocol

SNMP	Simple Network Management Protocol
SQL	Structured Query Language
T-PTSL	Textual PTSL
TCL	Tool Command Language
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Unified Resource Locator
WAN	Wide Area Network

Lista de Figuras

Lista de Tabelas

Resumo

Este trabalho propõe a utilização da arquitetura Trace como um sistema de detecção de intrusão. A arquitetura Trace oferece suporte ao gerenciamento de protocolos de alto nível, serviços e aplicações através de uma abordagem baseada na observação passiva de interações de protocolos (traços) no tráfego de rede. Para descrever os cenários a serem monitorados, é utilizada uma linguagem baseada em máquinas de estado. Esta linguagem permite caracterizar aspectos observáveis do tráfego capturado com vistas a sua associação com formas de ataque. O trabalho mostra, através de exemplos, que esta linguagem é adequada para a modelagem de assinaturas de ataques e propõe extensões para permitir a especificação de um número maior de cenários ligados ao gerenciamento de segurança. Em seguida, é descrita a implementação do agente de monitoração, componente-chave da arquitetura Trace, e sua utilização para detectar intrusões. Esse agente (a) captura o tráfego da rede, (b) observa a ocorrência dos traços programados e (c) armazena estatísticas sobre a sua ocorrência em uma base de informações de gerenciamento (MIB – *Management Information Base*). O uso de SNMP permite a recuperação destas informações relativas à ocorrências dos ataques. A solução apresentada mostrou ser apropriada para resolver duas classes de problemas dos sistemas de detecção de intrusão: o excesso de falsos positivos e a dificuldade em se modelar certos ataques.

Palavras-chave: Detecção de intrusão, redes de computadores, segurança, gerência de redes de computadores.

TITLE: "A PROPOSAL TO USE A TRACE ARCHITECTURE AS AN INTRUSION DETECTION SYSTEM"

Abstract

This work proposes the use of Trace Architecture as an intrusion detection system. The Trace Architecture supports the management of high-level protocols, services and applications through an approach based on passive observation of protocol interactions (traces) in the network traffic. A language based on finite state machine is used to describe scenarios to be monitored. This language allows to characterize noticeable features of the traffic being captured in order to associate it with attack forms. Through examples, this work shows that this language is suitable for attack signature modelling and proposes some extensions to allow the specification of a higher number of scenarios related to security management. Next, the implementation of the monitoring agent (key-component of Trace Architecture) is described as well as its use as an intrusion detection system. This agent (a) captures network traffic, (b) observes the occurrence of the programmed traces and (c) stores statistics about their occurrence in a management information base (MIB). SNMP allows the retrieval of information relative to the attack events in a simple manner. The solution presented here addresses two problems of intrusion detection systems: the great number of false positives and the inability to model some attacks.

Keywords: intrusion detection, computer networks, security, network management.

1 Introdução

O crescimento acelerado dos sistemas de computação interconectados tem levado a uma enorme dependência das pessoas e organizações em relação aos dados armazenados e transmitidos por esses sistemas [STA 95]. A proteção desses recursos e dados aparece como uma necessidade urgente. Os sistemas de detecção de intrusão (conhecidos por IDS - *Intrusion Detection System*) participam dessa proteção ao atuarem como separadores entre indícios de atividade maliciosa e atos que são atividades normais dentro do contexto do sistema de computação.

Para suprir a demanda de sistemas de detecção de intrusão, várias ferramentas foram lançadas no mercado nos últimos anos. Muitas dessas ferramentas são resultado direto de grupos de pesquisa, que utilizam abordagens diversas para detectar intrusões. Muitos IDSs fazem uso de uma técnica conhecida como análise por assinatura [CAM 2001, NOR 2001]. Essa técnica procura, em alguma fonte de informações, pela ocorrência de características de ataques previamente conhecidas. Essas fontes de dados podem ser registros do sistema ou dados coletados diretamente da rede. Outra técnica, usada em conjunto ou não com a descrita anteriormente, é a análise por anomalia [CAM 2001, NOR 2001], que utiliza métodos estatísticos para separar o comportamento normal do não esperado.

Alguns problemas são relatados em relação aos IDSs existentes atualmente, como excesso de falsos positivos, dificuldade para modelar determinadas assinaturas e descarte de pacotes sob alto tráfego [NOR 2001, NFR 2002]. A ocorrência de falsos positivos e falsos negativos está ligada à forma como os IDSs fazem a análise dos dados e fornecem o diagnóstico. Se houver alguma característica nos dados que indique atividade maliciosa ou anômala, então é lançado um alarme. Essa característica procurada nos dados é a assinatura do ataque. Em geral, a assinatura representa apenas o início do processo de ataque ou parte dele, e não uma seqüência de passos que compõe o ataque. Um alarme gerado pela detecção de um indício de ataque não pode ser considerado como um ataque completo e, muitas vezes, nem sequer como um ataque em andamento. Se o ataque puder ser modelado como uma seqüência de passos que o atacante realizaria para comprometer a segurança de um sistema de computação, então o sistema fará a detecção de intrusão com maior exatidão, sinalizando menos falsos positivos e negativos [VIG 2000].

O problema mencionado acima está intimamente relacionado com o pequeno poder de expressão das linguagens disponíveis para modelar assinaturas de ataques. Em geral, uma assinatura preocupa-se em observar campos de um pacote. A observação de muitos pacotes `TCP` com o *flag* `RST` ligado é um exemplo de assinatura usada por diversas ferramentas para detectar um dos tipos de sondagem de porta [DON 98]. No entanto, `TCP RST` não é gerado apenas por uma estação que não possui determinado serviço disponível ao receber uma requisição de conexão; esse mesmo tipo de pacote é usado por uma estação para reiniciar uma conexão em andamento. Como as ferramentas não correlacionam pacotes, não conseguem distinguir entre `TCP RST`s que representam sondagem de portas e os que são usados ao longo de uma conexão convencional, gerando alarmes em ambos os casos.

O descarte de pacotes sob alto tráfego, tais como os gerados por redes operando a 1 Gigabit por segundo, tem sido apontado como um dos maiores problemas enfrentado pela geração atual de IDSs [NFR 2002, ROE 99]. Trata-se de um problema

de escalabilidade das arquiteturas empregadas (arquiteturas centralizadas em várias soluções), que não conseguem analisar em tempo real um volume de dados dessa magnitude. Uma arquitetura distribuída ou hierárquica pode diminuir o impacto do aumento da largura de banda sob o ponto de vista dos IDSs.

Nesse contexto, este trabalho apresenta um agente para detecção de intrusão que faz uma análise baseada em estados (*stateful inspection*) de dados coletados diretamente da rede. Na verdade, o agente é parte integrante da arquitetura Trace, proposta em [GAS 2002], que oferece suporte ao gerenciamento de protocolos de alto nível, serviços e aplicações através de uma abordagem baseada na observação passiva de interações de protocolos (traços) no tráfego de rede. Embora o agente também se preste à monitoração de traços de protocolos com vistas ao gerenciamento de falhas, contabilização e desempenho, este trabalho procura focar o agente como uma ferramenta ligada à segurança (detecção de intrusão).

A linguagem PTSL (*Protocol Trace Specification Language*) faz parte da arquitetura Trace e foi proposta para permitir que gerentes de rede possam definir os traços de protocolos que devem ser monitorados [GAS 2002]. É uma linguagem concebida para ser facilmente assimilada pelo gerente ao mesmo tempo que oferece flexibilidade na modelagem. Uma das contribuições da linguagem é a capacidade que ela oferece para modelar assinaturas de ataque em que seja possível a correlação de pacotes, uma das limitações das soluções existentes. Observou-se, entretanto, que da forma como foi originalmente concebida, a linguagem não permitia a especificação de um conjunto significativo de assinaturas, sendo necessárias algumas extensões à linguagem.

1.1 Trabalhos Relacionados

Os primeiros trabalhos sobre monitoração voltados ao gerenciamento de segurança surgiram no início dos anos 80. Em 1984 foi desenvolvido o primeiro sistema de detecção de intrusão, denominado IDES ou *Intrusion Detection Expert System* pela SRI International. Desde então, uma série de abordagens e ferramentas foi proposta, embora, somente em 1999, tenha-se observado uma verdadeira profusão de ferramentas e soluções para essa área. O capítulo 2 faz uma descrição mais aprofundada dos principais sistemas de detecção de intrusão disponíveis atualmente.

O que se observa na maior parte das soluções mencionadas é a ausência de uma ferramenta que consiga reduzir os falsos positivos de maneira efetiva. Dessa forma, busca-se a modelagem de ataques de uma forma natural, através da descrição de uma seqüência de pacotes (interações) que, quando observada no tráfego da rede, seja um indício real de que um ataque está em andamento (reduzindo sobremaneira o número de alarmes falsos). A simplicidade na descrição desses ataques através de uma linguagem de fácil aprendizado também é importante. As soluções supracitadas falham em ambos os ítens. Além de oferecem suporte limitado à especificação de assinaturas, oferecem linguagens de muito baixo nível (que o gerente acaba ignorando pela sua complexidade).

1.2 Objetivos deste Trabalho

Este trabalho tem como objetivo avaliar o uso da plataforma Trace como um sistema de detecção de intrusão que evidencie melhorias em algumas das falhas encontradas nos IDSs atuais. A análise da linguagem PTSL para este fim é importante neste contexto, e foi realizada através da modelagem de cenários reais de ataque. Como resultado desta análise, uma extensão da linguagem é apresentada, no sentido de aumentar o escopo de ataques passíveis de serem descritos. Para validar o sistema, foi realizada a implementação do agente de monitoração. Cabe ressaltar que buscou-se a criação de uma ferramenta e não apenas um protótipo. Uma metodologia de avaliação do sistema para o fim proposto é apresentada, assim como os testes e resultados.

1.3 Organização do Trabalho

A dissertação está organizada da seguinte forma: no capítulo 2, são apresentados conceitos relativos a sistemas de detecção de intrusão, assim como a nomenclatura da área, taxonomia e sistemas existentes. O capítulo 3 apresenta uma descrição da arquitetura Trace. As extensões propostas à linguagem PTSL são apresentadas no capítulo 4. O capítulo 5 descreve a implementação do agente de monitoração, peça chave no contexto deste trabalho. No capítulo 6, é apresentada uma metodologia de teste para validar o sistema, assim como uma análise de desempenho do agente. O capítulo 7 encerra o trabalho, com considerações finais e perspectivas de trabalhos futuros.

2 Sistemas de Detecção de Intrusão

Como foi mencionado na introdução, os sistemas de detecção de intrusão atuam como protetores ativos ou passivos dos sistemas computacionais, na medida em que identificam a ocorrência de indícios de atividade maliciosa nos mesmos. Este capítulo busca explorar temas correlatos a sistemas de detecção de intrusão. Na primeira seção discute-se, de forma genérica, a segurança em sistemas computacionais. A terminologia adotada na área, embora ainda não completamente padronizada, é apresentada em seguida. Finalmente, são tecidas algumas considerações acerca da classificação dos sistemas de detecção de intrusão.

2.1 Segurança

De uma forma abrangente, Garfinkel e Spafford [GAR 96] definem um sistema computacional seguro como sendo aquele que se comporta da maneira esperada. A observação do comportamento esperado, em comparação com o comportamento apresentado, pode ser entendido como o nível de confiança do sistema e indica o quanto pode-se confiar no seu funcionamento. O comportamento esperado é formalizado dentro da política de segurança do sistema, e regula as metas que este deve cumprir. Dessa forma, geralmente um evento de segurança está relacionado a uma violação de normas ou procedimentos, que dependem do sistema computacional em uso.

Outra definição menciona segurança como sendo a tentativa de minimizar a vulnerabilidade de bens (qualquer coisa de valor) e recursos, sendo vulnerabilidade qualquer fraqueza que possa ser explorada para atacar um sistema [SOA 95]. Uma definição mais rígida de segurança de computadores baseia-se na confidencialidade, na integridade e na disponibilidade dos recursos do sistema [RUS 91]. Confidencialidade requer que a informação seja acessível somente àquelas pessoas autorizadas para tal; integridade requer que a informação permaneça intacta e inalterada por acidentes ou ataques; e disponibilidade requer que o sistema computacional funcione adequadamente, sem degradação de acesso, fornecendo recursos aos usuários autorizados, quando eles assim necessitarem. A confidencialidade pode ser importante para o sucesso comercial ou até mesmo sobrevivência de uma empresa; integridade pode ser importante para um hospital que mantém históricos médicos de seus pacientes e os utiliza para tomadas de decisão em situações críticas. Já a disponibilidade de um recurso pode ser fator determinante no controle de um sistema de tráfego aéreo ou possibilidade de retaliar um ataque militar. Nesta definição, um sistema computacional que não forneça todos os recursos necessários no devido momento, é tratado como não-confiável, uma vez que o conjunto representa os requisitos de segurança.

Souza [SOU 87] cita que o grande objetivo a ser alcançado é dotar os sistemas de computação de, pelo menos, duas características básicas: confiabilidade e disponibilidade. Confiabilidade é definida como sendo a capacidade que um certo sistema tem em responder a uma dada especificação dentro de condições definidas e durante um certo tempo de funcionamento. Desta forma, independente do tipo de falha ocorrida, seja uma ação mal intencionada ou não, o sistema deve continuar comportando-se como especificado [CAM 2001].

O que se pretende mostrar aqui é que, apesar de possuir muitos pontos comuns, a definição de segurança em sistemas de computadores é extremamente flexível e dependente de uma série de fatores e características. Percebe-se que segurança computacional é um ramo de uma área maior, que tem como preocupação a garantia de que os sistemas permaneçam confiáveis e disponíveis. Este é o principal objetivo da grande área chamada tolerância a falhas [CAM 2001].

Vale salientar outras características igualmente importantes no contexto de segurança: privacidade, autenticidade e integridade. Privacidade preocupa-se em garantir acesso autorizado as informações trocadas por pessoas ou programas. Autenticidade trata da possibilidade de identificar sem equívocos a autoria de determinada ação. Integridade preocupa-se em garantir que a informação não seja modificada, intencionalmente ou não.

2.2 Terminologia

Detecção de Intrusão é um campo jovem, e muitos termos ainda não são usados de forma consistente. Até mesmo os termos ataque e intrusão podem ser utilizados de forma incorreta. Existem vários termos usados para representar os diversos métodos para detecção de intrusão. Esta seção busca uniformizar a nomenclatura utilizada neste trabalho.

- *ataque*: ação conduzida por um adversário, o invasor, contra um alvo, a vítima. O invasor realiza um ataque com um objetivo em mente. Da perspectiva de um administrador responsável por manter o sistema, um ataque é uma série de um ou mais eventos que podem ter uma ou mais conseqüências em relação à segurança. Do ponto de vista de um invasor, um ataque é um mecanismo que permite atingir um objetivo;

- *exploit*: processo de usar uma vulnerabilidade para violar uma política de segurança. Uma ferramenta ou um método que pode ser usado para violar uma política de segurança é freqüentemente citado como um *exploit script*;

- *falso negativo*: evento em que o IDS falha ao identificar uma intrusão, quando, de fato, ela ocorreu;

- *falso positivo*: evento em que o IDS identifica como sendo uma invasão, quando, em verdade, nada ocorreu;

- *incidente*: coleção de dados representando um ou mais ataques. Os ataques podem estar relacionados com o atacante, tipo de ataque, objetivos, sites ou tempo;

- *invasor / atacante / intruso*: pessoa que realiza um ataque. Atacante e intruso são sinônimos comuns para invasor. Estas palavras apenas aplicam-se após um ataque ter ocorrido. Um invasor em potencial pode ser referenciado como um adversário;

- *intrusão*: é um sinônimo comum a palavra “ataque”, ou melhor, um ataque bem sucedido;

- *vulnerabilidade*: característica ou a combinação de características que permitem um adversário colocar um sistema em um estado que é contrário aos desejos das pessoas responsáveis por ele e que aumenta a probabilidade ou magnitude de um comportamento anormal do sistema;

- *assinatura*: uma especificação de aspectos, condições, arranjos e inter-relações entre eventos que significam um ataque, uma violação de acesso, outro tipo de abuso ou uma tentativa destes;

- *conexão*: indica um conjunto de pacotes de informações cujos endereços origem-destino são os mesmos. Não possui relação com os termos "orientado a conexão" e "sem conexão", normalmente presentes na literatura especializada;

- *host*: no contexto deste trabalho, significa um computador, com algum sistema operacional, e ao qual foi atribuído um número IP.

Segundo esta taxonomia, o termo detecção de intrusão é utilizado de forma errônea. Intrusão refere-se a incidentes já concretizados, embora a maioria dos autores trate intrusão como um sinônimo de ameaça. Possivelmente o correto seria utilizar-se o termo detecção de ataques.

Como já foi mencionado, ainda existe uma certa falta de uniformidade na terminologia, e isto se deve a falta de maturidade da área. Segundo [HEA 90], uma intrusão é definida como "qualquer conjunto de ações que tentem comprometer a integridade, confidencialidade ou disponibilidade de um recurso computacional", definição que vem ao encontro do que foi mencionado até aqui. Para ser coerente com a terminologia utilizada comumente na literatura, nesse trabalho será utilizado o termo detecção de intrusão englobando incidentes já concretizados, tentativas de ataques, obtenção de informações e ameaças internas e externas.

2.3 Classificação dos IDSs

A classificação dos sistemas de detecção de intrusão tem sido bastante discutida e existem pelo menos duas formas de classificação que são um consenso entre os pesquisadores: classificação pelo método de detecção e pela arquitetura do sistema.

A classificação pelo método de detecção envolve a forma como os dados serão tratados no esforço de detectar a intrusão. Dois grandes grupos de técnicas podem ser citados: técnicas baseadas em comportamento (ou por anomalia) e técnicas baseadas em assinaturas.

Na classificação realizada pela arquitetura, dois fatores são considerados fundamentais: a fonte dos dados que serão trabalhados e a forma pela qual os componentes da ferramenta estão arranjados entre sí. Segundo Mukherjee et al [MUK 94], os sistemas detectores de intrusão podem ser divididos em duas categorias amplas: os baseados em hosts (*host-based*) e os baseados em rede (*network-based*), o que define a fonte dos dados que serão utilizados no IDS. Mais recentemente, sistemas híbridos, que possuem características das duas arquiteturas, tem sido empregados. Segundo [CAM 2001], ainda dentro da classificação por arquitetura, existem os sistemas de detecção de intrusão centralizados, hierárquicos e distribuídos.

2.3.1 Classificação pelo Método de Detecção

A detecção por anomalia é baseada na determinação de comportamento anômalo no uso de recursos do sistema [CAN 97]. Por exemplo, se normalmente um determinado servidor possui poucos acessos externos e, repentinamente, passa-se a observar um volume de tráfego acima do normal, pode-se considerar este comportamento anômalo. A detecção por anomalia tenta quantificar o comportamento usual ou aceitável, enquanto busca indicar outros comportamentos como sendo potencialmente intrusivos.

Por outro lado, a detecção por abuso refere-se a intrusões que seguem um padrão bem definido de ataque e que explora vulnerabilidades no sistema ou nos softwares aplicativos. Tais padrões podem ser descritos previamente e facilmente identificáveis a partir dos dados coletados no alvo monitorado. Boa parte dos IDSs existentes no mercado fazem uso deste método (não exclusivamente), por ser preciso e de implementação mais simples. Por exemplo, situa-se nessa categoria a exploração das falhas de fingerd e sendmail usados no ataque conhecido como Internet Worm [SPA 89]. Esta técnica representa o conhecimento sobre o comportamento impróprio ou inaceitável [SMA 92] e procura localizá-lo diretamente, de maneira oposta à detecção de intrusão por anomalia, a qual busca detectar o complementar do comportamento normal. De maneira genérica, pode-se dizer que a detecção por abuso trata e usa diretamente o conhecimento sobre o comportamento das ações intrusivas, enquanto a detecção por anomalia usa o conhecimento sobre o comportamento normal e procura identificar o que foge a este comportamento [CAN 97]. A detecção por abuso também é conhecida pelo termo "detecção por assinatura de ataque", termo que será adotado neste trabalho.

Modelos estatísticos são usados na detecção de anomalia. A monitoração rigorosa de padrões de ataque bem conhecidos são adotados quando se usa detecção de abuso. Ambos os métodos trazem implícitas algumas suposições sobre a natureza das intrusões, que podem ser detectadas por eles e, obviamente, possuem determinadas aplicações e limitações. Além disso, os métodos de detecção podem variar pela maneira como são implantados e pelo tipo de dados ou análises que utilizam dentro dos sistemas computacionais.

2.3.2 Classificação pela Arquitetura

A arquitetura de um IDS está diretamente ligada a forma como seus componentes interagem entre si. Dois fatores influenciam diretamente na arquitetura: localização e fonte dos dados.

A localização dos componentes da arquitetura define se é um sistema centralizado, hierárquico ou distribuído. No primeiro caso, tem-se todos os componentes do IDS localizados no mesmo ponto, realizando as funções de coleta de dados até a configuração e gerência da ferramenta. Em uma arquitetura hierárquica, tem-se os componentes parcialmente distribuídos, mas com fortes relações de hierarquia entre eles. Geralmente, existirá a figura de um componente principal, que irá gerenciar o processo através da delegação de tarefas a outros componentes, que, por sua vez, poderão fazer o mesmo. Caracteriza-se, desta forma, uma relação de hierarquia entre os componentes. Já em uma arquitetura distribuída, os componentes estão espalhados pelo sistema, com uma relação mínima de hierarquia entre eles. Estes componentes poderiam, por exemplo, trabalhar em conjunto para alcançar

um objetivo comum, no caso, a detecção de intrusão.

Além da classificação pela localização, apresentada acima, os sistemas detectores de intrusão podem ainda ser divididos em duas categorias mais amplas: IDSs baseados em hosts (*host-based*) e IDSs baseados em rede (*network-based*). Os sistemas host-based empregam os registros de auditoria como fonte principal para identificar intrusões, enquanto os sistemas network-based constroem seu próprio mecanismo utilizando os dados coletados diretamente da rede. Os sistemas network-based podem ainda utilizar análises dos registros de auditoria dos computadores da rede sobre a qual atuam, de maneira a complementar informações ou eliminar casos duvidosos ou ambíguos. Alguns sistemas são híbridos e agregam várias técnicas utilizadas em conjunto.

A ferramenta proposta neste trabalho pode ser enquadrada como tendo uma arquitetura hierárquica. Existe uma forte relação de hierarquia implícita ao sistema: o gerente delega tarefas aos gerentes intermediários, que por sua vez disparam agentes de monitoração e de ação, de acordo com as tarefas solicitadas. Embora os agentes intermediários tenham autonomia em detectar eventos e trata-los, em última instância, sempre serão relatados os fatos ao gerente principal. Em relação à fonte dos dados, esta ferramenta baseia-se nos dados coletados diretamente da rede, podendo ser caracterizada integralmente como baseada em rede.

2.3.3 Outras Classificações

Outro esquema proposto por [SMA 88] classifica as intrusões em seis tipos, agrupando-as com base em seu efeito final e no método de ação.

- tentativa de Invasão (*Attempted Break-in*): freqüentemente detectado através de perfis de comportamento atípico ou tentativa de violação dos mecanismos de validação e contenção;
- ataque Dissimulado (*Masquerade Attack*): também determinado através de perfis de comportamento atípico ou tentativa de violação dos mecanismos de validação e contenção, mas de determinação mais sensível;
- penetração do Controle de Segurança do Sistema (*Penetration*): usualmente detectado pela monitoração de padrões específicos de atividade;
- escoamento (*Leakage*): normalmente detectado por uso atípico de recursos de entrada e saída (E/S), que não se encaixa nos casos anteriores;
- comprometimento de Recursos (*Denial of Services*): detectado por uso atípico dos recursos do sistema, freqüentemente levando-os a tornarem-se indisponíveis;
- uso Mal-intencionado (*Malicious use*): normalmente relacionado a usuários legítimos do sistema que abusam de seus privilégios; pode ser detectado a partir de perfis de comportamento atípico, violações ou tentativas de violações de mecanismos de bloqueio, ou uso de privilégios especiais.

Axelsson [AXE 2000], por outro lado, propõe uma classificação baseada em princípios de detecção. A proposta expande os dois métodos de detecção citados anteriormente (por assinatura e por anomalia), criando sub-divisões para cada um dos métodos. Um resumo da proposta está na fig. 2.1.

Anomalia	Auto-aprendizado	Sem série de tempo	Modelamento por regras
			Estatísticas descritivas
		Com série de tempo	Rede neural artificial
	programado	Estatísticas descritivas	Estatística simples
			Baseada em regras
			limiares
		Bloqueia por default	Modelagem por séries de tempo
Assinatura	programado	Modelagem por estados	Transição e estados
			Redes de Petri
		Inteligência artificial	
		Comparação de strings	
		Baseado em regras	

FIGURA 2.1 – Classificação de IDS segundo [AXE 2000]

A ferramenta proposta neste trabalho (arquitetura Trace) enquadra-se como detecção por assinatura, programada, com a utilização de modelagem por estados. O casamento de *strings*, contemplado na classificação, pode também ser facilmente atingido. A detecção por anomalia pode ser feita se for levado em conta a arquitetura Trace completa. Neste caso, ainda poderia-se enquadrá-la como detecção por anomalia e programável. Os sub-itens relativos a "programável" poderiam ser alcançados através do gerente intermediário.

Campello [CAM 2001] sugere um esquema de classificação que estende a classificação quanto à arquitetura e método de análise. Este esquema é apresentado na fig. 2.2.

Segundo a classificação apresentada na fig. 2.2, os IDSs podem ser classificados em 4 categorias: pelo método de detecção, pela arquitetura, pelo comportamento pós-detecção e pela freqüência de uso.

A classificação pelo método de detecção e pela arquitetura já foram detalhados anteriormente. Segundo o esquema de classificação proposto, o sistema de detecção de intrusão poderia ter um comportamento passivo ou ativo, após realizada a detecção positiva de intrusão. Por exemplo, durante a detecção de uma varredura de portas, poderia-se modificar dinamicamente as regras de um *firewall*, no sentido de bloquear o acesso da máquina atacante. Esta seria uma atitude de um IDS ativo. Em uma outra situação, o IDS poderia simplesmente ativar um alarme em uma console, ou enviar uma mensagem via e-mail ou pager, na esperança que o administrador tomasse providências.

Em relação à classificação via freqüência de uso, basicamente classifica-se o sistema como atuando na monitoração de forma contínua ou em processamentos periódicos. Um sistema que atua de forma contínua possui a capacidade de atuar de forma ativa, mas exige grande capacidade de processamento de alguns componentes do sistema. Um sistema que faz o processamento periodicamente, não será capaz de

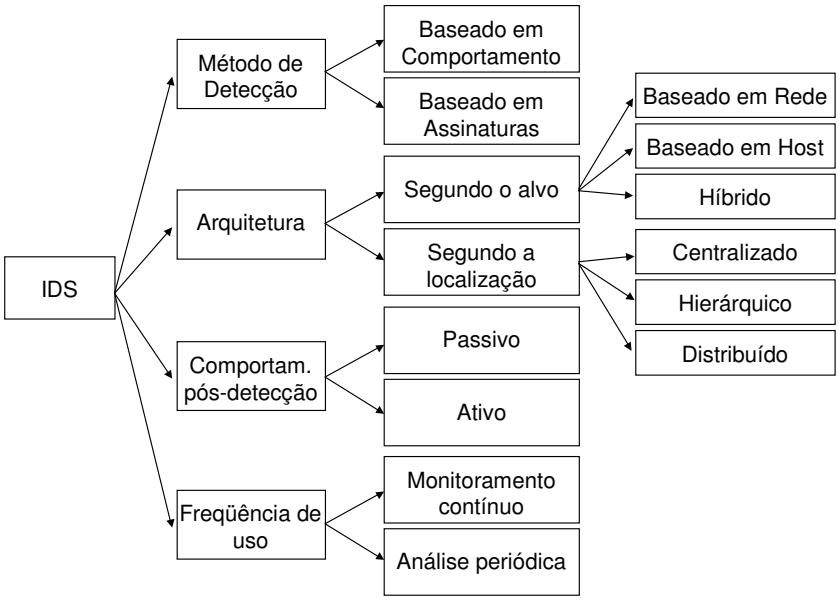

FIGURA 2.2 – Classificação de IDS segundo [CAM 2001]

atuar de forma ativa. Entretanto, esta abordagem pode ser útil quando deseja-se analisar uma grande quantidade de dados coletados em um período de tempo e onde uma atitude passiva após a detecção seja aceitável.

2.4 Análise pelo Método de Detecção

A busca por indícios de intrusão em trilhas de auditoria ou registros do sistema é um procedimento realizado há bastante tempo. Os IDSs utilizam técnicas que, de certa forma, derivam deste modelo. Um dos primeiros trabalhos que sugere esta relação surgiu em 1980 [AND 80]. Neste trabalho, Anderson sugere que certos padrões encontrados nos logs do sistema podem ser indícios de usuários que estejam abusando de privilégios, usuários não autorizados, ataques internos e externos entre outros.

Esta técnica traz um grande problema aos administradores dos sistemas: a grande quantidade de informação gerada nestes registros. O uso de técnicas manuais para manipular estes dados ou fazer correlações entre diferentes fontes de dados em tempo hábil, torna este processo praticamente inviável em boa parte dos sistemas modernos. Para resolver este problema, foram desenvolvidas ferramentas de apoio, que possibilitam o tratamento menos manual destes dados, facilitando a descoberta de indícios ou danos causados por ataques.

Outro problema que freqüentemente é citado em relação a esta técnica, é a dificuldade em proteger as próprias trilhas de auditoria. Em um sistema comprometido, as trilhas de auditoria são um dos primeiros alvos do atacante, como forma de encobrir os seus passos e o próprio ataque.

Além da análise de trilhas de auditoria, outra forma de detectar manualmente possíveis problemas de intrusão é valendo-se de dados coletados pela própria rede. Ferramentas de captura de tráfego, tais como o *tcpdump* [TCP 2002], são capazes de capturar o tráfego observado no segmento para posterior análise por pessoal es-

pecializado. A maior parte destas ferramentas de captura permite a especificação de filtros de captura, possibilitando assim um maior direcionamento dos dados capturados para o ataque ou evento que se procura. De qualquer forma, assim como no caso dos registros de auditoria, a quantidade de dados coletados é grande e a tarefa de localizar indícios é bastante árdua.

2.4.1 Análise Baseada em Assinaturas

A principal consideração na detecção por assinaturas de ataque é que há ataques que podem ser precisamente codificados, de maneira a capturar e registrar arranjos e variações acerca de atividades que exploram a mesma vulnerabilidade. Entretanto, na prática, nem todas as maneiras possíveis de se efetuar uma intrusão podem ser eficientemente capturadas e codificadas. A limitação principal desse método é que ele busca por vulnerabilidades conhecidas, e pode não ser de muita utilidade na detecção de intrusões futuras. Outra limitação deste método tem a ver com considerações práticas sobre o que é auditorado e de onde deve vir estes dados. Por exemplo, as práticas convencionais de auditoria não registram mudanças nas variáveis dos processos, devido ao impacto que isso pode causar na performance do sistema e devido às exigências de disco para armazenar os dados auditorados. Se uma intrusão só puder ser identificada a partir daqueles dados, há um comprometimento do método, ainda que alguns dados possam ser inferidos a partir de outras condições do sistema. A escolha das fontes de dados adequadas e as possíveis correlações existentes entre as fontes escolhidas são questões importantes também.

Por outro lado, os métodos de detecção por assinatura de ataque têm sido preferidos em muitos casos, devido ao custo computacional reduzido e ao pequeno comprometimento de desempenho do sistema efetuando a análise.

Segundo Campello [CAM 2001], várias são as formas de relacionar os dados coletados com as assinaturas existentes, otimizando o desempenho e diminuindo a ocorrência de falsos positivos e/ou negativos. O uso de sistemas especialistas, análise por transição de estados, uso de redes de Petri ou a adoção de informações facilmente encontradas nos dados coletados, são alguns exemplos de técnicas existentes para viabilizar a detecção de intrusão por assinatura. Este método apresenta alguns pontos positivos e negativos. Podem-se citar como pontos positivos o baixo índice de falsos positivos e o bom desempenho em geral, visto que não requer processos computacionalmente intensos nem uma grande quantidade de dados. A principal desvantagem é o fato de somente detectar ataques conhecidos. Como o número de ataques tem crescido muito, isto leva a períodos em que o sistema é incapaz de detectar certos ataques recém lançados. A alta manutenção requerida por estes sistemas também é um reflexo direto deste fato.

2.4.2 Análise Baseada em Comportamento

Segundo [CAN 97], a consideração central da detecção de intrusão por comportamento ou por anomalia é que a atividade intrusiva é um sub-conjunto da atividade anormal. Considerando-se que um atacante tem uma grande chance de agir de forma diferente de um usuário legítimo, se for possível medir o quanto diferente é este comportamento, teremos uma boa quantificação da probabilidade deste comportamento ser ou não ser intrusivo.

Em geral, o conjunto de atividades anormais é o mesmo conjunto de atividades intrusivas. Assim, em tese, localizando-se todas as anomalias, seria possível localizar todas as atividades intrusivas. Entretanto, a atividade intrusiva nem sempre coincide com atividade anômala. Nos sistemas por detecção de anomalia, há quatro casos possíveis, cada um dos quais com probabilidade diferente de zero de ocorrer [SMA 88]:

- *intrusivo e anômalo*: são os verdadeiros positivos. A atividade é intrusiva e é apontada como tal por ser também anômala;
- *não intrusivo e não anômalo*: são verdadeiros negativos. A atividade não é anômala e não é apontada como intrusiva;
- *intrusivo mas não anômalo*: são falso negativos, isto é, a atividade é intrusiva mas, como não é anômala, há uma falha em sua detecção;
- *não intrusivo mas anômalo*: são os falso positivos. A atividade não é intrusiva, mas como é anormal, o sistema a aponta, erroneamente, como sendo intrusiva.

Para eliminar ou reduzir o problema de falso-negativos e falsos-positivos, é preciso redefinir os limites que apontam a anomalia. Sistemas que apresentam um grande número de falsos-positivos tendem a cair em descrédito com o passar do tempo. Por outro lado, sistemas que apresentam falsos-negativos, representam uma falha no propósito do próprio IDS. Diversas técnicas têm sido propostas para analisar o comportamento do sistema. Uma das técnicas mais utilizadas, faz uso de métodos estatísticos para estabelecimento de perfis do sistema. Desta forma, o comportamento do usuário é medido em variáveis amostradas durante o tempo. O tempo de amostragem pode variar de alguns minutos até vários dias ou meses. De posse destas amostragens, são estabelecidas médias relacionadas a estas variáveis, que servem como limiares para estabelecer o grau de anormalidade do comportamento do usuário.

Sistemas especialistas também são utilizadas para analisar o comportamento do sistema e/ou usuário. Nesse caso, as atividades do sistema e/ou usuário são amostradas e regras são criadas contendo estas estatísticas. De posse destas regras, o sistema é capaz de fazer inferências periódicas acerca do comportamento atual deste usuário.

De forma geral, este método possui pontos positivos e negativos. A principal vantagem deste método está relacionada com a possibilidade de detectar ataques desconhecidos e complexos, não dependendo de uma base que armazene todos os ataques e vulnerabilidades possíveis. Outro ponto positivo é a baixa manutenção do sistema, visto que não há necessidade de atualizações freqüentes. Como pontos negativos, pode-se citar a dificuldade do ajuste dos limiares do que é anômalo ou não. Um ajuste errôneo pode ocasionar um grande número de falsos positivos, levando o sistema ao descrédito, como já citado anteriormente. A exigência computacional para este método é maior do que o necessário para a análise baseada em assinaturas. Finalmente, é difícil lidar com mudanças de comportamento sutis, de forma que se o atacante atuar desta forma, possivelmente o sistema irá considerar o comportamento do atacante como normal.

2.5 Arquiteturas de IDS

A arquitetura de um IDS refere-se a como os componentes funcionais são dispostos para garantir um desempenho adequado e o funcionamento correto do sistema. A seguir, serão descritas algumas formas de organização de um IDS, explorando as diferenças entre sistemas baseados em host e baseados em rede, além de arquiteturas centralizadas, distribuídas e hierárquicas.

2.5.1 IDS Baseado em Host

Os sistemas de detecção de intrusão que utilizam informações coletadas do próprio computador como base de dados foram os precursores nesta área. Com estes dados, é possível analisar as atividades do sistema e/ou usuário com uma boa precisão, determinando de forma exata quais processos e usuários estão envolvidos em um ataque determinado [BAC 2001]. O alvo das primeiras ferramentas desenvolvidas eram os mainframes, que por sua natureza centralizada, facilitavam a monitoração do sistema com base nos registros do próprio sistema. Em um estágio posterior, ferramentas auxiliares faziam a monitoração automática de eventos suspeitos, gerando alarmes quando necessário. Em geral, a fonte de informação utilizada são os registros de eventos do sistema (system logs) e as trilhas de auditoria (normalmente geradas em nível de kernel). Esta técnica possui uma boa precisão na detecção. Segundo [BAC 2001], as principais vantagens desta abordagem são:

- capacidade de monitorar dados em uma rede com criptografia;
- detectar ataques internos, possivelmente abuso de privilégio ou atividades não autorizadas;
- independência do tipo de rede utilizada.

Como principais desvantagens, [BAC 2001] cita:

- maior dificuldade de configuração e manutenção;
- impõe uma certa carga ao equipamento sendo monitorado;
- nos casos em que o alvo do ataque é o próprio host, o IDS pode ser desabilitado;
- a quantidade de informação gerada nas trilhas de auditoria pode ser muito grande;
- ataques de rede direcionados à infra-estrutura de rede não podem ser monitorados.

2.5.2 IDS Baseado em Rede

As desvantagens citadas nos IDSs baseado em host, em conjunto com o uso explosivo da Internet, observado nos últimos anos, levou à necessidade de se considerar os ataques à própria rede. Os IDSs baseado em host são particularmente ineficazes na monitoração de ataques do tipo negação de serviços, seqüestro de conexão e outros. A solução era obter os dados para análise diretamente da rede. Desta forma,

poderia-se monitorar os ataques à própria rede e, através da interpretação dos dados transportados pela rede, monitorar também determinadas máquinas.

Pelos motivos expostos acima, atualmente a maioria dos sistemas de detecção de intrusão são baseados em rede ou híbridos. Capturando pacotes em segmentos de rede ou diretamente de switches, é possível monitorar o tráfego da rede sem interferir no funcionamento normal das máquinas afetadas, além de possibilitar a detecção de ataques complexos. Em geral, este tipo de IDS faz uso de sensores espalhados em ponto estratégicos da rede, fazendo a análise do tráfego e relatando os indícios de ataques a uma estação central. Como estes sensores, muitas vezes, são dedicados a esta tarefa, eles são mais fáceis de ser gerenciados do ponto de vista de segurança e, portanto, mais imunes a ataques contra o próprio IDS.

As principais vantagens deste tipo de IDS são [BAC 2001]:

- alguns poucos e bem colocados IDSs podem monitorar uma grande rede;
- a instalação (ou desinstalação) é facilmente realizada;
- independência de plataforma;
- pouco impacto na rede.

Algumas desvantagens que podem ser citadas [BAC 2001]:

- em redes com alto tráfego, os IDSs podem ter dificuldade em processar todos os pacotes observados;
- em redes que fazem uso de switches, muitas vezes não existe o recurso no equipamento para copiar o tráfego de todas as portas para uma porta apenas, de forma que fica difícil colocar o IDS em um ponto que capture todo o tráfego da rede;
- não são capazes de analisar dados criptografados, prática cada vez mais comum nas redes;
- ataques em andamento nem sempre possibilitam uma reação adequada deste tipo de IDS;
- alguns IDSs deste tipo possuem dificuldades em tratar tráfego fragmentado.

2.6 Sistemas de Detecção de Intrusão Existentes

Esta seção apresenta um resumo das principais características de alguns sistemas de detecção de intrusão disponíveis atualmente.

2.6.1 SNORT

Snort [SNO 2002] é um sistema para detecção de intrusão leve, capaz de realizar análise de tráfego em tempo real, além de fazer registro de pacotes em uma rede IP. É um dos IDSs mais utilizados no momento. Pode ainda realizar análise de protocolo, casamento de padrões ou procura no conteúdo dos pacotes, além de ser usado para detectar uma variedade de ataques e sondagens, como estouro de

buffer, varredura de portas "discretas", ataques via CGI, tentativas para determinar o tipo de sistema operacional (*OS fingerprinting*), etc. O Snort utiliza uma linguagem flexível para criar regras como forma de determinar o tráfego que deve ser coletado ou descartado. Possui, também, um sistema de alerta em tempo real que pode ser direcionado para um arquivo específico, para um *socket* UNIX, para o "*syslog*" (mecanismo de registro de eventos utilizado no Sistema Operacional Unix) ou ainda um "*WinPopup*" para os clientes Windows.

O Snort preenche um nicho importante no domínio de segurança em redes: uma ferramenta para detecção de intrusão para rede multi-plataforma que pode ser instalada para monitorar pequenas redes `TCP/IP` e detectar uma grande quantidade de tráfego suspeito ou ataques efetivos. A ferramenta fornece ao administrador uma grande quantidade de dados que possibilitam a tomada de decisão em relação ao curso que deve ser tomado quando um ataque é detectado. Pode ainda ser rapidamente instalado para preencher algum problema urgente na segurança da rede, tal como um novo ataque recentemente detectado e que ainda não existam versões de softwares comerciais disponíveis.

O Snort possui três usos principais. Pode ser usado diretamente como um *packet sniffer* de rede, assim como o *tcpdump*, um registrador de pacotes (*packet logger*) ou como um poderoso sistema para detecção de intrusão. Como ele utiliza a biblioteca de funções *libpcap* para capturar pacotes da rede, ele pode ser instalado em praticamente qualquer sistema operacional que suporte esta biblioteca.

A arquitetura deste sistema é focada em desempenho, simplicidade e flexibilidade. Existem três sub-sistemas principais: o decodificador de pacote, o mecanismo de detecção e o sub-sistema de alerta e registro. Estes sub-sistemas estão acima da *libpcap*, o que fornece a capacidade de sniffer e filtragem de forma bastante portável.

A configuração do programa, análise das regras (*parsing*) e a geração da estrutura de dados são realizadas antes do sniffer ser inicializado, mantendo o processamento necessário para cada pacote no nível mais baixo possível.

O decodificador de pacote é organizado de forma semelhante às camadas da pilha de protocolo definidos para o `TCP/IP`. As sub-rotinas de decodificação são chamadas em ordem, de acordo com a pilha de protocolo, indo da camada de enlace até a camada de aplicação. A velocidade é importante nesta fase, sendo que a maior parte da funcionalidade do decodificador é de ajustar ponteiros nos pacotes de dados para análise posterior pelo mecanismo de detecção.

O mecanismo de detecção mantém as regras de detecção em uma lista interligada, que contém os elos para cabeçalhos e opções. Trata-se de listas de regras que foram condensadas em uma lista de atributos comuns dos cabeçalhos, com as opções modificadoras de detecção contidas no elo das opções. Por exemplo, se várias regras compartilham o mesmo endereço `IP` ou porta de destino, isto será condensado em um elo de cabeçalho e as assinaturas individuais serão colocadas nos elos de opções.

O sub-sistemas de alerta e registro é determinado em tempo de execução através de chaves na linha de comando. Existem três formas de registro e cinco formas de alerta. O Snort registra os pacotes no formato binário do *tcpdump* ou em um formato `ASCII`, onde são colocados os diretórios baseados em nomes ou endereços `IP` do host remoto.

A linguagem que descreve as regras é simples e flexível, embora não deixe de ser poderosa. As regras são divididas em duas sessões lógica, o cabeçalho e as opções da regra. O cabeçalho contém as ações, o protocolo, o endereço IP de origem

e destino e as máscaras de rede, além de informação das portas de origem e destino. A sessão de opções contém as mensagens de alerta e informação sobre qual parte do pacote deve ser inspecionado no sentido de definir se a ação da regra deve ser tomada ou não.

Um exemplo de regra:

```
alert tcp any any -> 192.168.1.0/24 111 (content:"—00 01 86 a5—"; msg: "mountd";)
```

A interpretação da regra:

- os dados até o primeiro parênteses correspondem a sessão do cabeçalho da regra;
- os dados entre parênteses correspondem a sessão de opções da regra;
- as palavras antes do ponto e vírgula são chamadas palavras chave da opção (*option keywords*).

A sessão de opções da regra não é obrigatória para nenhuma regra, mas são importantes para descrever definições mais precisas do que deve ser coletado ou descartado e do que deve ou não gerar um alerta. Todos os elementos que compõem a regra devem ser verdadeiros para que uma ação seja tomada. Considerados em conjunto, os elementos podem ser considerados como formando uma operação lógica E entre estes elementos (todos os elementos devem ser verdadeiros para que a expressão seja verdadeira). Por outro lado, considerando todas as regras do sistema, pode-se pensar como sendo uma longa expressão OU entre todas as regras do sistema (se alguma regra for verdadeira, emite-se um alarme).

2.6.2 Bro

Bro é uma ferramenta de pesquisa desenvolvida pelo Laboratório Nacional de Lawrence Livermore. Ela está sendo construída, em parte, para explorar as características relacionadas a robustez de um sistema de detecção de intrusão, isto é, avaliando quais características fazem um IDS capaz de resistir a ataques contra ele próprio. Os objetivos para o Bro incluem [PAX 98]:

- *monitoração sob alta carga*: a habilidade em manusear altas taxas de transferência de dados e volume de tráfego sem descartar pacotes é importante. Um invasor pode usar mecanismos para sobrecarregar a rede com pacotes estranhos para sobrecarregar o IDS. Isto pode forçar o IDS a descartar pacotes para os quais a rede estava vulnerável;
- *notificação em tempo real*: isto é necessário para garantir uma resposta em tempo hábil as ameaças de intrusão;
- *desacoplar os mecanismos da política*: separando a identificação de eventos de filtragem de dados das reações relacionadas a política adotada para os eventos resulta em um design do software mais limpo, uma implementação mais fácil além de uma manutenção mais direta;
- *extensibilidade*: o grande número de ataques conhecidos, junto com o crescente descobrimento de vulnerabilidades, exigem que o Bro tenha a capacidade de adicionar novos scripts a sua biblioteca;

- *capacidade de impedir ataques*: os atacantes mais sofisticados vão provavelmente procurar falhas no próprio sistema de detecção de intrusão.

Bro tem uma hierarquia de funções de três níveis. No nível mais baixo, Bro usa a *libpcap*, uma biblioteca largamente utilizado em sistemas Unix para extrair pacotes da rede. Isto desacopla a funcionalidade principal da detecção de intrusão dos detalhes de rede. Também permite uma significativa redução no número de pacotes a serem tratados, já que eles podem ser descartados no nível mais baixo. A *libpcap* vai capturar apenas os pacotes relacionados aos protocolos de aplicação (*finger*, *ftp*, *telnet*, etc) aos quais o Bro deve se preocupar.

O próximo nível, o nível de eventos, realiza a verificação da integridade nos cabeçalhos dos pacotes. Se o cabeçalho estiver mal formado, um evento identificando o problema é gerado e o cabeçalho é descartado. Uma verificação é então realizada para determinar se todo o conteúdo do pacote deve ser armazenado, somente a informação contida no cabeçalho ou se nada deve ser armazenado.

Os eventos são gerados a partir deste processo e colocados em uma fila para serem analisados pelo script interpretador de política, que reside no terceiro nível. Este script foi escrito em uma linguagem própria do Bro que usa uma forte tipificação para fornecer um amplo suporte a análise do cabeçalho de um pacote, tais como porta e domínio e outras informações importantes dentro do conceito de redes. O interpretador associa eventos a códigos para o tratador de eventos e então executa o código. Executando o código pode resultar em gerar mais eventos, registrar notificações em tempo real ou apenas registrar os dados. Para adicionar novas capacidades ao Bro, é necessário identificar os eventos associados com os protocolos das aplicações, e escrever tratadores de eventos correspondentes para estender a funcionalidade do script interpretador de política. Os desenvolvedores justificam que este desacoplamento do evento do seu tratador melhora a extensibilidade do Bro.

Até o momento, o Bro monitora 4 aplicações: *finger*, *ftp*, *portmapper* e *telnet*. Adicionar novas aplicações ao Bro é, de acordo com o desenvolvedor, bastante direto, algo como instanciar uma classe de C++ para analisar cada conexão, e planejar um conjunto de eventos correspondentes aos elementos significativos da aplicação. Bro roda em vários sabores de Unix e está sendo usado como parte do sistema de segurança do laboratório onde foi desenvolvido. Em uma rede `FDDI` com trafego da ordem de 25mbps, não foi noticiado perda de pacotes, o que representa uma capacidade de análise de 200 pacotes/segundo.

2.6.3 EMERALD

EMERALD (*Event Monitoring Enabling Responses to Anomalous Live Disturbances*) [EME 2002] é a mais recente ferramenta de pesquisa desenvolvida pela SRI International. Esta linha de ferramentas explora a detecção de intrusão como desvios de comportamento normal (anomalias) em conjunto com a detecção por assinaturas de ataques. A SRI foi pioneira em trabalhos relacionados a detecção de intrusão, tendo começado em 1983, quando um algoritmo estatístico de multi-variação foi desenvolvido para discriminar o comportamento entre usuários.

Após algum tempo, um subsistema que usa assinaturas de ataques baseado em P-BEST [LIN 99] foi investigado para dar suporte a detecção de atividades suspeitas. Estes esforços de pesquisa foram incorporadas à ferramenta IDES [LUN 92], um sistema que monitora atividades em múltiplos hosts em tempo real.

Baseado na experiência adquirida com o IDES, uma nova ferramenta orientada a produção foi desenvolvida entre 1992 e 1994 e chamada de NIDES [AND 95]. Assim como o IDES, esta ferramenta é baseada em host e usa também o sistema P-BEST.

Entretanto, foi além, adicionando um componente chamado de "*resolver*" que une os resultados dos componentes de análise estatístico e por assinatura. A interface para o usuário no NIDES foi melhorada de forma significativa, se comparada com o IDES.

EMERALD foi construído baseado na experiência anterior do IDES/NIDES, mas focando no suporte à rede e não ao host, ou seja, usando método não intrusivo ao contrário da abordagem anterior. O maior objetivo do EMERALD é de endereçar as questões associadas a grandes e fracamente acopladas redes de empresas. Estes ambientes são mais difíceis de monitorar e analisar, devido à natureza distribuída da informação que flue pela rede. EMERALD estrutura os usuários em uma "federação" de domínios administrados de forma independente. Cada domínio provê uma coleção de serviços de rede, como http ou ftp, que podem ter diferentes relações de confiança entre si e possuir diferentes políticas de segurança. Neste contexto, um repositório central pode resultar em uma degradação de desempenho significativa, devido à centralização da análise de todos os dados. Este assunto motivou o trabalho realizado no EMERALD e a demonstração da técnica "dividir para conquistar" que foi investigada.

A abordagem hierárquica provê três níveis de análise realizada por um sistema de três camadas de monitoradores: monitoradores de serviço, monitoradores de domínio e monitoradores do negócio. Estes monitoradores têm a mesma estrutura básica: um conjunto de mecanismos para traçar o perfil (para detecção de anomalia), mecanismos para reconhecimento de assinaturas, e um componente que integra os resultados gerados por estes mecanismos.

Cada módulo também contém um recurso que provê a manutenção de informações para personalizar os componentes modulares da aplicação alvo. Este recurso pode ser re-utilizado em múltiplos monitoradores dentro de uma aplicação EMERALD. Em um nível baixo, monitoradores de serviço suportam a detecção de intrusão para componentes individuais e serviços de rede dentro de um domínio, fazendo a leitura de dados (registros de atividade, eventos, etc) e realizando a comparação de assinaturas e análise estatística.

Monitoradores de domínio integram informação provenientes dos monitoradores de serviço para prover uma visão ampla do domínio quanto a intrusões, enquanto que os monitoradores de negócio realizam análise entre domínios para avaliar as ameaças a partir de uma perspectiva global. No nível de negócio, ameaças como ataques do tipo "vermes" ou ataques entre domínios nos serviços de rede são de interesse.

Trabalhos anteriores, como o NIDES, demonstraram que técnicas para criar perfis estatisticamente podem ser eficientes tanto com usuários como com aplicações como alvos. A monitoração de aplicações (como FTP anônimo por exemplo) é particularmente efetiva, já que poucos perfis de aplicação são necessários. EMERALD generaliza esta técnica para traçar perfis abstraindo a noção de perfil, separando a gerência do perfil da análise do perfil.

Em relação a análise das assinaturas, o mecanismo de assinaturas na camada de serviços monitora os componentes do domínio para determinar se atividade anor-

mal está ocorrendo, através do uso de *scripts* de exploração. Mecanismos de assinatura em camadas superioras retificam esta informação para avaliar se um ataque em larga escala está ocorrendo. EMERALD é um trabalho em progresso. Ele provê um exemplo da direção que os futuros sistemas para detecção de intrusão devem seguir. Como os invasores tendem a fazer ataques mais sofisticados, tornando as evidências de ataque na rede mais dispersas, a detecção de ataques distribuídos e coordenados torna-se cada vez mais difícil de ser realizada. Nestas situações, a habilidade em coletar, assimilar, correlacionar e analisar a informação vindo de várias fontes em tempo real, torna-se essencial. A flexibilidade da arquitetura escalável do EMERALD, a habilidade em abstrair a funcionalidade, a facilidade em incorporar ferramentas externas e a experiência anterior (NIDES por exemplo), fazem do EMERALD um precursor das futuras ferramentas para detecção de intrusão. Fica a ressalva de que gerenciar e manter a base de informação e construir a infraestrutura para suportar o sistema, podem requerer um esforço considerável.

2.6.4 STAT

STAT é um conjunto de ferramentas voltadas para a detecção de intrusão desenvolvido pelo Reliable Software Group da Universidade da Califórnia. É baseado em uma técnica de análise por transição de estados proposta em [POR 92] e que foi chamada de STAT. O STAT foi iniciado no início dos anos 90 e baseia-se na análise de transição de estado no sentido de suportar a detecção de intrusão em tempo real. Esta abordagem está fundamentada na premissa de que certas seqüências de ações refletem uma atividade não autorizada e indicam que existe um invasor modificando o estado de um sistema de normal para comprometido. Esta abordagem geral tem sido estendida no sentido de permitir a detecção de intrusão em diferentes domínios e ambientes. A versão mais recente deste sistema é composta por um módulo principal que provê as características de independência de domínio e ambiente.

Muitos sistemas de detecção de intrusão que atuam na categoria de detecção por anomalia, analisam evidências de intrusão nos arquivos de auditoria do sistema. Entretanto, o STAT transforma estes arquivos em um nível de informação mais abstrato, através da aplicação de um filtro nos arquivos originais. Esta abstração é mais apropriada para análise, portabilidade e entendimento humano e são chamadas de assinaturas, além de fundamentais para a abordagem adotada pelo STAT. Assinaturas de ações movem o sistema através de uma seqüência de estados, cada estado conduzindo o sistema próximo a uma configuração comprometida. Seqüências de intrusão são definidas por transições de estados, que são capturados nas regras do sistema em produção.

A implementação inicial deste método foi realizada em um Sistema Operacional Unix, e foi chamada de USTAT [KEM 99], composto de:

- um pré-processador;
- uma base de conhecimento (que inclui uma base de fatos e uma base de regras);
- um sistema de inferência;
- um sistema de decisão.

O pré-processador filtra e manipula os dados em um formato que seja independente dos arquivos de auditoria. O componente base de regras da base de co-

nhecimento armazena as regras de transição de estado, que indicam uma seqüência predefinida de intrusão, enquanto que a base de fatos armazena as mudanças de estado dinâmicas do sistema, no que diz respeito a possíveis invasões ocorrendo.

Fornecendo informações geradas pelo pre-processador, em conjunto com o atual estado do sistema como definido na base de fatos, possibilita ao sistema de inferência identificar qualquer mudança significativa de estado e atualizar a base de fatos. Uma função de atualização revisa, então, a base de fatos para refletir estas mudanças. O sistema de inferência também notifica o sistema de decisão sobre possíveis violações de segurança. O sistema de decisão, por sua vez, notifica o responsável pela segurança ou inicia uma ação por conta própria. Uma vantagem de uma abordagem orientada a estado é que um ataque pode ser reconhecido antes do sistema atingir um estado de comprometimento mais sério.

A abordagem baseada em estado usa uma tabela do sistema de inferência para capturar cada possível intrusão, e permite ao USTAT identificar um ataque coordenado proveniente de várias fontes. Isto é possível já que as seqüências de ataque são definidas não por quem está realizando o ataque, mas pelos estados do sistema. Então, se dois atacantes estão se baseando no mesmo estado do sistema, cada uma de suas ações posteriores pode ser acompanhada pela ramificação do estado prévio da seqüência de transições. Esta ramificação é implementada através da duplicação das linhas da tabela de inferência, cada linha representando diferentes seqüências de ataques.

NSTAT [KEM 97] foi o sucessor natural do USTAT. NSTAT tem como foco o suporte a redes de hosts, onde, por exemplo, arquivos são compartilhados. Assim, ações tomadas em um host, tais como montar diretórios, podem influenciar outras máquinas nesta rede. Tendo um sistema de detecção central, resulta em uma queda de desempenho menor nos hosts locais e também permite uma melhor avaliação quando uma análise direcionada a vários hosts esteja ocorrendo. Com o NSTAT, os hosts locais convertem seus arquivos de auditoria em um formato comum e em um fluxo de dados apenas.

Uma ferramenta que surgiu em seguida foi o NETSTAT [VIG 98], que diverge dos sistemas anteriores baseados em técnicas invasivas em favor de uma abordagem não intrusiva (baseado em rede). NetSTAT é composto por um conjunto de sondas que são responsáveis por detectar e avaliar intrusões em sub-redes sendo monitoradas. Cada sonda é suportada por um filtro configurável remoto, um sistema de inferência e um sistema de decisão. Estas sondas podem agir de forma autônoma. Entretanto, diferentes partes de uma rede podem detectar diferentes componentes de uma invasão (devido as diferentes localizações e filtros). Se um componente de intrusão for detectado, então um evento pode ser enviado a outras sondas com interesse neste evento, como forma de obter uma melhor compreensão da intrusão. Assim, intrusões que envolvem sub-redes separadas podem ser identificadas.

Na segunda metade de 1998, foi conduzida uma série de experimentos, que utilizou o USTAT e o NetSTAT [DUR 99], operando em conjunto. Deste experimento, foi estabelecido que havia uma série de similaridades entre a forma como os ataques eram descritos para as duas ferramentas. Como resultado, direcionou-se o desenvolvimento para a criação de um módulo principal que suportasse a especificação de ataques independentes do ambiente (sistemas operacionais por exemplo) ou do domínio (dados provenientes da rede ou de registros de auditoria). Este módulo principal foi chamado simplesmente de *STAT core*. Um dos elementos definidos

neste módulo foi uma nova linguagem, chamada STATL, usada para representar cenários de ataques usando estados e transições. Vigna [VIG 2000] descreve esta arquitetura em detalhes.

2.6.5 GrIDS

GrIDS significa "*Graph-based Intrusion Detection System*", sistema para detecção de intrusão baseado em grafo. Trata-se de uma linguagem para análise de atividade das conexões de redes locais ou metropolitanas. O sistema contém uma linguagem para especificação em uma forma de grafos da atividade em uma rede, incorporando informações de IDSs baseados em host, sniffers de rede e componentes de análise de conexões. O GrIDS faz parte do projeto "detecção de intrusão para grandes redes", sendo conduzido pela UC Davis's [CHE 99].

O ponto chave do GrIDS é detectar ataques automatizados em larga escala. O mecanismo proposto baseia-se em construir grafos de atividade. Os nós de um grafo de atividade correspondem a hosts em um sistema, enquanto que os arcos do grafo correspondem à atividade de rede entre estes hosts. Tomando como exemplo um ataque do tipo verme (*worm*), constrói-se o exemplo a seguir.

O verme inicia no sistema A, então inicia uma conexão com o host B e C, contaminando estes hosts. Quando o mecanismo de agregação recebe o relato de uma conexão entre A e B, ele inicia um novo grafo com dois nós e um arco entre eles (representado aqui como "A→B"). O mecanismo ajusta um tempo máximo de existência deste grafo. Se nada for recebido dentro do tempo de atividade, o grafo é descartado. Em seguida, o mecanismo recebe o relato da conexão A→C e adiciona o host C ao grafo, assim como uma borda representando a conexão entre os nós.

O verme vai então se espalhar para D, E e assim por diante. A representação deste ataque está na fig. 2.3.

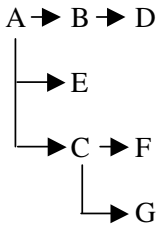

FIGURA 2.3 – Grafo de funcionamento do GrIDS

Neste ponto, o mecanismo de agregação verifica uma estrutura em rede do grafo, e reporta a suspeita de um verme. Neste exemplo, foram colocadas arcos no grafo puramente baseado no tempo de incidência. Em um algoritmo mais elaborado, poderia-se também usar outras informações para decidir se os arcos estão relacionadas, tal como se os nós usam as mesmas portas de destino ou se foram para hosts com sistemas operacionais semelhantes.

Além de formar grafos de conexões, é permitido que tanto os nós quanto os arcos possuam atributos que forneçam informações extras sobre o host ou a conexão. Por exemplo, se em uma conexão aparecer indícios de transferência de arquivo de senhas, esta anotação pode ser colocada neste link.

Uma dificuldade explícita no exemplo do verme, é que somente vermes que se espalham rapidamente, e que irão criar grandes grafos, serão detectados. Vermes

que se espalham lentamente, dificilmente serão detectados. A melhor solução para este problema é associar o tipo de atividade da rede relacionado com o ataque. Se um verme é conhecido por se espalhar usando *rlogin* e *telnet*, então um grafo pode ser construído somente se estas conexões estiverem presentes.

Obviamente, para numerosos tipos de ataques que devem ser procurados, são necessários vários tipos de grafos. O mecanismo de criação de grafos é capaz de manter múltiplos espaços de grafos, onde cada espaço possui apenas um tipo de grafo. O tipo do grafo é determinado por um conjunto de regras, que especifica como devem ser constituídos os grafos no espaço correspondente.

Assumindo que uma organização possua vários departamentos semi-independentes e considerando que a maior parte das atividades estejam dentro de cada departamento, cada departamento possui seu próprio módulo que computa os grafos de atividade para aquele departamento. Se os departamentos estão inseridos em uma hierarquia da organização, os sub-departamentos serão nodos no grafo de atividades deste departamento. Ataques detectados serão atributos destes nodos.

Um aspecto importante deste sistema é que ele deve ser capaz de construir grandes grafos, sem considerar sua própria comunicação. Por isto, opções de filtragem de mensagens conhecidas por serem enviadas por componentes de segurança da rede são importantes.

As fontes de informações possíveis são:

- IDS baseado em host;
- TCP wrapper que forneça algum tipo de registro de conexões;
- sniffers de rede.

2.6.6 RealSecure

RealSecure [SYS 2002], produzido pela Internet Security Systems (ISS), é um outro exemplo de IDS de tempo real. Usa uma arquitetura de três camadas consistindo de um mecanismo de reconhecimento baseado em rede, outro baseado em host e um módulo de administração. O mecanismo de reconhecimento baseado em rede roda em uma estação dedicada para prover detecção de intrusão e resposta em relação a rede. Cada mecanismo observa cada pacote trafegando por um segmento específico da rede a procura de assinaturas de ataques que evidenciem uma tentativa de ataque. Quando este mecanismo detecta uma tentativa de ataque, ele pode responder através do término da conexão, enviando uma mensagem de correio eletrônico ou alerta por pager, armazenando a sessão, reconfigurando os *firewalls* apropriados, ou, finalmente, tomando outras atitudes configurada pelo usuário. Além disto, o mecanismo envia um alarme ao módulo administrador ou a um software de gerência de terceiros, como lembrete ou para exame mais aprofundado.

O mecanismo de reconhecimento baseado em host é um complemento ao mecanismo citado anteriormente, baseado em rede e fica residente no próprio host. Ele analisa os registros do host como forma de reconhecer ataques, determinar se o ataque foi efetivo ou não e para prover informação judicial, não disponível em um ambiente de tempo real. Cada mecanismo é instalado em uma estação de trabalho ou host, e examina profundamente os padrões que revelam quebra de segurança. Este mecanismo reage no sentido de prevenir incursões mais aprofundadas do invasor, através do término dos processos do usuário e suspendendo esta conta. Pode

ainda, enviar alarmes, registrar eventos, enviar traps, enviar mensagens de correio eletrônico ou ainda executar ações definidas pelo usuário.

Todos os mecanismos de reconhecimentos reportam-se e são configurados pelo módulo de administração, uma console de gerência que monitora o estado de qualquer combinação de UNIX com Windows NT/2000 mecanismos de reconhecimento. O resultado é uma proteção efetiva, facilidade de configuração e administração de um ponto único

2.6.7 Network Flight Recorder

Network Flight Recorder [NFR 2002] é um sistema para detecção de intrusão que surgiu como uma versão de domínio público e tornou-se, recentemente, uma ferramenta apenas comercial.

O Network Flight Recorder (NFR) usa uma versão modificada da *libpcap* para extrair pacotes de forma promíscua e passiva da rede. Entretanto, o NFR vai além de apenas analisar o cabeçalho dos pacotes, sendo possível reconstruir todo um fluxo `TCP`. A coleta e análise de dados são usualmente realizadas na mesma plataforma, fora do *firewall*. Entretanto, cópias do NFR podem ser também colocadas em pontos internos estratégicos como forma de detectar possíveis ameaças internas. Acima do nível de extração de pacotes, uma máquina de decisão filtra e remonta os pacotes.

NFR inclui uma linguagem de programação completa, chamada "N", desenvolvida para análise de pacotes. Os filtros são escritos nesta linguagem, que é compilada em *byte-code* e interpretada pela máquina virtual de execução. Através de programas escritos em "N", o casamento de padrões é realizado para remontar os pacotes, como um exemplo. As funções alerta e grava são usadas para extrair dados depois da operação de filtragem e para suportar outros módulos que façam processamento posterior (*back end*). A função de alerta pode enviar eventos via correio eletrônico ou fax. A função de gravação ajusta os dados em formatos exigidos por vários módulos de análise.

Dois exemplos de *back end* de uso geral são o "histograma" e "lista". O primeiro fornece a possibilidade de capturar dados em uma matriz multi-dimensional. Os totais das categorias relevantes são acumulados em células da matriz. Os alertas podem ser gerados baseados no número absoluto ou freqüência relativa em cada célula. O *back end* "lista" grava os registros históricos, fornecendo assim um nível de detalhe não fornecido pela função anterior.

O NFR fornece também uma função de consulta que permite que o próprio usuário analise os dados. Esta função foi desenvolvida de forma a não degradar muito o desempenho da máquina de execução, o que poderia ocasionar a perda de pacotes por descarte. Esta função possui sua própria interface `CGI`, além de fornecer uma capacidade gráfica para visualização.

Entre as características diferenciadas desta ferramenta citadas pelo fabricante, está a capacidade de monitorar redes operando a 1 Gbit/segundo.

3 A Arquitetura Trace

A arquitetura Trace é uma extensão da infra-estrutura centralizada de gerenciamento SNMP, para, através de um modelo em três camadas, suportar o gerenciamento distribuído de protocolos de alto nível e serviços de rede [GAS 2001]. Trata-se de uma arquitetura proposta para fins de gerência de redes de computadores, na qual pretende-se utiliza-la com um enfoque específico para a detecção de intrusão. A maior parte deste trabalho está centrado no agente de monitoração e na linguagem PTSL, visto serem os pontos mais importantes no enfoque desejado, embora a compreensão da arquitetura completa também seja importante dentro do contexto. Este capítulo descreve os componentes da arquitetura, exemplificando os procedimentos envolvidos nos diversos processos. A linguagem PTSL é apresentada em suas duas versões, a gráfica e a textual, conforme a especificação original [GAS 2002].

A fig. 3.1 ilustra um esquema da arquitetura. Trata-se de uma arquitetura hierárquica ou parcialmente distribuída, onde existem relações de subordinação entre os componentes. A principal contribuição da arquitetura é a presença de agentes de monitoração programáveis, o que a difere da abordagem tradicional utilizada nas soluções de gerenciamento SNMP.

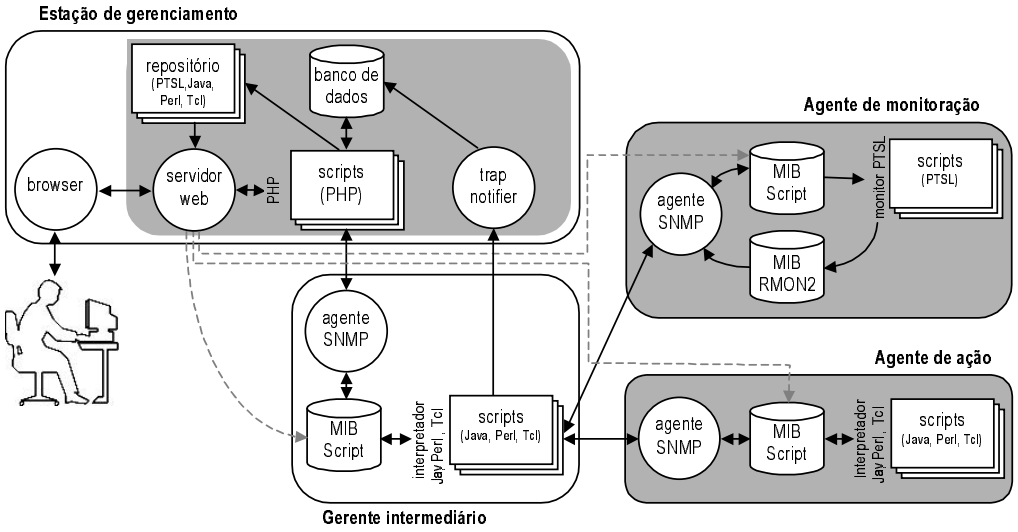

FIGURA 3.1 – Componentes da arquitetura Trace (extraída de [GAS 2001])

Os principais componentes da arquitetura são:

- *gerente*: delega tarefas de gerenciamento a gerentes intermediários. A arquitetura é composta por uma ou mais estações de gerenciamento (gerentes). A interface do gerente de rede com a arquitetura de gerenciamento é baseado em WEB. Através de um navegador Internet, o gerente acessa o ambiente de gerenciamento em um servidor WEB. Foi utilizada a linguagem PHP [PHP 2002] e o banco de dados MySQL [MYS 2002] para desenvolver este ambiente;

- *gerente intermediário*: o gerente intermediário tem por função executar e acompanhar tarefas de gerenciamento, delegadas pela estação central de gerenciamento, e reportar eventos significativos a ela. Uma rede pode possuir um ou mais gerentes intermediários. Os gerentes intermediários programam os

agentes de monitoração através da linguagem PTSL, que será vista a seguir. Podem ainda requisitar a agentes de ação a execução de procedimentos (*scripts Perl*), viabilizando a automação de diversas tarefas de gerenciamento;

- *agente de monitoração*: os agentes de monitoração contabilizam a ocorrência de traços no segmento onde se encontram. São denominados programáveis porque os traços a serem monitorados podem ser configurados dinamicamente, sem a necessidade de recompilar esses agentes. Esta flexibilidade é obtida através da linguagem PTSL. Na prática, os agentes realizam a leitura de arquivos PTSL, organizam algumas estruturas em memória e iniciam o processo de monitoração;

- *agente de ação*: através dos agentes de monitoração, o gerente intermediário tem condições de avaliar a ocorrência ou não de um traço. Os traços podem indicar a falha em um serviço de rede, a tentativa de intrusão, a queda de desempenho em algum serviço, entre outros. Nesse contexto, os agentes de ação são responsáveis pela execução de um procedimento de gerenciamento criado para combater, sem a intervenção humana e de forma automática, o problema detectado.

Genericamente, o funcionamento do sistema está representado através da fig. 3.2. O administrador da rede modela uma assinatura de ataque e uma tarefa de gerenciamento com o auxílio de uma interface `WEB`. A assinatura de ataque é descrita por traços de protocolos. Essa tarefa é delegada a um gerente intermediário, através de operações `SNMP` (*Simple Network Management Protocol*). O gerente intermediário programa os agentes de monitoração com os traços especificados na linguagem PTSL. As estatísticas coletadas pelos agentes de monitoração são armazenadas em uma `MIB` (*Management Information Base*) `RMON2` [WAL 97]. A programação do agente de monitoração é realizada pelo gerente intermediário através de um *script*. Este mesmo *script* faz a monitoração dos dados armazenados na `MIB` pelo agente e, dependendo das condições observadas, gera um alarme ao gerente (através de uma *trap SNMP*). O administrador da rede é informado da ocorrência de uma determinada condição estabelecida para ser monitorada. Esta forma de monitoração tem como grande desvantagem o alto tráfego de gerenciamento, visto que é necessário amostrar os valores coletados pelos agentes periodicamente (*polling*).

Em um segundo caso, deseja-se a monitoração de uma determinada situação, porém o agente de monitoração é capaz de gerar eventos ao gerente intermediário. Estes eventos são gerados através de *traps SNMP*, geradas pelo agente de monitoração com destino ao gerente intermediário. Este recurso está disponível através de grupos específicos da `MIB RMON`, que permitem a especificação de limiares que, quando excedidos, deverão gerar *traps SNMP*. Como o agente e monitoração implementa parcialmente os grupos da `MIB RMON2` e não implementa os grupos da `MIB RMON`, este recurso não está disponível. No estágio atual de desenvolvimento do agente de monitoração, a única forma de interação com o agente de monitoração é através de *polling*.

Do ponto de vista de detecção de intrusão, procura-se por determinadas variações de valores dos objetos da `MIB` em um período de tempo (Δ/t). Desta forma, considerando que o agente de monitoração fosse capaz de monitorar limiares, sempre que o agente de monitoração observar a ocorrência de traços por um período de

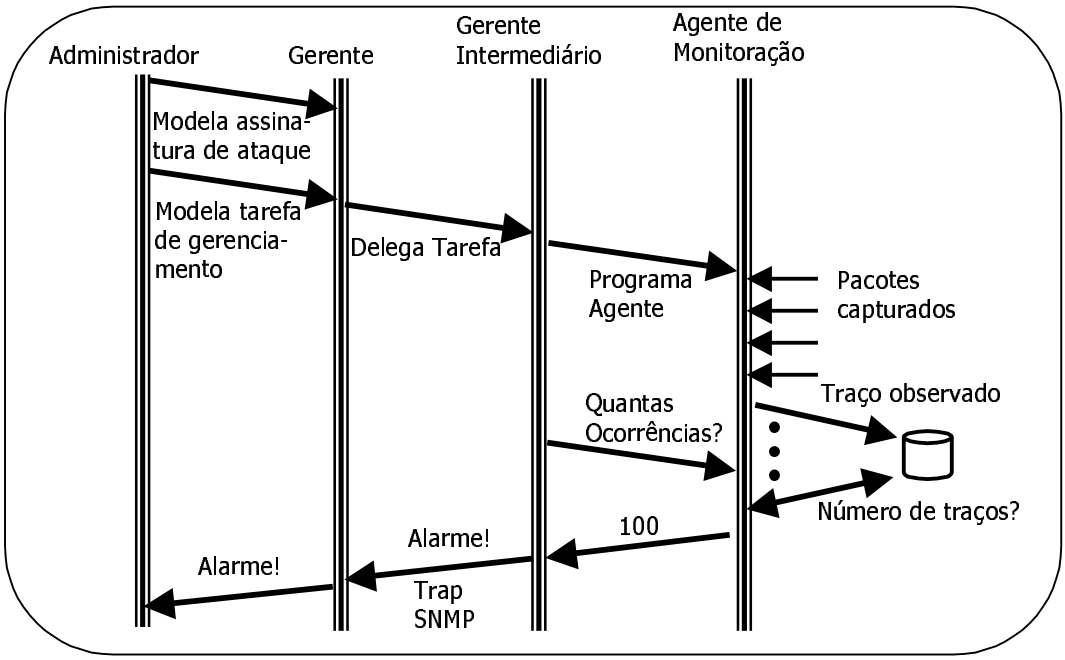

FIGURA 3.2 – Fluxo de informações para programar uma nova assinatura de ataque (via *polling*)

tempo superior a um limiar, um evento é gerado. Esta abordagem para a monitoração é mais eficiente que a baseada em polling. A fig. 3.3 representa esta abordagem.

Nos casos onde é necessário a execução de uma ação ao observar uma determinada condição reportada pelo agente de monitoração, então o gerente intermediário pode solicitar ao agente de ação a execução de um determinado procedimento. Os procedimentos disparados por esse agente poderiam realizar, por exemplo, a reconfiguração de um *firewall*. A fig. 3.4 apresenta um caso onde o agente de ação é utilizado.

A arquitetura foi projetada com o objetivo de atender aos requisitos das cinco áreas funcionais de gerenciamento FCAPS (*Fault*, *Configuration*, *Accounting*, *Performance* e *Security*). Ao observá-la com o foco voltado ao gerenciamento de segurança, a arquitetura seria classificada como um sistema de detecção de intrusão baseado em rede, já que os dados são coletados diretamente da rede, sem a necessidade de haver algum agente sendo executado nos hosts monitorados. Se a classificação for pelo método empregado, pode-se classificar como um sistema de detecção de intrusão baseado em assinatura e em anomalia, uma vez que se pode empregar assinaturas conhecidas para reconhecimento de invasões, bem como criar traços que representem situações anormais (anômalas). O próprio gerente intermediário pode realizar a detecção por anomalia, como, por exemplo, a ocorrência de um determinado traço em um horário não usual.

3.1 A Linguagem PTSL

Esta seção apresenta a linguagem para representação de traços de protocolos denominada PTSL (*Protocol Trace Specification Language*). Uma especificação PTSL é parte da descrição de um cenário de ataque. O ataque é modelado como

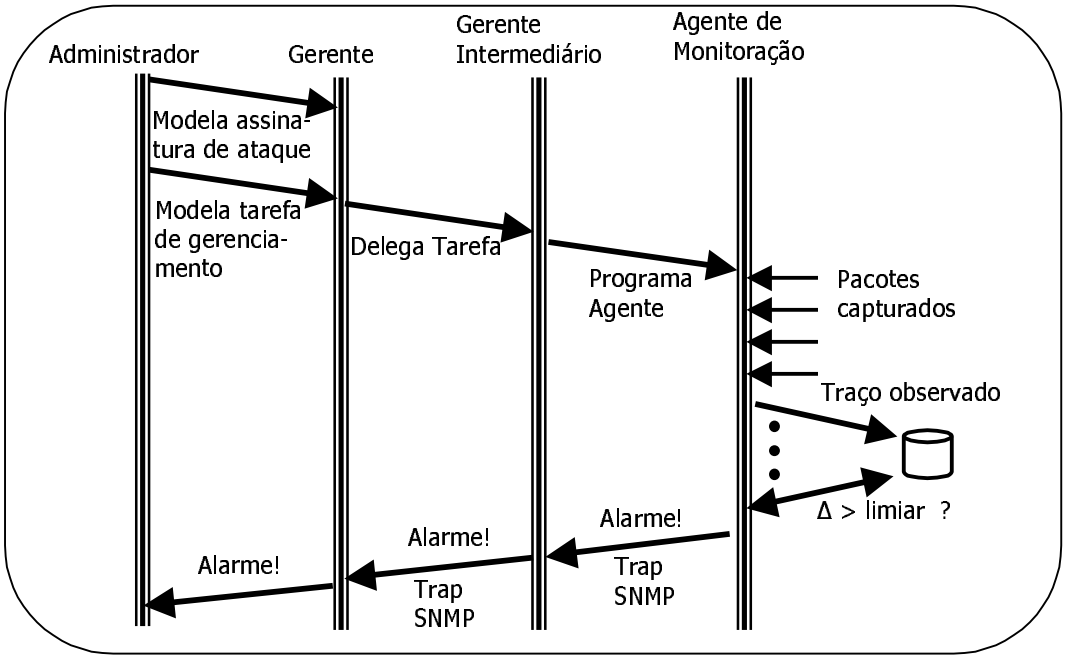

FIGURA 3.3 – Fluxo de informações para programar uma nova assinatura de ataque (via *trap SNMP*)

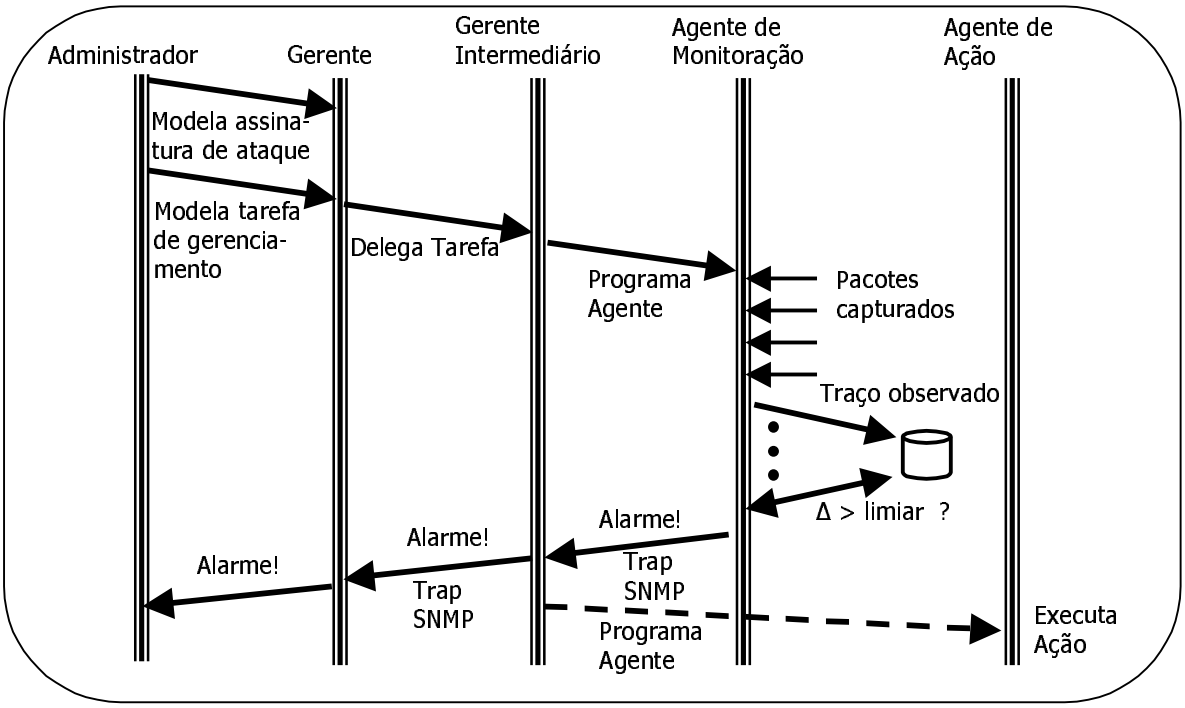

FIGURA 3.4 – Fluxo de informações para programar uma nova assinatura de ataque com agente de ação

uma seqüência de passos que sinalizam uma atividade típica de um atacante, ou uma atividade anormal dentro de um determinado contexto. Embora a linguagem tenha sido desenvolvida originalmente para descrever protocolos de alto nível, ela também é adequada para descrever os passos que compõem um ataque. Na taxonomia utilizada no trabalho que descreve a arquitetura Trace [GAS 2002], PTSL descreve traços de protocolos. Através da descrição de traços de protocolos, é possível a caracterização de um cenário de ataque, e, portanto, avançar com o processo de detecção do ataque (ou de intrusão). É importante salientar que um ataque será caracterizado por traços que ocorram uma ou mais vezes, ou por traços combinados.

A especificação PTSL por si nem sempre é capaz de descrever completamente um cenário de ataque, uma vez que a detecção do ataque pode depender da interpretação de outra entidade. Geralmente esta entidade será o gerente intermediário, apresentado anteriormente. Em outras situações, a interpretação final será realizada pelo próprio administrador da rede. Por exemplo, um ataque que seja descrito pela ocorrência de um pacote com uma determinada característica, fica plenamente descrito pela linguagem. Basta observar o número de traços observados e a ocorrência simples de um traço sinaliza o ataque. Um outro caso seria um ataque descrito pela ocorrência de vários pacotes com uma determinada característica em um período de tempo (varredura de portas, por exemplo). Neste caso, quem avalia a situação como sendo um ataque ou não será o gerente intermediário. Como a descrição dos passos que compõem o ataque em PTSL corresponde a parte mais crítica do processo, neste texto os ataques serão referenciados como plenamente descritos pela linguagem.

A abordagem utilizada na linguagem é de uma máquina de estados finita. Embora PTSL seja uma linguagem baseada em texto, pode-se utilizar diagramas de estado e transição para representar graficamente a máquina de estados finita. Por esta razão, a linguagem foi dividida em `Graphical` e `Textual PTSL`. As linguagens não são equivalentes. A linguagem textual permite a representação completa de um traço, incluindo a especificação da máquina de estados e os eventos que provocam as transições. A linguagem gráfica, por sua vez, equivale a um sub-conjunto da textual, oferecendo a possibilidade de representar pictoricamente a máquina de estados e, em um alto grau de abstração, de identificar as transições. É possível especificar um traço completo, de forma textual, sem fazer uso de `Graphical PTSL`. O contrário não é possível, uma vez que não há como mapear toda a sintaxe textual para a sintaxe gráfica.

A linguagem permite representar um ataque como uma composição de estados e transições. O estado corresponde a um dos passos que compõem o ataque. A transição corresponde a uma certa combinação de características do pacote coletado que leva a máquina de estados para um novo estado.

Um ataque descrito em PTSL sempre possuirá um estado inicial e um estado final. O estado inicial representa a condição de espera da máquina de estado. O estado final representa a condição de traço observado. Quando um traço completo for observado, uma `MIB RMON2` será atualizada. A atualização da `MIB` está descrita na seção 5.4.

As transições geram o caminhamento pela máquina de estados. Os estados possuem transições incidentes de e para outros estados. Cada pacote capturado da rede pode gerar uma transição em um traço. O evento que gera uma transição é uma determinada combinação de valores observados no pacote. Esta combinação de valores caracteriza uma mensagem. Portanto, através da ocorrência de mensagens

se dá o caminhamento pela máquina de estados. As mensagens podem ainda ser especificadas como provenientes do cliente ou do servidor. Uma descrição mais aprofundada destes conceitos será exposta nos capítulos seguintes.

A seguir são apresentadas as linguagens. `Graphical PTSL` ou `G-PTSL` é introduzida primeiro. Em seguida, é a vez da linguagem `Textual PTSL` ou `T-PTSL`. Para esta última acrescentou-se às explicações a gramática textual concreta da mesma. As duas linguagens serão descritas conforme a especificação original voltada para a monitoração de protocolos de alto nível [GAS 2002]. No capítulo 4 serão descritas as extensões propostas à linguagem, para suportar de forma adequada o processo de detecção de intrusão.

3.2 A Linguagem Graphical PTSL

A linguagem gráfica, denominada `Graphica PTSL`, permite a visualização parcial de um traço de protocolo. Em verdade, `Graphical PTSL` é uma representação gráfica do traço, facilitando a visualização e a compreensão do traço. Trata-se de um sub conjunto da linguagem textual que permite a visualização da máquina de estados e de algumas informações relativas ao traço. Na prática, a especificação PTSL será escrita em notação textual e visualizada em notação gráfica.

3.2.1 Representação de Informações para Catalogação e Controle de Versão

A notação gráfica oferece um construtor onde são incluídas informações sobre o traço, que são relevantes para a catalogação e o controle de versão das especificações (fig. 3.5). As informações armazenadas para um traço são:

- `Version`: versão da especificação;
- `Description`: breve comentário a respeito do traço;
- `Key`: palavras-chave relacionadas ao traço;
- `Owner`: identificação do indivíduo que especificou o traço;
- `Last Update`: data e hora da última atualização da especificação.

Além dessas informações, existe o campo `Port`, usado para indicar a porta `TCP` ou `UDP` do protocolo sendo monitorado. Este campo só tem significado se o traço for restrito a apenas um protocolo de aplicação.

3.2.2 Representação de Estados e Mensagens

Os elementos básicos que fazem parte da especificação de um traço são `state`, `client message` e `server message`:

- `State`: descreve o estado atual de um processo cliente;
- `Client message`: evento responsável por fazer o processo cliente mudar de estado; este evento é provocado pelo próprio processo cliente, possivelmente pelo envio de uma mensagem para o servidor;

- `Server message`: evento responsável por fazer o processo cliente mudar de estado; este evento é provocado pelo processo servidor, possivelmente em uma mensagem de resposta a uma requisição prévia;

- `Agrupamento`: quando duas ou mais mensagens distintas podem causar a transição de um mesmo estado para outro, elas são agrupadas. Nesse caso, a transição ocorrerá se qualquer uma das mensagens agrupadas for observada na rede.

Um traço é composto por estados e mensagens. O traço inicia por um estado especial, denominado *idle*, que corresponde ao estado de espera da máquina de estados. Através da ocorrência de mensagens, acontece o caminhamento pela máquina de estados, passando pelos demais estados especificados, até que se atinja o estado final, quando será caracterizada a ocorrência de um traço.

A fig. 3.5 ilustra um traço, definido por dois estados (*idle* e *2*), a mensagem do cliente `SYN` e a mensagem do servidor `RST`. Os estados são representados por círculos identificados; as mensagens do cliente são representadas por setas contínuas e as mensagens oriundas do servidor são representadas por setas pontilhadas. Nesse exemplo, os estados inicial e final são o mesmo (*idle*). Como é possível observar, este último é representado graficamente por dois círculos concêntricos. O traço ilustrado permite monitorar os acessos a portas não relacionadas a serviços. Este seria um traço típico usado para a detecção de varredura de portas.

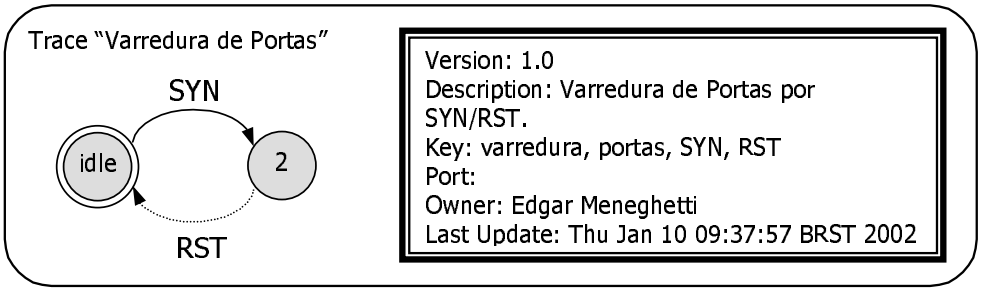

FIGURA 3.5 – Representação gráfica de um traço

Os textos que identificam as mensagens ou agrupamentos indicam, em um alto grau de abstração, o evento que precisa ocorrer para que a máquina de estados evolua. Na representação gráfica, contudo, não é possível descrever em detalhe o que causa a transição, sendo necessário assim complementar a especificação de cada mensagem ou agrupamento usando a notação textual (descrita na seção 3.3).

O agrupamento de mensagens possibilita a transição de um estado para outro pela ocorrência de alguma das mensagens especificadas. A fig. 3.6 representa um traço igualmente usado para varredura de portas, porém mais abrangente. Em [DON 98] são descritos vários métodos usados para varredura de portas. Um destes métodos descritos consiste em enviar um pacote `TCP` com a *flag* `FIN` ligada e aguardar pela resposta com a flag `RST` ligada, segundo o comportamento esperado definido pela RFC 793.

Na especificação do traço em PTSL não se definem os endereços relativos ao cliente ou servidor. Cada ocorrência de uma mensagem que leve a máquina de estados a sair do estado de repouso (idle), gera uma nova instância deste traço que

contém, entre outras informações, os endereços dos hosts envolvidos. Uma discussão mais aprofundada sobre este mecanismo pode ser encontrada no capítulo 5.1 que descreve a arquitetura.

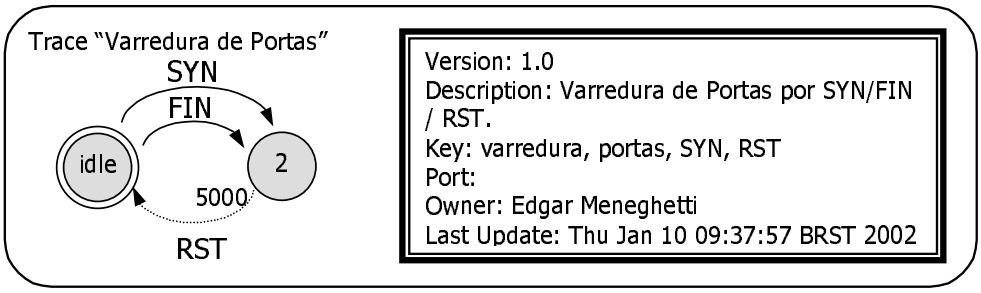

FIGURA 3.6 – Representação gráfica de um traço com agrupamento de mensagens

3.2.3 Representação de Temporizadores

Da forma como foram apresentadas até o momento, as transições não possuem limite de tempo para ocorrer. Para determinar um *timeout* para uma transição, é preciso associar um valor (em mili-segundos) a ela. No exemplo ilustrado na fig. 3.7, a transição do estado 2 para o estado inicial deve ocorrer em até 5 segundos (após a observação na rede da primitiva `SYN`). Se esse tempo for excedido, a monitoração do traço em questão é cancelada.

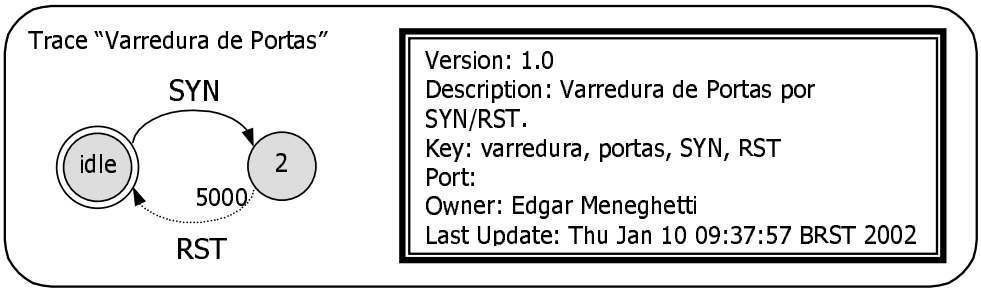

FIGURA 3.7 – Representação gráfica de um traço com temporizador

3.3 A Linguagem Textual PTSL

A linguagem textual, denominada `Textual PTSL`, permite a definição completa de um traço de protocolo. Ao longo desta seção são apresentados exemplos e a gramática textual correspondente à sintaxe da linguagem. Para tal, foi utilizada uma *BNF* simplificada, similar à utilizada por Almeida em [ALM 94].

3.3.1 Unidades Léxicas Utilizadas na Descrição da Linguagem

As unidades léxicas também podem ser chamadas de símbolos terminais da sintaxe textual. `Textual PTSL` possui os seguintes símbolos terminais:

```
  <Char> ::= qualquer caracter ASCII (octetos 0-127)
  <Char-sp> ::= qualquer caracter ASCII, exceto espaço
  <UpAlpha> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O
| P | Q | R | S | T | U | V | W | X | Y | Z
  <LoAlpha> ::= a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
| p | q | r | s | t | u | v | w | x | y | z
  <Alpha> ::= <UpAlpha> | <LoAlpha>
  <Digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
  <CR> ::= caracter ASCII, carriage return (13)
  <LF> ::= caracter ASCII, linefeed (10)
  <SP> ::= caracter ASCII, espaço (32)
  <UL> ::= caracter ASCII, underline (95)
  <DQ> ::= caracter ASCII, aspas duplas (34)
  <DT> ::= caracter ASCII, ponto final (46)
  <SL> ::= caracter ASCII, barra (47)
  <WildCard> ::= caracter ASCII, asterisco (42)
  <CRLF> ::= <CR> <LF>
  <Informal text> ::= <Char>+
  <Double quotted informal text> ::= <DQ> <Informal text> <DQ>
  <Protocol name> ::= <Alpha> [<Alpha>]*
  <Protocol encapsulation> ::= <Protocol name> [<SL><Protocol
name>]*
  <Keyword> ::= <Alpha> [<Alpha> | <SP>]*
  <Identifier> ::= <Alpha> [<Alpha> | <Digit> | <UL> | <SP>]*
  <Double quotted identifier> ::= <DQ> <Identifier> <DQ>
  <Verb identifier> ::= <Char-sp>+
  <Unsigned int> ::= qualquer número inteiro entre 0 e 65.535
  <Version number> ::= <Digit>+ <DT> <Digit>+
  <Date and time> ::= Wkday ``,'' SP Date SP Time SP ``GMT''
  <Wkday> ::= ``Mon'' | ``Tue'' | ``Wed'' | ``Thu'' | ``Fri'' |
``Sat'' | ``Sun''
  <Date> ::= <Digit><Digit> SP Month SP
<Digit><Digit><Digit><Digit>
  <Month> ::= ``Jan'' | ``Feb'' | ``Mar'' | ``Apr'' | ``May'' |
``Jun'' | ``Jul'' | ``Aug'' | ``Sep'' | ``Oct'' | ``Nov'' | ``Dec''
  <Time> ::= <Digit><Digit> ``:'' <Digit><Digit> ``:''
<Digit><Digit>
```

3.3.2 Organização de uma Especificação Textual

A especificação textual permite a descrição completa de um traço. Um traço é composto por 4 seções:

- Nome do traço e informações para catalogação;
- Descrição das mensagens;
- Descrição dos agrupamentos de mensagens;
- Descrição dos estados e da máquina de estados.

A seção relativa ao agrupamento de mensagens é opcional. As demais são obrigatórias e deverão existir para cada traço especificado. O nome do traço deverá ser único dentro do conjunto de traços existentes em um agente de monitoração. As demais seções fazem parte do contexto apenas do traço a que pertencem. Desta forma, é possível utilizar nomes iguais para definir as estruturas existentes em cada seção. Não é possível reutilizar uma definição de alguma estrutura de um traço em outros traços. Nestes casos, é necessário especificar a estrutura novamente.

3.3.3 Seção de Identificação do Traço

A seção de identificação do traço é a primeira seção que deve ser especificada e é obrigatória. Identifica o traço através de um nome e, opcionalmente, fornece informações para catalogação e controle de versão. O formato é semelhante ao adotado na notação gráfica.

A gramática desta seção está descrita a seguir. A fig. 3.8 exibe um exemplo de especificação desta seção.

```
  TraceSpecification ::= Trace <SP> <Double quotted identifier>
<CRLF>
                              [<Header>]
  <Header> ::= [Version:  <Version number> <CRLF>]
                   [Description:  <Informal text> <CRLF>]
                   [Key:  <Keyword> , <Keyword>* <CRLF>]
                   [Port:  <Unsigned int> <CRLF>]
                   [Owner:  <Informal text> <CRLF>]
                   [Last Update:  <Date and time> <CRLF>]
```

```
1  Trace ‘‘Varredura de Portas’’
2  Version:  1.0
3  Description:  Varredura de Portas por SYN/RST.
4  Key:  varredura, portas, SYN, RST
5  Port:
6  Owner:  Edgar Meneghetti
7  Last Update:  Thu Jan 10 09:37:57 BRST 2002
8  •••
9  EndTrace
```

FIGURA 3.8 – Exemplo de especificação da seção de identificação do traço

3.3.4 Descrição das Mensagens

A seção de descrição das mensagens define as transições que causarão o caminhamento pelos diversos estados que compõem o traço. Esta seção é obrigatória. Através desta seção, será possível definir quais características devem ser avaliadas em cada pacote. A linguagem fornece três abordagens para isolar a característica desejada no pacote capturado: localização absoluta de bits, localização relativa a campos separados por espaços em branco e localização de seqüência de caracteres

em posição livre. Além disso, a linguagem permite a especificação da origem da mensagem, ou seja, se a mensagem está sendo gerada pelo cliente ou pelo servidor. Esta notação é meramente uma convenção, visto que ela é usada apenas para identificar as duas estações participando da comunicação, não importando se elas são de fato clientes ou servidores.

A gramática desta seção está descrita a seguir.

```
  <MessagesSection> ::= MessagesSection
                        <MessagesSectionContent>
                        EndMessagesSection
  <MessagesSectionContent> ::= <MessageContent>+
  <MessageContent> ::= Message <SP> <Double quotted identifier>
<CRLF>
                       MessageType:  {client | server} <CRLF>
                       [MessageTimeout:  <Unsigned int> <CRLF>]
                       {<FieldCounter> | <BitCounter> |
<NoOffset>}+
                       EndMessage <CRLF>
  <FieldCounter> ::= FieldCounter <SP>
                     <Protocol encapsulation> <SP>
                     <Unsigned int> <SP>
                     {<WildCard>|<Verb identifier>} <SP>
                     <Double quotted informal text> <CRLF>
  <BitCounter> ::= BitCounter <SP>
                   <Protocol encapsulation> <SP>
                   <Unsigned int> <SP>
                   <Unsigned int> <SP>
                   {<WildCard>|<Verb identifier>} <SP>
                   <Double quotted informal text> <CRLF>
  <NoOffset> ::= NoOffset <SP>
                 <Protocol encapsulation> <SP>
                 <Verb identifier> <SP>
                 <Double quotted informal text> <CRLF>
```

A mensagem é composta por uma declaração do tipo de mensagem, do tempo em que ela deve ocorrer medido em mili-segundos e uma ou mais abordagens para localização de dados em um pacote. A seguir estão descritos com mais detalhes estes ítens.

- `MessageType`: pode ser declarado como *client* ou *server*. Como já foi mencionado anteriormente, identifica as duas estações participando do traço, através do endereço IP de cada uma delas;

- `MessageTimeout`: estipula o tempo máximo em mili-segundos para que esta mensagem ocorra, após o estado da qual ela parte ter sido atingido. Se este tempo for excedido, o traço é abortado, fazendo com que a máquina de estados retorne ao estado *idle*. É importante salientar que este tempo é relativo ao par de estações participando do traço, sendo possível que haja vários traços iguais ocorrendo entre pares de estações diferentes, em tempos diferentes;

- `FieldCounter`: usada em protocolos baseados em caracter, esta abordagem trata uma mensagem como um conjunto de primitivas separadas por espaços em branco; a identificação de um campo, nesse, caso, ocorre pela posição do mesmo na mensagem. Os demais argumentos são, respectivamente, o encapsulamento, o deslocamento, a string a ser comparada e uma breve descrição da mensagem;

- `BitCounter`: usada em protocolos binários; a identificação de um campo é determinada por um offset em bits a partir do início do cabeçalho do protocolo em questão até o início do campo desejado; além da posição inicial do campo, é preciso indicar ainda o número de bits que o campo ocupa Os demais argumentos são, respectivamente, o encapsulamento, o deslocamento em base decimal, o número de bits em base decimal, a seqüência de bits a ser comparada e uma breve descrição da mensagem;

- `NoOffset`: usada para localizar seqüências de caracteres (strings) em um determinado protocolo. O demais argumentos são, respectivamente, o encapsulamento, a string a ser comparada e uma breve descrição da mensagem; Protocol encapsulation: identifica onde o campo deve ser buscado; um encapsulamento do tipo Ethernet/IP/`TCP` indica que o campo deve ser buscado no campo de dados (*payload*) do protocolo `TCP`;

A fig. 3.9 apresenta um fragmento do traço da fig. 3.5 usando a linguagem textual. A abordagem para localização de dados no pacote é `BitCounter`. A mensagem do exemplo isola um bit localizado na posição *110* da área de dados do `IP` (cabeçalho do protocolo `TCP`) e o compara com o valor do quinto campo, que no exemplo é igual a um.

```
1  MessagesSection
2  Message ‘‘SYN’’
3  MessageType:  client
4  // OffsetType Encapsulation FieldNumber Verb Description
5  BitCounter Ethernet/IP 110 1 1 ‘‘Campo SYN/TCP’’
6  EndMessage
7  • • •
8  EndMessagesSection
```

FIGURA 3.9 – Descrição de mensagens usando PTSL textual

3.3.5 Seção de Agrupamento de Mensagens

Esta seção oferece a possibilidade de agrupar duas ou mais mensagens em uma única transação. É equivalente a uma operação booleana “OU” entre as mensagens, assim como as declarações da seção anterior (`bitcounter`,`fieldcounter`,`nooffset`) têm entre si uma semântica de “E”. Esta seção é opcional e pode ser descrita de forma equivalente por outros meios, porém perdendo a legibilidade e a otimização feita dentro do agente de monitoração.

A gramática desta seção está a seguir.

```
  <GroupsSectionContent> ::= <GroupContent>+
  <GroupContent> ::= Group <SP> <Double quotted identifier>
<CRLF>
                       Messages:  <Double quotted identifier>
                       {, <Double quotted identifier>}* <CRLF>
                       EndGroup <CRLF>
```

Para exemplificar esta seção, foram definidas duas mensagens e deseja-se que a ocorrência de qualquer uma das duas ocasione uma transição na máquina de estados. Este fragmento de um traço corresponde ao exemplo da fig. 3.10. A declaração do agrupamento está apresentado entre as linhas 17 e 19 da mesma figura.

```
 1  MessagesSection
 2  // definição de uma mensagem
 3  Message ‘‘SYN’’
 4  MessageType:  client
 5  // OffsetType Encapsulation FieldNumber Verb Description
 6  BitCounter Ethernet/IP 110 1 1 ‘‘Campo SYN/TCP’’
 7  EndMessage
 8  // definição de uma mensagem
 9  Message ‘‘FIN’’
10  MessageType:  client
11  // OffsetType Encapsulation FieldNumber Verb Description
12  BitCounter Ethernet/IP 110 1 1 ‘‘Campo SYN/TCP’’
13  EndMessage
14  EndMessagesSection
15  // agrupamento de duas mensagens
16  GroupsSection
17  Group ‘‘SYN || FIN’’
18  Messages:  ‘‘SYN’’, ‘‘FIN’’
19  EndGroup
20  EndGroupsSection
```

FIGURA 3.10 – Agrupamento de mensagens

3.3.6 Seção de Definição de Estados e da Máquina de Estados

Nesta seção são especificados os estados que compõem o traço, assim como quais as mensagens que geram o caminhamento pela máquina de estados. Esta seção é obrigatória, e deve conter pelo menos um estado inicial denominado "*idle*" e uma declaração especificando o estado final da máquina de estados. O estado final pode ser o próprio estado inicial. A gramática desta seção está apresentada a seguir.

```
  <StatesSectionContent> ::= FinalState <SP> <Identifier> <CRLF>
                               <StateContent>+
  <StateContent> ::= State SP <Identifier> <CRLF>
                       <Double quotted identifier> <SP> GotoState
<SP>
                       <Identifier>+ <CRLF>
```

```
EndState <CRLF>
```

A máquina de estados do traço ilustrado na fig. 3.5 pode ser mapeada para a especificação textual descrita na fig. 3.11. O estado final é identificado logo após o início da seção `StatesSection` (linha 2). Os estados *idle* e 2 são definidos, respectivamente, nas linhas 3 a 5 e 6 a 8. A especificação de um estado resume-se em listar os eventos (mensagens e agrupamentos) que podem provocar uma transição e indicar, para cada um deles, o próximo estado (linhas 4 e 7).

```
1  StatesSection
2  FinalState idle
3  State idle
4  ‘‘SYN’’ GotoState 2
5  EndState
6  State 2
7  ‘‘RST’’ GotoState idle
8  EndState
9  EndStateSection
```

FIGURA 3.11 – Especificação de estados e da máquina de estados

3.4 Exemplo de Especificação Textual em PTSL

Esta linguagem foi usada, no âmbito deste trabalho, para especificar traços que caracterizam ataques (assinatura de ataque). O exemplo da fig. 3.12 mostra uma especificação textual de um traço a ser observado, e que indicaria a ocorrência de um ataque do tipo varredura de portas. O traço apresentado no exemplo é equivalente ao apresentado na fig. 3.10 em notação gráfica.

Este exemplo mostra a especificação das mensagens “`SYN`”, “`FIN`” e “`RST`” (linhas 7 a 23) isolando o bit que corresponde as flags homônimas no protocolo `TCP`. As linhas 24 a 27 apresentam o agrupamento das mensagens “`SYN`” e “`FIN`”, nomeando este agrupamento de “`SYN || FIN`”. O nome do agrupamento sugere que existe uma operação lógica OU entre as mensagens, o que corresponde a semântica do agrupamento de mensagens. Entre as linhas 28 e 36 estão declarados os estados e as respectivas transições. O estado inicial ou de repouso é definido através da declaração do estado com nome *idle*, assim como o estado final é declarado na linha 30.

```
1  Version:  1.0
2  Description:  Varredura de Portas por SYN/RST.
3  Key:  varredura, portas, SYN, RST
4  Port:
5  Owner:  Edgar Meneghetti
6  Last Update:  Thu Jan 10 09:37:57 BRST 2002
7  MessagesSection
8  Message ‘‘SYN’’
9  MessageType:  client
10 // OffsetType Encapsulation FieldNumber Verb Description
11 BitCounter Ethernet/IP 110 1 0 ‘‘Campo SYN/TCP’’
12 EndMessage
13 Message ‘‘FIN’’
14 MessageType:  client
15 // OffsetType Encapsulation FieldNumber Verb Description
16 BitCounter Ethernet/IP 111 1 1 ‘‘Campo FIN/TCP’’
17 EndMessage
18 Message ‘‘RST’’
19 MessageType:  client
20 // OffsetType Encapsulation FieldNumber Verb Description
21 BitCounter Ethernet/IP 109 1 1 ‘‘Campo RST/TCP’’
22 EndMessage
23 EndMessagesSection
24 GroupsSection
25 Group ‘‘SYN || FIN’’
26 Messages:  ‘‘SYN’’, ‘‘FIN’’
27 EndGroup
28 StatesSection
29 FinalState idle
30 State idle
31 ‘‘SYN || FIN’’ GotoState 2
32 EndState
33 State 2
34 ‘‘RST’’ GotoState idle
35 EndState
36 EndStatesSection
```

FIGURA 3.12 – Exemplo completo de especificação textual

4 Análise da Linguagem PTSL para Fins de Detecção de Intrusão

A linguagem PTSL, em sua versão gráfica e textual, possibilita a modelagem de vários ataques, abrangendo todos os níveis de uma arquitetura de redes modernas. Entretanto, muitos modelos de ataque são impossíveis de serem descritos sem que se altere ou acrescente algumas funcionalidades à especificação original da linguagem. Este capítulo apresenta um estudo dos limites impostos pela linguagem atual e as extensões propostas à linguagem para aumentar o escopo de ataques possíveis de serem modelados.

É importante salientar que a implementação (descrita no capítulo 5) do agente de monitoração é onerosa computacionalmente. Desta forma, extensões à linguagem devem causar um impacto mínimo ao desempenho do agente de monitoração. Tendo em vista esta limitação, foram propostas extensões que satisfizessem os requisitos para modelagem de ataques, mas que possam ser implementadas de forma simples e eficiente.

4.1 Modelagem de Ataques Usando uma Máquina de Estados Finita

A arquitetura Trace tem como característica forte prover mecanismos que possibilitem o gerenciamento distribuído de protocolos de alto nível. Um estudo mais aprofundado de suas características evidencia que ela também pode ser utilizada para outros fins. Os requisitos básicos a um sistema de detecção de intrusão são atendidos pela arquitetura, que ainda oferece alguns benefícios quando comparadas com outras soluções, conforme foi mencionado no capítulo 2.

A linguagem PTSL faz parte da arquitetura Trace e foi proposta para permitir que gerentes de rede possam definir os traços de protocolos que devem ser monitorados. É uma linguagem concebida para ser facilmente assimilada pelo gerente, mas oferece flexibilidade na modelagem. O processo de análise da viabilidade de utilizar esta linguagem dentro de um sistema de detecção de intrusão iniciou-se pela modelagem de ataques descritos na literatura [NOR 2001, CAM 2001, DON 98, BAC 2001], no intuito de evidenciar possíveis problemas relacionados a funcionalidade da linguagem. A abordagem utilizada, baseada em uma máquina de estados finita, mostrou-se eficiente e simples, possibilitando a modelagem de vários ataques. Porém, alguns ataques importantes no contexto de detecção de intrusão não puderam ser descritos. Para aumentar a abrangência da ferramenta, algumas extensões foram propostas e serão descritas neste capítulo.

Um dos principais problemas observado em sistemas de detecção de intrusão está ligado a ocorrência de falsos positivos e falsos negativos em demasia. Este fato está ligado à forma como os IDSs fazem a análise dos dados e fornecem o diagnóstico. Se houver alguma característica nos dados que indique atividade maliciosa ou anômala, então é lançado um alarme. Essa característica procurada nos dados é a assinatura do ataque. Em geral, a assinatura representa apenas o início do processo de ataque ou parte dele, e não uma seqüência de passos que compõe o ataque. Um alarme gerado pela detecção de um indício de ataque não pode ser con-

siderado como um ataque completo e, muitas vezes, nem sequer como um ataque em andamento. Se o ataque puder ser modelado como uma seqüência de passos que o atacante iria realizar para comprometer a segurança de um sistema de computação, então o sistema fará a detecção de intrusão com maior exatidão, sinalizando menos falsos positivos e negativos [VIG 2000].

O problema mencionado acima está intimamente relacionado com o pequeno poder de expressão das linguagens disponíveis para modelar assinaturas de ataques. Em geral, uma assinatura preocupa-se em observar campos de um pacote. A observação de muitos pacotes `TCP` com o flag `RST` ligado é um exemplo de assinatura usada por diversas ferramentas para detectar um dos tipos de sondagem de porta. No entanto, `TCP RST` não é gerado apenas por uma estação que não possui determinado serviço disponível ao receber uma requisição de conexão; esse mesmo tipo de pacote é usado por uma estação para reiniciar uma conexão em andamento. Como as ferramentas não correlacionam pacotes, não conseguem distinguir entre pacotes `TCP RST` que representam sondagem de portas e os que são usados ao longo de uma conexão convencional, gerando alarmes em ambos os casos.

Uma abordagem baseada em estados permite uma descrição mais precisa do ataque. Ao descrever o ataque de forma mais precisa, o número de falsos positivos e falsos negativos é reduzido drasticamente. Como forma de identificar as funcionalidades ausentes na linguagem PTSL, foram modelados vários cenários de ataques, descritos parcialmente no anexo 1. Para fins de uma melhor compreensão da extensão proposta na linguagem, algumas modelagens que apresentam problemas de representação são apresentadas nas próximas seções.

4.1.1 Sondagem de Portas

Um passo importante executado pelo atacante quando em um ataque, é a determinação de quais serviços o equipamento alvo oferece. Geralmente, a forma como se determina os serviços oferecidos é através do teste de quais portas estão ativas. Este processo é realizado através do envio de um pacote com determinadas características e pela espera da resposta do equipamento alvo. Baseado no comportamento observado, o atacante conclui se a porta está ativa ou não. Várias técnicas foram descritas [DON 98] para realizar esta varredura, sendo que as técnicas mais simples são facilmente identificáveis pela maior partes dos IDSs.

A fig. 3.5 ilustra um traço que permite monitorar a ocorrência de varredura de portas. No caso do `TCP`, ao enviar um pacote com o bit `SYN` ligado, a resposta esperada é um pacote com os bits `SYN` e `ACK` ligados, caso aquela porta possua um serviço associado. Caso não haja nenhum serviço nessa porta, a resposta será um pacote `TCP` com o bit `RST` ligado. Desta forma, ao modelar uma situação onde ocorrem pacotes `TCP` com o bit `SYN` ligado seguidos de pacotes com o bit `RST` ligado, a ocorrência em demasia desta situação caracteriza uma varredura de portas.

Outras técnicas de varredura de portas são facilmente descritas através de PTSL. Por exemplo, a técnica na qual é enviado um pacote `TCP` com os bits `SYN` e `ACK` ligado e espera-se um pacote de resposta com o bit `RST` ligado (caso a porta não esteja associada a algum serviço) pode ser especificada de forma similar. Essa técnica também é conhecida por mapeamento reverso, visto que as respostas obtidas do alvo correspondem às portas que não possuem serviços (por dedução, as demais portas possuem). A fig. 3.6 exibe este traço.

Os artifícios para encobrir a varredura de portas, tais como o uso de uma

seqüência aleatória ou com intervalos de tempo grandes, são problemas tratados pontualmente pelas ferramentas para detecção de intrusão. No caso dos traços modelados anteriormente, esse problema não se aplica. A seqüência de números de portas não é importante para a ocorrência do traço, assim como o intervalo entre uma conexão e outra. Esse último problema deverá ser endereçado pelo gerente intermediário, no sentido de definir quais são os limites aceitáveis de ocorrência deste traço em um período de tempo.

4.1.2 Comportamento Anômalo

A fig. 4.1 ilustra um caso de comportamento anômalo, onde o pacote de início da conexão `TCP` (`SYN`) carrega dados [NOR 2001]. Este recurso é utilizado para enganar algum sistema de segurança que procure pela ocorrência de determinadas palavras-chave no *payload* do `TCP`, mas que em geral não fazem essa verificação no *handshake*. Para a estação alvo, os dados que vierem após o *handshake* serão adicionados aos dados presentes no pacote inicial.

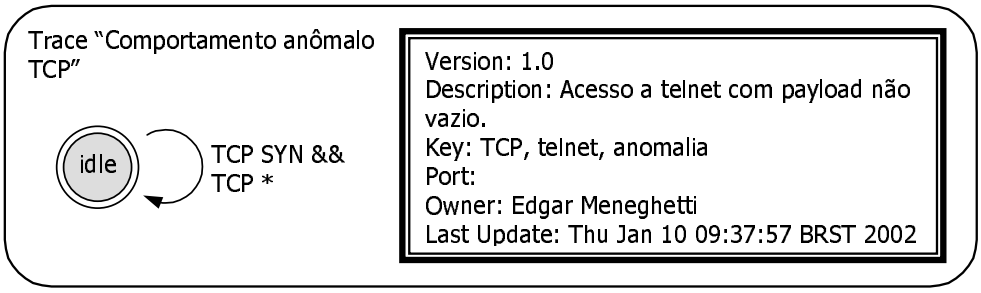

FIGURA 4.1 – Comportamento anômalo

A especificação deste ataque necessita de apenas um estado e uma transição. A transição é ocasionada pela observação de um pacote endereçado a porta 23 e com o bit `SYN` ligado em um pacote `TCP`. Além desta combinação, é necessário testar a existência de algum dado sendo carregado na área de dados do `TCP` (protocolo `TELNET`). A notação original de PTSL propõe a colocação de um asterisco como indicador da existência de um ou mais caracteres no campo especificado. O asterisco tem sido usado universalmente como um caracter coringa, indicando a ocorrência de zero ou mais caracteres de qualquer natureza (e não um ou mais caracteres). Uma mudança na semântica deste caractere seria prejudicial ao administrador que esteja especificando os cenários de ataques. Uma solução mais correta seria especificar limiares de tamanho máximo e mínimo aceitáveis para o pacote em questão. Limiares são tratados pelos operadores relacionais, apresentados na seção 4.5.

4.1.3 Ataques de Negação de Serviço

Os ataques de negação de serviços entraram em evidência no início de 2000, quando alguns servidores WEB de empresas conhecidas ficaram inoperantes por algum tempo devido a um ataque desta natureza. Embora os ataques de negação já fossem conhecidos a bastante tempo, nunca tinham sido utilizados de forma coordenada e maciça como ocorreu naquela data. Este tipo de ataque tem como propósito final afetar o funcionamento normal de um serviço. Em muitas situações, ocorre o travamento do sistema.

O ataque conhecido por `Land` [NOR 2001] aproveita uma falha em alguns sistemas operacionais da *Microsoft* (*Windows 95* e *Windows NT*), ocasionando lentidão e, ocasionalmente, uma falha geral no mesmo. Trata-se, portanto, de um ataque de negação de serviço clássico. Esta falha já está corrigida em versões mais recentes. O ataque consiste, basicamente, em enviar um pacote `TCP` com o bit `SYN` ligado e com os endereços de origem e destino iguais. Algumas variações incluem portas de origem e destino também iguais.

Este ataque não pode ser descrito pela linguagem. Como é necessária uma comparação entre valores existentes em um mesmo pacote, os operadores e funções existentes na linguagem não fornecem nenhum mecanismo que suporte esta comparação. Assim como neste caso específico existe essa necessidade, observa-se que esta funcionalidade é necessária na descrição de outros ataques. A fig. 4.2 procura exibir uma possível modelagem para este ataque, embora a transcrição deste ataque para a notação textual não seria possível.

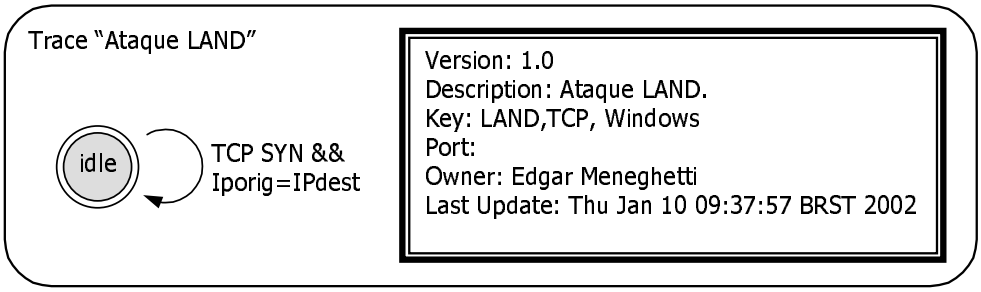

FIGURA 4.2 – Ataque `Land`

Outro exemplo de ataque de negação de serviço onde se faz necessário este tipo de comparação ocorre quando é enviado um pacote `ICMP` do tipo `ICMP-redirect` apontando o próximo roteador como o próprio equipamento sofrendo o ataque [NOR 2001]. Em alguns sistemas operacionais, tipicamente *Microsoft Windows 95 e 98*, este tipo de pacote causa o travamento do sistema. A fig. 4.3 apresenta uma modelagem parcial deste ataque, embora também não seja possível transcrever o ataque para a notação textual.

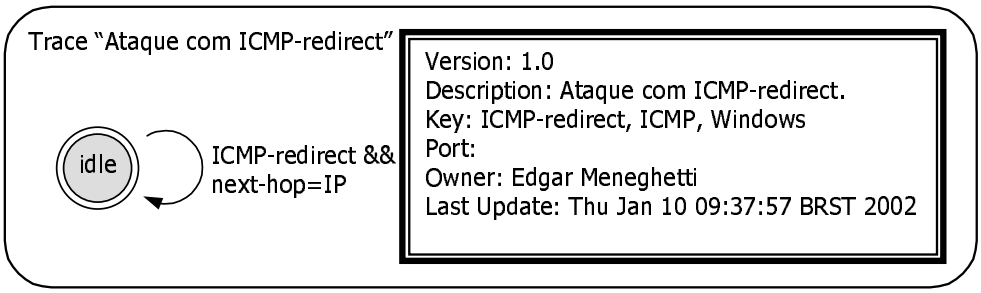

FIGURA 4.3 – Ataque com `ICMP-redirect`

Os ataques de negação distribuídos mais conhecidos, como o *Trinoo*, *TFN* ou *Stacheldraht* [ZIR 2000] são facilmente identificáveis. Como utilizam portas não usuais (porém conhecidas), basta observar a ocorrência desta porta em um segmento de rede para detectar o ataque. O problema maior em relação a este tipo de

ataque está relacionado com a dificuldade em prevenir ou suspender um ataque em andamento, embora este tema fuja ao objetivo deste trabalho.

4.1.4 Ataques no Nível de Aplicação

Uma das maiores contribuições da linguagem PTSL é a possibilidade de modelagem de ataques no nível de aplicação. A maior parte dos sistemas de detecção de intrusão baseados em rede operam até o nível de rede, sendo relativamente limitados em modelar as camadas superiores.

Um exemplo de cenário de ataque no nível de aplicação pode ser descrito pela utilização freqüente do comando *rpcinfo*, disponível em implementações Unix. Esse comando retorna uma relação de processos servidores que aceitam requisições do tipo `RPC` (*Remote Procedure Call*) e pode ser uma informação valiosa do ponto de vista do atacante. O comando baseia-se no envio de uma mensagem ao servidor de conversão de número de programas `RPC` para portas `TCP/UDP` (*portmapper daemon*), solicitando que seja enviado um *dump* de todos os servidores `RPC` disponíveis nessa máquina. Como resposta, o *portmapper* envia uma mensagem contendo uma relação desses servidores e as respectivas portas e números de programa `RPC`. A modelagem deste traço está ilustrada na fig. 4.4.

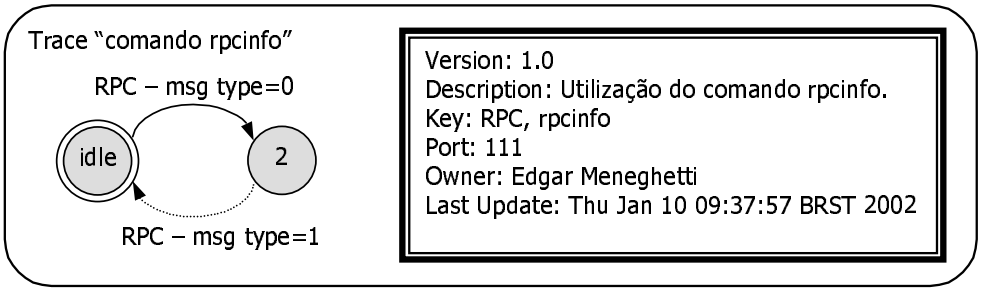

FIGURA 4.4 – Traço de ocorrência do comando *rpcinfo*

É importante salientar que esse traço irá capturar tráfego legítimo também, ou seja, a ocorrência do mesmo, analisado de forma estanque, não tem um significado conclusivo. O sistema `NFS` (*Network File System*) faz uso de rotinas que envolvem um processo idêntico ao descrito no traço. Porém, esse traço só será observado durante a negociação para a montagem de sistemas de arquivos externos e não mais durante o resto do período em que esse sistema de arquivos estiver montado. Dessa forma, a ocorrência desse traço em horários não usuais ou em grande quantidade (varredura de estações que possuam *portmapper* sendo executado, por exemplo), pode ser interpretada como um indicativo de ataque, ou de futuro ataque.

Uma situação comum em redes comprometidas é a existência de estações capturando tráfego em modo promíscuo. Geralmente corresponde a uma estação que sofreu um ataque com êxito, e está sendo utilizada para coletar mais informações sobre o próprio segmento de rede ou para futuras invasões. Algumas técnicas podem ser aplicadas para detectar a existência de uma estação em modo promíscuo. Uma técnica simples consiste em observar consultas de `DNS` (*Domain Name System*) partindo de uma estação que não esteja envolvida na comunicação entre duas outras estações. A fig. 4.5 exemplifica a técnica.

Este tipo de modelagem não pode ser feita em PTSL. Como é necessário a

correlação entre o par de estações se comunicando e a estação capturando tráfego (3 estações), o modelo cliente-servidor adotado pela linguagem não possui recursos para tal especificação.

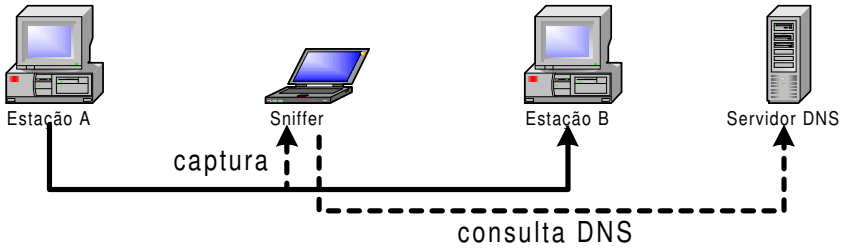

FIGURA 4.5 – Detecção de estação em modo promíscuo

Outra forma de realizar a detecção da existência de um *sniffer* em um segmento de rede seria através do envio de um pacote do tipo ICMP echo request com endereço MAC de destino igual a 255.255.255.255.255.255. Todas as estações do segmento tenderiam a responder, inclusive o próprio *sniffer*. Como o agente de monitoração é uma entidade passiva, seria necessário dota-lo da capacidade de executar procedimentos. Uma solução possível seria a utilização da `MIB SCRIPT` [LEV 2001]. Porém, a linguagem PTSL não possui recursos para tal e a sua implementação demandaria modificações profundas em sua estrutura.

Outros ataques em nível de aplicação estão descritos no anexo 1.

4.1.5 Ataques usando Fragmentação

Um artifício bastante utilizado em redes locais como forma de burlar mecanismos de segurança é fazer uso de pacotes fragmentados. Como a maior parte dos sistemas de detecção de intrusão procura pela ocorrência de marcas em pacotes que signifiquem um indício de ataque, ao fragmentar estes pacotes é possível mascarar esta marca.

A modelagem de ataques que utilizam fragmentação como forma de ocultar o próprio ataque é relativamente difícil de ser realizada. Ptacek em [PTA 98] apresenta vários problemas relacionados a este tipo de detecção. Cita-se como uma das principais dificuldades o problema de simular o comportamento que a estação alvo do ataque deverá ter ao receber pacotes fragmentados. Sabe-se que o comportamento varia de acordo com a implementação da pilha de protocolos `TCP/IP`. Desta forma, a interpretação do detector de intrusão em relação aos efeitos que estes pacotes irão ocasionar na estação alvo podem não serem iguais.

A fragmentação maliciosa também é utilizada em vários ataques conhecidos, tais como *Teardrop*, *Ping o'death* [NOR 2001] e geralmente causam o congelamento ou lentidão em alguns sistemas operacionais mais antigos. O *Teardrop*, por exemplo, envia pacotes com o valor de deslocamento posterior ao endereço de fim do fragmento. Alguns sistemas operacionais se desestabilizam ao tentar remontar estes fragmentos.

A linguagem PTSL oferece poucos recursos para lidar com fragmentação. A maior parte dos ataques que utiliza este artifício não pode ser modelada com os recursos existentes na linguagem.

4.2 Proposta de Extensão à Linguagem

Como descrito nas seções anteriores, constata-se que alguns ataques podem ser modelados com os recursos atuais da linguagem, outros poderiam ser modelados com algumas extensões e uma terceira classe de ataques não podem ser modelados sem uma alteração maior da linguagem.

As seções seguintes apresentam extensões que aumentam a quantidade de cenários de ataques passíveis de serem modelados. Na conclusão, são enumerados alguns trabalhos futuros buscando aumentar ainda mais a quantidade de cenários possíveis de serem descritos.

4.2.1 Novas Unidades Léxicas Utilizadas na Descrição da Linguagem Estendida

As unidades léxicas apresentadas a seguir serão necessárias a medida em que as extensões forem sendo propostas.

```
<GT> ::= '>'
<LT> ::= '<'
<Equal> ::= '='
<Exc> ::= '!'
<EQ> ::= <Equal>+<Equal>
<GE> ::= <GT> + <Equal>
<LE> ::= <LT> + <Equal>
<NE> ::= <Exc>+<Equal>
<BackSlash> ::= \
<Dolar> ::= '$'
<Pipe> ::= caracter ASCII, barra vertical (124)
```

4.3 Flexibilização na Especificação do Verbo de Comparação

Segundo a especificação original da linguagem, o verbo de comparação deve ser uma sequência de caracteres ASCII (de 0 a 127), não contendo espaços em branco. Para fins de detecção de intrusão, esta especificação mostra-se muito restritiva, no sentido que não é possível especificar verbos de comparação contendo espaços em branco, tampouco seqüências binárias de valores.

A proposta para as duas situações é a inclusão de uma nova unidade léxica na gramática da linguagem, conforme apresentado a seguir.

```
<Hexa> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F
<BinaryValue> ::= <Pipe><Hexa><Hexa><Pipe>
```

A unidade léxica <BinaryValue> define um valor binário qualquer, entre 0 e 255, especificada em hexa (00 a FF). Esta notação é bastante utilizada para esta finalidade, sendo a adotada pelo IDS Snort. A fig. 4.6 exibe dois exemplos onde esta extensão é aplicada.

No primeiro exemplo, apenas explora-se a possibilidade de se incluir espaços em branco na string. O caracter ASCII equivalente à 20 em hexadecimal é o espaço em branco. O segundo exemplo modela a assinatura de um ataque contra um servidor de correio eletrônico.

```
FieldCounter Ethernet/IP/TCP 0 teste|20|do|20|espaco "|20| é espaço em branco"
NoOffSet Ethernet mail|20|from|3a||20||22||7c| "ataque contra servidor SMTP"
```

FIGURA 4.6 – Especificação PTSL usando valores binários no verbo

4.4 Inclusão de Referência à Fonte de Descrição do Ataque

Com a disseminação dos sistemas de detecção de intrusão, em conjunto com uma maior disponibilidade de informações acerca das vulnerabilidade existentes nos sistemas, nota-se uma tendência ao desencontro e duplicidade de informações relativas a descrições de ataques. Em virtude disto, algumas instituições [CVE 2002], [WHI 2002], [SEC 2002] vem fazendo um trabalho no sentido de organizar e disponibilizar a descrição dos ataques. O grupo mantenedor do CVE tem procurado criar uma nomenclatura padronizada para descrição dos ataques, buscando diminuir as diferenças dos jargões usados nesta área. Embora ainda exista falta de homogeneidade entre estas próprias instituições, é muito importante associar a modelagem de um ataque a uma descrição padronizada do mesmo.

Na descrição de um traço, entre outras informações associadas ao mesmo, tais como autor ou data, propõem-se a inclusão de um campo opcional contendo uma referência à fonte de descrição do ataque. Esta referência é composta pelo nome da instituição seguido da denominação do ataque adotada pela própria instituição. A gramática estendida da seção de descrição do traço é apresentada a seguir.

```
  TraceSpecification ::= Trace <SP> <Double quotted identifier>
<CRLF>
                              [<Header>]
  <Header> ::= [Version:  <Version number> <CRLF>]
                    [Description:  <Informal text> <CRLF>]
                    [Key:  <Keyword> , <Keyword>* <CRLF>]
                    [Port:  <Unsigned int> <CRLF>]
                    [Owner:  <Informal text> <CRLF>]
                    [Last Update:  <Date and time> <CRLF>]
                    [Reference:  <Informal text>,<Informal text>
<CRLF>]
```

Um exemplo de especificação usando referência a um ataque descrito pelo CVE é apresentado na fig. 4.7, linha 8.

4.5 Operadores Relacionais

A linguagem PTSL oferece três abordagens para localização e delimitação de dados em um pacote: `BitCounter`, `FieldCounter` e `NoOffSet`. Após a localização e delimitação do dado, é possível compará-lo com um valor pré-estabelecido no sentido de realizar o caminhamento pela máquina de estados. Como a única operação possível a ser realizada é a comparação de igualdade, restringe-se muito a flexibilidade em testar valores pré-estabelecidos com dados existentes no pacote. Operações de teste de limiares ou faixa de valores são impossíveis de serem realizados.

1	Trace “Sendmail Overflow”
2	Version: 1.0
3	Description: Overflow on sendmail version 5.5.8.
4	Key: sendmail
5	Port: 25
6	Owner: Edgar Meneghetti
7	Last Update: Tue, 16 Aug 2000 15:30:58 GMT
8	Reference: CVE, CVE-1999-0095
	...

FIGURA 4.7 – Uso de referência ao ataque

As sintaxes de cada abordagem de localização foram descritas nas seções 3.3.4. A seguir está apresentada uma análise para cada uma destas abordagens, com ênfasen na necessidade e conveniência de modificações, assim como na melhor forma sintática a ser adotada. As novas unidades léxicas que suportam as extensões propostas foram apresentadas na secão 4.2.1.

4.5.1 Operadores Relacionais para Abordagem `BitCounter`

A especificação `BitCounter` possui o formato apresentado na fig. 4.8.

BitCounter	Encapsulamento	Deslocamento	Tamanho	Valor	Descrição

FIGURA 4.8 – Formato da especificação `BitCounter`

Por exemplo, para comparar se a porta de origem do datagrama é a porta 5, deve-se posicionar o localizador na posição onde se encontra a porta de origem no cabeçalho do protocolo `TCP`, especificar o tamanho do campo em número de bits (16 bits) e finalmente colocar o valor que se espera encontrar nesta área, escrito em base dois. Um exemplo de especificação está apresentado na fig. 4.9.

BitCounter	Ethernet/IP	0	16	0000000000000101	“Porta de Origem = 5”

FIGURA 4.9 – Especificação PTSL para testar se porta de origem é igual a 5

Observando a fig. 4.10, constata-se que a porta de origem tem um deslocamento igual a zero no cabeçalho `TCP` e um comprimento de 16 bits. Portanto, a especificação acima localiza e delimita o campo ”porta de origem”(*source port*) de forma correta e única.

Um teste simples, que verifique se a porta de origem está dentro de uma faixa de valores, é impossível de ser descrito pela linguagem. Como o único operador relacional é o operador de igualdade (implícito), seria adequada a inclusão de outros operadores na linguagem. A nova sintaxe, adotando a mesma gramática utilizada em 3.3.1, em conjunto com as novas unidades léxicas apresentadas em 4.2.1, seria:

```
<BitCounter> ::= BitCounter <SP>
       <Protocol encapsulation> <SP>
       <Unsigned int> <SP>
```

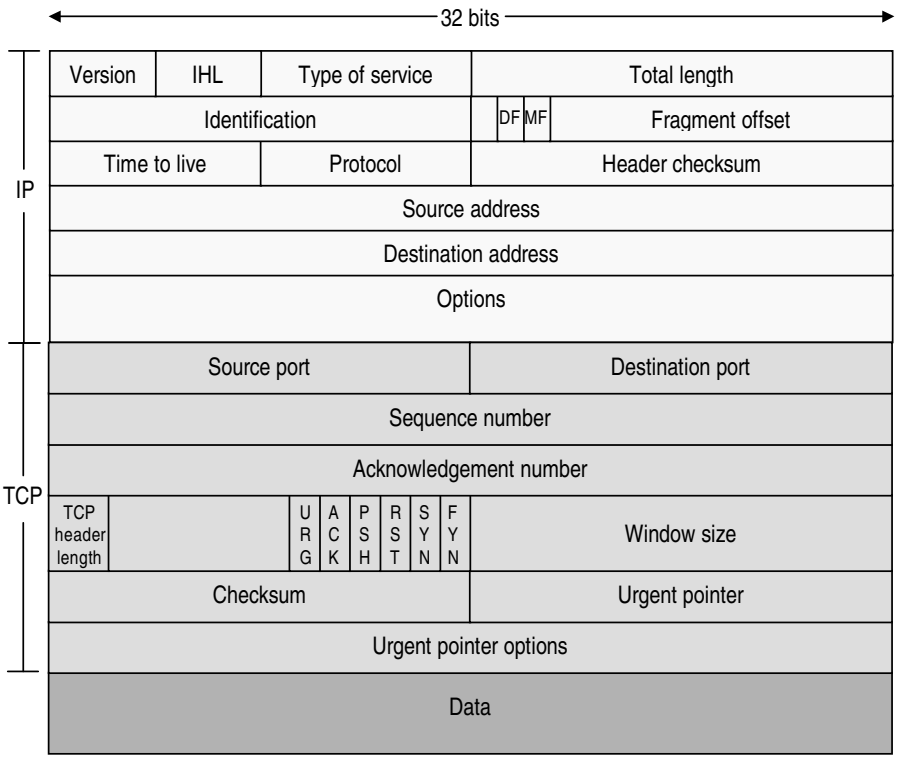

FIGURA 4.10 – Encapsulamento `IP/TCP`

```
<Unsigned int> <SP>
<WildCard>|<Verb identifier> <SP>
[ <GT> / <GE> / <LT> / <LE> / <EQ> / <NE> <SP>]
<Double quotted informal text> <CRLF>
```

A inclusão de novas unidades léxicas na gramática da especificação `BitCounter` possibilita a realização de operações de comparação mais amplas do que a existente originalmente na linguagem. Para manter compatibilidade com a gramática original, as unidades léxicas acrescentadas são opcionais. A implementação do *parser* (seção 5.3) continua considerando válidas a gramática original e a estendida. Se os operadores relacionais forem omitidos, o parser considera que o operador é a igualdade (<EQ>).

Como exemplo, um traço onde deseja-se mapear a incidência de início de conexões `TCP` (bit `SYN` ligado, `ACK` desligado) com portas de origem menores do que 1024, retrata uma situação atípica. Esta situação é, portanto, candidata a uma análise mais aprofundada do administrador. A especificação deste traço, em notação textual através da extensão proposta, está apresentada na fig. 4.11.

As linhas 11 e 12 incluem os operadores de comparação "=" e "<" respectivamente, conforme definido na nova gramática. É importante salientar que a inclusão destes novos operadores assume que os dígitos são números inteiros e positivos. Para os casos em que os números sejam representados com outra notação (complemento de 2, por exemplo), a comparação não pode ser feita de forma direta e sim através de uma função de conversão. A representação de valores em "*host order*" ou "*network order*" não é considerada para efeitos de comparação. O mecanismo de comparação não utiliza as funções *hton* e *ntoh* para converter as devidas representações e assume

1	Trace "Portas de Origem < 1024"
2	Version: 1.0
3	Description: Procura por portas de origem menores do que 1024.
4	Key: TCP, portas, anômalo
5	Port:
6	Owner: Edgar Meneghetti
7	Last Update: Tue, 16 Aug 2000 15:30:58 GMT
8	MessagesSection
9	Message "SRC PORT<1024"
10	MessageType: client
11	BitCounter Ethernet/IP 110 1 1 = "Chave SYN ligada"
12	BitCounter Ethernet/IP 107 1 0 = "Chave ACK desligada"
13	BitCounter Ethernet/IP 0 16 0000010000000000 < "SRC PORT < 1024"
14	EndMessage
15	EndMessagesSection
16	StatesSection
17	FinalState idle
18	State idle
19	"SRC PORT<1024" GotoState idle
20	EndState
21	EndStatesSection

FIGURA 4.11 – Traço para `TCP SYN` com porta menor do que 1024

que o bit mais significativo está a esquerda. Estas funções são apresentadas como trabalho futuro no capítulo 7.

A implementação deste operador, tanto em nível de *parser* quanto em nível da máquina de estados, não apresenta uma queda significativa de desempenho ou de complexidade. Do ponto de vista do *parser*, a inclusão desta nova unidade léxica não representa dificuldade em sua implementação, mesmo sendo este do tipo manual. Quanto ao desempenho, as operações de comparação de igualdade são mais simples e mais eficientes do que as demais. Na seção 5.2.4 discute-se o algoritmo e estruturas de dados usados para realizar esta comparação.

4.5.2 Operadores Relacionais para Abordagem `FieldCounter`

A abordagem `FieldCounter` é bastante conveniente para modelagem de traços de protocolos, porém se torna pouco útil quando se modela ataques. Em geral, os ataques exploram situações que fogem do convencional do ponto de vista do esperado pelos protocolos, de forma que são poucos os ataques que podem ser descritos utilizando esta abordagem. Assim, por uma questão de coerência, os operadores relacionais também são definidos na linguagem, embora saiba-se do seu uso restrito para fins de detecção de intrusão. A abordagem NoOffSet é mais conveniente na maioria das vezes, visto que a localização de uma determinada seqüência de caracteres em um pacote é uma das técnicas mais utilizadas para descrever um ataque.

De forma análoga à seção 4.5.1, a sintaxe estendida desta abordagem para suportar operadores relacionais é apresentada a seguir.

```
<FieldCounter> ::= FieldCounter <SP>
       <Protocol encapsulation> <SP>
```

```
<Unsigned int> <SP>
<WildCard>|<Verb identifier> <SP>
[ <GT> | <GE> | <LT> | <LE> | <EQ> | <NE> <SP>]
<Double quotted informal text> <CRLF>
```

4.5.3 Operadores Relacionais para Abordagem `NoOffSet`

A abordagem `NoOffSet` permite a localização de seqüência de caracteres em protocolos textuais ou na área de dados de qualquer protocolo. A implementação de operadores relacionais diferentes do comparador de igualdade seria de pouca utilidade neste caso. Como a finalidade desta abordagem é a localização de seqüências em posições desconhecidas, qualquer outro tipo de operação que não a comparação de igualdade permitiria a localização de caracteres quaisquer e sem significado para o traço em si. Além disso, a execução desta rotina pela máquina de estados seria bastante dispendiosa, mais do que ela já é sem nenhum tipo de extensão.

4.6 Variáveis

PTSL é uma linguagem não procedural, que apresenta algumas características de uma linguagem declarativa. Embora as linguagens declarativas não necessitem de variáveis, e por PTSL não poder ser inteiramente classificada como tal, o uso de estruturas similares a variáveis seria altamente benéfico do ponto de vista de aumento do escopo de ataques que podem ser descritos. Desta forma, se propõe o uso de estruturas semelhantes a variáveis dentro da gramática da linguagem.

4.6.1 Escopo de Variáveis

É possível identificar três escopos relativos ao funcionamento normal de um agente de monitoração: pacote, traço de protocolo e global. Considerando um escopo global, uma variável declarada em um traço teria significado para todos os demais traços. Em um escopo de traço de protocolo, a variável declarada em um determinado traço possui significado apenas para aquele traço, enquanto ele durar. Várias instâncias de um mesmo traço de protocolo não compartilham variáveis. Para o escopo relativo a um pacote, a variável teria significado apenas para um dado pacote capturado da rede, sendo eliminada tão logo o pacote tenha sido processado.

A fig. 4.12 exibe um exemplo onde o escopo de variáveis pode ser visualizado. No exemplo, existem dois traços idênticos, instanciados para pares de cliente e servidor diferentes. Uma variável local irá possuir significado durante uma transição entre um estado e outro. Durante a existência de um traço, instanciado a partir de um par de estações, pode-se definir variáveis de traço que irão existir enquanto o traço existir. Finalmente, as variáveis globais são definidas para todos os traços e respectivas instâncias.

O ataque conhecido por `LAND` (apresentado na seção 4.1.3) ilustra um caso onde o uso de variáveis locais possibilitaria a descrição do mesmo. Uma comparação entre o campo de endereço IP de origem e de destino caracterizaria o ataque.

A identificação de estações com a interface de rede em modo promíscuo, sugerida na seção 4.1.4, necessita de uma estrutura de variável de traço. Como é necessário identificar e armazenar alguns endereços no decorrer do traço, fica bem

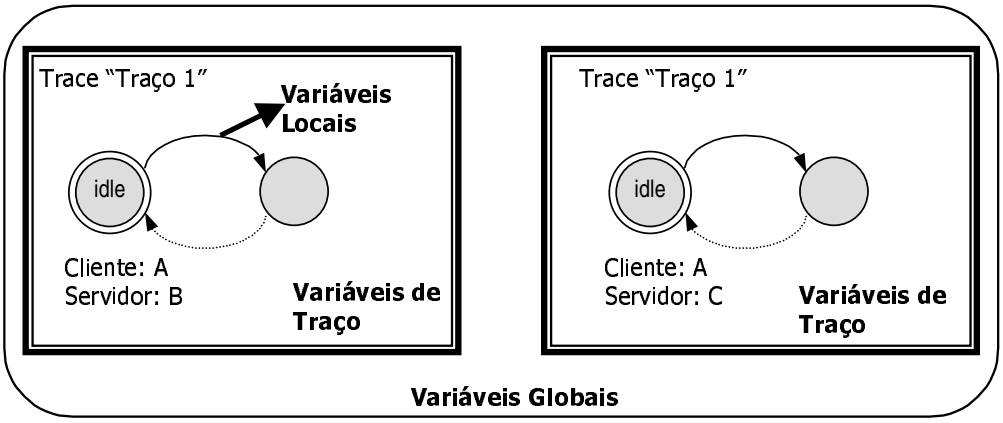

FIGURA 4.12 – Escopo de variáveis

caracterizado o uso de variáveis de traço para descrever o ataque. Este ataque ainda necessita de outras estruturas ainda não apresentadas.

O uso de variáveis globais não possui aplicação direta. Em muitos casos, o seu uso estaria mais voltado a utilização de constantes globais para fins de melhoria na legibilidade das especificações. Por exemplo, referências ao endereço de rede interno poderiam ser definidas como uma variável global recebendo um nome sugestivo (tal como *RedeInterna*, por exemplo).

4.6.2 Sintaxe para Especificação de Variáveis

A inclusão de variáveis na gramática da linguagem requer a alteração na sintaxe de algumas estruturas. Como forma de reduzir a complexidade e aumentar o desempenho do agente, as variáveis locais e de traço são tratadas e especificadas de forma diferente. As variáveis globais são suprimidas, em favor do uso de constantes globais. Durante a fase de modelagem de ataques, o administrador dispõe de uma ferramenta gráfica para especificação dos mesmos, de forma que o uso de estruturas que aumentem a legibilidade é desnecessário em nível de agente de monitoração.

As variáveis são definidas aproveitando a estrutura sintática que localiza e delimita o dado buscado. O uso das estruturas `BitCounter` e `FieldCounter` recebem uma nova denominação para especificar variáveis:

- `BitCounter_LV`: define uma variável local do tipo `BitCounter`
- `FieldCounter_LV`: define uma variável local do tipo `FieldCounter`
- `BitCounter_TV`: define uma variável de traço do tipo `BitCounter`
- `FieldCounter_TV`: define uma variável de traço do tipo `FieldCounter`

A sintaxe da estrutura léxica `BitCounter_LV` alterada está apresentada a seguir.

```
<BitCounter_LV> ::= BitCounter_LV <SP>
       <Protocol encapsulation> <SP>
       <Unsigned int> <SP>
       <Unsigned int> <SP>
```

```
        <Variable identifier><SP>
        <Double quotted informal text> <CRLF>
```

A sintaxe para `BitCounter_TV`, `FieldCounter_LV` e `FieldCounter_TV` é análoga, e estão apresentadas a seguir.

```
<BitCounter_TV> ::= BitCounter_TV <SP>
        <Protocol encapsulation> <SP>
        <Unsigned int> <SP>
        <Unsigned int> <SP>
        <Variable identifier><SP>
        <Double quotted informal text> <CRLF>

<FieldCounter_LV> ::= FieldCounter_LV <SP>
        <Protocol encapsulation> <SP>
        <Unsigned int> <SP>
        <Variable identifier><SP>
        <Double quotted informal text> <CRLF>

<FieldCounter_TV> ::= FieldCounter_TV <SP>
        <Protocol encapsulation> <SP>
        <Unsigned int> <SP>
        <Variable identifier><SP>
        <Double quotted informal text> <CRLF>
```

Um exemplo de especificação de uma variável local do tipo `BitCounter`, que armazena a porta de destino de um pacote `TCP`, está apresentada na fig. 4.13.

```
BitCounter_LV Ethernet/IP 16 16 port_dst "Porta de Destino"
```

FIGURA 4.13 – Exemplo de especificação de variáveis para `BitCounter_LV`

Um exemplo de especificação que utiliza variáveis do tipo `FieldCounter` é apresentado na fig. 4.14.

```
FieldCounter_LV Ethernet/IP/TCP 0 oper "faz oper = fieldcounter"
```

FIGURA 4.14 – Exemplo de especificação de variáveis para `FieldCounter_LV`

O exemplo define uma variável chamada "oper" e atribui a ela o valor obtido a partir do primeiro campo da área de dados do `TCP` (considerando o espaço em branco como delimitador de campos).

4.6.3 Uso de Variáveis

Conforme visto no item anterior, através de poucas alterações na estrutura sintática da gramática de PTSL, consegue-se especificar variáveis com o tipo definido automaticamente. O uso destas variáveis é feito de forma similar, através das mesmas unidades léxicas `BitCounter` e `FieldCounter`, apenas alterando o verbo de

identificação pelo nome da variável, prescindido do caracter "$". Como alterou-se a semântica do "*verb identifier*" com a inclusão do caracter "$", não seria mais possível especificar, neste campo, strings contendo o caracter citado. A solução adotada é o uso do caracter "\" como normalizador do "$", tornando-o um caracter sem significado especial. Trata-se de solução amplamente adotada para casos correlatos. Por analogia, o uso do próprio "\" também pode ser normalizado.

A sintaxe destas unidades léxicas estendidas está apresentada a seguir.

```
  <BitCounter> ::= BitCounter <SP>
        <Protocol encapsulation> <SP>
        <Unsigned int> <SP>
        <Unsigned int> <SP>
        <WildCard>|[<BackSlash]<Verb
identifier>|[<Dol>]<Variable identifier> <SP>
        [ <GT> | <GE> | <LT> | <LE> | <EQ> | <NE> ]
        <Double quotted informal text> <CRLF>

  <FieldCounter> ::= FieldCounter <SP>
        <Protocol encapsulation> <SP>
        <Unsigned int> <SP>
        <Unsigned int> <SP>
        <WildCard>|[<BackSlash]<Verb identifier>|
[<Dol>]<Variable identifier> <SP>
        [ <GT> | <GE> | <LT> | <LE> | <EQ> | <NE> ]
        <Double quotted informal text> <CRLF>
```

A sintaxe proposta suporta os operadores de comparação, a especificação de variáveis e o respectivo tipo, e o uso de variáveis. Alguns exemplos que exploram esta nova gramática são apresentados na fig. 4.15.

```
BitCounter_LV Ethernet/IP/TCP 0 op "Define variável local op"
BitCounter Ethernet/IP/TCP 0 $op "Compara campo na posição zero com variável op"
FieldCounter Ethernet/IP/TCP 0 \$ "Compara campo na posição zero com caracter $"
FieldCounter Ethernet/IP/TCP 0 \\ "Compara campo na posição zero com caracter \"
```

FIGURA 4.15 – Exemplo de utilização de variáveis para `FieldCounter`

O primeiro exemplo ilustra a definição de uma variável. O segundo exemplo descreve uma mensagem onde o campo zero deve ser comparado com o conteúdo da variável "oper". O segundo e terceiro exemplo ilustram casos onde é necessário o uso de contra-barra.

É importante salientar que o tipo e comprimento da variável são atribuídos a partir da abordagem e especificação desta. Desta forma, por exemplo, não é possível definir uma variável do tipo `BitCounter` e tamanho igual a 16 bits e compará-la com um campo de mesmo tipo e tamanho diferente, ou apenas de tipo diferente.

4.7 Mudança na Semântica da Abordagem `FieldCounter`

Conforme a especificação original da abordagem `FieldCounter`, o caracter que separa os campos para efeito de posicionamento é o espaço em branco. Como a maior parte dos protocolos de aplicação adota esse caracter como separador de estruturas, é bastante razoável o seu uso na abordagem. Entretanto, ocorrem situações onde seria conveniente a localização de seqüência de caracteres utilizando outro separador que não o espaço em branco. Na descrição de ataques relacionados com servidores `WEB`, por exemplo, é comum a necessidade de se especificar a posição de campos em relação aos caracteres "&" e "?". Este tipo de uso da abordagem `FieldCounter` não seria possível.

A proposta desta extensão é possibilitar o uso de delimitadores de campos diferentes do espaço em branco na abordagem `FieldCounter`. A mudança sintática necessária para tal alteração está apresentada a seguir.

```
  <FieldCounter> ::= FieldCounter <SP>
         <Protocol encapsulation> <SP>
         <Unsigned int> <SP>
         <Unsigned int> <SP>
         <WildCard>|[<BackSlash]<Verb identifier>|
[<Dol>]<Variable identifier> <SP>
  <Equal>}{<Alpha>} <SP>
         [ <GT> | <GE> | <LT> | <LE> | <EQ> | <NE>
         [ <BackSlash<DQ> | <Alpha> ] ]
         <Double quotted informal text> <CRLF>
```

A inclusão de mais um argumento na abordagem `FieldCounter` é a solução proposta. Como o parser deve continuar considerando PTSL sem extensões, algumas considerações devem ser observadas:

- Se o delimitador for especificado (possivelmente diferente do espaço em branco), então é obrigatório o uso dos operadores de comparação. De outra forma, poderia haver ambigüidades quando se desejasse adotar algum símbolo de comparação como caracter delimitador;
- Se o caracter delimitador for a unidade léxica <DQ>, então obrigatoriamente deverá vir precedida de <BackSlash>;
- O caracter delimitador deve ser único, conforme a gramática especificada.

O exemplo apresentado na fig. 4.16 ilustra uma requisição `HTTP`, onde o atacante procura passar a *string* como argumento "/c+dir+c:\" (possivelmente a um servidor IIS). Uma `URL` como essa pode indicar que o atacante está querendo executar algum *script* ou `CGI` no servidor `HTTP` a fim de obter uma listagem dos arquivos existentes no servidor.

```
GET /scripts/..%C0%AF../winnt/system32/cmd.exe?/c+dir+c:\
```

FIGURA 4.16 – Uma requisição HTTP suspeita

A definição de uma mensagem que permita identificar a sub-string "/c+dir+c:\" no segundo campo da mensagem, deve ser realizada como ilustra a fig. 4.17. Como pode ser observado, deseja-se delimitar o conteúdo existente após o "?" (presente no campo 1) e compara-lo à string "/c+dir+c:\".

```
FieldCounter Ethernet/IP/TCP 1 /c+dir+c:\ == ? "2º campo, com separador '?"'
```

FIGURA 4.17 – Utilização de outros separadores além do espaço

É importante salientar que este tipo de mensagem poderia também ser descrita através da abordagem `NoOffSet`. Porém, o esforço computacional para a execução das comparações seria muito maior do que a forma proposta.

4.8 Modificadores de Comportamento do Agente

Durante o período de modelagem de ataques e análise de desempenho do agente, ficou evidenciado a necessidade de se poder controlar alguns aspectos de comportamento do agente. Esta seção descreve a inclusão de dois modificadores de comportamento do agente, opcionais, mas importantes no contexto do agente: NoGrabPort e DisableGenericRmon2.

4.8.1 Modificador NoGrabPort

O agente de monitoração observa pacotes na rede procurando por características descritas nas transições entre estados. Ao iniciar um traço (em uma transição partindo do estado *idle*), o agente instancia um novo traço para este par de estações. Os dados que caracterizam esta instância são tipicamente os endereços IP de origem e destino, e as portas de origem e destino. De posse destes quatro dados, será possível determinar se os pacotes vindouros fazem parte desta sessão ou não. Se fizerem parte da mesma sessão, então será avaliado se eles ocasionam caminhamento na máquina de estados ou não.

A opção em utilizar os endereços IP e portas de origem e destino se deve ao fato de facilitar a modelagem dos traços. Se o agente não realizasse este controle, caberia ao administrador realizar este controle por intermédio das estruturas disponíveis em PTSL. Mesmo que o agente controlasse apenas os endereços IP, ainda assim restaria a possibilidade de haver várias conexões simultâneas entre um par de estações, e talvez a modelagem não estivesse contemplando esta possibilidade.

Entretanto, a adoção deste mecanismo de controle automático de endereços e portas leva a problemas quando o traço especificado lida com protocolos da camada de rede ou enlace. Em algumas circunstâncias, a porta não pode ser considerada, mesmo que ela exista sob forma do protocolo `TCP` ou `UDP`. Por exemplo, em uma modelagem de um ataque que utilize fragmentação sob forma de encobrir o verdadeiro ataque, algum pacote pode estar fragmentado justamente no protocolo de transporte. Neste caso, o valor da porta obtido do protocolo de transporte poderia não estar correto (pois apenas uma parte do cabeçalho está disponível). Mesmo assim, o agente iria armazenar a porta e só consideraria os próximos pacotes que possuissem o mesmo par de portas, e que provavelmente não iria acontecer. As fig. 4.18 e 4.19 ilustram um cenário deste tipo. Uma ferramenta de varredura de portas chamada

nmap [NMA 2002] é capaz de realizar este processo utilizando fragmentação para ocultar e facilitar a penetração em uma rede.

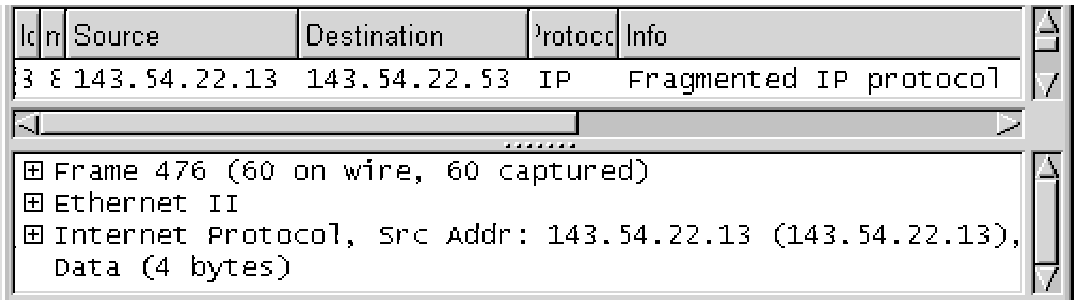

FIGURA 4.18 – Fragmentação no cabeçalho TCP

A fig. 4.18 exibe um datagrama IP capturado de um segmento de rede, carregando apenas 4 bytes em sua área de dados. Esta área deveria estar carregando todos os bytes do cabeçalho do protocolo de transporte, mas não foi possível devido a fragmentação forçada. A fig. 4.19 apresenta o datagrama IP seguinte, onde os demais bytes do cabeçalho do protocolo de transporte estão presentes (e que a ferramenta de captura apresenta como cabeçalho mal formado).

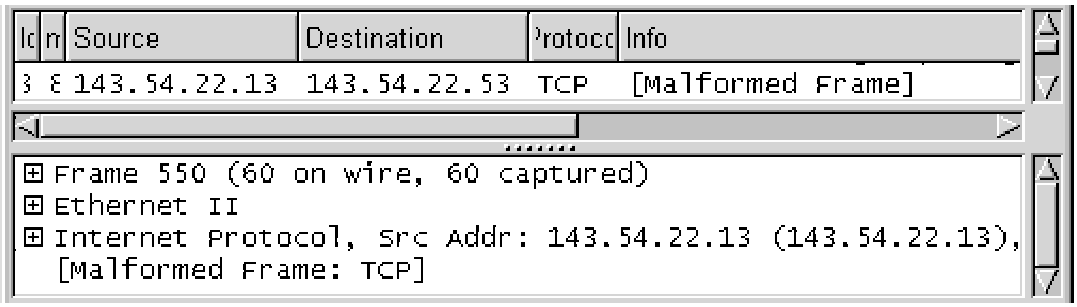

FIGURA 4.19 – Fragmentação no cabeçalho TCP

A solução adotada para resolver este tipo de problema é a inclusão de um modificador, que força o agente de monitoração a não fazer controle por porta de origem e destino. Desta forma, o único controle que será realizado é através do endereço IP de origem e destino. A fig. 4.20 ilustra um traço que faz uso do modificador NoGrabPort (linha 7) para forçar o agente a se comportar da forma descrita acima, especificamente para este traço. Neste traço, procura-se pela ocorrência de pacotes que possuam a *flag* `More Fragments` ativa (linha 13), e que sejam datagramas com tamanho inferior a 24 bytes (linha 14). Como um pacote `TCP` ou `UDP` possui mais do que 24 bytes (incluindo o próprio IP que ocupa 20 bytes), fica caracterizada a ocorrência de fragmentação no cabeçalho de transporte

4.8.2 Modificador DisableGenericRmon2

Como já foi mencionado anteriormente, o agente de monitoração é um agente RMON2 convencional, porém estendido para suportar a contabilização de traços. Durante o período de validação do agente (vide capítulo 6), constatou-se o baixo desempenho do agente. Embora a causa do baixo desempenho seja apresentada no capítulo supracitado, uma alternativa encontrada seria transformar o agente de monitoração em um agente RMON2 dedicado a contabilizar traços de protocolos.

```
1  Trace "Fragments"
2  Version: 1.0
3  Description: MISC Tiny Fragments
4  Key:
5  Owner: Edgar Meneghetti
6  Last Update: 2001-11-29 22:34:55
7  NoGrabPort
8
9  MessagesSection
10 Message "Fragments"
11 MessageType: client
12 MessageTimeout: 1
13 BitCounter Ethernet 48 4 0010 "MF"
14 BitCounter Ethernet 16 16 0000000000011000 < "Total Length < 24 bytes"
15 EndMessage
16 EndMessagesSection
17
18 GroupsSection
19 EndGroupsSection
20
21 StatesSection
22 FinalState idle
23
24 State idle
25 "Fragments" GotoState idle
26 EndState
27
28 EndStatesSection
29
30 EndTrace
```

FIGURA 4.20 – Traço para monitorar datagramas `IP` fragmentando o protocolo de transporte

Desta forma, pacotes que não façam parte de nenhum traço sendo monitorado são descartados. Dependendo do propósito do agente de monitoração, esta abordagem pode ser razoável. Em testes posteriores (vide seção 6.2), constatou-se uma melhora sensível na capacidade do agente de monitoração de tratar os pacotes lidos da rede, mesmo com tráfego em rajada.

Este modificador tem um efeito global, ou seja, qualquer traço que declare este modificador força o agente de monitoração a descartar pacotes não relacionados aos traços em monitoração. A fig. 4.21 apresenta um fragmento de código que contém este modificador.

4.9 Considerações Finais

Neste capítulo, foram apresentados formas de utilizar a linguagem PTSL para descrever ataques. Considerações positivas e negativas acerca da adequação da arquitetura TRACE como um sistema de detecção de intrusão são descritas, assim

```
1  Trace "Fragments"
2  Version: 1.0
3  Description: MISC Tiny Fragments
4  Key:
5  Owner: Edgar Meneghetti
6  Last Update: 2001-11-29 22:34:55
7  DisableGenericRmon2
8
9  MessagesSection
10 ....
```

FIGURA 4.21 – Fragmento de um traço que utiliza o modificador DisableGenericRmon2

como procura-se modelar situações reais no contexto da arquitetura e da linguagem. Destas modelagens, surgiu situações onde a linguagem carece de algumas funcionalidades que são fundamentais para o propósito em questão.

Como decorrência da identificação da falta de algumas funcionalidades na linguagem PTSL, extensões foram propostas e a nova gramática da linguagem foi apresentada para cada extensão. Salienta-se que as extensões foram propostas tendo como premissas principais a real necessidade da extensão, facilidade de implementação e pouca implicação em termos de alteração no desempenho do agente. No anexo 1 são exibidos vários modelos de descrição de ataques onde as extensões apresentadas foram utilizadas.

5 O Agente de Monitoração PTSL

A arquitetura Trace corresponde ao sistema de gerenciamento de protocolos de alto nível proposto em [GAS 2002]. Esta arquitetura, acrescida das extensões e considerações propostas neste trabalho, pode ser utilizada como um sistema de detecção de intrusão. Este capítulo apresenta detalhes da implementação do agente de monitoração, que é um dos componentes chave da arquitetura Trace. O agente de monitoração implementa o interpretador da linguagem PTSL, além de implementar parcialmente um agente `RMON2`, com as devidas extensões necessárias para integra-lo ao sistema. É o principal instrumento de validação da arquitetura como um sistema de detecção de intrusão eficiente e efetivo.

Cabe ressaltar, que o agente de monitoração pode ser utilizado de forma estanque ao sistema de gerenciamento, principalmente no contexto de detecção de intrusão. A forma como o agente está construído permite que ele seja autônomo, tanto do ponto de vista de sua programação quanto do ponto de vista da observação de resultados. Como o agente de monitoração é também um agente `SNMP`, é possível coletar informações relacionadas aos traços através do uso de um software genérico de consulta a agentes `SNMP`.

Conforme abordado superficialmente no capítulo 3, a ocorrência de traços é armazenada em uma `MIB` similar a `RMON2`. Toda vez que a ocorrência do traço é observada entre qualquer par de estações, informações são armazenadas nesta `MIB`. A solução proposta para armazenar estas informações na `MIB`, sem alterar a semântica da especificação original também é descrita neste capítulo.

A integração do agente com o sistema de gerenciamento através da implementação dos mecanismos adequados para tal, não é contemplado neste trabalho, embora esteja previsto em [GAS 2002]. Sucintamente, a proposta é utilizar a `MIB Script` [LEV 2001] desenvolvida pelo grupo de trabalho *Distributed Network Management* (DISMAN), do IETF. Segundo [GAS 2002], a `Script MIB` permite que processos remotos (ou scripts) sejam iniciados, controlados e terminados através do framework de gerenciamento `SNMP`. A especificação da `MIB` é muito sucinta ao descrever o que são scripts; até onde lhe diz respeito, um script é um pedaço de código capaz de ser executado pelo agente remoto que implementa a `MIB`. Isso significa que quem implementa a `MIB` pode escolher as linguagens de programação em que os scripts são implementados e, portanto, inclusive PTSL.

O capítulo está organizado em 4 seções. A seção 5.1 apresenta a arquitetura do agente. Na seção 5.2, são apresentadas as principais estruturas de dados que compõe o agente. A base de informações de gerenciamento é apresentada na seção 5.4.

5.1 Arquitetura do Agente

Como foi mencionado no início deste capítulo, o agente de monitoração é o principal item a ser considerado em termos de validação das propostas apresentadas neste trabalho. A arquitetura do agente é composta por alguns módulos que realizam a interpretação dos traços (ataques modelados), a monitoração a partir da rede, a contabilização e finalmente a disponibilização destas informações através de uma `MIB RMON2` estendida. Os módulos que compõem o agente estão apresentados na fig.

5.1.

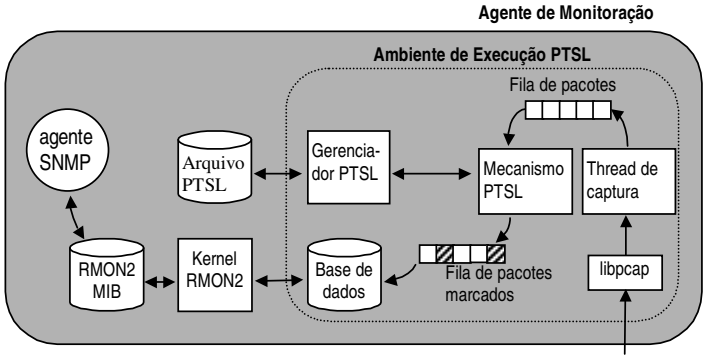

FIGURA 5.1 – Arquitetura do agente de monitoração

O agente de monitoração coleta informações diretamente da rede, através da colocação da interface de rede em modo promíscuo. Foi utilizada a biblioteca de captura `pcap` [TCP 2002], por tratar-se de um padrão *de facto* em aplicações que demandem o uso de dados provenientes da rede. Esta biblioteca fornece o pacote capturado em estado bruto, em conjunto com um cabeçalho contendo a data de captura e tamanho do pacote capturado em bytes. Esta é a base de dados do agente.

De posse dos pacotes fornecidos pela biblioteca `pcap`, o agente passa a monitorar a ocorrência de traços nesses pacotes, marcando-os quando for necessário. Esses pacotes, seletivamente marcados, são entregues ao núcleo do agente `RMON2` estendido, que disponibiliza as informações relativas aos grupos da `MIB` implementados.

O fluxograma da fig. 5.2 ilustra todo o processo com mais detalhes.

O agente de monitoração é composto basicamente por três fluxos concorrentes: a captura de pacotes, a máquina de estados e a contabilização no formato da `MIB RMON2`. Os fluxos concorrentes são implementados com o auxílio de `threads POSIX`. O laço principal também tem como funcão controlar o tempo de traços em andamento. Quando o tempo de um traço em andamento exceder o tempo máximo previsto, esta instância do traço deve ser cancelada (seção 3.2.3 e 5.2.5).

A especificação PTSL é lida de um arquivo, que contém uma lista com o nome de um ou mais arquivos. Esses arquivos devem conter a descrição dos ataques, modelados conforme a linguagem. O *parser* aceita a especificação dos traços individualmente (um por arquivo) ou através de um único arquivo contendo todos as especificações.

5.1.1 Captura de Pacotes

O módulo responsável pela captura de pacotes faz parte de um algoritmo produtor-consumidor, atuando como produtor de dados a serem consumidos pelos demais módulos. A biblioteca *pcap* fornece os pacotes em estado bruto, acrescentando uma estrutura auxiliar contendo o tamanho e uma marca de tempo (*timestamp*). A principal função desse módulo é inserir o pacote em uma lista circular, monitorando e contabilizando as situações onde a lista porventura encha. Esta situação caracterizaria descarte de pacotes na captura, embora não tenha sido observada no ambiente de teste (seção 6.2). Esse módulo é implementado através da execução de uma *thread*.

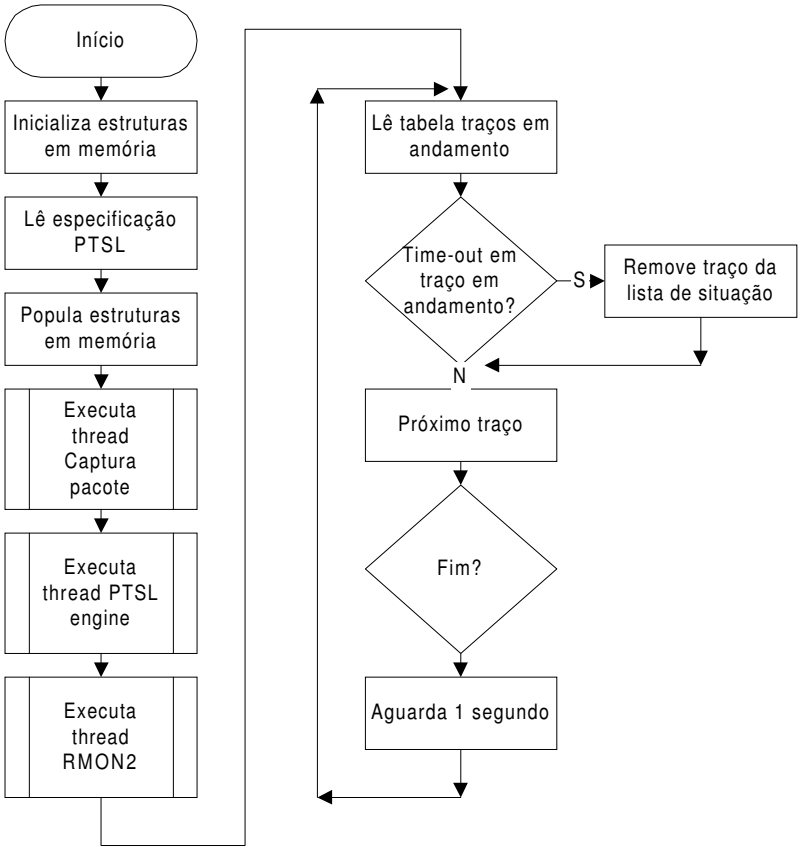

FIGURA 5.2 – Fluxograma "rotina principal"

Cabe salientar a opção tomada, no que tange descarte de pacotes. Como pode ser observado no fluxograma da fig. 5.3, quando a lista circular enche, o procedimento tomado é descartar o pacote recém recebido em vez dos pacotes mais antigos (ou que ainda não puderam ser processados). Desta forma, beneficia-se os traços em andamento em detrimento de possíveis novos traços. Se não fosse realizado dessa forma, ao descartar pacotes antigos estaria-se automaticamente descartando traços em andamento que dependessem deles para mudar de estados ou ser finalizado.

5.1.2 Máquina de Estados PTSL

A máquina de estados corresponde ao principal componente do agente. A principal função desse módulo é observar os pacotes capturados da rede (colocados em uma fila circular) e providenciar a implementação de uma máquina de estados de acordo com as especificações PTSL e com as transições decorrentes desses pacotes observados. A fig. 5.4 apresenta um fluxograma dos principais passos executados por esse módulo. Esse módulo é implementado através de uma *thread*.

A decodificação do pacote representa a localização dos cabeçalhos dos protocolos existentes no pacote, e a respectiva criação de ponteiros para eles. As áreas de dados de cada protocolo também são identificadas e ponteiros são criados para cada uma delas. Estes ponteiros irão agilizar o processamento desse pacote nos passos seguintes.

A máscara é uma estrutura criada (seção 5.2.3) para melhorar o desempenho da máquina de estados PTSL. A máscara corresponde a um valor binário, onde cada

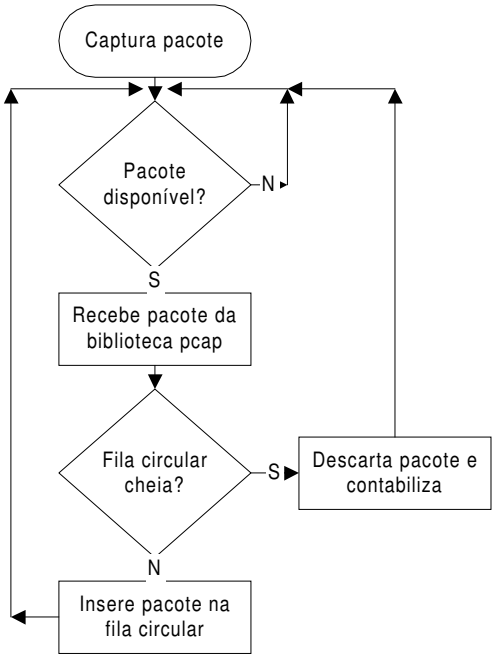

FIGURA 5.3 – Fluxograma "captura de pacotes"

bit representa um determinado protocolo. A máscara de cada um dos traços está colocada em uma lista de máscaras. Ao comparar a máscara do pacote com cada uma das máscaras dessa lista, verifica-se se este pacote pode fazer parte ou não de um traço. Dessa forma, pacotes que não contenham protocolos de interesse para a máquina de estados, não chegam sequer a serem analisados pela mesma.

Quando a máscara do pacote indica que se trata de um candidato a uma maior análise, então é chamada uma rotina específica para testar o pacote em relação à máquina de estados propriamente dita. O fluxograma dessa rotina está apresentado na fig. 5.5.

A rotina apresentada nesse fluxo corresponde a boa parte do esforço computacional demandado pelo agente. Tanto a manutenção das instâncias de traços, quanto a implementação das abordagens `FieldCounter`, `BitCounter` e `NoOffSet`, estão a cargo dessa rotina, além do próprio controle dos estados e transições da máquina de estados.

Os primeiros passos dessa rotina procuram posicionar o pacote e sua máscara dentro da situação atual da máquina de estados. Assim, se a máscara em questão não corresponde a uma transição inicial (partindo de um estado inicial), então deve-se analisar se já existe um traço em andamento para este par de estações. Caso não exista, não há porque prosseguir na análise, já que esse pacote não vai causar nenhuma alteração na máquina de estados. Se a transição for inicial, ou já exista um traço em andamento para esse par de estações e as máscaras conferem, então pode-se passar a analisar o pacote quanto as abordagens requisitadas pelo traço.

Os testes das abordagens `FieldCounter`, `BitCounter` e `NoOffSet` são realizados sequencialmente (visto que uma transição pode conter mais do que uma abordagem), conforme os algoritmos descritos a seguir.

- `FieldCounter`: é implementado com auxílio da função `strtok_r`, que separa uma string em substrings, de acordo com um caracter de separação (*token*).

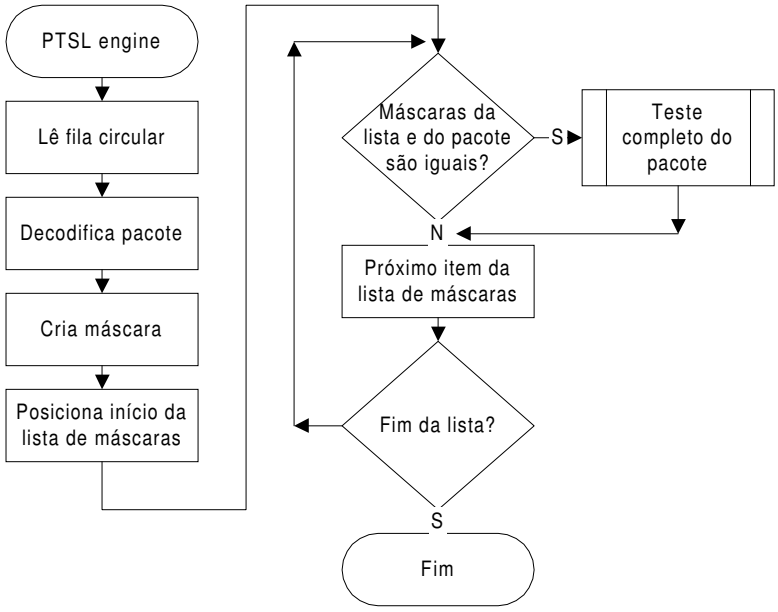

FIGURA 5.4 – Fluxograma "máquina de estados"

De posse da substring, procede-se com a operação solicitada pelo operador relacional, retornando verdadeiro ou falso.

- `BitCounter`: a implementação dessa abordagem considera somente operações relacionais entre *bytes*. Dessa forma, quando a especificação contém valores binários menores ou maiores do que um *byte*, estes valores são convertidos para o equivalente em *bytes*. O *parser* faz o mesmo, quando na leitura da especificação. Assim, a comparação entre o valor proveniente do pacote e o valor a ser comparado pode ser facilmente realizada.

- `NoOffSet`: trata-se da abordagem mais onerosa computacionalmente. A comparação entre o valor pretendido e os dados provenientes do pacote tem que ser feita n vezes, onde (n = tam_dado_pacote - tam_valor_pretendido + 1). Assim, por exemplo, um pacote que possua uma área de dados com 200 bytes, ao ser avaliado por esta abordagem contra uma string de 5 caracteres, irá requerer 196 comparações.

Se o teste de todas as abordagens resultarem em verdadeiro, o próximo passo é fazer o caminhamento na máquina de estados. Dependendo do estado a que esta transição está levando, pode-se iniciar uma nova instância desse traço (traço em andamento) ou considerar esse traço como encerrado e contabiliza-lo. Na seção 5.2 são descritas as estruturas de dados que suportam a implementação de instâncias de traços. A contabilização do traço se dá pela marcação do pacote em processamento. Essa marcação indicará à *thread* responsável por manter os grupos da `MIB RMON2` que ocorreu um traço completo durante o processamento desse pacote.

5.1.3 Conversor de Pacotes e Traços em Grupos da MIB RMON2

Esse módulo é o responsável por receber os pacotes capturados da rede e fazer as devidas contabilizações necessárias para a criação dos grupos `RMON2`. Trata-se de

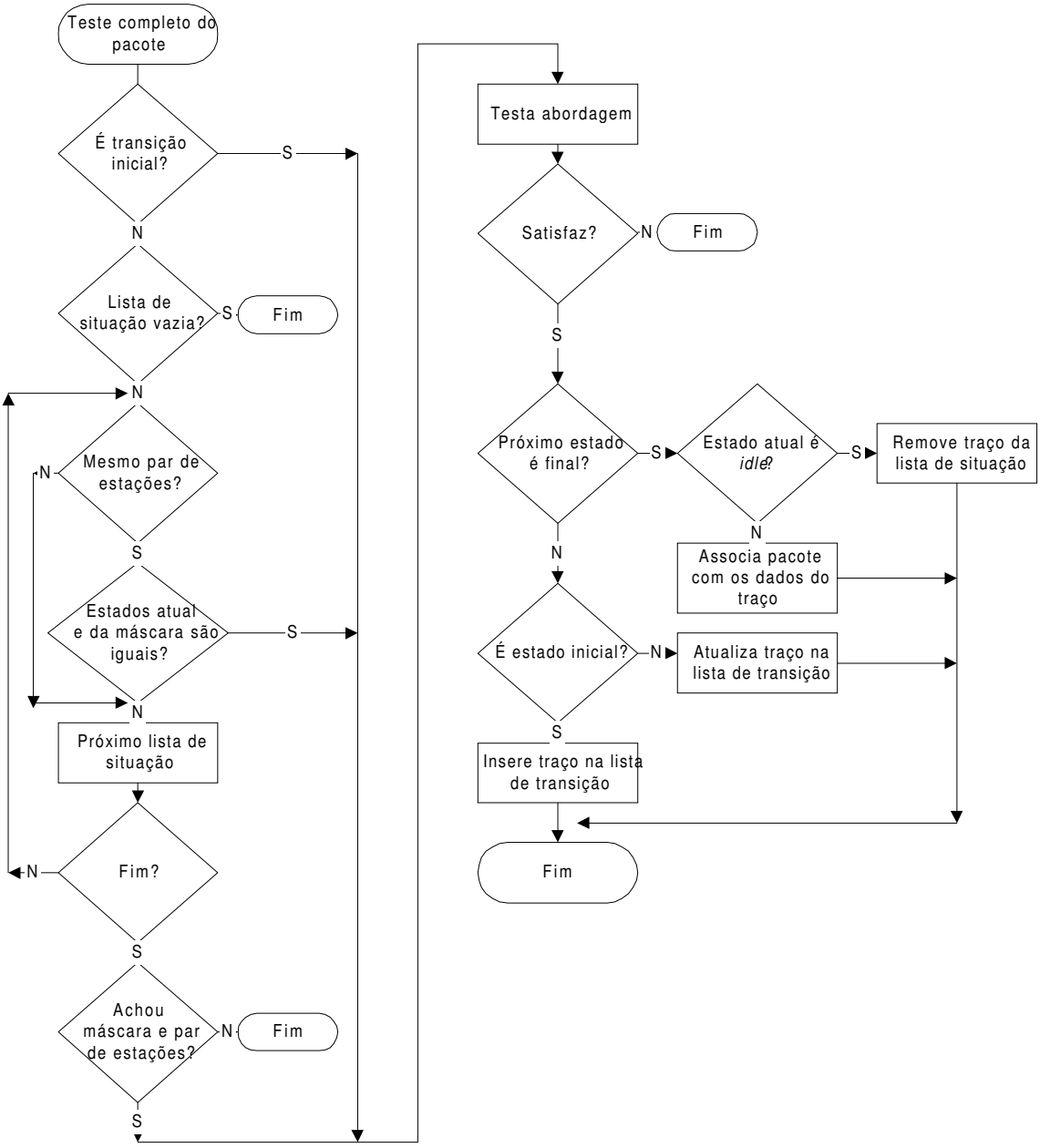

FIGURA 5.5 – Fluxograma “teste completo do pacote”

um agente RMON2 convencional que foi estendido para suportar a contabilização de traços de protocolos. Este módulo também é implementado através de uma *thread*, e está fortemente baseado no trabalho desenvolvido por Braga em [BRA 2002]. Nesta seção, é apresentada sucintamente a proposta original e as modificações propostas e implementadas.

O agente RMON2 original faz uso de um banco de dados relacional para armazenar os pacotes coletados da rede. Os grupos RMON2 também são armazenados nesse mesmo banco de dados. Boa parte do trabalho efetuado pelo agente diz respeito a contabilizar pacotes lidos do banco de dados em tabelas necessárias para implementar os grupos da MIB RMON2. As consultas ao agente RMON2 são efetuadas através do agente NET-SNMP [HAR 2002], que foi estendido para suportar os grupos dessa MIB. A fig. 5.6 ilustra a arquitetura do agente original.

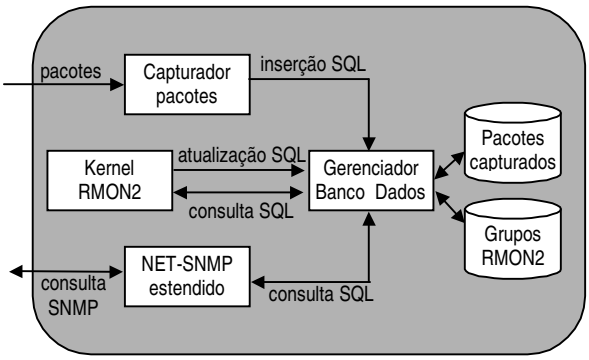

FIGURA 5.6 – Agente RMON2 original

Como pode ser observado na figura, o gerenciador do banco de dados possui uma iteração muito grande com os demais módulos do agente. Em testes posteriores (descritos na seção 6.2), foi constatado que o gerenciador do banco de dados é o ponto mais fraco em termos de desempenho do agente.

As principais alterações efetuadas no agente original estão elencadas a seguir.

- *Otimizações quanto a desempenho*: o módulo que captura pacotes passa a armazenar pacotes em uma lista circular alocada em memória (seção 5.2). Como a única operação SQL (*Structured Query Language*) realizada durante a inserção de um pacote no banco de dados é o próprio *insert*, a implementação da tabela de pacotes capturados em memória é simples e representa um bom ganho em desempenho;

- *Inclusão automática de traços de protocolos em* ProtocolDir: este grupo da MIB RMON2 é configurável manualmente via SNMP e contém os protocolos sendo monitorados pelo agente RMON2. Como os traços de protocolos são declarados como protocolos (e seus respectivos encapsulamentos), este grupo deve também refletir todos os traços sendo monitorados. Este requisito é implementado através de uma atualização periódica desse grupo no sentido de mantê-lo em sincronia com os traços de protocolos em monitoração;

A arquitetura do agente RMON2 modificado é apresentada na fig. 5.7. Embora as alterações realizadas tenham refletido positivamente em relação ao desempenho, o gerenciador de banco de dados continua sendo um ponto fraco do agente. Uma

mudança na arquitetura do agente, no sentido de substituir o banco de dados relacional por uma alternativa mais eficiente - arquivos DBM por exemplo - implicaria na reescrita de grande parte do código do agente. Isto se deve a grande dependência do kernel RMON2 em relação à consultas SQL, que somente estão disponíveis em um banco de dados desse tipo.

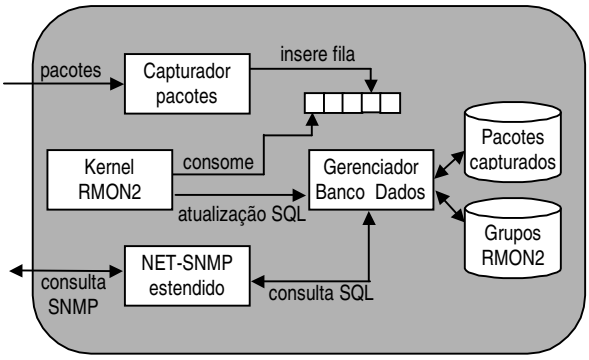

FIGURA 5.7 – Agente RMON2 modificado

5.2 Estruturas de Dados

A implementação de um sistema de detecção de intrusão, tal como o proposto nesse trabalho, requer o uso de várias estruturas de dados que suportem adequadamente e eficientemente os requisitos dessa classe de ferramenta. A escolha de estruturas de dados adequadas pode significar uma diferença considerável em termos de tempo de implementação e depuração, assim como no desempenho geral da ferramenta. Nessa seção serão apresentadas as principais estruturas de dados usadas na ferramenta.

O agente usa extensivamente estruturas alocadas em memória, principalmente filas encadeadas. A fig. 5.8 apresenta um diagrama com as principais estruturas de dados utilizadas e seus relacionamentos. Foi utilizada a notação conhecida por "crow's foot" [MAR 92] para representar a cardinalidade entre as relações.

As principais estruturas estão descritas nas seções subseqüentes.

5.2.1 Traços

A estrutura "Traços" é uma lista encadeada que contém descritores de todos os traços sendo monitorados pelo agente. Além de conter informações descritivas de cada traço, ainda armazena um ponteiro para uma lista de estados que compõe este traço (*state_list). Os campos com denominação iniciando com "rmon2" estão relacionados ao agente RMON2. Tratam especificamente de como este traço será inserido no grupo ProtocolDir da MIB RMON2. O campo rmon2_protocoldir_cod contém componentes do OID (*Object Identifier*) relativas aos códigos dos protocolos das camadas de enlace, rede e transporte. O campo rmon2_protocoldir_name contém a denominação textual deste traço quando inserido no grupo ProtocolDir. O campo rmon2_protocoldir_name_fault armazena o nome do mesmo traço quando contabilizando traços que não foram bem sucedidos (vide seção 5.4 para uma maior explicação).

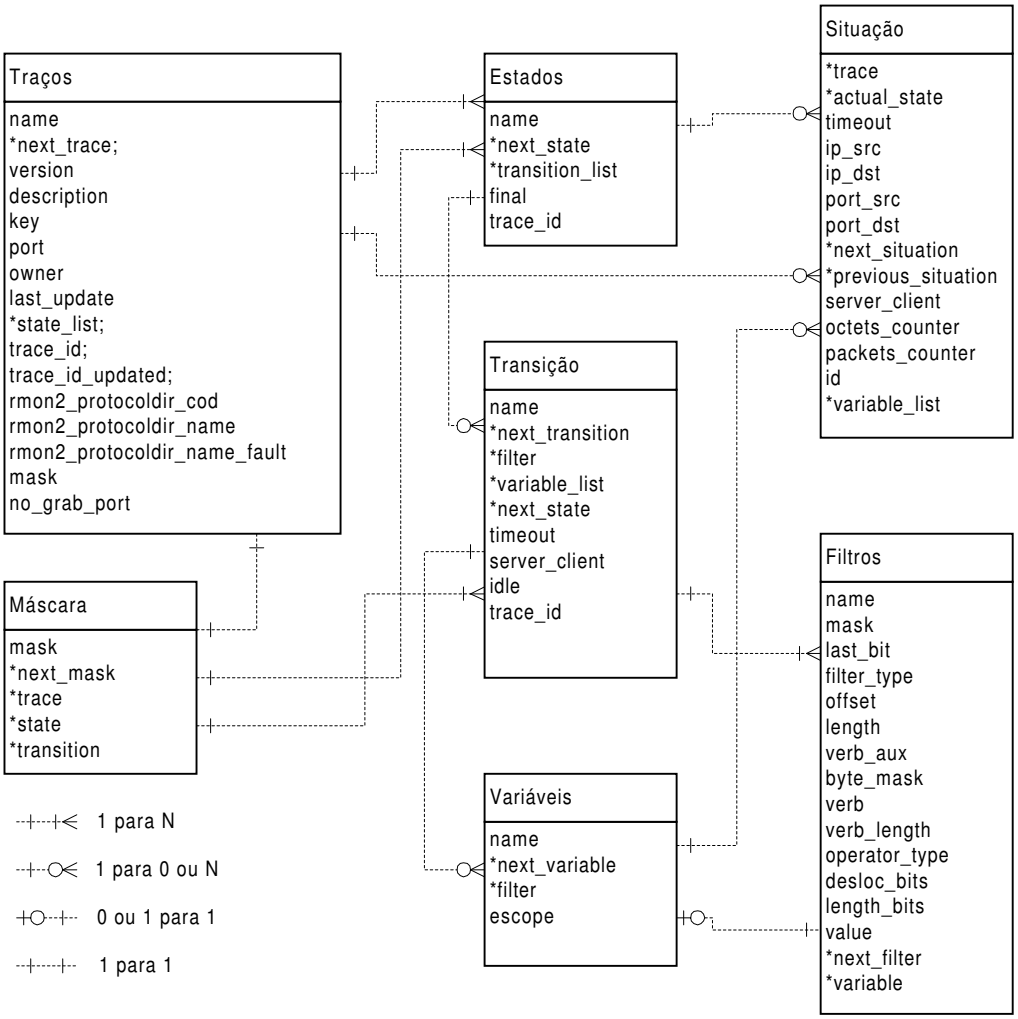

FIGURA 5.8 – Estruturas de Dados

5.2.2 Estados

A estrutura "Estados" corresponde a várias listas encadeadas, com início apontado pela estrutura Traços. Cada registro desta estrutura corresponde a um estado descrito no traço. Os registros possuem um ponteiro para as transições que partem do estado em questão (`transition_list`), além de uma chave que sinaliza se este estado corresponde a um estado final ou intermediário (`final`).

5.2.3 Máscara

A estrutura "Máscara" tem como propósito otimizar o processamento do pacote sob análise. A idéia principal é permitir que um pacote que contenha protocolos não relevantes para um traço seja descartado o mais rápido possível (ao menos para o mecanismo que implementa a máquina de estados). Embora na implementação atual exista suporte apenas para os protocolos `IP`, `TCP` e `UDP`, o mecanismo está pronto para suportar outros protocolos sem maiores alterações.

Durante o processo de leitura dos traços em PTSL (*parser*), é criada uma lista de máscaras de protocolos que compõe cada transição de cada traço. Ao decodificar o pacote capturado da rede, esta mesma máscara também é criada. Desta forma, é possível decidir se o pacote capturado da rede possui os protocolos descritos na transição. Se não possuir, para esta transição não é necessário fazer mais nenhum teste. Ainda não é possível descartar o pacote, visto que as demais transições podem ter este(s) protocolos(s) descrito(s).

A máscara é formada por bits que correspondem aos protocolos reconhecidos. Cada protocolo corresponde a um bit específico. Por exemplo, *ethernet* corresponde ao bit menos significativo, assim como IP é o segundo menos significativo, `TCP` é o terceiro menos significativo e `UDP` é o mais significativo. O exemplo da fig. 5.9 ilustra o processo de formação da máscara. Um pacote com encapsulamento `TCP` sobre `IP` teria uma máscara 01011. Este pacote só seria testado contra transições com máscara 01011 (TCP sobre IP). Se a transição contivesse testes relativos a protocolos diferentes entre sí (`TCP` sobre `IP` e `UDP` sobre `IP`, por exemplo), então a máscara da transição seria 01111, e forçaria qualquer pacote `TCP` ou `UDP` a ser analisado pela máquina de estados.

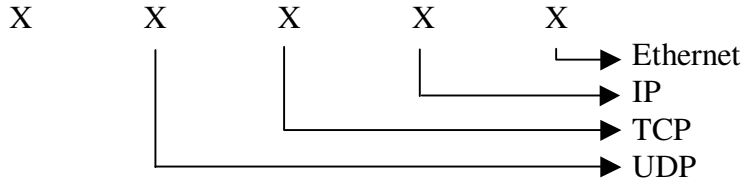

FIGURA 5.9 – Formação da máscara

5.2.4 Transições e Filtros

Estas duas estruturas correspondem aos descritores das mensagens entre estados. A estrutura "Transições" corresponde a uma lista encadeada, apontada pelo registro da estrutura "Estados" da qual a transição parte. Contém uma referência ao estado ao qual a transição chega (*next_state*), além de um campo para armazenamento de informações relativas ao tempo máximo desta transição (*timeout*), se esta

estrutura é do tipo cliente ou servidor (*server_client*) e ainda se é uma transição partindo de um estado inicial (*idle*). Cada transição possui um ou mais filtros associados. Cada filtro descreve uma abordagem especificamente.

A abordagem `BitCounter` é descrita com o auxílio de campos que otimizam o processo de comparação de valores binários. Basicamente, a idéia é armazenar os valores dos bits que deverão ser comparados já em sua posição dentro de um byte. A fig. 5.10 ilustra um exemplo.

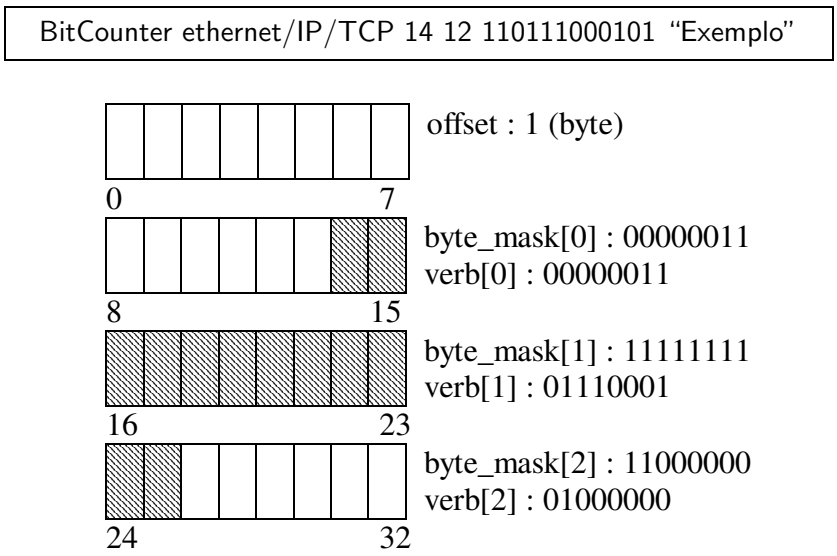

FIGURA 5.10 – Exemplo de tratamento da abordagem BitCounter

Como o deslocamento inicial é 14, significa que deve haver um deslocamento de 1 byte (8 bits) e deve-se considerar apenas os dois bits finais do segundo byte (*verb* e *mask* com índice zero). Os 10 bits restantes correspondem ao terceiro byte integralmente e aos dois primeiros bits do quarto byte. Desta forma, a comparação em tempo de execução é uma simples operação "E lógico" entre o pacote deslocado em *offset* bytes e *byte_mask*, e uma comparação simples do resultado desta operação com *verb*.

Para operações relacionais diferentes da igualdade, este processo não é usado. Neste caso, adota-se uma comparação por deslocamento de bits (*shift*), que é mais onerosa computacionalmente.

Para a abordagem `FieldCounter`, é armazenado o valor a ser comparado e o respectivo deslocamento. O algoritmo usado na comparação é uma versão semelhante ao de Boyer-Moore [BOY 77]. Cabe ressaltar que os operadores relacionais diferentes de igualdade e diferença somente serão permitidos quando o valor a ser comparado for numérico, inteiro e positivo.

A abordagem `NoOffSet` armazena apenas a string a ser comparada, sendo vetado o uso de operadores relacionais diferentes da igualdade e diferença. O algoritmo de comparação é o mesmo utilizado para a abordagem `FieldCounter`, porém o deslocamento inicial não ocorre, sendo necessário percorrer toda a área delimitada do pacote afim de localizar a string em questão. É a abordagem mais onerosa computacionalmente.

5.2.5 Situação

A estrutura de dados "Situação" armazena dados relativos aos traços em andamento, ou instâncias de traços. Entre os dados armazenados para cada traço em andamento de cada par de estações, destacam-se:

- *endereço IP de origem e destino*;
- *portas de origem e destino*;
- *timeout*: tempo decorrido até então desde o início desta transição;
- *contador de pacotes e octetos*: conta pacotes e traços ocorridos para este traço;
- *ponteiro para variáveis deste traço*: armazena uma referência para a área de variáveis globais relativas a este traço.

Como trata-se de uma lista com freqüentes inserções e remoções, optou-se por implementa-la através de uma lista duplamente encadeada. Como todos os traços em andamento estão armazenados nesta estrutura, é necessário dimensioná-la de forma adequada para evitar a impossibilidade de se criar novas instâncias de traços.

5.2.6 Variáveis

Em relação ao que foi definido na seção 4.6.1, as variáveis podem ter um escopo local ou global. A forma como as variáveis são tratadas faz uso das mesmas rotinas utilizadas pelas abordagens `BitCounter` e `FieldCounter` para localizar e delimitar a informação. No caso de uma variável local, assim que todos os filtros da transição forem aplicados, a informação armazenada na variável é descartada. Se a variável foi definida como global, ao final da aplicação dos filtros ela será salva em uma área de variáveis e será referenciada através da estrutura "Situação" pelo campo `variable_list`.

5.3 O *Parser* PTSL

Devido a simplicidade da gramática da linguagem PTSL, os traços são registrados diretamente a partir de um arquivo tipo texto. O analisador léxico, implementado através de um autômato finito, lê este arquivo e efetua o reconhecimento das palavras reservadas da linguagem. A análise sintática é feita em duas passadas pelo arquivo com a descrição do traço a fim de popular as estruturas em memória com o traço correspondente.

As máscaras dos traços (seção 5.2.3) são calculadas a partir deste arquivo e é criada também a lista que contém todas as máscaras dos traços. Esta lista será comparada com cada pacote capturado da rede e é importante no sentido de selecionar apenas os pacotes adequados aos traços em monitoração.

5.4 A Base de Informações de Gerenciamento

Toda vez que a ocorrência do traço é observada entre qualquer par de estações, informações são armazenadas em uma `MIB` similar à `RMON2`. É importante salien-

tar que o agente de monitoração atua como um agente RMON2 convencional, armazenando na MIB os valores relativos aos pacotes observados no segmento da rede e que estejam presentes no grupo protocolDir. Na presença de traços de protocolos, apenas inclui-se a definição desses no grupo ProtocolDir. Os grupos AlMatrix podem ser, então, consultados para determinar o número de traços ocorridos. Como a semântica dos campos não foi alterada, o gerente intermediário necessita converter o número de pacotes exibido pelo grupo em número de traços equivalentes. Cabe ressaltar que existe a possibilidade de forçar o agente a considerar apenas pacotes pertinentes aos traços em monitoração (seção 4.8.2), descartando os demais. Desta forma, o desempenho do agente melhora sensivelmente, embora se altere a semântica da MIB RMON2.

Na implementação atual, existe suporte para Ethernet, IP, TCP e alguns protocolos de aplicação tradicionais (HTTP, SMTP, FTP, SNMP e TELNET). Nem todos os grupos que compõem as MIBs RMON e RMON2 estão implementados. A implementação do agente RMON2 está fortemente baseada no trabalho efetuado por Braga em [BRA 2002], que se constitui de uma extensão do agente NET-SNMP [HAR 2002].

Uma das diferenças percebidas com a possibilidade de se incluir traços de protocolos no grupo protocolDir é o aumento da granularidade da monitoração. Ao invés de armazenar estatísticas globais sobre o tráfego gerado por um determinado protocolo, as estatísticas são geradas de acordo com a ocorrência de traços especificados. A tabela 5.1 mostra um exemplo resumido do grupo *protocolDir* onde, além dos protocolos normais, foram adicionados alguns traços de protocolo que descrevem cenários de ataque.

TABELA 5.1 – Grupo *protocolDir* da MIB RMON2

ID	*Descr*
00-00-00-01-00-00-08-00	ether2.ip
00-00-00-01-00-00-08-00-00-00-00-17	ether2.ip.tcp
00-00-00-01-00-00-08-00-00-00-00-17-00-00-00-19	ether2.ip.tcp.smtp
00-00-00-01-00-00-08-00-**00-01-00-01**	ether2.ip.varredura portas
00-00-00-01-00-00-08-00-00-00-00-17-**00-01-00-02**	ether2.ip.tcp.ataque serv IIS

A inclusão de traços nesse grupo é feita de forma automática pelo agente de monitoração, tão logo um novo traço passe a ser monitorado. A regra de formação do OID (*Object Identifier*) para os componentes deste grupo (que define os encapsulamentos) faz uso de 4 bytes. Em redes locais, a identificação utilizada para Ethernet, IP ou protocolos de camadas superiores nunca ultrapassa os 16 bits (2 bytes). Assim, os traços recebem identificadores (*ProtocolDirID*) superiores a 65535, para não colidirem com os dos protocolos normais. Essa regra se aplica aos traços que sejam compostos a partir de protocolos do nível de rede, transporte ou aplicação.

O grupo *alMatrix*, da MIB RMON2, armazena estatísticas sobre o traço, quando observado entre cada par de estações. A tabela 5.2 ilustra o conteúdo da tabela *alMatrixSD*. Ela contabiliza o número de pacotes e octetos entre cada par de máquinas (cliente/servidor). A semântica dos objetos da tabela foi mantida, de forma que o número de pacotes não representa o número de traços, e sim o número de pacotes observados que compõe o traço. Em um traço onde existam duas transições, a

ocorrência de um traço irá contabilizar dois pacotes para aquele par de estações. O número de octetos segue a mesma idéia.

TABELA 5.2 – Informações obtidas com consulta à tabela alMatrixSD

Src Addr	*Dst Addr*	*Protocol*	*Packets*	*Octets*
125.120.10.200	172.16.108.25	Acesso inválido a serviço TCP	250	53.256
125.120.10.100	172.16.108.25	Acesso inválido a serviço TCP	20	3.204
172.16.108.1	172.16.108.2	Comportamento anômalo TCP	4	4.350
172.16.108.32	172.16.108.2	Comando rpcinfo	8	7.300

É importante salientar que a interpretação dos valores computados para os objetos das tabelas fica a cargo do administrador. Quando o sistema estiver sendo usado como agente de monitoração da arquitetura Trace, o gerente intermediário será o responsável por interpretar os valores lidos sem seu real significado (número de traços).

6 Validação do Agente

Este capítulo descreve, de forma mais específica, o uso do agente como um IDS, assim como suas qualidades e limitações. Como a integração do agente com a plataforma Trace está disponível parcialmente (o gerente intermediário pode consultar o agente de monitoração, mas não programa-lo), a validação está mais focada no próprio agente como um IDS autônomo. Como já foi mencionado neste trabalho, o agente de monitoração pode ser acessado a partir de qualquer ferramenta capaz de realizar consultas SNMP. Ao realizar consultas periódicas, e comparando com uma *baseline*, o administrador da rede pode observar indícios de ataques em andamento na rede sob monitoração. No contexto da plataforma trace, o gerente intermediário é capaz de realizar esta observação e lançar alarmes ao administrador de maneira bem mais eficiente e automatizada.

Este capítulo está organizado em 3 seções: na seção 6.1 é proposta uma metodologia para validação do agente, no sentido de contemplar alguns requisitos buscados com a extensão da linguagem e características da implementação; na seção 6.2 são apresentados os resultados dos testes, segundo a metodologia de validação e na última seção faz-se comentários acerca das limitações do agente e possíveis soluções.

6.1 Metodologia

Algumas metodologias para avaliação de sistemas de detecção de intrusão têm sido propostas nos últimos anos. A iniciativa do DARPA [KEN 98] e [LIP 99], em 1998 e 1999, no sentido de definir uma metodologia e disponibilizar ferramentas e tráfego de ataques talvez seja a iniciativa mais completa até então. A idéia principal é criar um tráfego de fundo que reflita o uso normal de uma rede de dados, e inserir neste tráfego pacotes que contenham indícios de ataques ou todo o ataque. Vários sistemas de detecção de intrusão foram avaliados durante o período de alguns dias, tendo gerado um dos primeiros testes de avaliação de IDS padronizado.

Em [HAI 2001], é apresentado uma extensão ao modelo proposto originalmente. Neste trabalho, são apresentados três cenários de ataques: dois ataques de negação de serviço distribuídos e um ataque direcionado a um servidor Microsoft Windows NT. Como resultado do primeiro esforço, realizado entre 1998 e 1999, é apresentada uma ferramenta de geração de tráfego e ataques específica para fins de avaliação de IDS. Embora ainda não esteja concluída, esta ferramenta reflete a dificuldade em reunir cenários de ataques e tráfego de fundo completamente caracterizados [1].

McHugh [MCH 2000] faz uma crítica ao modelo de 1998 e 1999, lançando dúvidas em relação à validade de certos procedimentos. Em relação ao tráfego de fundo adotado, trata-se de pacotes capturados de uma base da força aérea americana. Este tráfego foi tratado, no sentido de ocultar dados não públicos. Duas das principais críticas são a falta de caracterização deste tráfego (mencionado como tráfego "normal") e a falta de um estudo mais aprofundado de quanto este tráfego

[1] segundo os próprios autores, uma das maiores dificuldades no teste anterior foi justamente estabelecer o cenário de teste.

poderia facilitar ou prejudicar o desempenho de um IDS em específico. Além disso, a própria manipulação do tráfego poderia induzir resultados falsos na avaliação.

Esta seção apresenta uma metodologia simples para validar o agente de monitoração como capaz de reduzir o número de falsos positivos quando comparado com outros IDS (no experimento foi utilizado o Snort).

6.1.1 Descrição do Cenário

A metodologia utiliza um dos cenários de ataque propostos em [HAI 2001], denominado *Lincoln Laboratory Scenario (DDoS) 1.0*. Trata-se de um ataque de negação de serviço distribuído, utilizando uma técnica de estouro de buffer em um servidor *sadmind* [2] e sendo executado em uma estação com sistema operacional Solaris. O ataque pode ser dividido em cinco fases bem definidas:

- *fase 1 - varredura de estações*: utilizam-se pacotes do tipo `ICMP echo request` e `ICMP echo reply` (idêntico ao que o comando *ping* faz) para mapear quais são as estações disponíveis no segmento;
- *fase 2 - verificação de quais estações estão executando o sadmind*: realiza-se uma consulta ao portmapper a fim de descobrir em qual porta TCP o daemon está escutando. Em seguida, abre-se uma conexão TCP com esta porta;
- *fase 3 - tentativa de invasão ao host alvo*: o atacante utiliza uma técnica de estouro de buffer no servidor *sadmind*, que permite a execução de um script com privilégio de super usuário. O comando executado edita o arquivo de senhas e inclui um novo usuário;
- *fase 4 - identificação das estações que foram comprometidas*: efetua um telnet para todas as estações, tentando iniciar uma sessão com o usuário criado na fase anterior;
- *fase 5 - disparo do ataque de negação de serviço distribuído*: manualmente, através de uma sessão telnet, o atacante dispara um ataque de negação de serviço distribuído. Esta parte do ataque não foi considerada nesta avaliação.

A metodologia consiste em executar as quatro primeiras fases do ataques, em conjunto com tráfego de fundo, e avaliar o número de alarmes reportados pelo agente de monitoração e pelo IDS Snort. A escolha do IDS Snort é motivada pela alta disseminação que este sistema tem apresentado nos últimos meses, sendo um dos mais utilizados atualmente.

O tráfego do ataque e de fundo será colocado na rede através de um software denominado *tcpreplay* [TCP 2002], que é capaz de reproduzir os pacotes armazenados em um arquivo com o mesmo intervalo em que foram recebidos. Este software permite ainda a variação na taxa de reprodução, além de possibilitar o envio dos pacotes a taxas pré-fixadas.

O ambiente de teste é apresentado na fig. 6.1.

O tráfego de fundo foi capturado em uma rede local ethernet (10Mbits/s), contendo dez computadores pessoais de uso essencialmente administrativo. A caracterização do tráfego é apresentada nas fig. 6.2, 6.3 e 6.4.

[2] *sadmind* faz parte do sistema de administração distribuída utilizado no sistema operacional Solaris da Sun.

FIGURA 6.1 – Ambiente do teste de validação

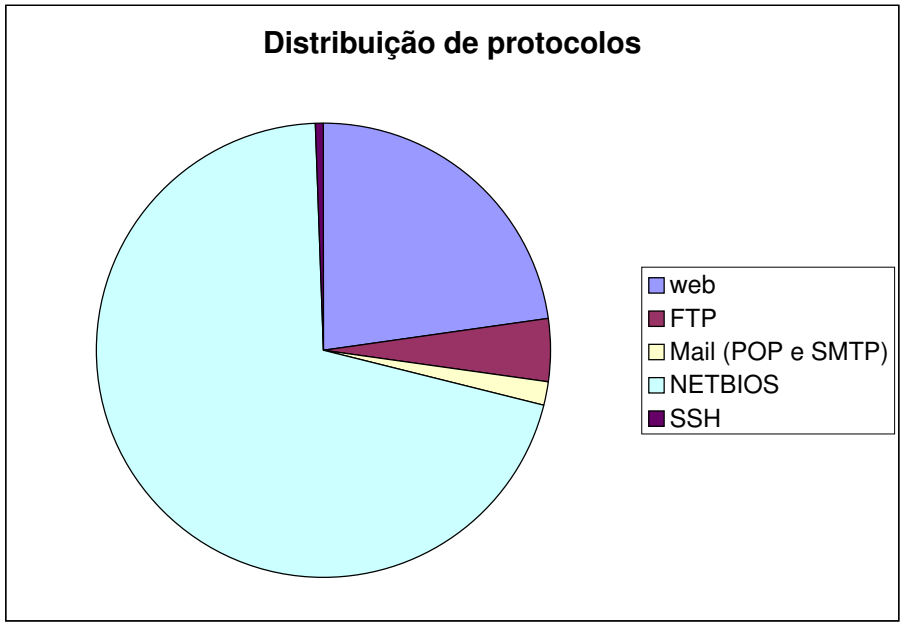

FIGURA 6.2 – Distribuição de protocolos no tráfego de fundo

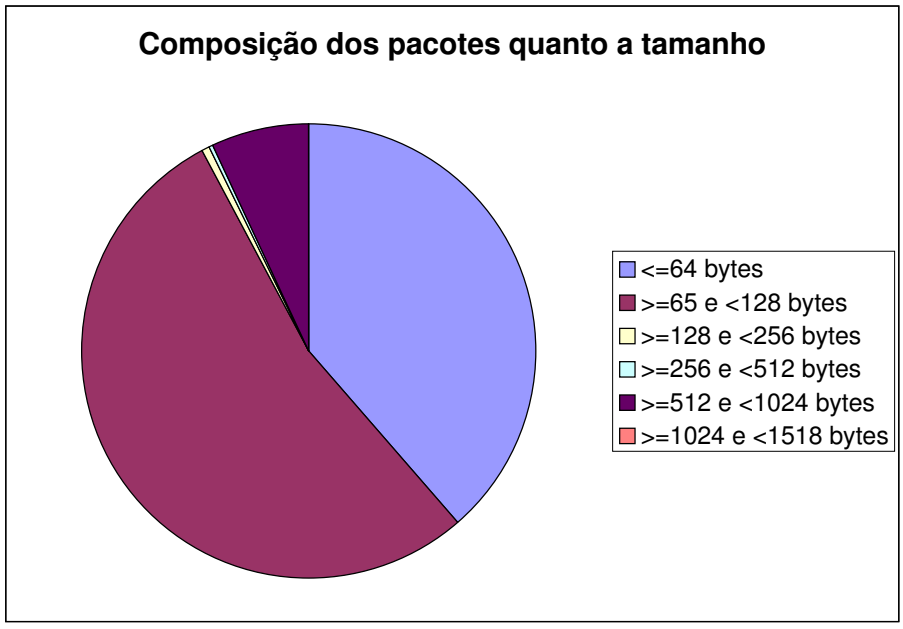

FIGURA 6.3 – Composição dos pacotes quanto a tamanho

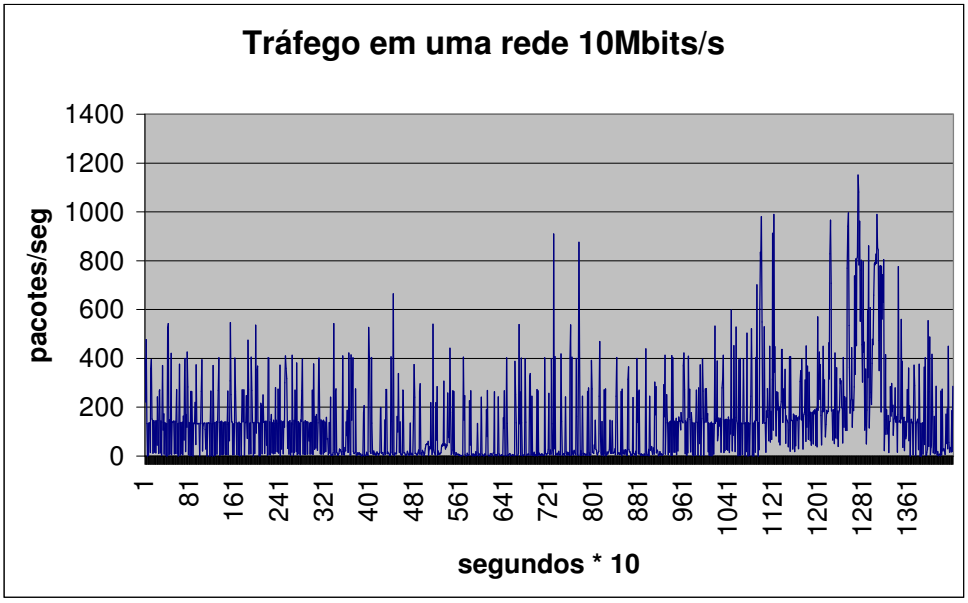

FIGURA 6.4 – Perfil do tráfego em pacotes/segundo

No tráfego de fundo foi inserido tráfego legítimo de acesso ao servidor *sadmind*. Os ataques foram lançados dez vezes, em tempos aleatórios. A taxa de envio do tráfego de fundo e dos ataques foi mantida baixa (máximo 1Mbit/segundo), em razão das limitações apresentadas na seção 6.3.

6.1.2 Resultados

Pretende-se demonstrar com este experimento que o agente de monitoração é capaz de gerar menos alarmes falsos (falsos positivos), mantendo uma boa taxa de acerto na detecção de ataques (baixo índice de falsos negativos).

O método para o reconhecimento de um ataque pelo agente de monitoração levou em conta os seguintes parâmetros:

- *período de amostragem*: as consultas SNMP à MIB RMON2 foram realizadas a cada segundo;
- *limiares*: foram ajustados especificamente para o teste. Estes valores são críticos para o agente e devem ser inicialmente determinados por estimativa e em seguida ajustados de acordo com as características da rede;
- *software de consulta*: qualquer software capaz de fazer consultas SNMP a uma base de informações RMON2 pode ser utilizado;

Um fragmento de amostragem efetuada no agente de monitoração PTSL está mostrada no gráfico da fig. 6.5. A detecção do ataque pode ser feita pela observação de variações acima dos limiares estipulados. No gráfico, a primeira variação abrupta ocorre durante a fase 3 do ataque. A segunda variação ocorre durante a fase 4 do ataque. A fase 1 não foi modelada, pois trata-se de tráfego corriqueiro em um segmento de rede (*ICMP echo request* e *ICMP echo reply*). Embora não tenha sido considerado neste teste, a modelagem da fase 1 poderia ser realizada de maneira bastante simples. A maior dificuldade, neste teste, seria ajustar os limiares aceitáveis

de ocorrência deste traço em função do tempo no sentido de minimizar a ocorrência de falsos positivos. O Snort não considera este tráfego em suas assinaturas de ataques. A detecção da fase 2 não está exibida no gráfico. O anexo 1 contém a descrição textual destas modelagens.

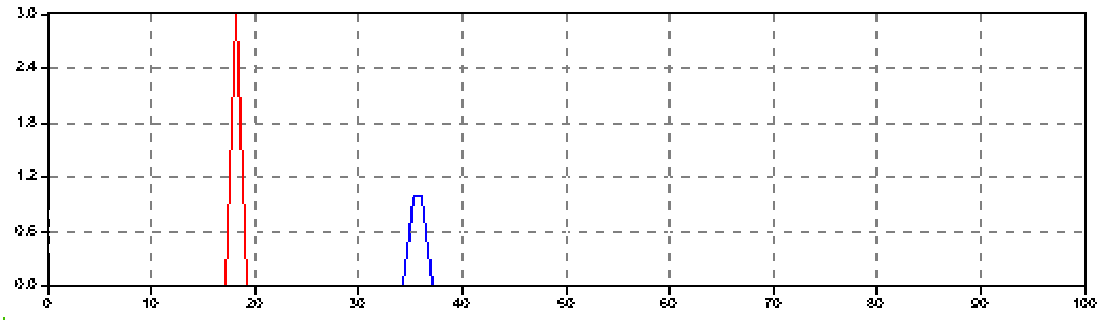

FIGURA 6.5 – Detecção do ataque pela variação de ocorrência dos traços

A tabela 6.1 exibe os resultados da avaliação.

TABELA 6.1 – Resultados da Avaliaçao

	Agente de monitoração PTSL	*Snort*	*esperado*
fase 1	0 alarme	0 alarme	0 alarme
fase 2	10 alarmes	460 alarmes	10 alarmes
fase 3	10 alarmes	250 alarmes	10 alarmes
fase 4	10 alarmes	130 alarmes	10 alarmes

Os resultados evidenciam o maior número de falsos positivos lançados pelo Snort. Este fato se dá pela natureza do mecanismo de detecção adotado por este software e pela maioria dos IDS. Em várias circunstâncias, o tráfego legítimo de uma determinada aplicação confunde-se com o tráfego de um ataque, justamente pela impossibilidade da correlação entre pacotes, característica intrínseca do agente de monitoração PTSL. Vários dos alarmes gerados pelo Snort decorrem do tráfego legítimo ao servidor *sadmind*, que não ocasiona alarme no agente de monitoração.

O índice de falsos negativos se manteve baixo em ambas as ferramentas, evidenciando uma detecção correta dos ataques. A fase 1 não acusou alarmes por tratar-se de tráfego comum em uma rede, e que em qualquer abordagem utilizada para modelagem, iria gerar falsos positivos. Desta forma, esta fase não foi modelada.

6.2 Análise de Desempenho

Esta seção descreve o comportamento do agente de monitoração PTSL em relação ao desempenho do mesmo. Um dos principais desafios dos IDS na atualidade é suportar o alto tráfego possível nas modernas redes de computadores. Embora este não tenha sido um dos focos deste trabalho, é importante avaliar o agente quanto a este aspecto.

A abordagem adotada (baseada em máquina de estados) mostrou-se onerosa computacionalmente. O ambiente de teste foi montado com três estações: uma

gerando tráfego, uma recebendo o tráfego e a terceira com o agente de monitoração. Alguns dados do teste estão resumidos abaixo. A estação, executando o agente de monitoração, possuía um hardware com as seguintes características: processador AMD K6II-450 MHz, 64 Mbytes de memória RAM, placa de rede 10Mbits/s.

- A capacidade do agente (sustentada) em termos de datagramas/segundo está em torno de 30 datagramas/segundo, com um traço sendo monitorado. Esse número foi obtido pela geração de seqüência de datagramas `UDP` que corresponde integralmente à modelagem feita no traço. Esta é uma situação de pior caso;

- Aumentando o número de traços monitorados, o desempenho do agente sofre degradação. Com 5 traços e tráfego gerado especialmente para esses traços, a capacidade do agente é reduzida para 22 datagramas/segundo;

- Utilizando a cláusula `DisableGenericRmon2` (seção 4.8.2), a capacidade do agente aumenta para 235 datagramas/segundo. Este aumento deve-se ao fato que o agente de monitoração deve contabilizar apenas os pacotes que compõem algum traço. Os pacotes que não fazem parte dos traços em monitoração, não são contabilizados nesta situação;

Cabe ressaltar que o agente, quando colocado em ambiente de produção e monitorando 10 traços, não descartou pacotes. O ambiente utilizado foi o mesmo caracterizado na seção 6.1.

6.3 Considerações acerca das Limitações

Com a crescente disponibilização de largura de banda nas redes locais, é necessário buscar alternativas que não comprometam o processo de detecção de intrusão. O desempenho medido do agente de monitoração não é adequado para o padrão mínimo atual de largura de banda nas redes locais (100Mbits/s).

O módulo que faz a atualização dos grupos da `MIB RMON2` é baseado integralmente em consultas e atualizações `SQL` efetuadas em um banco de dados relacional. Embora o banco de dados adotado seja bastante rápido (MySQL), este acaba sendo o ponto fraco em termos de desempenho. Uma simples observação no percentual de uso do processador evidencia um uso de 80% pelo banco de dados.

Algumas soluções a serem consideradas:

- *Uso de estações com mais de um processador*: como a implementação utiliza largamente threads, a existência de mais de um processador seria bastante benéfica;

- *Distribuição dos traços em mais de uma estação*: como os traços não possuem ligação entre si, a programação de diferentes traços em diferentes agentes não causa nenhum tipo de prejuízo; assim, os agentes, atuando de modo passivo na captura e observação do tráfego, assumiriam funções de inspeção complementares uns aos outros;

- *Implementação de filtros BPF (Berkeley Packet Filter)*: o uso destes filtros na biblioteca de captura, filtrando somente os pacotes que contém os encapsulamentos necessários para os traços, reduziria a quantidade de pacotes a serem

tratados pela ferramenta. É importante salientar que este processo alteraria a semântica de funcionamento do agente `RMON2`;

- *Substituição do banco de dados MySQL*: este banco de dados é usado pelo agente `RMON2` e poderia ser substituído por uma alternativa mais eficiente. A maioria das operações realizadas no banco de dados são de procura simples. Embora a linguagem `SQL` ofereça facilidades neste tipo de operação, o uso de arquivos *hashed* tais como os fornecidos pela biblioteca de funções `DBM` seria uma alternativa com melhor desempenho.

7 Conclusões

Este trabalho procurou mostrar a adequação da arquitetura Trace para fins de detecção de intrusão. Uma breve introdução aos temas correlatos, tais como os sistemas de detecção de intrusão e a própria arquitetura Trace, são importantes para a compreensão deste trabalho e foram apresentados nos capítulos 2 e 3. Uma análise mais aprofundada da linguagem que dá suporte à programação do agente de monitoração PTSL (parte integrante da arquitetura Trace) levou a evidências da necessidade de uma extensão da linguagem. Esta extensão foi apresentada no capítulo 4, assim como alguns estudos de casos que foram base para a definição desta extensão. Como forma de validar este trabalho, foi implementado o agente de monitoração, conforme especificação original [GAS 2002] acrescido das extensões propostas neste trabalho. A arquitetura do agente foi apresentada no capítulo 5. A metodologia empregada para validar o agente de monitoração como capaz de reduzir o número de falsos positivos em comparação com outras abordagens foi apresentada no capítulo 6, assim como uma análise de desempenho do mesmo.

Como resultados alcançados, pode-se citar a implementação do agente de monitoração PTSL, com suporte à linguagem PTSL estendida e capaz de responder requisições `SNMP` a uma `MIB RMON2` também estendida. Embora a integração completa com a plataforma Trace ainda não esteja implementada, é possível que o gerente intermediário faça consultas ao agente em questão e tome decisões baseadas em limiares pré-estabelecidos. Como foi mencionado no texto, o foco deste trabalho limitou-se em analisar o agente de monitoração PTSL individualmente, deixando ao administrador a interpretação dos traços de protocolos como evidências de ataques ou não. Uma integração com a plataforma Trace seria bastante benéfica, no sentido que este procedimento poderia ser parcialmente implementado pelo gerente e gerente intermediário. Cabe ressaltar a preservação dos mecanismos e semânticas normalizadas, tais como as da `MIB RMON2`. Houve uma preocupação em não propor novos padrões e sim utilizar os já existentes.

A redução do índice de falsos positivos é um dos objetivos buscados pela maioria dos sistemas de detecção de intrusão atualmente. Procurou-se mostrar com este trabalho que é possível uma redução neste índice utilizando uma abordagem baseada em máquina de estados, tal como o proposto pela arquitetura. A extensão da linguagem é importante para ampliar o escopo de ataques passíveis de serem modelados, sendo importante para classificar a arquitetura como adequada para a detecção de intrusão. Embora a metodologia adotada para a comprovação desta afirmação seja simples, fica evidente que esta abordagem efetivamente reduz o índice de falsos positivos.

A modelagem de ataques através da abordagem proposta na arquitetura é flexível e natural para o administrador. Um ataque é composto por fases que, após identificadas, podem facilmente ser modeladas em uma máquina de estados. Por se tratar de uma linguagem simples, o período de aprendizado do administrador em relação à linguagem é curto. Acredita-se que esta facilidade pode ser altamente benéfica em tempos que a incidência de novos ataques é alta.

Existem uma série de trabalhos futuros a serem desenvolvidos, envolvendo tanto a arquitetura quanto o próprio agente de monitoração:

- *conversor de regras do Snort para o agente*: o Snort é, sem dúvida, o IDS mais

utilizado na atualidade e dispõe de uma vasta relação de ataques modelados. Um conversor de regras para traços seria altamente desejável, no sentido de alimentar o agente de monitoração com mais modelos de ataques. Como os modelos de ataques do Snort são baseados em características encontradas em um único pacote (sem correlação), a conversão para traços é simples e direta;

- *integração com a arquitetura Trace*: na implementação atual, o agente de monitoração PTSL recebe os traços via um arquivo texto. Na concepção original, o agente deveria ser programado via `Script MIB`. Este já é um trabalho sendo desenvolvido pelo grupo;

- *substituição do banco de dados*: na seção dedicada a análise de desempenho, fica evidente a ineficência do banco de dados em tratar dados e requisições da natureza imposta pelo agente. Uma solução mais adequada seria fazer um tratamento dos dados em memória, armazenando os resultados periodicamente em disco. Considerando a especificação de um agente `SNMP`, os contadores deste são inicializados a partir do começo de funcionamento do agente. Desta forma, o armazenamento em disco nem seria necessário;

- *inclusão de novas funções à linguagem*: algumas formas de ataque permanecem sem poderem ser modeladas, mesmo com as extensões propostas. Uma das principais carências da linguagem é a inabilidade em tratar conversão entre números e caracteres. Funções desta natureza, assim como a capacidade de realizar operações aritmética e lógicas, seriam úteis;

- *estudo de técnicas para lidar com fragmentação*: embora seja possível modelar alguns ataques que fazem uso deste artifício, seria necessário uma modificação mais profunda na concepção da linguagem para dota-la de um suporte adequado para tal. A recomposição de fluxos (`TCP`, por exemplo) também não é facilmente modelada.

Como resultado desta dissertação, os seguintes artigos foram publicados em eventos relacionados aos temas do trabalho:

- Trabalhos publicados em caráter regimental:

  1. **Um estudo sobre ferramentas para detecção de intrusão**. *Trabalho Individual I, 2001.* É um estudo sobre o estado da arte em sistemas de detecção de intrusão.
  2. **Proposta de Extensão da Linguagem PTSL e de uma Ferramenta de Monitoração Programável voltadas à Detecção de Intrusão**. *Semana Acadêmica do PPGC, 2000.* Apresenta um estudo introdutório sobre a arquitetura trace e a necessidade de extensão da linguagem PTSL para fins de detecção de intrusão.

- Trabalhos publicados em Congressos (trabalhos completos):

  1. **Uma Ferramenta de Monitoração Programável Voltada à Detecção de Intrusão**. *Conferência Latino Americana de Estudos Avançados em Informática, 2001.* O artigo apresenta a arquitetura trace e como ela pode ser usada como um sistema de detecção de intrusão.

2. **Um Agente SNMP para Detecção de Intrusão Baseada na Monitoração de Interações de Protocolos**. *Simpósio Brasileiro de Redes de Computadores, 2002.* O artigo defende a arquitetura Trace como capaz de melhorar alguns pontos fracos dos sistemas de detecção de intrusão da atualidade, além de propor extensões à linguagem PTSL.

Anexo 1 Ataques Modelados em PTSL

Este anexo apresenta algumas modelagens de ataques utilizadas no capítulo relativo à validação. Várias das extensões propostas neste trabalho são utilizadas nestes exemplos.

A Varredura de portas usando fragmentação corresponde a uma forma de varredura de portas que procura encobrir o ataque através do envio de pacotes fragmentados. A detecção se dá pela teste de tamanho do pacote (menor do que 36 bytes), o que indica que o cabeçalho do protocolo TCP está fragmentado.

```
Trace "Fragmentos"
Version: 1.0
Description: MISC Tiny Fragments
Key: Fragmentos, fragmentação
Owner: Edgar Meneghetti
Last Update: 2001-11-29 22:34:55
Reference:
nograbport
MessagesSection
Message "Fragments"
MessageType: client
MessageTimeout: 1
BitCounter Ethernet 48 4 0010 "MF"
BitCounter Ethernet 16 16 0000000000100111 < "Total Length < 36 bytes"
EndMessage
EndMessagesSection
GroupsSection
EndGroupsSection
StatesSection
FinalState idle
State idle
"Fragments" GotoState idle
EndState
EndStatesSection
EndTrace
```

O ataque Land é de um ataque antigo de negação de serviço. Tem como propósito principal mostrar a utilização de variáveis locais.

A detecção de invasão via *.rhosts* procura pela ocorrência de pacotes que contenham "+ +" em conjunto com "rhosts". É um excelente indício de atividade maliciosa, através da exploração de uma característica dos serviços "r" do *Unix*.

A sondagem de RPC é modelada através dos acessos aos serviços `RPC` disponíveis no alvo. Como existem muitas vulnerabilidades de serviços `RPC` descritas na literatura, pode ser um ótimo indício de preparação a um ataque futuro.

O ataque ao servidor *sadmind* é modelado desde a descoberta da porta `UDP` que o servidor está utilizando (através de uma consulta ao servidor *portmapper*) até a assinatura do ataque via estouro de *buffer*.

```
Trace "LAND"
Version: 1.0
Description: Detecção do ataque Land
Key: Land
Owner: Edgar Meneghetti
Last Update: 2001-11-29 22:34:55
Reference:
MessagesSection
Message "LAND"
MessageType: client
MessageTimeout: 1
BitCounter_LV Ethernet 96 32 src "SRC IP"
BitCounter Ethernet 128 32 $src "DST IP"
EndMessage
EndMessagesSection
GroupsSection
EndGroupsSection
StatesSection
FinalState idle
State idle
"LAND" GotoState idle
EndState
EndStatesSection
EndTrace
```

```
Trace "rhosts com acesso global"
Version: 1.0
Description: deteccao de .rhosts com acesso global
Key: .rhosts
Owner: Edgar Meneghetti
Last Update: 2001-11-29 22:34:55
Reference: arachnids,385
nograbport
MessagesSection
Message "Rhosts"
MessageType: client
MessageTimeout: 1
NoOffSet Ethernet/IP/TCP +|20|+ "Procura + +"
NoOffSet Ethernet/IP/TCP .rhosts "Procura .rhosts"
EndMessage
EndMessagesSection
GroupsSection
EndGroup
EndGroupsSection
StatesSection
FinalState idle
State idle
"Rhosts" GotoState idle
EndState
EndStatesSection
EndTrace
```

```
Trace “Sondagem de RPC”
Version: 1.0
Description: Sondagem de RPC
Key: RPC
Owner: Edgar Meneghetti
Last Update: 2001-11-29 22:34:55
Reference:
nograbport
MessagesSection
Message “PortMapper1”
MessageType: client
MessageTimeout: 2
BitCounter Ethernet/IP 16 16 0000000001101111 “PortMapper”
BitCounter Ethernet/IP/UDP 160 32 00000000000000000000000000000011 “RPC Get Port”
BitCounter Ethernet/IP/UDP 32 32 00000000000000000000000000000000 “Type Call”
EndMessage
Message “PortMapper2”
MessageType: server
MessageTimeout: 2
BitCounter Ethernet/IP/UDP 32 32 00000000000000000000000000000001 “Type Reply”
EndMessage
Message “PortMapper3”
MessageType: client
MessageTimeout: 2
BitCounter Ethernet/IP/TCP 64 32 00000000000000000000000000000000 “Type Call”
BitCounter Ethernet/IP/TCP 128 32 00000000000000011000011010100101 “Program Mount”
EndMessage
EndMessagesSection
GroupsSection
EndGroupsSection
StatesSection
FinalState idle
State idle
“PortMapper1” GotoState RPCState2
EndState
State RPCState2
“PortMapper2” GotoState RPCState3
EndState
State RPCState3
“PortMapper3” GotoState idle
EndState
EndStatesSection
EndTrace
```

```
Trace “ataque ao sadmind”
Version: 1.0
Description: Ataque ao sadmind
Key: sadmind
Owner: Edgar Meneghetti
Last Update: 2001-11-29 22:34:55
Reference: arachnids,20
nograbport
MessagesSection
Message “PortMapper1”
MessageType: client
MessageTimeout: 2
BitCounter_TV Ethernet/IP/UDP 0 32 xid “PortMapper”
BitCounter Ethernet/IP 16 16 0000000001101111 “PortMapper”
BitCounter Ethernet/IP/UDP 32 32 00000000000000000000000000000000 “Type Call”
BitCounter Ethernet/IP/UDP 96 32 00000000000000011000011010100000 “PortMap”
BitCounter Ethernet/IP/UDP 160 32 00000000000000000000000000000011 “GetPort”
BitCounter Ethernet/IP/UDP 320 32 00000000000000011000011110001000 “Sadmind” EndMessage
Message “PortMapper2”
MessageType: server
MessageTimeout: 2
BitCounter_TV Ethernet/IP/UDP 208 16 port “porta UDP do sadmind”
BitCounter Ethernet/IP/UDP 0 32 $xid “PortMapper”
BitCounter Ethernet/IP/UDP 32 32 00000000000000000000000000000001 “Type Reply”
EndMessage
Message “PortMapper3”
MessageType: client
MessageTimeout: 2
BitCounter Ethernet/IP 16 16 $port “Porta do sadmind”
NoOffSet Ethernet/IP/UDP /etc/passwd “Procura /etc/passwd”
EndMessage
EndMessagesSection
GroupsSection
EndGroupsSection
StatesSection
FinalState idle
State idle
“PortMapper1” GotoState RPCState2
EndState
State RPCState2
“PortMapper2” GotoState RPCstate3
EndState
State RPCState3
“PortMapper3” GotoState idle
EndState
EndStatesSection
EndTrace
```

Bibliografia

[ALM 94] ALMEIDA, M. J. B. **Especificação de sistemas na área de redes de computadores**: uma abordagem orientada a objetos. 1994. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[AND 95] ANDERSON, D.; FRIVOLD, T.; VALDES, A. **Next-generation intrusion detection expert system (NIDES)**. Disponível em: <http://www.csl.sri.com/reports/html/tr-abstracts.html≠csl9507>. Acesso em: abr. 2002.

[AND 80] ANDERSON, J. P. **Computer security threat monitoring and surveillance**. Fort Washington, USA: James P. Anderson Co., 1980.

[AXE 2000] AXELSSON, S. **Intrusion detection systems**: a taxomomy and survey. Sweden: Dept. of Computer Engineering, Chalmers University of Technology, 2000. (99-15).

[BAC 2001] BACE, R.; MELL, P. **Nist special publication on intrusion detection systems**. [S.l.]: NIST, 2001. Disponível em: <http://www.nist.org>. Acesso em: abr. 2002.

[BOY 77] BOYER, R. S.; MOORE, J. S. A fast string searching algorithm. **Communications of the ACM**, New York, n.20, p.762–772, 1977.

[BRA 2002] BRAGA, L.; TAROUCO, L. **Monitoracao de protocolos de alto nivel atraves da implementacao de um agente rmon2**. 2002. Dissertação (Mestrado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[CAM 2001] CAMPELLO, R.; WEBER, R. Sistemas de detecção de intrusão. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 19., 2001, Florianópolis. **Minicurso...** Florianópolis: UFSC, 2001.

[CAN 97] CANSIAN, A. **Desenvolvimento de um sistema adaptativo de detecção de intrusos em redes de computadores**. 1997. 154p. Tese (Doutorado em Ciência da Computação) — Universidade de São Paulo, São Carlos, SP. Disponível em: <http://www.dcce.ibilce.unesp.br/ adriano/adrtese-pdf.zip>. Acesso em: abr. 2002.

[CHE 99] CHEUNG, S. et al. **The design of GrIDS**: a graph-based intrusion detection system. [S.l.]: U.C. Davis Computer Science Department, 1999. (CSE-99-2).

[CVE 2002] CVE. **Cve - common vulnerabilities and exposures homepage**. Disponível em: <http://cve.mitre.org/>. Acesso em: abr. 2002.

[DON 98] DONKERS, A. The art and detection of port scanning. **Sys Admin Magazine**, [S.l.], Nov. 1998.

[DUR 99] DURST, R. et al. Testing and evaluating computer intrusion detection systems. **Communications of the ACM**, New York, v.42, n.7, p.53–61, July 1999.

[EME 2002] EMERALD. **EMERALD**: Event Monitoring Enabling Responses to Anomalous Live Disturbances. Disponível em: <http://www.sdl.sri.com/emerald/index.html>. Acesso em: abr. 2002.

[GAR 96] GARFINKEL, S.; SPAFFORD, G. **Pratical unix and internet security**. California, EUA: O'Reilly and Associates, 1996.

[GAS 2001] GASPARY, L. et al. Uma arquitetura para gerenciamento distribuído e flexível de protocolos de alto nível e serviços de rede. In: SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES, SBRC, 19., 2001, Florianópolis. **Anais...** Florianópolis: UFSC, 2001.

[GAS 2002] GASPARY, L. P. **Gerenciamento distribuido e flexivel de protocolos de alto nivel, servicos e aplicacoes em redes de computadores**. 2002. Tese (Doutorado em Ciência da Computação) — Instituto de Informática, Universidade Federal do Rio Grande do Sul, Porto Alegre.

[HAI 2001] HAINES, J. et al. Extending the 1999 evaluation. In: DISCEX 2001, 2001, Anaheim, CA. **Proceedings...** California: [s.n.], 2001.

[HAR 2002] HARDAKER, W. **The netsnmp project**. Disponível em: <http://netsnmp.sourceforge.net/>. Acesso em: abr. 2002.

[HEA 90] HEADY, R.; LUGER, G.; SERVILLA, A. M. M. **The architecture of a network level intrusion detection system**. USA: Departament of Computer Science, University of New Mexico, 1990.

[SNO 2002] HOMEPAGE, S. **Snort**. Disponível em: <http://www.snort.org>. Acesso em: abr. 2002.

[KEM 97] KEMMERER, R. A. **Nstat**: a model-based real-time network intrusion detection system. Santa Barbara, CA: [s.n.], 1997. Disponível em: <http://www.cs.ucsb.edu/ kemm/>. Acesso em: abr. 2002.

[KEM 99] KEMMERER, R. A. **Stat projects**. Santa Barbara, CA: [s.n.], 1999. Disponível em: <http://www.cs.ucsb.edu/ kemm/>. Acesso em: abr. 2002.

[KEN 98] KENDALL, K. **A database of computer attacks for the evaluation of intrusion detection systems**. 1998. Master Thesis in Computer Science — Massachusetts Institute of Technology.

[LEV 2001] LEVI, D.; SCHÖNWÄLDER, J. **Definitions of managed objects for the delegation of management scripts**. Request for Comments 3165. Disponível em: <http://www.ietf.org/rfc.html>. Acesso em: abr. 2002.

[LIN 99] LINDQVIST, U.; PORRAS, P. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In: IEEE SYMPOSIUM ON SECURITY AND PRIVACY, 1999., 1999, Oakland, CA. **Proceedings...** Oakland: [s.n.], 1999. p.9–12.

[LIP 99] LIPPMANN, R. P. et al. Results of the darpa 1998 offline intrusion detection evaluation. In: RAID CONFERENCE, 1999, West Lafayette, Indiana. **Proceedings...** [S.l.: s.n.], 1999.

[LUN 92] LUNT, T. F. **A real-time intrusion detection expert system (IDES)**. 1992. Disponível em: <http://www2.csl.sri.com/projects/intru.shtml>. Acesso em: abr. 2002.

[MAR 92] MARTING, J.; ODELL, J. **Object-oriented analysis and design**. Englewood Cliffs, New Jersey: Prentice-Hall, 1992.

[MCH 2000] MCHUGH, J. the lincoln laboratory intrusion detection evaluation: a critique. In: DARPA INFORMATION SURVIVABILITY CONFERENCE AND EXPOSITION, 2000, Lexington, MA. **Proceedings...** Massachusetts: Massachusetts Institute of Technology, 2000.

[MUK 94] MUKHERJEE, B.; HEBERLEIN, L. T.; LEVITT, K. N. Network intrusion detection. **IEEE Network**, New York, p.26–41, May/June 1994.

[MYS 2002] MYSQL. **Mysql homepage**. Disponível em: <http://www.mysql.org>. Acesso em: abr. 2002.

[NFR 2002] NFR. **Nfr - network flight recorder homepage**. Disponível em: <http://www.nfr.org/>. Acesso em: abr. 2002.

[NMA 2002] NMAP. **Nmap homepage**. Disponível em: <http://www.insecure.org/nmap/>. Acesso em: abr. 2002.

[NOR 2001] NORTHCUTT, S.; NOVAK, J.; MCLACHLAN, D. **Segurança e prevenção em redes**. São Paulo, Brasil: Berkeley, 2001.

[PAX 98] PAXSON, V. Bro: a system for detecting network intruders in real-time. In: USENIX SECURITY SYMPOSIUM, 7., 1998, San Antonio, Texas. **Proceedings...** San Antonio: [s.n.], 1998.

[PHP 2002] PHP. **Php homepage**. Disponível em: <http://www.php.net>. Acesso em: abr. 2002.

[POR 92] PORRAS, P. **Stat - a state transition analysis tool for intrusion detection**. 1992. Master Thesis in Computer Science — University of California, Santa Barbara, California.

[PTA 98] PTACEK, T.; NEWSHAM, T. **Insertion, evasion and denial of service**: eluding network intrusion detection. [S.l.]: Secure Networks Inc, 1998.

[ROE 99] ROESCH, M. Snort - lightweight intrusion detection for networks. In: LISA, 1999. **Proceedings...** [S.l.: s.n.], 1999. Disponível em: <http://www.snort.org/>. Acesso em: abr. 2002.

[RUS 91] RUSSEL, D.; GANGEMI, G. **Computer security basics**. California: O'Reilly and Associates, 1991.

[SEC 2002] SECURITYFOCUS. **Securityfocus homepage**. Disponível em: <http://online.securityfocus.com/>. Acesso em: abr. 2002.

[SMA 92] SMAHA, S. Questions about cmad. In: WORKSHOP ON FUTURE DIRECTIONS IN COMPUTER MISUSE AND ANOMALY DETECTION, 1992, Davis, California. **Proceedings...** Davis: University of California, 1992. p.17–21.

[SMA 88] SMAHA, S. E. Haystack - an intrusion detection system. In: AEROSPACE COMPUTER SECURITY APPLICATION CONFERENCE, 1988, Austin, Texas. **Proceedings...** Austin: [s.n], 1988. p.37–44.

[SOA 95] SOARES, L. F. G.; COLCHER, S.; LEMOS, G. **Redes de computadores**: das LANs, MANs e WANs às Redes ATM. Rio de Janeiro: Campus, 1995.

[SOU 87] SOUZA, J.; MARTINI, M. Avaliação de confiabilidade de sistemas. In: SIMPÓSIO EM SISTEMAS DE COMPUTADORES TOLERANTES A FALHAS, 2., 1987, Campinas. **Anais...** Campinas: Unicamp, 1987. p.24–28.

[SPA 89] SPAFFORD, E. Crisis and aftermath. **Communications of the ACM**, New York, v.32, n.6, p.678–686, June 1989.

[STA 95] STALLINGS, W. **Network and internetwork security**. Upper Saddle River, New Jersey: Prentice Hall, 1995.

[SYS 2002] SYSTEMS, I. S. **Real secure**. Disponível em: <http://www.iss.net>. Acesso em: abr. 2002.

[TCP 2002] TCPDUMP and libpcap. Disponível em: <http://www.tcpdump.org>. Acesso em: abr. 2002.

[VIG 2000] VIGNA, G.; ECKMANN, S. T.; KEMMERER, R. A. The stat tool suite. In: DISCEX, 2000, USA. **Proceedings...** [S.l.]: IEEE Press, 2000.

[VIG 98] VIGNA, G.; KEMMERER, R. A. NetSTAT: a network-based intrusion detection approach. In: ANNUAL COMPUTER SECURITY APPLICATIONS CONFERENCE, 14., 1998, Scottsdale, AZ. **Proceedings...** Scottsdale: [s.n.], 1998.

[WAL 97] WALDBUSSER, S. **Remote network monitoring management information base version 2 using smiv2**. Request for Comments 2021. Disponível em: <http://www.ietf.org/rfc.html>. Acesso em: abr. 2002.

[WHI 2002] WHITEHATS. **Whitehats homepage**. Disponível em: <http://www.whitehats.com/>. Acesso em: abr. 2002.

[ZIR 2000] ZIRKLE, L.; DRAKE, G.; DITTRICH, D. **Incident handling step by step**: unix trojan programs. USA: SANS Institute, 2000. Disponível em: <http://www.sans.org/y2k/DDoS.htm>. Acesso em: abr. 2002.