Um Agente SNMP para Detecção de Intrusão Baseada na Monitoração de Interações de Protocolos

Edgar Meneghetti[1], **Luciano Gaspary**[2], **Liane Tarouco**[1]

[1]Universidade Federal do Rio Grande do Sul, Instituto de Informática
Av. Bento Gonçalves, 9500 - Agronomia - CEP 91591-970 - Porto Alegre, Brasil

[2]Universidade do Vale do Rio dos Sinos, Centro de Ciências Exatas e Tecnológicas
Av. Unisinos 950 - CEP 93.022-000 - São Leopoldo, Brasil

edgar@cesup.ufrgs.br, paschoal@exatas.unisinos.br, liane@penta.ufrgs.br

Resumo

Este artigo propõe a utilização da arquitetura Trace como um sistema de detecção de intrusão. A arquitetura, apresentada em [1, 2, 3], 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, o gerente de rede utiliza uma linguagem baseada em máquinas de estados. O artigo defende, através de exemplos, que essa linguagem é adequada para a modelagem de assinaturas de ataques e propõe algumas extensões para permitir a especificação de um número maior de cenários ligados ao gerenciamento de segurança. Em seguida, o artigo descreve 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 facilita a recuperação de informações relativas à ocorrências dos ataques. A solução apresentada aborda três problemas de sistemas de detecção de intrusão: o excesso de falsos positivos, a dificuldade em se modelar certos ataques e o descarte de pacotes (quando usados em redes de alta velocidade).

Abstract

This paper proposes the use of Trace architecture as an intrusion detection system. The architecture, presented in [1, 2, 3], supports high-layer protocol, service and application management through an approach based on passive observation of protocol interactions (traces) in the network traffic. To describe the scenarios to be monitored, the network manager uses a state machine-based language. By using examples, the paper shows that this language is suitable for attack signature modeling and proposes some extensions to allow the specification of a higher number of scenarios related to security management. Next, the paper describes the implementation of the monitoring agent, key-component from Trace architecture, and its use to detect intrusions. 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). The use of SNMP let queries to the monitoring agent be easily made. The solution presented addresses three problems from intrusion detection systems: the great number of false positives, the inability to model some attacks and packet discards (when used in high speed networks).

Palavras-chave: detecção de intrusão, segurança, gerenciamento de redes de computadores, monitoração, RMON2.

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 [4]. 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 dos indícios de atividade maliciosa dos 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 [5, 6]. 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, é a análise por anomalia [5, 6], que utiliza métodos estatísticos que procuram separar o comportamento normal do não esperado.

Vários problemas são relatados em relação aos IDSs existentes atualmente: excesso de falsos positivos, dificuldade para modelar determinadas assinaturas e descarte de pacotes sob alto tráfego [6, 7]. 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 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 [8].

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 [9]. 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 RSTs 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 enfrentados pela geração atual de IDSs [7, 10]. 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, o artigo 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 [1, 2, 3], 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, apenas cenários ligados à segurança (detecção de intrusão)

são abordados no artigo. Um aspecto importante do agente é que as informações armazenadas por ele são disponibilizadas via SNMP através de objetos gerenciáveis da MIB RMON2 [11]. Foi implantado a partir de uma expansão do agente NET-SNMP [12].

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 [1, 2, 3]. 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. Para resolver esse problema, algumas extensões foram propostas. O artigo está organizado da seguinte forma. Na seção 2 são citados alguns trabalhos relacionados. Na seção 3 é apresentada uma descrição sucinta da arquitetura Trace. A linguagem PTSL e algumas assinaturas de ataque descritas com a linguagem (uma das importantes contribuições do artigo) são apresentadas na seção 4. Na seção 5 são apresentadas as extensões incorporadas à linguagem. A arquitetura do agente de monitoração é abordada na seção 6. Na mesma seção, uma análise do agente quanto ao desempenho e à escalabilidade é apresentada. A seção 7 encerra o artigo com algumas considerações finais e perspectivas de trabalhos futuros.

2 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.

A ferramenta Snort [10] aparece como o sistema mais utilizado atualmente para detecção de intrusão, possuindo um desempenho muito bom e uma linguagem procedural flexível para descrição de ataques. O processo de detecção de intrusão do Snort baseia-se na aplicação de regras de filtragem em cada pacote (o que impossibilita a correlação de eventos), a fim de procurar assinaturas de ataques conhecidos. Embora possua uma vasta coleção de ataques descritos através de sua linguagem, o Snort não faz uma análise baseada em estados dos dados coletados, o que acaba ocasionando falsos positivos em excesso.

Uma ferramenta semelhante à anterior é o Bro [13], desenvolvido pelo *Lawrence Berkeley National Laboratory*. Conceitualmente é dividida em dois componentes: uma máquina de eventos, responsável por analisar um fluxo de pacotes já previamente filtrados, e um interpretador de scripts, responsável pelo processamento da linguagem de descrição de políticas. A linguagem adotada para analisar os dados coletados da rede é sintaticamente semelhante à linguagem C. Alguns ataques foram modelados a título de experimento, mas a modelagem destes mesmos ataques não é uma tarefa trivial.

No âmbito comercial, vários sistemas de detecção de intrusão surgiram nos últimos anos. O RealSecure [14] é muito usado no mercado, possuindo uma arquitetura híbrida baseada em gerentes e agentes. Como ponto forte pode-se destacar a fácil administração e a boa integração com outros mecanismos de segurança, tais como um *firewall*.

O sistema para detecção de intrusão NFR [7] apresenta um mecanismo mais elaborado para o processo de detecção de intrusão, fornecendo uma linguagem procedural bastante extensa e de difícil aprendizado (N-code), bem como a possibilidade de se medir o grau de anormalidade de um determinado pacote. O grau de anormalidade é medido através da ocorrência, freqüente ou não, dos pares endereço IP e porta do serviço em um intervalo de tempo. Um ponto forte da ferramenta é a remontagem de fluxo de dados, seja de pacotes fragmentados, seja de um fluxo

de dados TCP.

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. O que se busca é 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 itens. Além de oferecem suporte limitado à especificação de assinaturas, oferecem linguagens de muito baixo nível (que o gerente acaba ignorando pela sua complexidade).

3 A arquitetura Trace

A arquitetura Trace é uma extensão da infra-estrutura centralizada de gerenciamento SNMP, para, através de um modelo em três níveis, suportar o gerenciamento distribuído de protocolos de alto nível e serviços de rede. A figura 1 ilustra um esquema da arquitetura. A principal contribuição da mesma é a presença de agentes de monitoração extensíveis ou programáveis, capazes de receber dinamicamente descrições de traços de protocolos (especificações PTSL) e passar a monitorar a sua ocorrência na rede. As estatísticas coletadas são armazenadas em uma MIB construída a partir da RMON2. 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, instancia e dispara um script em um agente de ação. O agente de ação reside na estação onde o serviço monitorado se encontra. Os scripts disparados por esse agente realizam, por exemplo, o reinício do serviço ou a reconfiguração do mesmo. A arquitetura apresenta ainda mecanismos de notificação (traps) para que o gerente intermediário possa reportar eventos mais significativos à estação de gerenciamento.

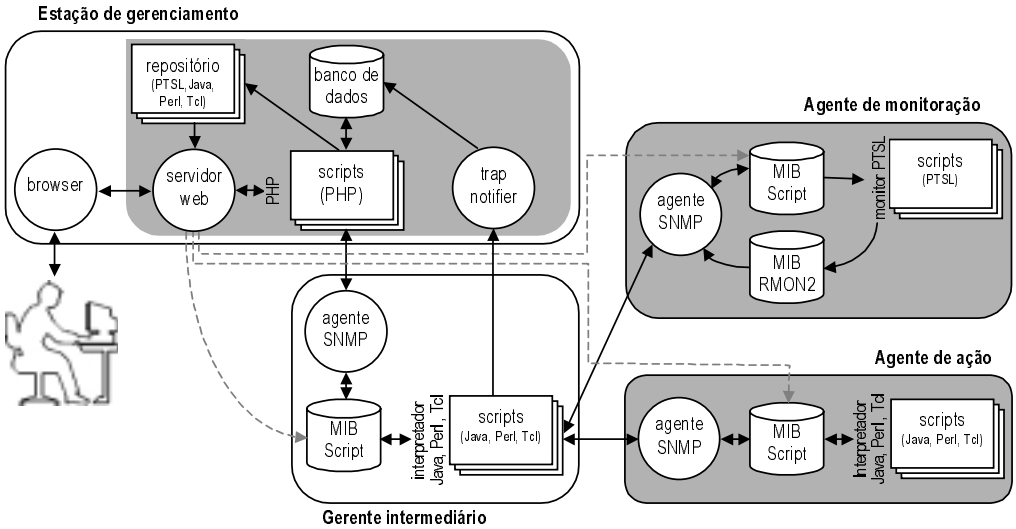

Figura 1: Componentes da arquitetura Trace (extraída de [2])

Conforme já mencionado, 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 nas estações monitoradas. Se a classificação for pelo método empregado, pode-se classificar

a arquitetura 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, por exemplo, ao observar a ocorrência de um determinado traço em um horário não usual.

Os passos executados e o fluxo de informações para a execução de uma tarefa de gerenciamento ligada à detecção de intrusão, usando a arquitetura Trace, são apresentados na figura 2. A partir da estação de gerenciamento, o gerente precisa definir um traço de protocolo usando a linguagem PTSL e uma tarefa de gerenciamento que, na verdade, é uma regra que determina quantas vezes o traço precisa ser observado em um determinado intervalo de tempo para se identificar a ocorrência de uma intrusão. Após realizadas as especificações, a tarefa de gerenciamento é delegada a um gerente intermediário (o mais próximo do serviço a ser monitorado). A partir daí, o gerente intermediário programa um agente de monitoração para que passe a observar a ocorrência do traço. Além disso, ele consulta periodicamente (via SNMP) a MIB RMON2 estendida onde os resultados da monitoração são armazenados. Se em um intervalo de consulta o número de ocorrências superar um determinado valor, definido pelo gerente na especificação da tarefa de gerenciamento, o *script* gera uma notificação à estação central de gerenciamento. É possível ainda disparar uma ação (ex: reconfiguração do *firewall*).

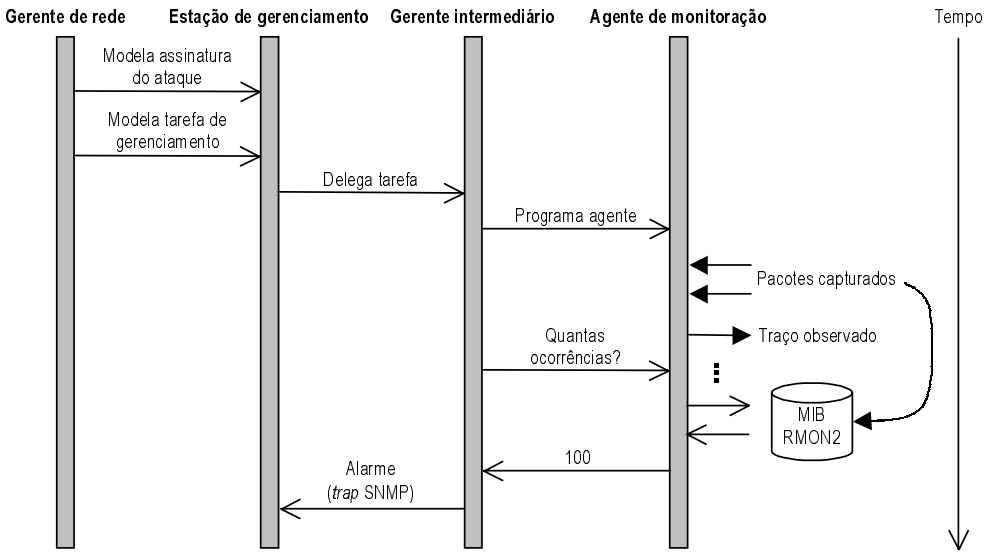

Figura 2: Fluxo de informações para programar uma nova assinatura de ataque

4 Modelagem de assinaturas de ataques usando PTSL

A arquitetura Trace fornece uma linguagem gráfica e textual (PTSL) que permite descrever traços de protocolos. Essa descrição é feita através de estados e transições. Utilizando a linguagem PTSL para descrever cenários de ataques, obtém-se uma descrição precisa do processo de ataque e, por conseqüência, consegue-se realizar a detecção de intrusão com menor índice de falsos positivos. Uma série de ataques conhecidos [6, 9, 15, 16] foi modelada utilizando as notações gráfica e textual da linguagem PTSL. A seguir são apresentados alguns exemplos.

Conforme se pode observar na figura 3, a notação gráfica corresponde a uma máquina de estados, onde as transições em linha cheia representam mensagens do cliente para o servidor e as linhas pontilhadas representam mensagens do servidor para o cliente. É importante salientar que os termos "cliente" e "servidor" utilizados têm como função apenas identificar as duas partes envolvidas na comunicação.

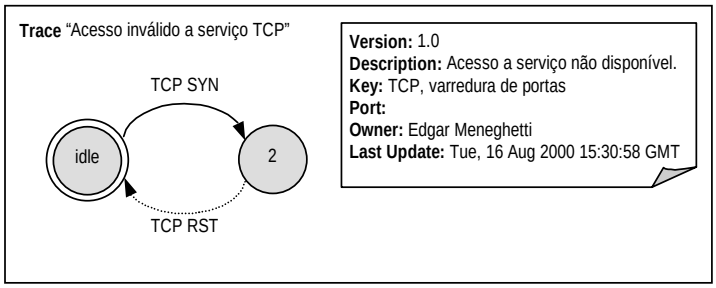

Figura 3: Traço para detectar varredura de portas

A figura 3 ilustra um traço que permite monitorar a ocorrência de varredura de portas, técnica usada por atacantes para sondar quais são os possíveis serviços TCP e UDP disponíveis em um equipamento alvo. Existem várias técnicas para realizar essa varredura [9], sendo a mais simples o envio de pacotes para todas as portas de uma estação. 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.

Outras técnicas de varredura de portas são facilmente implementadas 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).

Os artifícios para encobrir a varredura de portas tais como os algoritmos que usam uma seqüência aleatória de portas ou com intervalos de tempo grandes são problemas tratados pontualmente pelas ferramentas para detecção de intrusão. No caso do traço modelado acima, esse problema não se aplica. A seqüência de números de portas não é importante para a detecção da ocorrência do traço, assim como o intervalo entre uma conexão e outra. Esse último problema deverá ser endereçado pelo gerente intermediário, ao definir quais são os limites aceitáveis de ocorrência desse traço em um período de tempo.

A especificação textual do traço apresentado na figura 3 pode ser observada na figura 4. Os principais construtores da linguagem são descritos a seguir. A especificação completa da linguagem pode ser obtida em [2]. A especificação de um traço inicia com a palavra-chave *Trace* e termina com *EndTrace* (linhas 1 e 27). O nome do traço (linha 1) é seguido de um conjunto de informações opcionais (linhas 2 a 7). Em seguida, a especificação é dividida em três seções: *MessagesSection* (linhas 8 a 17), *GroupsSection* (não utilizada no exemplo) e *StatesSection* (linhas 18 a 26).

Em *MessagesSection* são definidas as mensagens que causarão a evolução do traço. Quando duas ou mais mensagens causam a transição de um mesmo estado para outro, é possível agrupá-las, tornando a especificação mais clara. Esses agrupamentos são definidos na seção *GroupsSection*. Por fim, na seção *StatesSection* é definida a máquina de estados que representa o traço.

A evolução da máquina de estados ocorre quando um pacote com campos idênticos aos especificados em uma *client* ou *server message* é observado na rede. No exemplo apresentado na figura 3, a máquina de estados evolui do estado *idle* para o estado 2 ao observar uma mensagem (ou pacote) TCP-SYN.

Para identificar essa (TCP- SYN) e as demais transições, a linguagem PTSL oferece um mecanismo baseado em um sistema posicional para determinar os campos de um encapsulamento a serem analisados. A diferença entre os protocolos (baseados em caracter e binários) requer a utilização de duas abordagens distintas para a identificação desses campos. São elas:

```
1  Trace "Acesso inválido a serviço TCP"
2  Version: 1.0
3  Description: Acesso a serviço não disponível.
4  Key: TCP, varredura de portas
5  Port:
6  Owner: Edgar Meneghetti
7  Last Update: Tue, 16 Aug 2000 15:30:58 GMT
8  MessagesSection
9  Message "TCP SYN"
10 MessageType: client
11 BitCounter Ethernet/IP 110 1 1 "Campo SYN - 1 significa TCP Connect"
12 EndMessage
13 Message "TCP RST"
14 MessageType: server
15 BitCounter Ethernet/IP 109 1 1 "Campo RST"
16 EndMessage
17 EndMessagesSection
18 StatesSection
19 FinalState idle
20 State idle
21 "TCP SYN" GotoState 2
22 EndState
23 State 2
24 "TCP RST" GotoState idle
25 EndState
26 EndStatesSection
27 EndTrace
```

Figura 4: Especificação textual do traço para detectar varredura de portas

- *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 (não usado no exemplo).
- *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 (linhas 11 e 15).

Para identificar um campo *FieldCounter* é preciso definir:

- *Encapsulamento*: 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 do protocolo TCP;
- *Posição*: indica a posição do campo na mensagem; a primeira posição é identificada por 0, a segunda por 1 e assim por diante;
- *String de comparação*: uma vez identificado o campo no pacote, é feita a comparação entre ele e a string de comparação. A evolução da máquina de estados acontecerá quando, para um pacote capturado, as strings de comparação de todos os campos especificados coincidirem com os valores observados no pacote;
- *Descrição*: breve comentário a respeito do campo.

As informações associadas a um campo *BitCounter*, por sua vez, são as seguintes:

- *Encapsulamento*: identifica onde o campo deve ser buscado; no exemplo, o encapsulamento definido para ambos os campos é Ethernet/IP, indicando que o protocolo a ser monitorado será encontrado no campo de dados do protocolo IP;
- *Posição inicial*: indica o *offset* em bits a partir do início do cabeçalho do protocolo em questão até o início do campo desejado; o primeiro bit é o 0;
- *Comprimento do campo*: indica o tamanho em bits do campo;
- *String de comparação*: *string* usada na comparação com o campo selecionado no pacote;
- *Descrição*: breve comentário a respeito do campo.

A seção *StatesSection* permite especificar, de forma textual, a máquina de estados que representa o traço. O estado final é identificado logo após o início da seção *StatesSection* (linha 19). Os estados *idle* e 2 são definidos, respectivamente, nas linhas 20 a 22 e 23 a 25. A especificação de um estado se resume a listar os eventos (mensagens e agrupamentos) que podem provocar uma transição e indicar, para cada um deles, o próximo estado (linhas 21 e 24).

Além das abordagens *BitCounter* e *FieldCounter*, a linguagem ainda proporciona a abordagem *NoOffSet*, que permite a localização de seqüência de caracteres (*strings*) na área de dados do protocolo especificado. Agrupamento de mensagens também é possível através da declaração *GroupsSection*, o que equivale a uma operação lógica "ou" entre as mensagens especificadas na seção *MessagesSection*.

A figura 5 ilustra um caso de comportamento anômalo, onde o pacote de início da conexão TCP (SYN) carrega dados [6]. Esse recurso é utilizado para enganar algum sistema de segurança que procura pela ocorrência de determinadas palavras-chave no *payload* do TCP, mas que em geral não faz essa verificação no *handshake*.

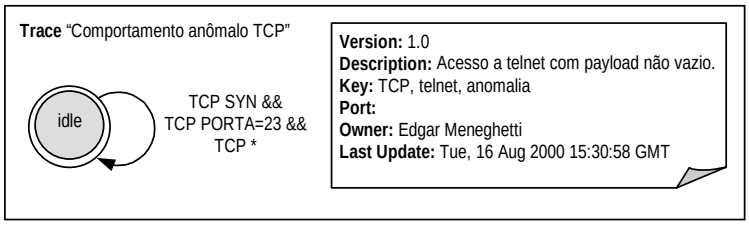

Figura 5: Comportamento anômalo

Um 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 na estação. 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 figura 6.

É importante salientar que esse traço irá capturar tráfego legítimo também, ou seja, a ocorrência do mesmo, analisada 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. Esse traço, contudo, só será observado durante a negociação para a montagem de sistemas de arquivos externos e não mais durante o 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.

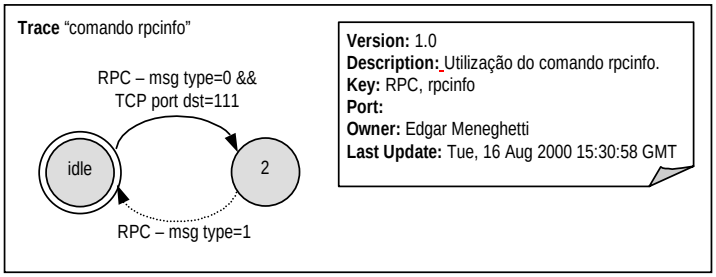

Figura 6: Traço de ocorrência do comando *rpcinfo*

5 Extensões à linguagem PTSL

Do ponto de vista de detecção de intrusão, embora a linguagem PTSL permita a modelagem de vários cenários de ataques, identificou-se algumas limitações. Para resolvê-las foi proposta uma extensão à mesma, incluindo novos construtores e algumas funcionalidades, discutidas a seguir.

A falta de operadores de comparação é uma das limitações. A única operação de comparação admitida é o teste de igualdade. Contudo, para modelar alguns traços, em especial os ligados à detecção de intrusão, é importante comparar o valor encontrado em campos de um pacote com alguns limiares. Para resolver esse problema os operadores menor (<), menor ou igual (<=), maior (>), maior ou igual (>=) e diferente de (! =) foram acrescentados à linguagem. Quando nenhum operador é informado o teste de igualdade é realizado. Um exemplo de uso desses operadores pode ser observado na modelagem do traço que descreve o acesso a servidores HTTP com URLs longas, comportamento que caracteriza muitos ataques a servidores HTTP. A figura 7 apresenta a descrição, em PTSL, da mensagem que permite identificar requisições HTTP (GET) com URLs longas.

```
...
Message "Requisição HTTP com URL longa"
MessageType: client
BitCounter Ethernet 16 16 0000000111110100 >= "URL longa, pacote maior do que 500"
BitCounter Ethernet/IP 16 16 0000000001010000 == "Porta destino 80"
FieldCounter Ethernet/IP/TCP 0 GET == "Operação GET"
EndMessage
...
```

Figura 7: Exemplo de utilização de operadores na linguagem PTSL

A semântica do uso de vários campos em uma mensagem corresponde a uma operação lógica "e". Assim, tem-se acima uma mensagem caracterizada por ser uma conexão na porta 80, realizando uma operação GET no servidor HTTP, na qual o tamanho do pacote seja maior do que 500. É importante notar que a URL longa é obtida indiretamente pelo tamanho do pacote.

A segunda funcionalidade não disponível na versão original da linguagem PTSL é a possibilidade de comparar o valor de um campo com o de outro campo (pertencentes a um mesmo pacote). A solução encontrada para a essa limitação foi a adoção do conceito de variáveis, com escopo limitado ao traço em andamento, conforme ilustra a figura 8. No exemplo, a variável a está sendo usada para comparar se os endereços IP origem e destino de um pacote são iguais. A observação dessa mensagem em um segmento de rede indica que uma estação a está sendo atacada através da técnica conhecida como *Land*.

Outra extensão está relacionada com a possibilidade de usar caracteres delimitadores diferentes do espaço, quando usada a abordagem FieldCounter. O exemplo apresentado na figura 9 ilustra uma requisição HTTP onde o atacante procura passar 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

```
...
Message "Mensagem com IP origem e destino iguais"
MessageType: client
BitCounter Ethernet 96 32 = a "Endereço IP origem é atribuído à variavel a"
BitCounter Ethernet 128 32 $a == "IP destino é igual ao valor de a ?"
EndMessage
...
```

Figura 8: Exemplo de utilização de variáveis na linguagem PTSL

arquivos existentes no servidor.

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

Figura 9: 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 figura 10. Como pode ser observado, deseja-se observar 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:\ == ? "Segundo campo, com separador '?' "
```

Figura 10: Utilização de outros separadores além do espaço

Com a alteração na sintaxe da linguagem, alguns caracteres passaram a ter significado diferenciado, como o caracter "$" por exemplo. O tratamento para esses casos é o usual, onde se usa o caracter "\" como normalizador do caracter que vier em seguida.

A questão de remontagem de pacotes fragmentados ou recomposição de fluxos de dados não pode ser tratada de forma direta com a abordagem utilizada na linguagem. O tratamento dessas situações é importante no contexto de detecção de intrusão, sendo possivelmente alvo de estudo em versões futuras da linguagem.

6 O agente de monitoração

Do ponto de vista do uso da arquitetura Trace como ferramenta para detecção de intrusão, os agentes de monitoração programáveis são o ponto mais importante da solução. Esta seção descreve como é feita a interação entre o agente e o gerente intermediário, assim como mostra detalhes de como o agente foi implementado.

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. Essa flexibilidade é obtida através da linguagem PTSL, apresentada nas seções 4 e 5. Na prática, os agentes realizam a leitura de arquivos PTSL, organizam algumas estruturas em memória e iniciam o processo de monitoração.

A determinação de que traços devam ser monitorados em um dado momento é realizada pelo gerente intermediário. A interface de comunicação entre o gerente intermediário e o agente de monitoração é a MIB Script [17]. No *script* executado pelo gerente intermediário, descrito em [1, 2, 3], é possível observar como é feita a programação de um agente de monitoração. Um dos parâmetros informados no *script* executado no gerente intermediário é o endereço URL do script (especificação PTSL) a ser executado. Ao receber do gerente intermediário a solicitação de execução de um determinado *script*, esse é recuperado do repositório via HTTP e executado.

6.1 Implementação do agente de monitoração

O agente de monitoração foi implementado na plataforma Linux usando a linguagem C e a biblioteca de *threads* POSIX. Uma visão mais detalhada da arquitetura do agente é apresentada na figura 11. A *thread* gerente PTSL é responsável pela integração entre a MIB Script e a máquina de execução PTSL. Ela atualiza as estruturas usadas pela máquina PTSL sempre que um novo traço é programado para ser monitorado ou um traço existente é removido (por não ser mais necessário). Outras três *threads* – fila, máquina PTSL e RMON2 – operam segundo o paradigma produtor/consumidor. A primeira é responsável por capturar todos os pacotes usando a biblioteca *libpcap* e adicioná-los a uma fila circular. Embora essa biblioteca suporte a especificação de filtros utilizando a notação BPF (*Berkeley Packet Filter*), esse recurso não está sendo utilizado pelo agente. A segunda *thread* processa cada pacote presente na fila, sem retirá-lo dela, com o objetivo de identificar se o mesmo possui as características esperadas para permitir que um ou mais traços evoluam na máquina de estados. Caso afirmativo, o pacote recebe marcações especiais. A *thread* RMON2, por fim, retira cada pacote da fila e, de acordo com as marcações, faz as devidas contabilizações no banco de dados (mySQL). Essa base de dados é consultada por uma extensão ao agente NET-SNMP [12] que implementa a MIB RMON2 estendida (detalhada a seguir).

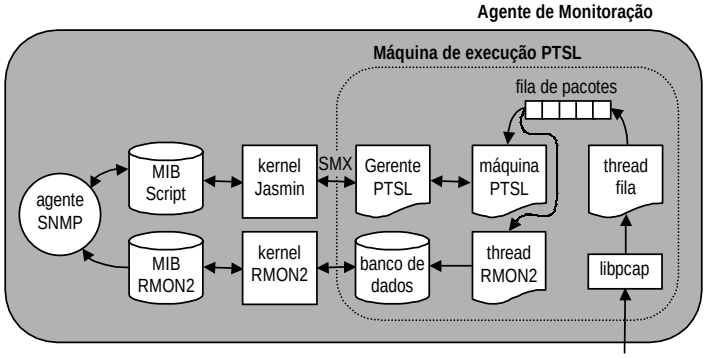

Figura 11: Agente de Monitoração

A thread responsável pela captura e classificação dos pacotes insere-os em uma fila circular, anexando a cada pacote dados que facilitam o processamento destes pela máquina de estados. Entre esses dados auxiliares, estão ponteiros para os cabeçalhos e carga de cada um dos protocolos encapsulados no pacote. Uma máscara que contém todos os protocolos presentes no pacote também é anexada ao mesmo. Ao encher a lista circular, os pacotes mais antigos passam a ser descartados.

A implementação da máquina de estados é realizada através da definição de várias listas. O conteúdo de cada uma destas listas esta listado a seguir:

- dados de cada um dos traços, tais como nome do traço, versão, autor entre outros;
- dados de cada estado, definindo se é um estado inicial ou final e a qual traço esta relacionado, assim como o tempo máximo de permanência nesse estado (timeout);
- relação de mensagens existentes em cada traço, apontando o estado de origem e destino para cada mensagem;
- tipos de estratégia de localização da informação no pacote para cada mensagem;
- relação de máscaras de protocolos que foram definidos nos traços. Esta lista otimiza o processo, visto que apenas os pacotes que contenham protocolos definidos em algum traço serão analisados;
- Relação de traços em andamento para cada par de estações.

Mais informações sobre a implementação do agente de monitoração são encontradas em [18].

6.2 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 [11]. É importante salientar 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 implementação atual, existe suporte para Ethernet, IP, TCP e algumas aplicações. 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 [19], que constitui-se de uma extensão do agente NET-SNMP [12].

Uma das diferenças da MIB usada com relação à RMON2 é que o grupo *protocolDir*, que indica os protocolos que o agente é capaz de monitorar, deixa de ser um grupo de leitura e passa a permitir a inclusão de traços monitorados pelo agente de monitoração. Além disso, a granularidade da monitoração torna-se maior. 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 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 1: Grupo *protocolDir* da MIB RMON2

ID	*LocalIndex*	*Descr*
00-00-00-01-00-00-08-00	1	ether2.ip
00-00-00-01-00-00-08-00-00-00-00-17	2	ether2.ip.tcp
00-00-00-01-00-00-08-00-00-00-00-17-00-00-00-19	3	ether2.ip.tcp.smtp
00-00-00-01-00-00-08-00-**00-01-00-01**	4	ether2.ip.varredura de portas
00-00-00-01-00-00-08-00-00-00-00-17-**00-01-00-02**	5	ether2.ip.tcp.ataque servidor 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 desse 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 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 obedece a mesma regra.

6.3 Desempenho

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 utilizada foi um K6II-450 MHz, 64 Mbytes de memória RAM.

Tabela 2: Informações obtidas com consulta à tabela alMatrixSD

Source Address	*Destination Address*	*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

- 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 correspondem à modelagem feita no traço;
- 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;
- Gerando tráfego na ordem de 10Mbits/s (em torno de 5000 datagramas/segundo), com todos os datagramas fazendo parte do traço, o agente descarta pacotes;

O agente colocado em ambiente de produção e monitorando 10 traços não descarta pacotes. O ambiente está caracterizado a seguir.

- Rede ethernet 10Mbits/s em um segmento conectado por *HUB*;
- 10 estações rodando sistema operacional Microsoft Windows, com compartilhamento de arquivos e impressoras;
- Tráfego médio de 150 pacotes por segundo em horário comercial;
- Tamanho médio de pacotes variando entre 65 e 256 bytes (75%) e maior do que 1024 bytes (21%);
- Perfil de protocolos de aplicação variando entre acesso à servidores WEB (15%), NetBIOS (7%) e correio eletrônico (2%).

Com a crescente disponibilização de largura de banda nas redes locais, é necessário buscar alternativas que não comprometam o processo de detecção de intrusão. 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 na biblioteca de captura, filtrando os pacotes que contêm os encapsulamentos necessários para os traços. Essa alternativa reduziria a quantidade de pacotes a serem tratados pela ferramenta, embora alteraria a semântica de funcionamento do agente RMON2;
- Substituição do banco de dados mySQL (usado pelo agente RMON2) por uma alternativa mais eficiente. A maioria das operações realizadas no banco de dados são de procura simples. Embora a linguagem SQL (*Structured Query Language*) ofereça facilidades neste tipo de operação, o uso de arquivos *hashed* tais como os fornecidos pela biblioteca de funções DBM seriam uma alternativa com melhor desempenho.

7 Conclusões

Este artigo apresentou como a arquitetura Trace pode ser utilizada como uma ferramenta para detecção de intrusão baseada em rede. Alguns cenários de ataques foram descritos e pôde-se verificar a naturalidade com que se modelam ataques usando-se estados e transições. A linguagem PTSL mostra-se bastante adequada para a especificação de cenários de ataques, tanto pela sua simplicidade quanto pela facilidade e rapidez em descrever um traço. A proposta de extensão da linguagem foi apresentada com o objetivo de aumentar o escopo de ataques passíveis de serem modelados.

O poder de expressão de PTSL e da extensão proposta é um ponto a ser destacado. A possibilidade de correlacionar pacotes, pertençam ele a um mesmo fluxo ou não, permite identificar, com menor chance de erros, indícios de ataques, reduzindo sensivelmente o número de alarmes falsos. Além disso, enquanto boa parte das abordagens permite selecionar pacotes com base em alguns campos pré-determinados de protocolos até, no máximo, a camada de transporte, PTSL vai além ao possibilitar que a filtragem seja feita com base em quaisquer campos de quaisquer protocolos, incluindo os do nível de aplicação. Como deficiência da linguagem, fica a impossibilidade de se recompor fluxos de dados. Em uma sessão de telnet, por exemplo, a monitoração de senhas fracas não seria possível, devido à impossibilidade de agregar todos os dados encapsulados no protocolo TCP para fazer essa análise.

O uso de uma MIB RMON2 para armazenar os traços observados é particularmente interessante. Embora tenha sido apresentada toda a arquitetura Trace, é possível montar uma ferramenta para detecção de intrusão apenas com o agente de monitoração. Para tal, basta adotar algum software que permita programar e consultar objetos SNMP. O desempenho do agente é adequado para redes com baixo tráfego, mas deve ser repensado para uma infra-estrutura de redes operando na ordem de Gigabits por segundo. Entretanto, a distribuição ordenada desses agentes nos segmentos de rede mais significativos e a utilização de poucos traços em cada um deles reduzem o problema sem causar problemas de escalabilidade (graças aos gerentes intermediários).

A integração do agente com os demais componentes da arquitetura está em andamento. As próximas atividades previstas são o estudo da utilização do mecanismo de filtragem da biblioteca libpcap como forma de melhorar o desempenho, além da investigação de algoritmos mais eficazes para tratamento dos pacotes capturados da rede.

Referências

[1] GASPARY, L. P.; BALBINOT, L. F.; STORCH, R.; WENDT, F.; TAROUCO, L. R. *Towards a Programmable Agent-based Architecture for Enterprise Application and Service Management.* Proc. First IEEE/IEC Enterprise Networking Applications and Services Conference, Atlanta, June 2001.

[2] GASPARY, L. P.; BALBINOT, L. F.; STORCH, R.; WENDT, F.; TAROUCO, L. R. *Uma Arquitetura para Gerenciamento Distribuído e Flexível de Protocolos de Alto Nível e Serviços de Rede.* Anais do XIX Simpósio Brasileiro de Redes de Computadores, Florianópolis, Maio de 2001.

[3] GASPARY, L. P.; BALBINOT, L. F.; STORCH, R.; WENDT, F.; TAROUCO, L. R. *Distributed Management of High-layer Protocols and Network Services Through a Programmable Agent-based Architecture.* Proc. IEEE International Conference on Networking, Colmar, France, July 2001.

[4] STALLINGS, W. *Network And Internetwork Security.* Upper Saddle River, New Jersey: Prentice Hall, 1995.

[5] CAMPELLO, R.; WEBER, R. *Sistemas de Detecção de Intrusão*. Minicurso do XIX Simpósio Brasileiro de Redes de Computadores, Florianópolis, Maio de 2001.

[6] NORTHCUTT, S; NOVAK, J; MCLACHLAN, D. *Segurança e Prevenção em Redes*. São Paulo, Brasil: Editora Berkeley, 2001.

[7] NETWORK FLIGHT RECORDER INC., *Network Flight Recorder Homepage*, Disponível em http://www.nfr.com.

[8] VIGNA, G., ECKMANN, S.T., KEMMERE, R.A. *The STAT Tool Suite. Proc. of DISCEX 2000*, Hilton Head, South Carolina, January 2000, IEEE Press.

[9] DONKERS, A. *The Art and Detection of Port Scanning*, Sys Admin. Mag., Nov. issue, 1998.

[10] ROESCH, M. *Snort - Lightweight Intrusion Detection for Networks*, Proc. LISA'99, Nov. 1999.

[11] WALDBUSSER, S. *Remote Network Monitoring Management Information Base Version 2 using SMIv2*. Request for Comments 2021, January 1997.

[12] HARDAKER, W. *The NETSNMP Project*. Sourceforge, Disponível em http://net-snmp.sourceforge.net/.

[13] PAXSON, Vern. *Bro: A System for Detecting Network Intruders in Real Time*. In: USENIX Security Simposium, 7. January 1998, San Antonio. 1998.

[14] INTERNET SECURITY SYSTEMS, INC. *Real Secure Homepage*, Disponível em http://www.iss.net

[15] BELLOVIN, STEVEN M. *Security Problems in the TCP/IP Protocol Suite*, Computer Communications Review, Vol. 19, No. 2, pp. 32-48, April 1989.

[16] IRWIN, Vicki; NORTHCUTT, Stephen; RALPH, Bill. (Naval Surface Warfare Center). *Building a Network Monitoring and Analysis Capability–Step by Step*. 1998. Disponível em http://www.nswc.navy.mil/ISSEC/CID/step.htm.

[17] LEVI, D.; SCHÖNWÄLDER, J. *Definitions of Managed Objects for the Delegation of Management Scripts*. Request for Comments 3165, August 2001.

[18] MENEGHETTI, E.,GASPARY, L. P.,TAROUCO, L. M. R. *Uma Ferramenta de Monitoração Programável Voltada à Detecção de Intrusão*. In Proc. of the XXVII Conferencia Latinoamericana de Informática (CLEI'2001), Merida, Venezuela, September 24 - September 28, 2001.

[19] BRAGA, L., TAROUCO,L. *Monitoracao de Protocolos de Alto Nivel Atraves da Implementacao de um Agente RMON2*. Dissertação de mestrado. PPGC da UFRGS. Dezembro de 2001.