Leonardo Lemes Fagundes

Uma Abordagem para Detecção de Ataques Distribuídos e de Múltiplas Etapas baseada na Composição de Serviços Web voltados à Segurança

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

Orientador: Prof. Dr. Luciano Paschoal Gaspary

São Leopoldo

2006

Ficha catalográfica elaborada pela Biblioteca da
Universidade do Vale do Rio dos Sinos

F156a Fagundes, Leonardo Lemes
Uma abordagem para detecção de ataques distribuídos e de múltiplas estapas baseada na composição de serviços Web voltados à segiurança / por Leonardo Lemes Fagundes. – 2006.
77 f. : il. ; 30cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, 2006.
"Orientação: Prof. Dr. Luciano Paschoal Gaspary, Ciências Exatas e Tecnológicas".

1. Seurança de rede - Computador. 2. Detecção de intrusão. 3. Web services. I. Título.

CDU 004.56

Catalogação na Publicação:
Bibliotecária Eliete Mari Doncato Brasil - CRB 10/1184

Dedico esta dissertação, que é fruto de muito esforço e dedicação, a minha família, em especial aos meus queridos avós, Pedro e Sunilda, a minha grande mãe, Tania e a minha amada noiva, Karine.

Agradecimentos

Finalmente é chegado o momento de registrar o meu profundo e sincero reconhecimento a todos aqueles que contribuiram para a realização de mais essa etapa. Começo agradecendo a minha noiva, não só pelo carinho e dedicação que já seriam o suficiente, mas pela compreensão e apoio em todos os momentos. A minha querida mãe pela torcida e confiança inabalável, muito obrigado. Aos meu avós, pela fé e pelas orações, as gurias, “mana”, “prima” e “Pilo” e aos meus tios, pelo carinho, obrigado de coração.

Meu muito obrigado ao amigo Vinícius Costa de Souza pelo incentivo, pela ajuda e pela sua amizade, aos colegas de mestrado Gilberto Iraja Muller, Glauco Antonio Ludwig e Marcelo Scopel pelas inúmeras horas de devaneios que nos ajudaram a acalmar os ânimos e persistir na luta.

Agradeço, ainda, ao Professor Dr. Luciano Paschoal Gaspary pela excelente orientação, pelas diversas revisões e correções dessa dissertação e pelo intenso acompanhamento.

Por fim, mas não menos importante, agradeço aos colegas do Instituto de Informática, principalmente, pela colaboração, sem a qual essa caminhada teria sido ainda mais difícil.

Senhor Deus obrigado.

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

AFRL	Air Force Research Laboratory
CAML	Correlated Attack Modeling
CIDS	Collaborative Intrusion Detection System
CT	Connection Tracker
ED	Elementary Detector
G-MADL	Graphical Multistage Attack Description Language
HIDS	Host Intrusion Detection System
HTTP	Hiper-Text Transport Protocol
IDMEF	Intrusion Detection Message Exchange Format
IDS	Intrusion Detection System
IHFB	Incident Handling Fact Base
IP	Internet Protocol
LAMBDA	Language to Model a Database for Detection of Attacks
LLDOS	Lincoln Laboratory Distributed Denial of Service
MADL	Multistage Attack Description Language
MQ	Message Queue
OASIS	Organization for the Advancement of Structured Information Standards
SOA	Services Oriented Architecture
SOAP	Simple Object Access Protocol
SSL	Secure Socket Layer
STATL	State Transition Analysis Technique Language
T-MADL	Textual Multistage Attack Description Language
TCP	Transport Control Protocol
TFN	Tribe Flood Network
TFTP	Trivial File Transfer Protocol
UDDI	Universal Description, Discovery and Integration
URI	Unique Resource Identifier
WS	Web Service

WSDL	Web Service Definition Language
WSN	Web Services Notification
XML	Extensible Markup Language

Sumário

Resumo

Com o uso em larga escala da Internet e a proliferação de ferramentas para realização de ataques, as instituições têm se tornado alvo de uma variedade de atividades intrusivas que vão desde simples varreduras de portas até ataques mais complexos, tais como negação de serviço distribuídos e *worms*. Com o objetivo de desenvolver soluções capazes de minimizar as chances de um intruso obter sucesso em suas atividades, diversos projetos de pesquisa têm sido realizados, sobretudo na área de detecção de intrusão. A grande parte dessas soluções apresentam as seguintes limitações: (a) ausência de uma forma adequada para representação e descrição de cenários de ataques de múltiplas etapas, que permita modelar o fluxo em que as atividades que os compõem devem ser observadas e (b) correlacionam alertas gerados por um conjunto reduzido de sensores, enquanto o ideal é observar as evidências registradas no maior número possível de serviços. No intuito de suprir essas limitações, esta dissertação propõe uma linguagem e uma arquitetura para detecção de ataques distribuídos e de múltiplas etapas. A linguagem proposta, denominada *Multistage Attack Description Language*, possui uma notação gráfica para representação visual e em alto nível do ataque e uma notação textual, baseada em XML, que permite a especificação detalhada do mesmo. A arquitetura, por sua vez, oferece um mecanismo uniforme para comunicação com diferentes serviços de segurança, que possibilita assinar pelos eventos que constituem os cenários especificados, detectar a sua ocorrência e acompanhar a evolução desses cenários. A arquitetura prevê, ainda, a execução de serviços de contenção que, uma vez invocados, realizam procedimentos para impedir a realização das demais etapas de um determinado cenário de ataque. O processo de comunicação entre os componentes desta arquitetura é realizado em conformidade com o padrão *Web Services Notification*.

Palavras-chave: Segurança de Redes de Computadores, Detecção de Intrusão, *Web Services*.

Abstract

With the wide use of the Internet and the proliferation of technologies to reproduce attacks, institutions have become target of a variety of intrusion activities, ranging from simple port scans to complex attacks, such as distributed denial of services and worms. Aiming to develop solutions to minimize the intruder's chances to succeed in his/her activities, several research projects have been carried out in the recent years, especially in the area of intrusion detection. Most of the solutions proposed present limitations since they: (a) do not provide an appropriate notation to represent and describe multistage attacks (that allows one to model the flow in which activities are expected be observed); and (b) correlate alerts produced by a reduced group of sensors, while the ideal is to observe evidences generated by the maximum number of available services. To fulfill this gap, this work proposes a language and an architecture to detect distributed multistage attacks. The proposed language, named Multistage Attack Description Language, provides a graphic notation to represent attacks in a high level manner, as well as a textual notation, based on XML, that allows for their detailed specification. The architecture offers a uniform mechanism to communicate with different security services, enabling the subscription for events that compose the intrusion scenario, the detection of their occurrence and the tracking of the intrusion scenario evolution as a whole. The architecture also allows the execution of contention services that, once invoked, execute procedures to prevent further phases of the attack to happen. The communication among the components of this architecture is performed in accordance to the Web Services Notification standard.

Key-words: Computer Network Security, Intrusion Detection, *Web Services*.

1 Introdução

1.1 Contextualização

Conforme [Durst et al., 1999; Campello, 2001], a sofisticação dos ataques é diretamente proporcional ao nível de automação das ferramentas disponíveis para a realização dos mesmos. A disponibilidade dessas ferramentas aliada ao uso não controlado da Internet faz com que as instituições estejam cada vez mais suscetíveis a inúmeras tentativas de invasões.

Através da exploração dos diferentes tipos de vulnerabilidades, tais como falhas de configuração, falhas de implementação e uso indevido de recursos disponíveis, surge um universo de ataques possíveis, que compreende desde simples varreduras de portas até ataques mais complexos, tais como negação de serviço distribuídos e *worms*. Esses ataques são compostos por uma série de atividades como identificação de portas abertas, exploração de serviços vulneráveis, criação de usuários, instalação de pequenos programas e, finalmente, realização de uma última ação, essa perceptível aos usuários devido aos danos causados [Northcutt, 2000].

No intuito de desenvolver soluções capazes de minimizar as chances de um intruso obter sucesso em suas atividades, diversos projetos de pesquisa têm sido realizados, sobretudo na área de detecção de intrusão. Entretanto, algumas limitações ainda permanecem, entre elas: (i) geração de altas taxas de falsos positivos, (ii) baixa escalabilidade, (iii) incapacidade de identificar novos ataques com precisão, (iv) inadequação para responder a tentativas de intrusões e (v) produção de um grande volume de alertas, o que exige uma análise criteriosa por parte dos profissionais de segurança [Fagundes, 2004].

1.2 Definição do problema

A grande parte das soluções de detecção de intrusão desenvolvidas até o momento é capaz, apenas, de detectar atividades muito pontuais não identificando o vínculo existente entre essas atividades. Sendo assim, resta aos administradores auditar o ambiente atingido em busca de evidências desses ataques, a fim de entender a estratégia existente por trás dos mesmos e, então, identificar e tomar medidas preventivas no intuito de evitar novos ataques dessa mesma natureza.

As estratégias ou planos de intrusões são compostos a partir do desenvolvimento de uma série de etapas que podem ocorrer de forma seqüencial ou concorrente no tempo. Essas etapas, quando devidamente identificadas, podem conduzir não só à detecção das atividades de intrusão, ainda nas etapas iniciais, mas também à interrupção dessas ações. A realização de um ataque, normalmente, obedece à seguinte metodologia: coleta de informações do alvo (*footprinting*), sondagem de portas, enumeração de usuários (acesso a contas de usuários), instalação de ferramentas e exploração das vulnerabilidades [Northcutt, 2000]. Portanto, o grande desafio é identificar o mais rapidamente possível a realização dessas etapas e prevenir que o intruso consiga realizar os passos subseqüentes.

Em geral os trabalhos de pesquisa, cujo objetivo é projetar e desenvolver arquiteturas ou *frameworks* para detecção de intrusão, contemplam apenas mecanismos para agregação de alertas [Cuppens, 2002; Debar, 2001]. Embora esse mecanismo permita reunir alertas gerados em diferentes pontos da infra-estrutura de uma rede, a detecção de um ataque depende, ainda, da identificação de toda uma sequência de atividades, o que caracteriza um ciclo longo e, portanto, pouco apropriado para detectar ataques em fase inicial.

Um segundo aspecto pouco explorado nas arquiteturas propostas refere-se à forma de representar os cenários de intrusões. Embora algumas dessas soluções apresentem notações, através das quais é possível definir atividades que compõe um ataque, elas deixam a desejar no que diz respeito à representação e especificação da relação entre essas atividades, o que é importante para detecção de um conjunto de atividades intrusivas.

Outra limitação das soluções propostas até o momento diz respeito à ausência de mecanismos para tomada de contramedidas. Sem a presença de tal mecanismo é possível somente reagir a uma intrusão quando a mesma já alcançou parte ou a totalidade de seus objetivos. Além disso, as ações de contramedidas utilizadas atualmente limitam-se, na maioria das vezes, à reconfiguração de *firewalls,* o que, por sua vez, pode não ser suficiente para bloquear outras tentativas de realizar esse mesmo ataque ou, ainda, causar transtornos à realização das atividades de usuários legítimos. Para que as ações de contramedidas de fato representem um recurso eficiente no contexto da detecção de intrusão é fundamental que sejam realizadas de forma dinâmica, ou seja, evoluam conforme as ações executadas pelo atacante e dispensem a necessidade de intervenção direta do gerente da rede.

Por fim, identifica-se algumas limitações comuns a grande parte das soluções propostas até o momento: a falta de uma maneira unifome e bem definida para relacionar alertas entre domínios administrativos diferentes e o fato de que os mecanismos de cooperação disponíveis têm como fontes de informações exclusivamente os sistemas de

detecção de intrusão, ignorando, assim, os alertas gerados por outras fontes (exemplo: *firewalls*, *syslog* e antivírus).

1.3 Objetivos

Este trabalho tem por objetivo principal propor uma linguagem e uma arquitetura para detecção de ataques distribuídos e de múltiplas etapas.

A linguagem proposta oferece uma forma flexível e adequada para representar a grande parte das especificidades inerentes aos cenários de ataque de múltiplas etapas. Através dessa linguagem é possível representar os diferentes fluxos das atividades que compõem esse tipo de ataque.

A arquitetura de detecção, por sua vez, foi projetada para suportar a composição de serviços *web* voltados à segurança. Para tal, a arquitetura se baseia na utilização de serviços, que provêem um mecanismo de comunicação uniforme com diferentes serviços de segurança que prima por satisfazer requisitos como interoperabilidade e extensibilidade. Outra característica relevante dessa arquitetura diz respeito à monitoração da evolução dos cenários de ataque, o que permite invocar medidas de contenção na tentativa de evitar a conclusão do mesmo.

1.4 Organização da dissertação

Esta dissertação está organizada da seguinte forma: o Capítulo 2 aborda conceitos referentes à anatomia de ataques e serviços *web*. O Capítulo 3 apresenta uma síntese de trabalhos relacionados ao tema dessa dissertação. Já o Capítulo 4 introduz a linguagem e a arquitetura proposta para detecção de cenários de ataques baseados em múltiplas etapas. A descrição dos procedimentos executados e do resultado obtido na avaliação experimental são tratados no Capítulo 5. O Capítulos 6 encerra esta dissertação com as considerações finais e perspectivas de trabalhos futuros.

2 Fundamentos de Segurança e Serviços *Web*

Este capítulo apresenta conceitos fundamentais relacionados à temática desta dissertação. Em relação aos fundamentos de segurança, enfatiza-se na Seção 2.1 a caracterização de ataques de múltiplas etapas. A composição de serviços na Internet e a notificação de eventos por meio de serviços *web* também são assuntos relevantes para o trabalho desenvolvido, sendo tratados nas Seções 2.2 e 2.3, respectivamente.

2.1 Ataques de múltiplas etapas

Ataques de múltiplas etapas são aqueles compostos por uma seqüência de atividades que ocorrem em diversas etapas, realizados de forma distribuída, ao longo de um intervalo de tempo. Essas etapas ou ataques intermediários possuem objetivos específicos e, em alguns casos, podem ser realizadas de diferentes maneiras e em uma ordem temporal qualquer [Cheung et al., 2003]. A natureza distribuída e, muitas vezes, concorrente desses ataques, aliada à possibilidade de execução de uma mesma etapa de diferentes maneiras, fazem com que a sua detecção represente uma tarefa bastante complexa.

Um exemplo simplificado dessa natureza de ataque pode ser ilustrado por um cenário em que um atacante deseja expor informações confidenciais que podem ser acessadas somente por usuários autorizados, a partir de um servidor *web*. Inicialmente esse atacante poderia identificar uma vulnerabilidade, por exemplo um *buffer overflow,* na implementação do módulo SSL (*Secure Socket Layer*) desse servidor. A partir da exploração dessa vulnerabilidade, o atacante obteria privilégios que lhe permitiriam executar comandos remotamente. Então, acessaria o sistema de arquivos, localizaria os dados que desejasse obter e alteraria a página *web* inicial desse servidor, de maneira que, através de uma requisição HTTP realizada a partir de qualquer estação, seria possível acessar esses dados.

As etapas intermediários que compõem o cenário supracitado poderiam ser identificadas da seguinte forma: (i) um sistema de detecção de intrusão de rede baseado em assinaturas detecta um *portscan* na rede, (ii) esse mesmo mecanismo alerta sobre a execução de um *exploit* e (iii) em seguida, um sistema de detecção de intrusão de *host* registra a alteração do arquivo *index.php*. A identificação desses eventos de forma isolada, contudo, não fornece ao gerente da rede uma noção real e abrangente do que de fato está ocorrendo.

Uma solução para prevenir ataques de múltiplas etapas tem que ser capaz de monitorar as diversas fases que compõem esses ataques e, ainda, executar contramedidas que sejam suficientes para inibir a evolução dos mesmos. Além disso, é importante mencionar que um cenário de intrusão composto de múltiplas etapas pode possuir diversas variações. Por exemplo, considerando o cenário anterior, ao invés de explorar uma vulnerabilidade no protocolo SSL poderia ter sido explorado um conjunto de outras vulnerabilidades existentes em protocolos ou serviços disponíveis nesse servidor.

Essas características dificultam a detecção e a prevenção de ataques constituídos por múltiplas etapas. Essa dificuldade é ainda maior porque, geralmente, o processo de detecção ocorre, somente, a partir da análise individual dos alertas gerados. Logo, é pouco provável que a detecção desse tipo ataque ocorra a tempo de evitar que o mesmo alcance seus objetivos.

Ataques de negação de serviços distribuídos, como *Tribe Flood Network* (TFN) e *Mstream*, também são exemplos de intrusões baseadas em múltiplas etapas. Esses ataques são compostos pela seguintes etapas: instalação de um programa *mestre* (geralmente em uma estação na qual o atacante possui maior controle), que irá coordenar o ataque e a instalação de um programa *zumbi* (em estações localizadas na *web* ou em uma rede corporativa). Esse, por sua vez, irá realizar múltiplas conexões simultaneamente à máquina alvo.

A partir da estação mestre o atacante envia um comando para as estações zumbis; essas, por sua vez, iniciam o ataque. Caso esse ataque esteja partindo de dentro da organização, a instalação de um determinado programa, não autorizado, em diversas estações pode ser um indicador de que essa rede será utilizada para a realização de um ataque de negação de serviços. A detecção dessa forma de ataque pode ocorrer facilmente através de sistemas de detecção de intrusão. Contudo, quando isso ocorrer possivelmente os objetivos do atacante já terão sido atingidos.

Além dos ataques de negação de serviços, os *worms* também são exemplos atuais de ataques de múltiplas etapas. O W32/Nimda, divulgado em 18 de setembro de 2001, constitui um dos mais famosos exemplos desse tipo de ataque. O *worm* explora uma série de vulnerabilidades existentes em estações e servidores Windows, tendo como principais objetivos: (i) permitir que intrusos executem comandos arbitrários nas estações comprometidas, (ii) possibilitar ao *worm* executar com os mesmos privilégios do usuário que foi, inicialmente, infectado e (iii) comprometer o alvo de forma que o mesmo fique vulnerável a outros ataques via Internet. O Nimda se propaga das seguintes maneiras [Cert, 2001]:

- ao explorar com sucesso algumas vulnerabilidades existentes em servidores IIS, o *worm* realiza o *download*, via tftp, de um arquivo chamado admin.dll que nada mais é do que uma réplica desse *worm* armazenada na estação da qual parte o ataque;
- o *worm* procura pela lista de contato dos clientes de e-mail e envia para todos os registros dessa lista um arquivo anexado (readme.exe) que, a exemplo do arquivo admin.dll, é o próprio *worm*;
- todas as páginas *web* encontradas pelo *worm* têm seu conteúdo modificado, de forma que o usuário, ao navegar nas páginas do servidor *web* contaminado, estará realizando o *download* para o sistema de arquivos local de um arquivo binário (.wav) que é a cópia do *worm*. Esse arquivo é gravado com o nome de readme.eml e, a partir desse momento, se inicia novamente o ciclo de propagação.

Assumindo como exemplo apenas a primeira forma mencionada de propagação, se observam as seguintes ações: sondagem de portas, a fim de identificar as vulnerabilidades do servidor *web*; em seguida, tentativas de execução de um *exploit*; uma vez que a execução do mesmo tenha obtido sucesso, inicia a etapa de conexão via tftp para uma determinada estação e, como estágio final, é realizado o *download* do arquivo admin.dll.

```
[**] IDS297 – WEB MISC – http-directory-traversal 1 [**]
09/19-10:29:47.817594 61.74.162.90:2421 -> 200.144.121.33:80 TCP TTL:106
TOS:0x0 ID:39592 IpLen:20 DgmLen:137 DF ***AP*** Seq: 0x87DB8B55 Ack:
0xFDF3DD74 Win: 0x2238 TcpLen: 20
```

Figura 2.1 – Alerta gerado pelo Snort

Um sistema de detecção de intrusão, por exemplo, o Snort poderia detectar a sondagem de portas e a execução do *exploit*, mas não seria capaz de identificar a estratégia do ataque. Além disso, o alerta resultante (Figura 2.1) é bastante genérico e poderia ser gerado por diversos ataques de inserção [Ptacek, 1998], por exemplo, por um ataque conhecido como *Reverse Traversal*. Nessa situação, o alerta gerado é pouco conclusivo e exige a busca por outras evidências no intuito de identificar corretamente o que está ocorrendo.

Conforme se pôde observar ao longo desta seção, os ataques baseados em múltiplas etapas podem assumir diversas variações na forma de execução. Assim, tão importante quanto detectar todas as etapas de um ataque é compreender que esses ataques fazem parte de uma estratégia, de um cenário de intrusão. Esse cenário deve ser devidamente modelado considerando todas as variações possíveis.

2.2 Serviços *web*

Esta seção aborda o conceito de serviços *web* e uma visão geral da sua arquitetura. Em seguida, é apresentado um modelo de comunicação (baseado na notificação de eventos) entre serviços *web*.

A Internet é composta por aplicações e plataformas heterogêneas. Entretanto, essas aplicações precisam ser capazes de realizar interações dinamicamente entre si e com outras aplicações [Walsh, 2002]. Isso é possível a partir do uso de serviços *web* [Cerami, 2002].

Um serviço *web* é um sistema identificado por um *Unique Resource Identifier* (URI), cujas interfaces públicas e pontos de ligação são definidos e descritos utilizando XML (*eXtensible Markup Language*). Sua definição pode ser descoberta por uma aplicação cliente, independente da linguagem na qual a mesma foi desenvolvida. Essas aplicações devem interagir com um serviço *web* de forma prescrita em sua definição, utilizando mensagens XML padronizadas e transmitidas através de protocolos da Internet [Booth et al., 2003]. Alguns dos principais objetivos dos serviços *web* são [Fuller et al., 2003]:

- solucionar as questões referentes à interoperabilidade através do uso de uma plataforma neutra;
- criar uma camada de abstração entre nodos heterogêneos em uma rede;
- ser acessível por todas as plataformas;
- prover solução de sobrepasso em relação aos *firewalls* para que, dessa maneira, o acesso a serviços seja possível a partir de diferentes domínios;
- desenvolver aplicações como um conjunto de serviços distribuídos e não somente como um componente que é executado dentro de um único processo.

A fim de viabilizar tais objetivos a arquitetura dos serviços *web* é composta de diversos padrões, como XML (*eXtensible Markup Language*), WSDL (*Web Service Description Language*) [Booth et al., 2005], UDDI (*Universal Description , Discovery and Integration*) [Clement et al., 2004] e SOAP (*Simple Object Access Protocol*) [Gudgin et al., 2003]. Os serviços *web* são baseados na arquitetura orientada a serviços (*services-oriented architecture – SOA*).

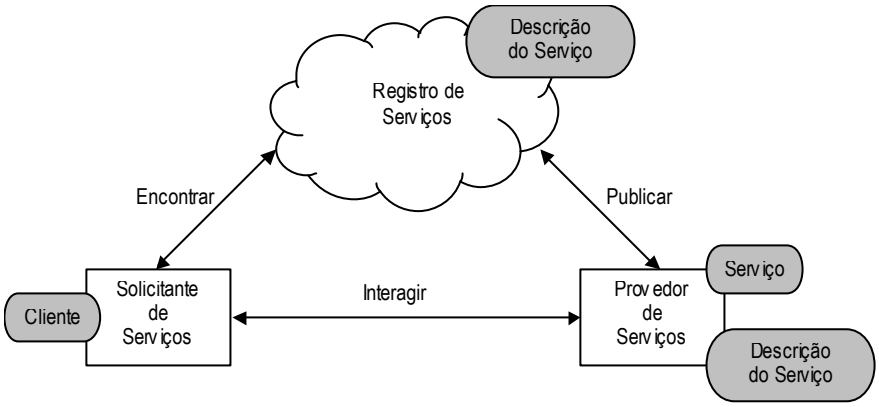

Figura 2.2 – Arquitetura dos serviços *web* [Ferris, 2003]

Conforme ilustrado na Figura 2.2, a arquitetura dos serviços *web* é composta pelos seguintes componentes: provedor de serviços, solicitante de serviços e registro de serviços. O provedor de serviços é responsável por disponibilizar serviços, armazenar as descrições dos mesmos em conformidade com o padrão WSDL e publicar essas descrições em um registro de serviços. O cliente ou solicitante de serviços é uma aplicação ou mesmo outro serviço *web* que localiza, através da descrição, o serviço desejado no registro de serviços. Nesta fase, o cliente recebe informações sobre como acessar o provedor e como a requisição ao serviço deve ser realizada. Através dessas informações, os serviços podem ser acessados diretamente. O registro de serviços é o componente com o qual tanto o provedor, quanto o solicitante de serviços interagem, pois os provedores publicam seus serviços e os clientes buscam por esses serviços [Ferris, 2003].

De forma resumida, serviços *web* são interfaces que descrevem um conjunto de operações acessíveis através da troca de mensagens em formato XML padronizadas. Esses serviços são descritos através de uma linguagem de descrição de serviços, publicados em um registro e descobertos por meio de um mecanismo padrão [Chung, 2003].

2.3 Notificação de eventos

O modelo de notificação de eventos é uma forma de comunicação na qual um ou mais provedores de serviços enviam mensagens, para *n* nodos, de acordo com os registros de interesse vinculados a cada nodo. O *Web Services Notification* [Graham et al., 2004] é uma especificação da OASIS (*Organization for the Advancement of Structured Information Standards*), cuja função é padronizar uma abordagem para que serviços *web* implementem a notificação de eventos utilizando o padrão *publish/subscribe* baseado em tópicos. Essa

especificação padroniza a terminologia e o formato das mensagens utilizadas e, ainda, fornece uma linguagem para descrição de tópicos (também referenciados como eventos).

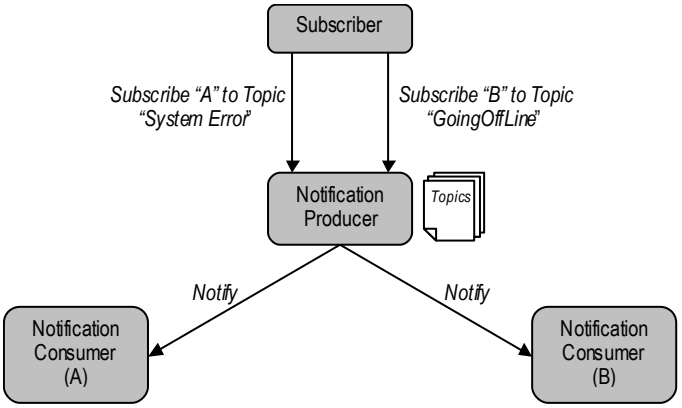

Figura 2.3 – Processo de subscrição e notificação de eventos

A Figura 2.3 ilustra um exemplo de subscrição e notificação de eventos. O serviço *web* denominado *NotificationProducer* suporta dois tópicos: *goingOffLine* e *SystemError*. Esse serviço pode disponibilizar um método ou função que permite ao requisitante, identificado na figura como *Subscriber*, obter a lista de tópicos suportados. O serviço requisitante, então, registra os eventos de interesse e quais nodos deverão ser notificados (*NotificationConsumer*) sobre a ocorrência dos respectivos eventos. Para esse exemplo a Figura 2.4 apresenta a seqüência de operações realizadas [Graham et al., 2004].

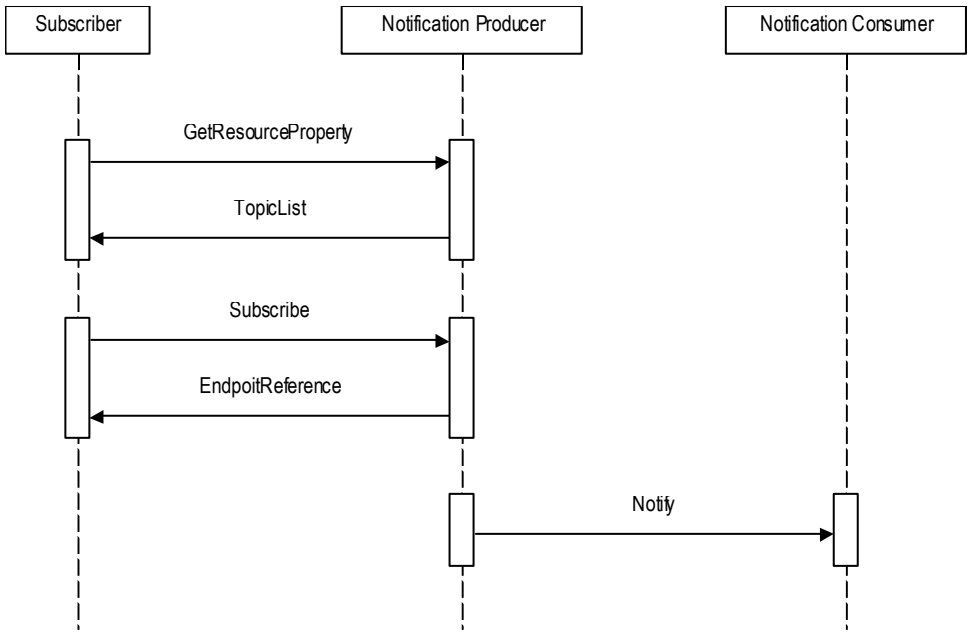

Figura 2.4 – Diagrama de seqüência do processo de subscrição e notificação de eventos

No diagrama de seqüência do processo de subscrição e notificação de eventos, observa-se que *Subscriber* envia uma mensagem (GetResourceProperty) ao *NotificationProducer* solicitando a lista de tópicos (TopicList) suportados pelo mesmo. Essa verificação pode ser omitida, ou seja, a aplicação ou serviço requisitante pode enviar uma solicitação sem que antes tenha sido realizada tal consulta. Uma vez de posse da lista de tópicos, o *Subcriber* envia uma mensagem de requisição (Subscribe) ao *NotificationProducer*, indicando o endereço do *NotificationConsumer* e o(s) tópico(s) de interesse. Em resposta a essa solicitação o *NotificationProducer* envia uma mensagem de confirmação (*EndpointReference*) ao remetente. Quando o *NotificationProducer* constata a ocorrência de um determinado evento de interesse que esteja relacionado a alguma solicitação já recebida, envia uma mensagem de notificação (*notify*) ao(s) *NotificationConsumer(s)*.

Outra forma de realizar notificações de eventos é incluindo um componente intermediário, ou *broker*, conforme apresentado no exemplo da Figura 2.5. Nesse exemplo, o requisitante (Subscriber) realiza uma solicitação ao *NotificationBroker*, um serviço *web* que implementa interface para comunicação tanto com o *NotificationProducer* quanto com o *Notification Consumer*. A função desse serviço é receber as notificações que partem das entidades *publisherA* e *publisherB*, responsáveis pelo monitoramento de alguns tópicos, e, em seguida, distribuir as mensagens de notificação conforme as solicitações recebidas.

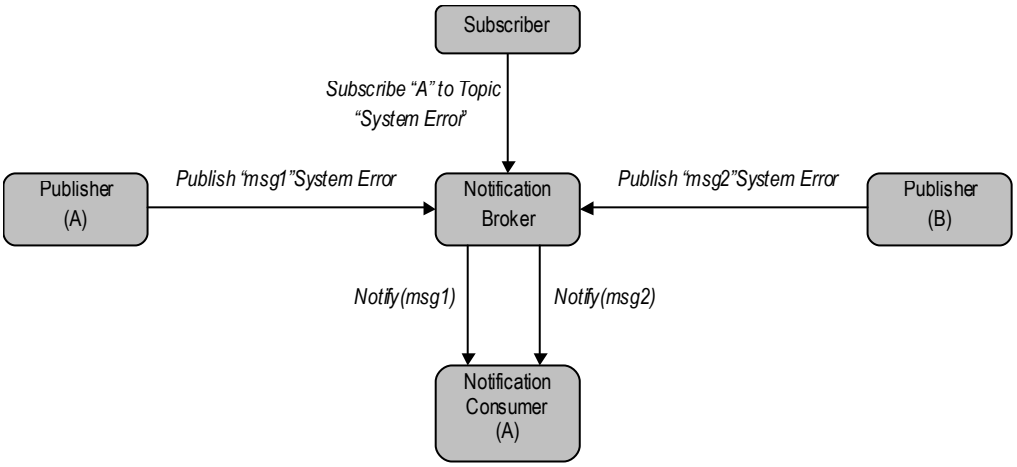

Figura 2.5 – Processo de subscrição e notificação de eventos com *broker*

A Figura 2.6 ilustra o diagrama de seqüência para esse modelo de notificação [Graham et al., 2004]. A seqüência de troca de mensagens entre o *Subscriber* e o *NotificationBroker* é análoga à seqüência de troca de mensagens entre o *Subscriber* e o *NotificationProducer* no modelo de notificação sem *broker*. Ao ser instanciado, o *Publisher* envia uma mensagem (*RegisterPublisher*) ao *Broker* comunicando os tópicos fornecidos. Em resposta, o *Broker*

envia outra mensagem (*EndpointReference*). A partir daí, está estabelecida a relação entre esses dois componentes. Para notificar o *NotificationBroker*, o *Publisher* envia uma mensagem de notificação (*notify*). O *NotificationBroker*, por sua vez, notifica o *NotificationConsumer*.

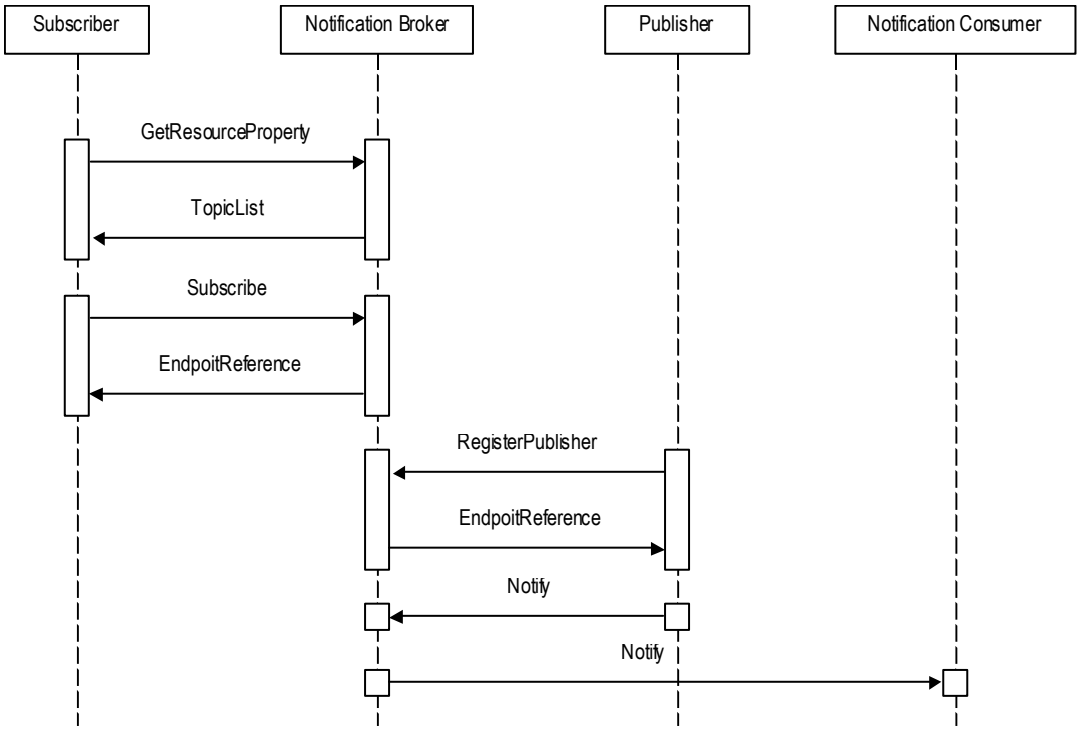

Figura 2.6 - Diagrama de seqüência de um processo de subscrição e notificação de eventos com *broker*

Através da utilização de serviços *web* para intermediar a relação entre as entidades que geram notificações e aquelas que as recebem é possível controlar o processo de notificação para que somente entidades (*Publishers*) previamente registradas possam enviar mensagens de notificação, evitando, assim, que componentes de publicação maliciosos possam interferir no funcionamento dos serviços responsáveis pelo recebimento das notificações. Além disso, o *broker* pode ser utilizado, também, para armazenar registros (*logs*) referentes a todas as transações ocorridas.

Publicação

Referente ao processo de notificação ou publicação de um tópico, o WS-Notification especifica como a disseminação de uma mensagem de notificação deve ocorrer, tanto para o modelo com entidades intermediárias quanto para o modelo de entrega direta.

No modelo em que uma mensagem de notificação é enviada diretamente ao *NotificationConsumer*, a entidade que observa a ocorrência de um evento, o

NotificationProducer, assume a responsabilidade de receber solicitações e enviar mensagens de notificação para as partes interessadas. Para os casos em que há uma entidade intermediária envolvida no processo de notificação, o *NotificationBroker* é responsável por disseminar as mensagens geradas pelas entidades *Publishers* para as partes interessadas (*NotificationConsumers*). Para tal, existem pelo menos duas formas de relacionar o *Publisher* e o *NotificationBroker*: publicação simples e publicação baseada em demanda.

Em um cenário simples de publicação a entidade *Publisher* é responsável somente por monitorar a ocorrência de um determinado tópico e formatar a mensagem de notificação que descreve o evento. A disseminação dessa notificação ocorre quando o *Publisher* envia a mensagem do tipo *Notify* ao *NotificationBroker* e esse, por sua vez, atua conforme representado no diagrama de atividades da Figura 2.4. Para ocasiões nas quais, por alguma razão, seja interessante que as solicitações partam do *Broker* diretamente para o *Publisher,* o mais adequado é utilizar o padrão de publicação baseado em demanda. Nesse caso, o *Publisher* deve aceitar solicitações de notificação vindas diretamente do *Broker*.

Tópicos

Tópicos são utilizados em mensagens de notificação e requisições para identificar eventos. Um tópico pode possuir zero ou mais tópicos filhos, bem como cada filho pode ter sub-tópicos. O tópico para o qual não existe um tópico pai é denominado tópico raiz (*Root Topic*). A estrutura composta por um tópico raiz e todos os seus descendentes forma uma hierarquia referenciada como árvore de tópicos. Essa estrutura reduz o número de solicitações realizadas por um *Subscriber*, pois uma requisição pode ser composta de uma árvore de tópicos inteira ou simplesmente de parte dessa estrutura. A Figura 2.7 apresenta um exemplo em que se observa dois tópicos raízes (t1 e t4) e dos seus filhos (t2 e t3) e (t5 e t6).

```
1  ...
2      <wsnt:topic name="t1">
3          <wsnt:topic name="t2"/>
4          <wsnt:topic name="t3"/>
5      </wsnt:topic>
6      <wsnt:topic name="t4">
7          <wsnt:topic name="t5"/>
8          <wsnt:topic name="t6"/>
9      </wsnt:topic>
10 ...
```

Figura 2.7 – Estrutura de tópicos

O *NotificationProducer* suporta um conjunto de tópicos de diversos *TopicSpaces* e os utiliza para agrupar mensagens de notificação relacionadas a alguma situação. Esse componente pode receber como entrada todo um grupo hierárquico de tópicos ou somente um subconjunto dessa estrutura. A relação de tópicos que uma determinada entidade é capaz de monitorar pode ser obtida através de uma consulta ao elemento chamado *Topic Resource Property*.

2.4 Considerações Parciais

No Capítulo 2 foram abordados temas relevantes ao trabalho em questão tais como: o funcionamento e as características dos ataques de múltiplas etapas, os conceitos sobre a arquitetura dos servicos web (*web services*) e, finalmente, foi detalhado o modelo de notificação de eventos denominado *Web Services Notification* (WSN).

O conteúdo referente aos ataques de múltiplas etapas será de particular importância já a partir do próximo Capítulo, em que será discutido as linguagens e as arquiteturas relacionadas a esse tipo de ataque. A especificação da OASIS para notificação de eventos entre *web services* será, novamente discutida e aplicada à arquitetura proposta ao longo do Capítulo 4.

3 Trabalhos Relacionados

Este capítulo apresenta uma síntese dos principais trabalhos relacionados com esta dissertação. Uma vez que essa dissertação propõe uma linguagem para especificação de ataques e uma arquitetura para detecção de intrusão, o capítulo foi dividido em duas partes. Algumas das principais linguagens propostas recentemente para modelagem de cenários de ataques são descritas na Seção 3.1. Na Seção 3.2 são apresentadas arquiteturas para detecção de intrusão.

3.1 Linguagens para modelagem de cenários de ataques

Os ataques compostos por diversas etapas são difíceis de detectar, pois através das soluções empregadas atualmente o que se obtém são alertas isolados, que representam apenas as etapas individuais ou intermediárias que compõem um cenário de invasão de múltiplas etapas [Cuppens, 2002].

Para detectar intrusões compostas por diversas etapas é necessário observar, além de alertas individuais, algumas características e particularidades inerentes a essa forma de ataque. Para isso, é fundamental a utilização de uma linguagem que viabilize a representação das diferentes estratégias que um atacante pode adotar com a finalidade de comprometer um determinado alvo.

Atualmente, são várias as linguagens disponíveis para modelagem de ataques. A seguir são apresentadas três dessas linguagens: *State Transition Analysis Technique Language* (STATL), *Correlated Attack Modeling Language* (CAML) e *Language to Model a Database for Detection of Attacks* (LAMBDA). Embora existam outras linguagens com o mesmo propósito, tais como ADELE [Michel, 2001] e SHEDEL [Meier, 2002], acredita-se que as três linguagens escolhidas representam um sub-conjunto bastante representativo em virtude do tempo de vida desses projetos e dos resultados reportados até o momento.

3.1.1 State transition analysis technique language (STATL)

A linguagem STATL [Eckmann, 2002] começou a ser desenvolvida a partir de 1992 pelo *Reliable Software Group* do Departamento de Ciência da Computação da *University of California*. Essa é uma linguagem descritiva que foi projetada para permitir a modelagem de cenários de ataques através de *máquinas de estados*.

O conjunto de construtores disponível nessa linguagem possibilita representar um ataque como uma composição de *estados* e *transições*. Os *estados* são utilizados para caracterizar diferentes comportamentos de um sistema durante a ocorrência de um ataque. Para auxiliar na definição de um estado são utilizadas variáveis. Elas possibilitam armazenar informações tais como o proprietário de um arquivo, a data de sua última modificação ou mesmo o valor de um contador. As transições são ações ou eventos que, quando ocorrem, causam a mudança de um estado para outro.

Uma transição em STATL pode ser do tipo *consuming*, *nonconsuming* e *unwinding*. Transições *nonconsuming* são utilizadas para representar ações que não inviabilizam a ocorrência de novos ataques a partir do estado original, ou seja, as condições necessárias para que essa etapa do ataque seja realizada novamente são preservadas. Em transições desse tipo tanto o estado de origem quanto o de destino permanecem válidos. Por exemplo, se um ataque consiste na criação de um *link* para um *shell script* com SUID e da execução desse *script* através do vínculo criado, então essa última ação não invalida o estado anterior. Sendo assim, um outro ataque pode ser realizado a partir da execução desse *link*.

Ao contrário da transição *nonconsuming*, uma transição do tipo *consuming* torna o estado de origem inválido e, portanto, não é possível realizar um novo ataque a partir desse estado, pois não haverá as condições necessárias para tal. Já transições do tipo *unwinding* representam uma forma de *rollback* e são utilizadas para descrever eventos e condições que podem impedir a evolução de um ataque, fazendo com que a máquina de estados retorne ao estado anterior. Por exemplo, caso um atacante tente acessar o arquivo de senhas do sistema operacional GNU/Linux e não consiga obter um usuário e sua respectiva senha, isso faz com que esse ataque retorne ao estado anterior, aquele antes da tentativa de acesso ao arquivo */etc/passwd*.

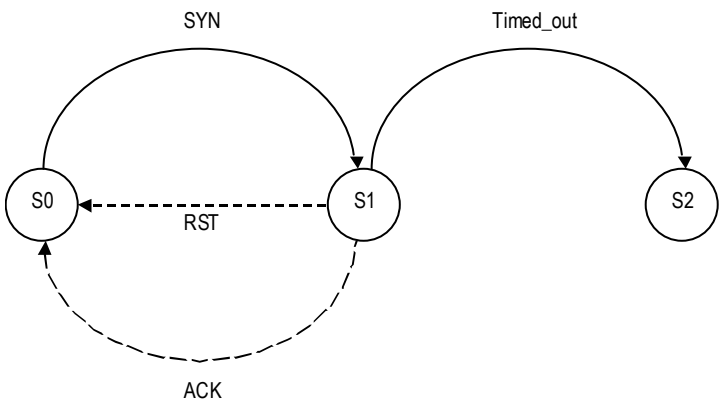

Figura 3.1 – Máquina de estados

As Figura 3.1 e 3.2 apresentam, respectivamente, a máquina de estados e a descrição do ataque denominado *halfopentcp*. Esse ataque consiste no envio de uma grande quantidade de solicitações de conexão a um servidor, impedindo o mesmo de atender solicitações realizadas por usuários legítimos.

```
use netstat;
   scenario halfopentcp(int timeout)
   {
        IPAddress victim_addr;
        Port victim_port;
        IPAddress attacker_addr;
        Port attacker_port;
        timer t0;

       initial state s0 {}
       transition SYN (s0 -> s1)
       nonconsuming
          {
             [IP ip [TCP tcp]] : (tcp.tcp_header.flags & TH_SYN) && !(tcp.tcp_header.flags & TH_ACK)
           {
             victim_addr=ip.header.dst;
             victim_port=tcp.header.dst;
             attacker_addr=ip.header.src;
             attacker_port=tcp.header.src;
           }
          }

       state s1
       {
             { timer_start(t0, timeout); }
       }
       transition ACK (s1 -> s0)
       unwinding
             {
             [IP ip [TCP tcp]]:(ip.header.dst==victim_addr) && (tcp.header.dst==victim_port) && (ip.header.src==attacker_addr) &&
              (tcp.header.src==attacker_port) && !(tcp.header.flags & TH_SYN) && (tcp.header.flags & TH_ACK)
             }
       transition RST (s1 -> s0)
       unwinding
             {
             [IP ip [TCP tcp]]:(ip.header.src==victim_addr) && (tcp.header.src==victim_port) && (ip.header.dst==attacker_addr) &&
               (tcp.header.dst==attacker_port) && (tcp.header.flags & TH_RST)
             }
       transition Timed_out (s1 -> s2)
       consuming
             {
              timer t0;
             }

```

```
45      state s2
46      {
47          {
48          HALFOPENTCP e;
49          e = new HALFOPENTCP(attacker_addr, attacker_port, victim_addr, victim_port, start);
50          enqueue_event(e, HALFOPENTCP, start);
51          }
52      }
53  }
```

Figura 3.2 – Codificação do cenário *halfopentcp* em STATL

A primeira linha dessa descrição informa que o cenário modelado será interpretado pelo módulo *netstat*, utilizado para descrever ataques a redes de computadores. Em seguida, é atribuído um nome ao cenário e algumas variáveis são declaradas conforme a sintaxe *tipo nome_da_variável*. O restante do código consiste na definição de *estados* e *transições*.

O estado *s0* representa o comportamento inicial do sistema (linha 10) e associado a esse estado há apenas a transição *SYN* (linha 11). Essa transição ocorre quando a condição da linha 14 é satisfeita. Durante a realização da transição são armazenados em variáveis locais os endereços IPs e as portas das estações envolvidas (linhas 16 a 19). A transição *SYN* ocorre toda vez que uma estação iniciar uma sessão TCP e conduz a máquina de estados para o estado *s1* (linhas 23 a 26). Nesse estado um temporizador é iniciado e o tempo máximo para a conexão é representado através do parâmetro *timeout*.

No estado *s1* existem dois eventos que definem que não há um ataque em andamento e, portanto, a máquina de estado deve retornar ao estado inicial *s0*. No primeiro evento (linhas 27 a 32) a estação cliente envia um pacote com a *flag* ACK habilitada indicando que a conexão foi estabelecida. No segundo evento (linhas 33 a 38) a porta está bloqueada e não há como estabelecer conexão. Esses dois eventos disparam as transições *ACK* e *RST*, respectivamente. Ambas são do tipo *unwinding* e, portanto, fazem com que o sistema retorne ao estado inicial. Associado ao estado *s1* há, ainda, a transição denominada *Timed_out* (linhas 39 a 43), do tipo *consuming*, que realiza a transição do estado *s1* para o estado *s2*. O estado *s2* (linhas 45 a 51) cria um evento do tipo *HALFOPENTCP* com as informações obtidas nos estados anteriores.

3.1.2 Correlated attack modeling language (CAML)

O *Correlated Attack Modeling* (CAM) é um projeto financiado pelo DARPA, através do *Air Force Research Laboratory* (AFRL) [Cheung et al., 2003]. Nesse projeto foram desenvolvidos métodos e uma linguagem para modelagem de cenários de ataques de múltiplas etapas. A CAML utiliza uma abordagem na qual cada módulo representa uma etapa (um

ataque) e esses módulos, quando relacionados uns com os outros, formam um cenário de intrusão. Essa linguagem possui uma biblioteca de predicados que atua como um vocabulário para descrever as propriedades dos eventos e dos estados de um sistema.

A modelagem de ataques através da CAML prevê a realização das seguintes tarefas: (a) identificar os ataques de forma individual, (b) caracterizar esses ataques do ponto de vista da detecção e (c) especificar a relação entre esses ataques.

Somente a identificação das etapas que compõem uma intrusão e a especificação dos eventos a serem monitorados não são suficientes para detectar que um determinado ataque está ocorrendo. Para tal, é preciso representar as relações existentes entre cada um dos ataques intermediários modelados. Essas relações podem ser expressas através do tempo (um ataque ocorre antes do outro) e do uso de variáveis (ex: o endereço IP das estações).

A Figura 3.3 ilustra a codificação de um módulo em CAML para ataques que exploram uma falha na implementação do protocolo *Secure Socket Layer (SSL) - SSL buffer overflow*. Essa especificação corresponde a uma única etapa do ataque. A codificação do módulo consiste (a) na identificação do mesmo, (b) na especificação de uma *activity* que, quando detectada por um sensor, inicia a ação do módulo e (c) pelo conjunto de condições necessárias (*pre-condition*) para que seja possível realizar a inferência (*post-condition*) sobre o que está ocorrendo.

Conforme pode ser observado no exemplo, o módulo é denominado *OpenSSL-Handshake-BO-2-Remote-Exec*. A primeira seção (linhas 2 a 11) descreve um evento através dos seguintes dados: endereço IP origem, endereço IP e porta de destino e uma descrição textual (CAN-2002-0656). A seção *pre-condition* (linhas 12 a 29) utiliza-se dos predicados *HasService*, *Depends, Subset* e da função *VersionCmp*. O predicado *HasService* informa que a estação alvo deve possuir um endereço IP igual ao da estação alvo da seção atividade (*address* == *t*) e fornecer um serviço associado à porta *tp*. Já o predicado *Depends* indica que nessa porta deve haver uma versão vulnerável desse serviço.

```
1   module OpenSSL-Handshake-BO-2-Remote-Exec (
2       activity:
3           r1: Event(
4               Source(
5                   Node(Address(s: address)))
6               Target(
7                   Node(Address(t: address))
8                   Service(tp: port))
9               Classification(
10                  origin == "cve"
11                  name == "CAN-2002-0656"))
```

```
12      pre:
13        p1: HasService(
14              Node(Address(address == t))
15              Service(
16                 imp: implement
17                 ver1: version
18                 port == tp))
19        p2: Depends(
20              Source(Service(
21                 implement == imp
22                 version == ver1
23                 port == tp))
24              Target(Service(
25                 implement == "OpenSSL"
26                 ver2: version)))
27                 VersionCmp(ver2, "0.9.6") < 0
28                 Subset(r1, p1)
29                 Subset(r1, p2)
30      post:
31              Event(
32              starttime == r1.starttime
33              endtime == DEFAULT_ENDTIME
34              Source(
35                 Node(Address(address == s)))
36              Target(
37                 Node(Address(address == t)))
38              Classification(
39                 origin == "vendor-specific"
40                 name == "CAM-Remote-Exec"))
41  )
```

Figura 3.3 – Codificação do ataque *OpenSSL-Handshake* em CAML

A função *VersionCmp* compara a versão atual do serviço que está disponível com a versão vulnerável ao ataque em questão. Por fim, o predicado *Subset* determina a relação temporal entre a atividade *r1* e os predicados *p1* e *p2*. Neste caso esta sendo representado que após a atividade *r1* pode ser disparado tanto o predicado *HasService* quanto *Depends*. Caso a atividade e a pré-condição sejam devidamente satisfeitas, a inferência desse módulo (definida entre as linhas 30 e 40) retorna um evento do tipo *CAM-Remote-Exec*. A saída de um módulo pode ser utilizado como entrada para outro módulo, por meio do predicado *data theft*.

3.1.3 Language to Model a database for Detection of Attacks (LAMBDA)

A linguagem LAMBDA [Cuppens, 2000] é uma linguagem para descrição de ataques baseada em XML. Nessa linguagem um ataque é especificado a partir de, no mínimo, três campos:

- *pre-condition*: descreve as condições necessárias para que um determinado ataque possa ser realizado;
- *post-condition*: corresponde à descrição dos efeitos causados por um determinado ataque quando realizado com êxito;
- *detection scenario*: representa uma combinação de eventos (ações realizada pelo intruso) que indica a ocorrência de um ataque.

As informações contidas nos campos mencionados acima são estruturadas a partir de predicados. Os predicados são utilizados para descrever características importantes de um ataque, tais como o nível de acesso (*access level*) que um intruso necessita para iniciar um ataque, os serviços que precisam estar ativos (*use service*) em uma determinada estação e os efeitos causados (*deny_of_service, alter* e *ilegal use*) por um ataque bem sucedido.

Os predicados dos campos *pre-condition* e *post-condition* podem ser combinados a partir dos conectores lógicos "," e "not" que significam conjunção e negação, respectivamente. As combinações entre eventos podem ser representadas através dos seguintes operadores: ";" (seqüencial), "|" (paralelo) e "?" (opcional).

A figura 3.4 ilustra o ataque TCPScan, especificado em LAMBDA. Nessa forma de especificação os termos que iniciam com letra maiúscula representam variáveis. A descrição inicia pela identificação do ataque (linhas 2 e 3) que é realizada através das *tags* <attack> e <name>. A *tag* <attack> possui um atributo denominado attackid, que corresponde a uma forma de classificação que segue o padrão "MIR-xxx", utilizado pelo projeto MIRADOR [Cuppens, 2000]. Essas informações são utilizadas, posteriormente, para correlacionar ataques.

Em seguida, entre as tags <pre> e </pre> (linhas 4 a 7), são utilizados os predicados use_soft, use_service e service_type para especificar: (a) a ferramenta que deverá estar instalada na estação a partir da qual o ataque será executado e (b) as características do serviço que necessita estar ativo na estação alvo para que o ataque TCPScan possa ocorrer.

O resultado da ação do atacante é formalmente descrito entre as tags <post> e </post> (linhas 8 a 10). No caso desse tipo de ataque, o resultado é expresso através do predicado knows(Source_user, use_service (Target_address, Target_service)), ou seja, como conseqüência do ataque, o invasor (Source_user) obtém as seguintes informações sobre o sistema alvo: endereço IP (Target_address) e serviços ativos (Target_service).

A *tag* <scenario>Action</scenario> (linha 11) define que o cenário de ataque é composto por um único evento "Action". Esse evento é detalhado através da *tag* <cond_scenario>...</cond_scenario> (linha 12) que, geralmente, tem como conteúdo o predicado script, cuja função é representar o

comando que o atacante deve realizar para executar o ataque. Já a *tag* <detection>...</detection> (linha 13) informa que o ataque TCPScan pode ser detectado a partir de um único evento "Alert". Os principais atributos que esse alerta deve apresentar constam entre as *tags* <cond_detection>...</cond_detection> (linhas 14 a 23).

```
1  <?xml version="1.0" encoding="UTF-8"?>
2     <attack attackid="MIR-0074">
3     <name>tcpscan</name>
4     <pre>use_soft(Source_address, tcpscan),
5          use_service(Target_address, Target_service),
6          service_type(Target_service,tcp)
7     </pre>
8    <post>
9          knows(Source_user, use_service (Target_address, Target_service))
10   <post>
11         <scenario>Action</scenario>
12         <cond_scenario>script(Action,'tcpscan $Target_address')</cond_scenario>
13         <detection>Alert</detection>
14         <cond_detection>alert(Alert),
15               source(Alert, Source),
16               source_node(Source, Source_node),
17               address(Source_node, Source_address),
18               source_user(Source, Source_user),
19               target(Alert, Target),
20               target_node(Target, Target_node),
21               target_service(Target, Target_service),
22               classification(Alert,"MIR-0074")
23    </cond_detection>
24    </attack>
```

Figura 3.4 – Ataque TCPScan especificado em LAMBDA

Uma vez que os ataques estejam devidamente representados conforme especificado pela LAMBDA, é possível definir as regras de correlação entre os alertas que podem compor um cenário de ataque de múltiplas etapas. Para tal, essa linguagem prevê o predicado alert_correlation que possui a seguinte sintaxe: alert_correlation (Alert1, Alert2), cujo significado representa que o Alert1 deve ocorrer antes do alerta Alert2.

3.1.4 Análise das linguagens

Embora não exista um consenso, projetos de pesquisa desenvolvidos durante os últimos cinco anos, tais como [Cheung et al., 2003] e [Eckmann, 2002], sugerem que (a) heterogeneidade, (b) extensibilidade e (c) expressividade sejam características presentes em linguagem para representação de cenários de ataques. Frente a esse contexto, optou-se por avaliar as linguagens CAML, STATL e LAMBDA tendo como base essas características.

Heterogeneidade

Esse requisito se refere à capacidade da linguagem de representar eventos oriundos de diversas fontes de informação (ex: pacotes IP e registros de auditoria de uma aplicação ou de um sistema operacional). Por exemplo, em STATL, através do comando use, se determina qual a origem dos eventos a serem tratados em um determinado cenário. As bibliotecas ustat, winstat e netstat são utilizadas para especificar cenários baseados em eventos gerados pelos sistemas operacionais Unix, Windows e pelo sistema de detecção de intrusão netstat, respectivamente.

Extensibilidade

Uma linguagem para modelagem de ataques deve possuir meios para incorporar novos recursos. Assumindo como exemplo a STATL, a inclusão de um novo tipo de transição é um recurso que, se implementado aumentaria a capacidade dessa linguagem.

Expressividade

Este requisito se refere à capacidade de representação da maior quantidade possível de tipos de ataques, bem como das suas variações. Segundo [Templeton, 2001], os métodos mais empregados para criar variações de ataques são:

- mutação: permite especificar variações para algumas informações que se deseja observar, tais como: endereço IP, número da porta e conta de usuário utilizados durante um ataque;
- ordenação aleatória de eventos: especifica que um determinado conjunto de eventos pode ocorrer em uma ordem temporal qualquer;
- substituição: define que ao executar o procedimento “X” ou “Y” os resultados obtidos são semelhantes, logo o atacante pode tanto realizar um quanto o outro que a conseqüência será a mesma;
- *looping*: o atacante pode executar de forma contínua um mesmo conjunto de procedimentos.

A tabela 3.1, apresenta uma síntese comparativa entre as linguagens discutidas na seção anterior.

Tabela 3.1 – Análise das linguagens

		CAML	STATL	LAMBDA
Heterogeneidade		IDMEF	Módulos STATL	IDMEF
Extensibilidade		Predicados	Limitada	Predicados
Expressividade	Mutação	Contempla	Contempla	Contempla
	Ordenação aleatória de eventos	Não contempla	Não contempla	Contempla Parcialmente
	Substituição	Não contempla	Não contempla	Não contempla
	Looping	Não contempla	Contempla	Contempla

Quanto à capacidade de especificar eventos gerados por diferentes fontes de informações (heterogeneidade) se observa que, entre as linguagens analisadas, a STATL é a mais restrita, uma vez que só podem ser representados eventos gerados pelos módulos acoplados ao *framework* STATL, não sendo possível utilizar eventos gerados por outras ferramentas de segurança. Já nos cenários de ataque especificados em LAMBDA e CAML é possível utilizar, apenas, eventos oriundos de sistemas de detecção de intrusão baseados em rede, cujos alertas sejam gerados em conformidade com o padrão IDMEF.

A extensibilidade é um requisito contemplado nas linguagens CAML e LAMBDA, através do desenvolvimento de predicados. Em STATL a inclusão de novas funções requer alterações diretamente no *framework* STAT.

No que se refere à expressividade das linguagens estudadas, mais especificamente à mutação, as linguagens STATL, CAML e LAMBDA contemplam plenamente este requisito. A mutação é especificada nessas linguagens através do uso de variáveis.

A ordenação aleatória de eventos é uma propriedade para a qual STATL e CAML não possuem construtores. Em LAMBDA, através do predicado Alert_Correlation é possível criar regras de correlacionamento. Entretanto, se for necessário especificar que três eventos diferentes podem ocorrer em uma seqüência qualquer, será preciso criar nove diferentes regras para contemplar todas as variações possíveis o que, em determinados casos, pode gerar uma quantidade muito grande de regras.

Ainda no que se refere à expressividade, as linguagens STATL, CAML e LAMBDA não são capazes de modelar cenários em que existam ações equivalentes para se alcançar um mesmo objetivo (Substituição). Por fim observa-se que apenas STATL permite especificar a ocorrência repetida de determinados eventos (looping) dentro de um mesmo cenário de ataque.

3.2 Arquiteturas para detecção de cenários de intrusão

O grande volume de alertas que é produzido através dos sensores dos sistemas de detecção de intrusão, dos *firewalls* e dos arquivos de *log* das aplicações e dos serviços que se encontram em uma rede, juntamente com a ausência de mecanismos para correlação dessas informações, faz com que analisar esses alarmes seja uma tarefa que demande muito tempo e esteja suscetível a diversos equívocos.

Quando se trata do processo de detecção de ataques que ocorrem em diversas etapas, distribuídas ao longo de um intervalo de tempo, o problema é ainda mais complexo, uma vez que é necessário não só detectar os ataques com precisão, mas também identificar a estratégia do atacante em relação aos seus alvos.

Na tentativa de diminuir o impacto dessas limitações e, por conseqüência, aumentar o nível de proteção dos sistemas de informação, diversos projetos de pesquisa estão em desenvolvimento. Entre eles destacam-se os seguintes: [Debar, 2001], [Cuppens, 2002], [Porras et al., 2002] e [Wu et al., 2003]. Após analisar estes estudos observou-se que os dois últimos trabalhos, denominados respectivamente de M-Correlator e CIDS, são alguns dos trabalhos maduros e reconhecidos na área de detecção de intrusão.

3.2.1 M-Correlator

M-Correlator [Porras et al., 2002] é uma aplicação que se propõe a reduzir o tempo e o esforço empregado para o gerenciamento de alertas, gerados a partir de múltiplos mecanismos de segurança. A estratégia empregada para atingir esse objetivo é priorizar e agregar alertas em função de atributos e condições pré-estabelecidas. Essa aplicação foi projetada para consolidar e classificar uma seqüência de incidentes de segurança, uma vez informados a topologia e os objetivos operacionais (missão) da rede protegida. Na Figura 3.5 são apresentados os elementos conceituais dessa ferramenta.

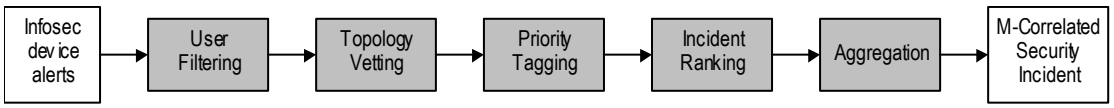

Figura 3.5 – Componentes do M-Correlator

Como pode ser observado, os dispositivos fornecem uma série de alertas (*infosec device alerts*), que são inicialmente processados por filtros controlados dinamicamente (*User Filtering*) a fim de eliminar alertas de baixo interesse. Em seguida, esses alertas são novamente avaliados agora em relação às informações sobre a topologia da rede (*Topology Vetting*). Dessa maneira, é possível verificar a relevância desse alerta frente à estrutura de

rede existente. Posteriormente é calculado o grau de prioridade por alerta (*Priority Tagging*), a fim de (i) indicar o quão crítico é esse ataque e (ii) verificar o nível de interesse que foi registrado pelos usuários do sistema para essa classe de ataque. Finalmente, o sistema gera um parecer final sobre cada incidente de segurança (*Incident Ranking*) e a probabilidade de que as atividades reportadas nesses alertas tenham realmente ocorrido.

Através de um algoritmo de agregação de alertas (*alert clustering algorithm*), o *M-Correlator* permite definir regras de consolidação e correlacionamento dos alertas referentes aos incidentes de segurança observados. A seguir são descritas algumas das principais funcionalidades dessa solução.

Incident Handling Fact Base

A *Incident Handling Fact Base* (IHFB) fornece informações importantes para representar os mais diversos tipos e formas de incidentes de segurança. Através dessa forma de representação é possível identificar as dependências entre os tipos de incidentes e as vulnerabilidades existentes. Outra importante função da IHFB é fornecer subsídios para a agregação de alertas. A tabela 3.2 descreve as informações armazenadas na IHFB.

Tabela 3.2 – Informações para a representação de incidentes

Tipo de Campo	Descrição
Código do incidente	Chave-primária que identifica um incidente de segurança. O mapeamento entre esse código e especificações como Bugtraq ID é possível utilizando o campo *Referências*.
Classe do incidente	Representa ações tais como: violação de privilégios, sondagem de portas e violação da integridade dos dados.
Descrição	Descrição textual do ataque.
Sistemas operacionais vulneráveis	Indicação dos tipos e versões de sistemas operacionais vulneráveis ao tipo de incidente definido.
Portas e aplicações	Lista de serviços de rede e aplicações ativas para que esse tipo de incidente tenha sucesso.
Cluster	Índices que podem ser associados com tipos de incidentes. Alertas que possuem os mesmos índices são candidatos a serem unidos na etapa de agregação.
Referências	Códigos de incidentes de segurança nos formatos *Bugtraq ID, CERT ID e Common Vulnerabilities and Exposures (CVE).*

Com as informações contidas nessa base de dados e com a especificação da topologia da rede é gerado um escore de relevância para cada um dos alertas processados. A escala de relevância é entre 0 e 255, sendo que 0 significa a ausência de condições favoráveis para o desenvolvimento da atividade reportada. Já um valor próximo de 127 indica que nem todas as condições para um determinado ataque ocorrer são satisfeitas. Valores próximos de 255

representam alertas para os quais foram constatadas vulnerabilidades que permitem o desenvolvimento das atividades reportadas.

Priorização dos alertas

No *M-Correlator* os alarmes são agregados, formando alertas de alto nível. Esses, por sua vez, são classificados em função do grau de ameaça que representam aos objetivos da missão[1] de uma determinada rede. A especificação de uma missão é realizada em duas etapas: (a) enumeração dos serviços (tais como servidor *web* ou banco de dados) e dos recursos considerados mais críticos em uma rede e (b) identificação das classes de incidentes que representam maior ameaça à missão, segundo a percepção do profissional de segurança.

Como resultado do cálculo da prioridade dos alertas se obtém um dos seguintes valores: baixo, médio baixo, médio, médio alto e alto. Esses valores representam a importância dos alarmes gerados em função da missão que foi especificada.

Classificação dos incidentes

A classificação dos incidentes constitui a avaliação final sobre cada incidente de segurança no que diz respeito ao impacto desse evento e à probabilidade de sucesso dessa tentativa de ataque ou intrusão. O cálculo da distribuição de probabilidade, realizado nessa etapa, está baseado nas premissas das redes *bayesianas,* conforme descrito em [Valdes, 2001].

3.2.2 Collaborative Intrusion Detection System (CIDS)

O *Collaborative Intrusion Detection System* (CIDS) [Wu et al., 2003] é um *framework* que se propõe a detectar intrusões em um ambiente distribuído, de forma mais eficiente e com maior exatidão do que os mecanismos convencionais. Para isso, esse *framework* conta com recursos para capturar, agregar e avaliar eventos.

A arquitetura do CIDS é composta pelos seguintes componentes: um conjunto de sensores (*Elementary Detectors - EDs*), uma fila de mensagens (*Message Queue – MQ*), um monitor de conexões (*Connection Tracker – CT*) e um gerente (*Manager*). Essa arquitetura está representada na Figura 3.6.

Os EDs são sensores instalados em diferentes estações e distribuídos em uma rede de computadores conforme a estratégia de monitoração previamente definida. Internamente os EDs são organizados em três camadas: rede (consiste na análise do tráfego de rede), *kernel* (monitoramento dos sub-sistemas de um sistema operacional) e aplicação (programas que estejam sendo executados na estação e cuja função seja monitorar algum serviço).

---

[1] A missão é a razão pela qual determinados recursos são alocados em uma rede.

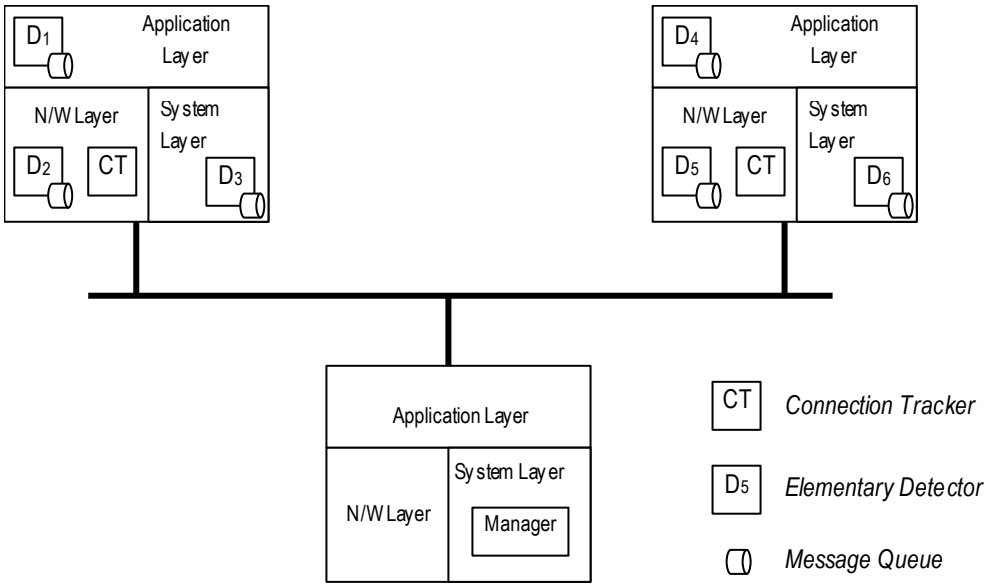

Figura 3.6 – Visão geral da arquitetura CIDS

Os experimentos realizados até o momento descrevem a utilização dos seguintes sensores: Snort [Roesch, 1999], Libsafe [Baratloo, 2000] e Sysmon [Mauch, 1998]. O Snort é um sistema de detecção de intrusão baseado em rede; já o Libsafe é uma aplicação para prevenir ataques que exploram vulnerabilidades como *buffer overflow;* por fim, o Sysmon é um *patch* para *kernel* Linux que fornece funcionalidade para realizar a monitoração de determinadas atividades do sistema operacional.

A comunicação entre os componentes dessa arquitetura é estabelecida através de uma fila de mensagens que utiliza como protocolo de transporte o TCP. As mensagens trocadas entre os sensores e o gerente possuem uma assinatura e um número de série. A cada ED está associada uma chave secreta que é compartilhada com o gerente. Dessa forma, segundo os autores, o CIDS não se torna vulnerável à utilização de mensagens forjadas ou mesmo à retransmissão de mensagens legítimas.

A função de agregar as informações de diferentes sensores, analisá-las e decidir sobre a existência de uma intrusão cabe, nessa arquitetura, ao gerente. Deve haver ao menos um gerente por domínio administrativo o qual se deseja monitorar. O gerente é constituído pelos seguintes componentes: motor de tradução, motor de inferência e motor de resposta.

O motor de tradução converte um alerta gerado por um ED para um formato de evento compreensível ao CIDS. Já o motor de inferência define se há ou não um ataque em andamento. Quando um ataque é observado, o motor de resposta gera um alerta e executa um procedimento de reação a essa intrusão. Atualmente o CIDS implementa apenas um tipo de resposta: encerrar o processo de conexão com as estações suspeitas.

3.2.3 Análise das arquiteturas

Para realizar a análise das arquiteturas de detecção de intrusão, foram utilizados os seguintes critérios: (a) tipo de detecção, (b) ataques de múltiplas etapas, (c) medidas de contenção, (d) fontes de informação e (e) extensibilidade.

Através do primeiro critério (tipo de detecção) busca-se identificar se a solução é capaz de detectar um ataque ainda na fase de desenvolvimento (detecção pró-ativa) ou somente após a conclusão do mesmo. O segundo critério (ataques de múltiplas etapas) avalia se a arquitetura é capaz de detectar um cenário de intrusão ou, apenas, eventos individuais. Já os critérios denominados medidas de contenção e fontes de informação permitem identificar se as arquiteturas prevêem a realização de ações de contenção para inibir a evolução de um ataque e quais os mecanismos de segurança utilizados, respectivamente. O último critério considerado diz respeito ao processo de agregação de novas fontes de informação (extensibilidade) à arquitetura.

Tabela 3.3 – Análise das arquiteturas

	M-Correlator	**CIDS**
Tipo de Detecção	Após ataque	Após ataque
Ataques de múltiplas etapas	Não	Não
Medidas de Contenção	Não	Limitada
Fontes de Informação	Limitada	Limitada
Extensibilidade	Limitada	Não

No que tange ao tipo de detecção, o M-Correlator só tem condição de detectar um ataque após um ciclo longo de análise que contempla, entre outras tarefas, a execução de filtros, cálculos de prioridade, agregação de eventos e, por fim, a geração de alertas. A exemplo do M-Correlator, o CIDS só detecta ataques após a realização de um conjunto de atividades que se inicia com a conversão do alerta recebido para um formato interno reconhecido por essa arquitetura. Em seguida, esse alerta é enviado para o motor de inferência que, por sua vez, analisa essas informações e gera um indicador que, posteriormente, será analisado por outro componente da arquitetura, o motor de resposta. É o motor de resposta que toma a decisão de gerar um novo alerta. Isso caracteriza um ciclo muito longo e, portanto, inadequado para detectar ataques ainda durante o seu andamento.

Quanto à detecção de ataques de múltiplas etapas, nenhuma das arquiteturas estudadas permite modelar e monitorar cenários de intrusão compostos por diversas atividades

(intrusivas ou não). Referente à execução de medidas de contenção, essas não estão previstas no projeto M-Correlator. Já no CIDS as ações de contenção limitam-se ao bloqueio das conexões previamente estabelecidas.

O conjunto de mecanismos de segurança utilizados pelo M-Correlator se reduz, atualmente, aos sistemas de detecção de intrusão e aos *firewalls*. Já o CIDS é ainda mais restrito limitando-se a coletar, apenas, alertas gerados por aplicações como Snort, Sysmon e libsafe.

Finaliza-se a síntese comparativa entre os dois projetos analisando a capacidade de inclusão de novos mecanismos (extensibilidade) a essas arquiteturas. Nesse requisito o M-Correlator, segundos seus autores, pode trabalhar com outras fontes de informações além daquelas já mencionadas, mas para isso há necessidade de adaptações na arquitetura, sobretudo, nos filtros e no componente que calcula a prioridade dos alertas. Além disso, todos os novos mecanismos (ferramentas) devem, obrigatoriamente, estar instalados em um mesmo domínio administrativo. Quanto ao CIDS, esse não apresenta possibilidades de atuar com outros sensores.

4 Multistage Attack Detection Architecture

O desenvolvimento de mecanismos de detecção de intrusão, sobretudo para identificar ataques compostos por múltiplas etapas, é de extrema relevância e representa uma questão de pesquisa em aberto. Nesse contexto, este trabalho apresenta (a) uma linguagem para especificação de ataques e (b) uma arquitetura baseada em *web services* para detecção de cenários de ataques compostos por múltiplas fases.

Em relação à linguagem proposta, denominada *Multistage Attack Description Language*, cabe mencionar as seguintes características. Possui uma notação gráfica para representação visual e em alto nível do ataque e uma notação textual, baseada em XML, que permite a especificação detalhada do mesmo. Através dessa linguagem é possível especificar eventos oriundos das mais diversas fontes de informações, o que viabiliza a especificação de diferentes fases de um ataque.

A arquitetura oferece um mecanismo uniforme de comunicação com diferentes serviços de segurança para subscrição e notificação de eventos. A correlação entre os eventos observados e os ataques especificados por meio da linguagem MADL permite detectar ataques distribuídos e de múltiplas etapas. O processo de comunicação entre os componentes desta arquitetura é realizado em conformidade com a especificação *Web Services Notification* [Graham et al., 2005]. A arquitetura prevê, ainda, a execução de serviços de contenção, que podem ser executados com o objetivo de impedir a evolução do ataque .

Este Capítulo está organizado da seguinte forma. Inicialmente, na Seção 4.1, é apresentada a linguagem proposta para modelagem e especificação dos cenários de ataques. Na Seção 4.2, são descritos os componentes da arquitetura, incluindo o detalhamento de suas funções e a relação estabelecida entre os mesmos. Os aspectos referentes à implementação da arquitetura são abordados na Seção 4.3. Na Seção 4.4. é realizada uma análise da linguagem e da arquitetura propostas.

4.1 Representação de cenários de ataques

Considerando as limitações das formas de representação de ataques discutidas no Capítulo 3, propõe-se a utilização de uma linguagem para modelar e descrever cenários de ataques baseados em múltiplas etapas. Esta linguagem é denominada *Multistage Attack Description Language* (MADL) e possui duas notações, uma notação gráfica (*Graphical*

MADL) e outra textual (*Textual* MADL). Essas notações não são equivalentes, pois a notação textual permite uma descrição mais detalhada das atividades que fazem parte de um cenário de ataque. Já a notação gráfica, por sua vez, equivale a um sub-conjunto da notação textual, e oferece a possibilidade de representar visualmente os cenários de ataque em um alto grau de abstração.

Através da notação textual é possível descrever um cenário a ser monitorado sem fazer uso da notação *Graphical* MADL. O contrário não é possível, uma vez que a representação gráfica não reúne todas as informações necessárias para a detecção de um ataque. Em seguida, são apresentados a notação *Graphical* MADL e os principais construtores da notação Textual MADL.

4.1.1 Graphical MADL (G-MADL)

A notação gráfica G-MADL é uma adaptação do diagrama de atividades da *Unified Modeling Language* (UML). Segundo [Fowler, 2003] esse tipo de diagrama teve a sua origem em técnicas como: diagrama de eventos de Jim Odell e modelagem de *workflows*. Além disso, é um caso especial do diagrama de estados, que permite a representação do fluxo das atividades realizadas em um sistema.

Pode-se afirmar que um diagrama de atividades é uma técnica para se representar interações entre diversas atividades, com a possibilidade de expressar: (a) quando determinadas ações são executadas, (b) onde elas ocorrem, (c) como tais ações são executadas e (d) a conseqüência da execução das mesmas (mudanças dos estados) [Medeiros, 2004].

Construtores elementares da linguagem

A Figura 4.1 apresenta os elementos básicos que fazem parte da especificação visual de um cenário de ataque. Esses elementos são:

- início: identifica o começo da representação gráfica de um ataque;
- setas: indicam o fluxo de realização das atividades. Estas setas podem indicar atividades seqüenciais, paralelas e também condicionalidade (desvios no fluxo);
- atividade: descreve um evento observado ou uma ação a ser executada;
- fim: define o término de um fluxo de atividades que caracteriza um cenário de ataque;

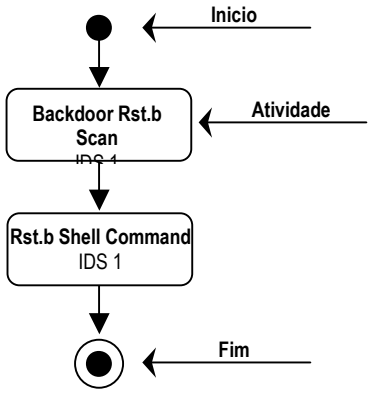

Figura 4.1 – Representação gráfica de um cenário de ataque

A especificação de um cenário de ataque inicia sempre a partir de uma atividade inicial representada por um círculo, cuja parte interna é totalmente preenchida. O término de uma especificação é indicado por um símbolo contendo dois círculos concêntricos onde o círculo interno é totalmente preenchido.

As atividades são representadas por retângulos de cantos arredondados; as setas contínuas indicam a ordem de execução dessas atividades. Os rótulos que identificam as atividades contêm uma pequena descrição do evento ou ação e a identificação do serviço de segurança que deverá monitorar ou executar essa atividade.

Representação de atividades seqüenciais

Atividades seqüenciais representam eventos a serem observados ou ações que devem ser executadas de maneira sucessiva. A Figura 4.1 ilustra um cenário de ataque composto por uma seqüência de duas atividades (*Backdoor Rst.b Scan* e *Rst.b Shell Command*). A primeira atividade é uma sondagem de portas, cujo objetivo é identificar estações comprometidas pelo *backdoor Rst.b*. A segunda etapa é a execução remota de comandos (*Rst.b Shell Command*) nas estações alvo.

Representação de atividades paralelas

A representação de um cenário de ataque pode envolver a observação de eventos e/ou execução de ações que ocorram em paralelo e em uma ordem qualquer. Esse é o caso do exemplo apresentado na Figura 4.2**,** em que se especifica o interesse de observar a instalação do software *Mstream* em três estações.

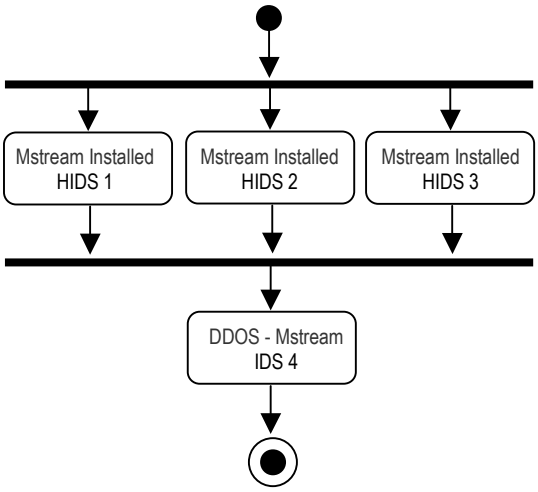

Figura 4.2 – Representação gráfica envolvendo atividades em paralelo

Para os casos em que seja necessário indicar que um número "x" de ocorrências dos fluxos de uma atividade em paralelo basta para que o ataque tenha andamento, foi prevista uma notação do tipo n..* que deve ser posicionada acima e a direita da primeira barra. Por exemplo, a notação 2..* informa que se pelo menos 2 alertas indicarem a instalação do *Mstream* isso já será o suficiente para que o ataque tenha andamento.

Representação de seletividade

A seletividade permite definir fluxos mutuamente exclusivos em cenários de ataque. No exemplo ilustrado na Figura 4.3, o ataque especificado tem início a partir de uma e, somente, uma das três atividades descritas (*Apache Chunked Worm, Directory Disclosure Attempt* e *Apache Chunked Exploit*). Todos esses ataques resultam na execução de comandos remotamente. Uma vez com acesso ao sistema, o ataque chega ao fim por meio de uma conexão *Trivial File Transfer Protocol* (TFTP), realizada para transferir informações para outro *host* na Internet.

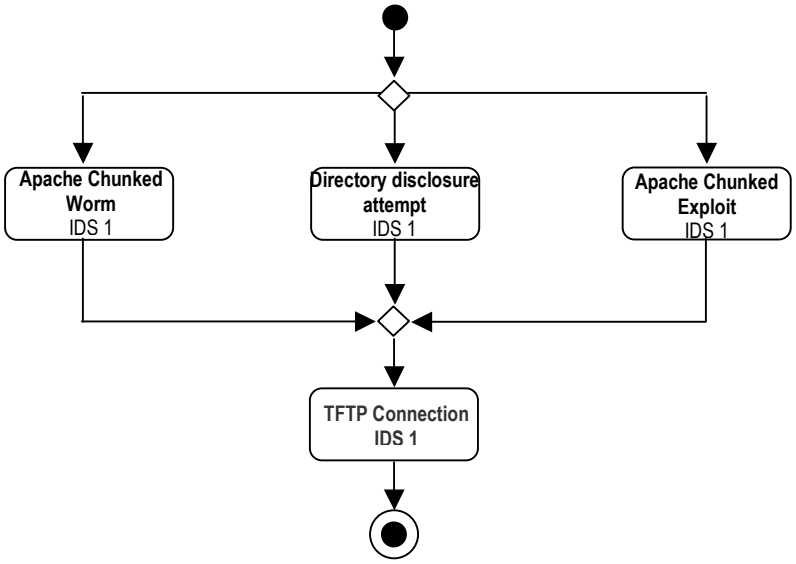

Figura 4.3 – Representação gráfica envolvendo condicionalidade

4.1.2 Textual-MADL (T-MADL)

Esta linguagem permite descrever, através de documentos no formato XML, o fluxo de atividades que constitui um ataque de múltiplas etapas. O formato de um documento T-MADL é semelhante à estrutura (aninhamento de elementos e utilização de atributos) empregada em outras linguagens também baseadas em XML. Por se tratar de uma linguagem para especificação de ataques cujas estratégias podem ser as mais variadas possíveis, T-MADL prevê um conjunto de elementos para descrever diferentes situações.

Representação de informações para documentação

Conforme apresentado na Figura 4.4, o elemento raiz de um documento T-MADL é a *tag* denominada scenario cujo atributo code (linha 1) identifica a especificação criada. Outro elemento utilizado para identificação e documentação de uma especificação de ataque é a *tag* catalogue (linha 2), nodo pai, que contém os seguintes nodos filhos (entre as linhas 3 e 5): description, author e date.

```
1  <scenario code="101">
2     <catalogue>
3        <description> Especificação do ataque Backdoor Rst.b </description>
4        <author> Leonardo Lemes Fagundes </author>
5        <date> 01/12/2005 </date>
6     </catalogue>
7     ...
8  </scenario>
```

Figura 4.4 – Sintaxe das *tags* scenario e catalogue

Elemento *event*

Um evento pode ser tanto um alerta de ataque gerado por um sistema de detecção de intrusão (ou de qualquer outro mecanismo de segurança), como por exemplo um *portscan*, quanto uma atividade considerada legítima, por exemplo, uma conexão via SSH a um determinado servidor. A *tag* event é o elemento que representa um evento e a sua sintaxe é ilustrada na Figura 4.5.

```
<event name="value" ip_src="value" ip_dst="value" date="value" time="value" />
```

Figura 4.5 – Sintaxe do elemento *event*

Conforme pode ser observado na Figura 4.5, o elemento event possui atributos que possibilitam especificar o nome do evento observado, o endereço IP do qual partiu o evento, o endereço IP da estação alvo e a data e hora de ocorrência do evento. Apenas o atributo referente ao nome do evento é obrigatório, todos os demais são opcionais.

A definição de um atributo pode ser realizada através da utilização de caracteres especiais (*wildcards*) como "*" (asterisco), "?" (interrogação) e "!" (negação). O símbolo "*" é utilizado para substituir uma cadeia de caracteres quaisquer, já os caracteres "?" e "!" são utilizados para indicar substituição de um único caractere e negação de uma cadeia de caracteres, respectivamente. O exemplo mostrado na Figura 4.6 descreve um evento denominado "Backdoor Rst.b Scan" cujo endereço IP da máquina alvo é 192.168.10.5.

```
<event name="Backdoor Rst.b Scan" ip_dst="192.168.10.5"/>
```

Figura 4.6 – Exemplo de descrição de um evento

Elemento *alert*

O elemento alert representa uma mensagem enviada à aplicação de gerenciamento no intuito de comunicar a ocorrência de um conjunto de atividades suspeitas que podem significar o desenvolvimento de uma atividade intrusiva.

A geração de uma mensagem de alerta ocorre através do construtor alert. Esse construtor pode ser inserido em qualquer parte de um documento T-MADL. A sintaxe dessa *tag* é ilustrada na Figura 4.7.

```
<alert   msg="value"   date="current-time" />
```

Figura 4.7 – Sintaxe do elemento *alert*

A *tag* alert possui possui apena um atributo. O atributo msg corresponde a um texto que será definido pelo administrador do sistema. O exemplo da Figura 4.8 mostra a geração de uma mensagem de alerta contendo a seguinte informação "Atenção, tentativa de intrusão em andamento.".

```
<alert   msg="Atenção, tentativa de intrusão em andamento." />
```

Figura 4.8 – Exemplo de geração de uma mensagem de alerta

Elemento *action*

Além da geração de alertas, T-MADL prevê o elemento action (Figura 4.9) que permite a invocação de *web services* visando a execução de um procedimento de contenção. Esses *web services* realizam alterações nas configurações de alguns serviços e executam procedimentos, no intuito de inibir o desenvolvimento da atividade intrusiva que está em andamento. A *tag* action tem como único atributo o endereço do serviço *web* (ws). Ao invocar um serviço de contenção é opcional a passagem de parâmetros.

```
<action  ws="path_url($param1, $param 2)" />
```

Figura 4.9 – Sintaxe do elemento *action*

O exemplo da Figura 4.10 mostra a invocação de um serviço, denominado "ws-accessdeny" localizado no *host* que responde pelo endereço 10.16.163.77. Este serviço bloqueia temporariamente as solicitações de acessos que partem do endereço $ip_src para a estação identificada pelo parâmetro $ip_host.

```
<action  ws="http://10.16.163.77/services/ws-accessdeny($ip_src, $ip_host) />
```

Figura 4.10 – Sintaxe do elemento *action*

Elemento *sequence*

O elemento sequence indica que um conjunto de eventos deve ocorrer de forma seqüencial, um após o outro. No exemplo apresentado na Figura 4.11 o cenário é composto por uma seqüência (entre as linhas 2 e 5) de dois eventos (linha 3 e 4).

```
1  ...
2     <sequence>
3        <event name="Backdoor Rst.b Scan" ip_dst="192.168.10.5"/>
4        <event name="Rst.b Shell Command" ip_dst="192.168.10.5"/>
5     </sequence >
6  ...
```

Figura 4.11 – Sintaxe do elemento *sequence*

Elemento *parallel*

A *tag* parallel permite descrever fluxos de atividades que ocorrem em paralelo. Um exemplo do uso dessa *tag* é apresentado na Figura 4.12. Nesse exemplo, se observa que os eventos declarados, entre as linhas 3 e 5, podem ocorrer em qualquer ordem desde que pelo menos dois desses eventos ocorram. Essa restrição é declarada através do atributo min da *tag* parallel (linhas 2).

```
1  ...
2     <parallel min="2">
3           <event name="Mstream Installed" ip_dst="192.168.10.*"/>
4           <event name="Mstream Installed" ip_dst="192.168.20.*"/>
5           <event name="Mstream Installed" ip_dst="192.168.30.*"/>
6     </parallel>
7  ...
```

Figura 4.12 – Sintaxe do elemento *parallel*

Elemento *choice*

O exemplo apresentado na Figura 4.13 representa parte da descrição de um cenário de ataque, na qual a tag choice (entre as linhas 2 e 6) indica que basta a observação de um dos eventos especificados, nas linhas 3, 4 e 5, para que essa etapa do ataque seja realizada.

```
1  ...
2       <choice>
3          <event name="Apache Chunked Worm" ip_dst="192.168.10.5"/>
4          <event name="Directory disclosure attempt" ip_dst="192.168.10.5"/>
5          <event name=" Apache Chunked Exploit" ip_dst="192.168.10.5"/>
6       </choice>
7   ...
```

Figura 4.13 – Sintaxe do elemento *choice*

Documentos escritos em T-MADL alimentam o motor de detecção (componente discutido na subseção 4.2.3), que recebe subscrições e monitora a realização de ataques.

Como pôde ser observado nos exemplos desta seção, a linguagem proposta permite representar não somente alertas referentes a ataques, mas também permite descrever outras atividades necessárias para a detecção de um ataque composto por múltiplos estágios. O esquema XML da linguagem MADL pode ser visto no Anexo B.

4.2 Visão conceitual da arquitetura

Como mencionado anteriormente, a idéia principal deste trabalho é a criação de uma arquitetura para detectar cenários de ataques baseados em múltiplas etapas. A Figura 4.14 ilustra uma visão geral da arquitetura proposta e as interações realizadas entre os diferentes componentes (aplicação de gerenciamento, serviços de notificação, serviço de detecção e serviços de contenção) que fazem parte da mesma.

A aplicação de gerenciamento é o componente da arquitetura através do qual o gerente da rede (i) especifica os cenários de intrusão, (ii) instancia o processo de monitoração e (iii) visualiza os alertas gerados pelo serviço de detecção.

Os serviços de notificação são responsáveis por observar e comunicar a ocorrência de eventos. As principais funções dos serviços de notificação são (i) publicar nos repositórios de serviços *web* as suas funcionalidades, (ii) receber subscrições por eventos e (iii) notificar o serviço de detecção à medida que os eventos subscritos forem sendo observados.

O serviço de detecção é o componente da arquitetura responsável pela monitoração das especificações de cenário de ataques (documentos T-MADL). As principais funções exercidas por esse serviço são (i) receber as mensagens de notificação , (ii) gerar alertas e (iii) invocar os serviços de contenção. Já os serviços de contenção são responsáveis pela execução de procedimentos cujo objetivo é inibir a evolução de um ataque.

A análise da seqüência das interações realizadas entre os componentes dessa arquitetura (Figura 4.14) resulta em um melhor entendimento sobre o funcionamento da mesma. Inicialmente, os serviços de notificação e contenção são publicados (fluxos 1 e 2 da figura) em um repositório de serviços *web* (Capítulo 2).

A partir do momento em que o gerente de rede estabelece, via *browser*, uma conexão (3) com a aplicação de gerenciamento, é possível realizar consultas a repositório de serviços (4) com o objetivo de se obter uma lista dos serviços de notificação e contenção de ataques disponíveis. As informações fornecidas por esses serviços são utilizadas na especificação dos cenários de intrusão que, posteriormente, são armazenados em uma base de dados local (5). Ao instanciar um cenário de intrusão junto ao serviço de detecção são geradas diversas

subscrições de eventos (6) aos serviços de notificação e, então, o serviço de detecção inícia a monitoração desse cenário (7).

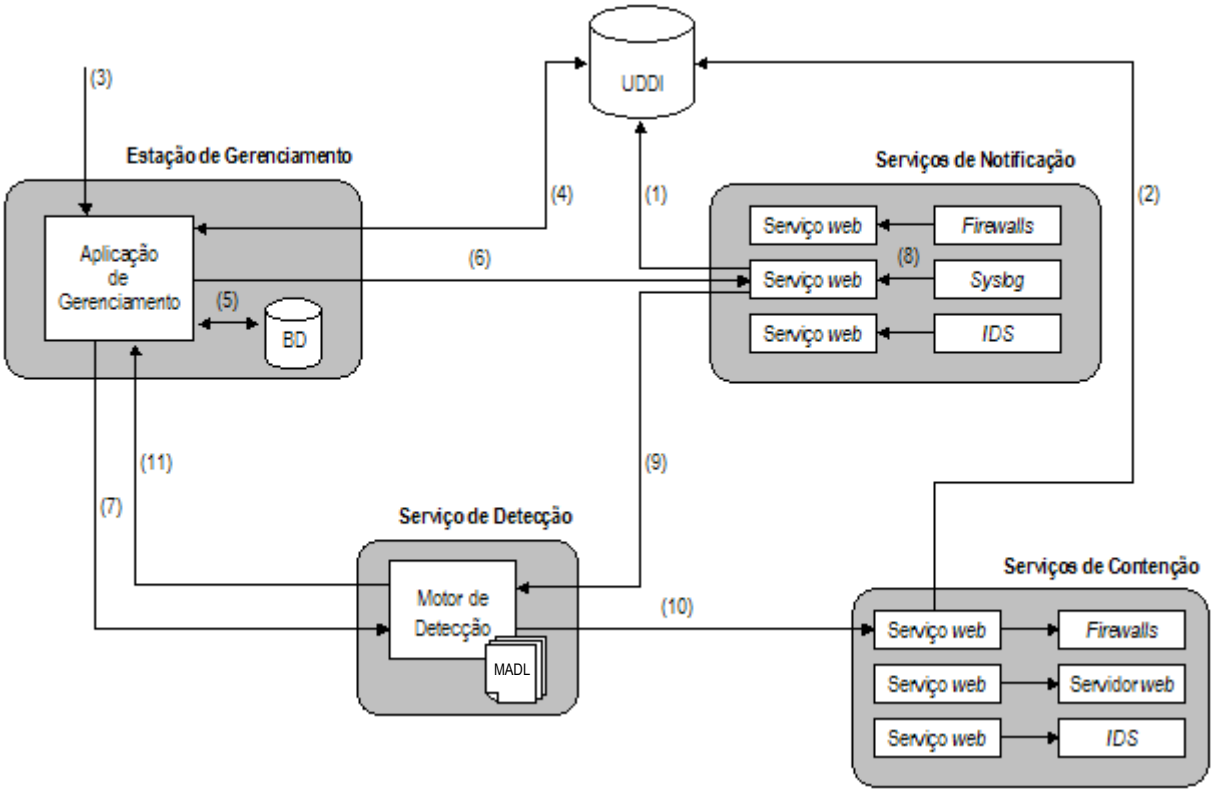

Figura 4.14 - Visão geral da arquitetura

Assim que um serviço de notificação recebe uma subscrição vinda da estação de gerenciamento, se inicia o processo de monitoração dos tópicos de interesse. Tão logo seja identificada a ocorrência de um desses tópicos (8), o serviço de detecção é comunicado (9). Para cada novo evento recebido, o serviço de detecção verifica se o mesmo não permite a evolução de um dos ataques já instanciados. Além disso, essa verificação pode resultar tanto na execução dos serviços de contenção (10) quanto no envio de alertas (11) à aplicação de gerenciamento.

4.2.1 Aplicação de gerenciamento

As atividades que podem ser realizadas a partir de uma estação de gerenciamento são descritas a seguir:

- especificação de cenários de intrusão: através de um ambiente de edição, o gerente da rede pode especificar o conjunto de atividades que serão monitoradas. Esses cenários podem ser modelados a partir do zero ou tendo como base um cenário já existente (uma derivação);

- consulta ao repositório de serviços *web*: através desta opção é possível identificar serviços para observação de eventos e execução de ações de contenção;
- instanciação dos cenários de ataque: corresponde à subscrição de eventos junto aos serviços de notificação e programação da monitoração dos cenários de ataque pelo serviço de detecção;
- visualização de alertas: consiste na apresentação dos alertas, referentes ao cenários de ataque monitorados, gerados pelo serviço de detecção.

4.2.2 Serviço de notificação

No contexto da arquitetura proposta, ferramentas tais como programas antivírus, sistemas de detecção de intrusão, *firewalls*, e serviços de monitoração de sistemas atuam como sensores, uma vez que coletam informações que serão analisadas pelo serviço de notificação (Figura 4.15). Essas ferramentas são recursos que, normalmente, se encontram disponíveis na infra-estrutura de segurança das instituições e geram alertas importantes para identificar cenários de ataques compostos por múltiplas etapas.

Ao publicar as suas funcionalidades (1) junto a um repositório de serviços *web*, o serviço de notificação recebe subscrições de eventos (6) que partem da aplicação de gerenciamento. Essas subscrições são interpretadas para que sejam identificados quais os eventos de interesse e quais as estações (no caso dessa arquitetura é o serviço de detecção) que devem ser comunicadas quando esses eventos ocorrerem.

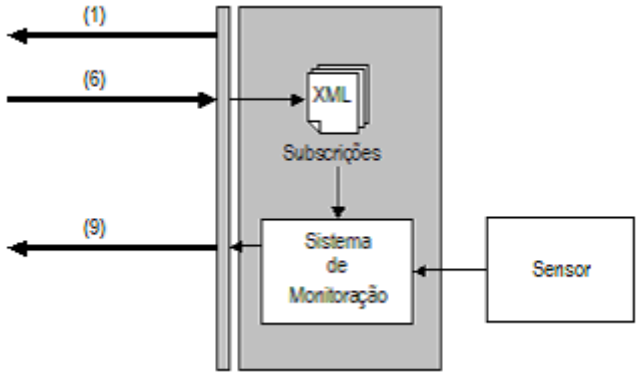

Figura 4.15 - Serviço de notificação

Uma vez que uma subscrição é recebida e interpretada com êxito a mesma permanece sendo monitorada até que tal subscrição seja removida. O sistema de monitoração implementa as funcionalidades de análise das informações geradas pelos sensores.

O serviço de notificação organiza em uma árvore de tópicos, ou propriedades, todas as informações que pode monitorar. Por exemplo, no caso de um sistema de detecção de intrusão, cada tipo de alerta fornecido representa uma propriedade. Para cada uma dessas propriedades as operações disponíveis são: subscribe, destroy e SetResourceProperties, ou seja, a aplicação de gerenciamento pode realizar a subscrição (subscribe) por um tópico e a exclusão (destroy) dessa subscrição. Já o serviço de notificação assim que observa o evento esperado atualiza (SetResourceProperties) as informações da propriedade em questão, para que a mesma seja notificada. Quando o serviço de notificação identifica que um evento de interesse ocorreu, uma mensagem XML contendo essa informação é enviada, através da interface de notificação, aos motores de detecção (9).

4.2.3 Serviço de detecção

As atividades realizadas pelo serviço de detecção (Figura 4.16) têm início assim que ocorre a instanciação dos cenários de ataque (7). A cada nova mensagem de notificação recebida (9), a mesma é armazenada em um banco de dados com as seguintes informações: nome do evento, endereço IP de origem, endereço IP de destino e data e hora de ocorrência do evento.

Caso esse evento faça parte de alguns dos cenários de ataque monitorados, o serviço de detecção pode, dependendo do cenário invocar os serviços de contenção (10) ou gerar alertas (11) para fornecer informações ao gerente da rede sobre as atividades detectadas.

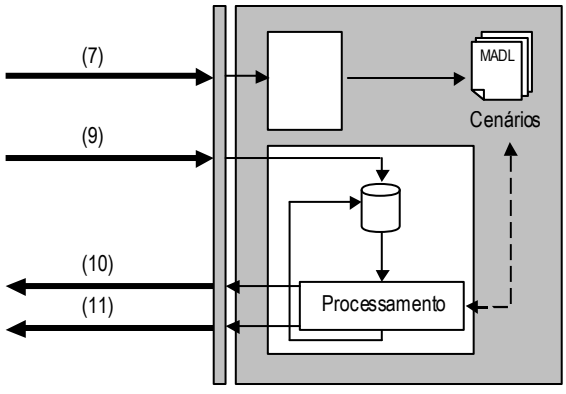

Figura 4.16 - Serviço de detecção

Para concluir essa subseção é importante registrar que, embora não esteja no escopo dessa proposta, a arquitetura em desenvolvimento pode ser composta por um ou mais serviços de detecção, desde que sejam respeitadas as interfaces com os demais componentes. A utilização de diversos serviços de detecção faz com que (i) aumente a disponibilidade da arquitetura, uma vez que existem réplicas capazes de permanecer realizando o processamento

das informações, mesmo quando um dos serviços de detecção estiver inoperante e (ii) fornece maior escalabilidade, uma vez que nem todos os cenários de intrusão são processados na mesma estação.

4.2.4 Serviço de contenção

O serviço de contenção (Figura 4.17) é um *web services* que tem como função realizar atividades, tais como reprogramar um serviço de segurança para inibir a evolução de um ataque.

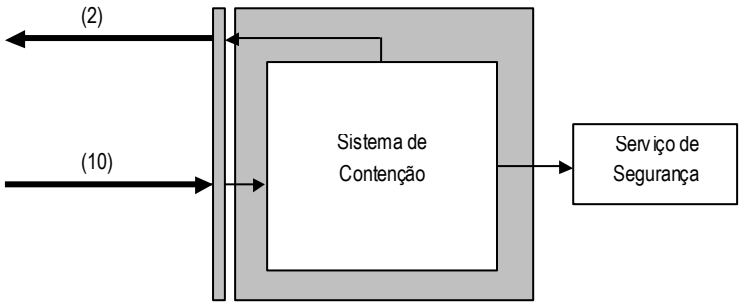

Figura 4.17 - Serviço de contenção

Após publicar as suas funcionalidades (2) junto a um repositório de serviços *web*, o serviço de contenção torna-se apto para ser invocado pelo motor de detecção (10). Uma vez invocado, este serviço executa procedimentos que alteram o funcionamento de um determinado mecanismos de segurança.

4.3 Implementação

Nesta seção os componentes da arquitetura são descritos com ênfase no projeto e na implementação dos mesmos. Ressalta-se que o modelo de comunicação utiliza o *Pubscribe*, uma implementação em Java da especificação *Web Services Notification* (Seção 2.2.1). Essa implementação é parte do projeto *Web Services Project @ Apache*[2].

4.3.1 Aplicação de gerenciamento

A aplicação de gerenciamento constitui-se de um conjunto de *scripts* que, na atual implementação do protótipo, oferece suporte à definição de especificações em MADL, organização do processo de instanciação dos cenários de ataque e visualização dos alertas gerados pelo serviço de detecção.

[2] Documentação disponível em: http://ws.apache.org/

A edição, formatação e verificação dos documentos que utilizam a notação T-MADL é realizada através do *XML Starlet Command Line XML Tollkit*[3], um conjunto de utilitários de código aberto desenvolvido em linguagem C que se utiliza das bibliotecas *libxml2* e *libxslt*. Uma vez que os documentos T-MADL estejam devidamente verificados em relação ao esquema XML da linguagem MADL, é o momento de instanciar o cenário de ataque. Essa tarefa consiste na subscrição pelos eventos envolvidos e na programação do serviço de detecção.

A Figura 4.18 representa, de forma simplificada, um modelo de subscrição para o evento *Rst.b Shell Command* (linha 7). Na linha 3 é informado ao serviço de notificação que todas as mensagens referentes ao evento em questão devem ser enviadas ao endereço http://10.16.172.77 na porta 8081, que corresponde ao serviço de detecção.

```
1  <wsnt:Subscribe>
2    <wsnt:ConsumerReference>
3      <wsa:Address>http:/10.16.172.77:8081</wsa:Address>
4      <wsa:ReferenceProperties/>
5    </wsnt:ConsumerReference>
6    <wsnt:TopicExpressionDialect="http://docs.oasis-open.org/wsn/2004/ 06/TopicExpression/Simple">
7      fs: Rst.b Shell Command
8    </wsnt:TopicExpression>
9  </wsnt:Subscribe>
```

Figura 4.18 - Formato simplificado de uma subscrição para o evento *Rst.b Shell Command*

A subscrição pelos eventos e a programação do serviço de detecção é realizada com o apoio de um script, desenvolvido em Bash (*Bourne Again Shell*), que automatiza o processo. A Figura 4.19 ilustra o *script* usado para a instanciar o cenário de ataque apresentado na Figura 4.11. Na linha 2 é realizada a subscrição pela propriedade BackdoorRst.bScan. Já na linha 3 a subscrição realizada refere-se a propriedade Rst.bShellCommand. Essas subscrições foram realizadas para o serviço ws-snort. A linha 4 invoca o serviço de detecção que, por sua vez, passa a monitorar a especificação MADL correspondente a esse cenário de ataque.

```
1  ...
2  ant –Durl=http://10.16.166.15:8080/pubscribe/services/arq/ws-snort -Dxml= requests/subscribe_BackdoorRst.bScan.soap
3  ant –Durl=http://10.16.166.15:8080/pubscribe/services/arq/ws-snort -Dxml= requests/subscribe_Rst.bShellCommand.soap
4  updDetectionService
5  ...
```

Figura 4.19 – Instanciação de um cenário de ataque

[3] O Software e a documentação estão disponíveis em http://xmlstar.sourceforge.net/

4.3.2 Serviço de notificação

Assim que uma mensagem de subscrição é recebida, o serviço de notificação (*publisher*) passa a monitorar o evento (também denominado tópico) contido nessa mensagem. Conforme já mencionado, esses eventos são gerados por diferentes mecanismos de segurança. Para realizar a atividade de monitoração dos tópicos subscritos, foram criados *scripts* que percorrem os arquivos de *log* gerados por esses mecanismos e, ao detectarem um evento de interesse, (a) normalizam as informações obtidas, (b) criam uma mensagem de notificação e, em seguida, (c) distribuem a mensagem.

O processo de normalização da informação consiste na extração das informações, sobre um determinado evento, no seguinte formato: [nome do evento] [ip_origem] [ip_destino] [data] [hora:minuto]. A partir do instante em que são obtidas as informações necessárias sobre um evento, o *script* de monitoração cria uma requisição do tipo SetResourceProperties.

```
1  <Envelope xmlns="http://schemas.xmlsoap.org/soap/envelope/"
2        xmlns:fs="http://ws.apache.org/resource/example/arq">
3
4    <Header xmlns:wsa=\"http://schemas.xmlsoap.org/ws/2004/03/addressing\">
5        <wsa:To mustUnderstand=\"1\"> http://localhost:8080/wsrf/services/ arq </wsa:To>
6        <wsa:Action
7          mustUnderstand=\"1\">http://ws.apache.org/resource/example/arq/arqPortType/yourWsdlRequestName</wsa:Action>
8        <fs:ResourceIdentifier mustUnderstand=\"1\"> /dev/vg00/lvol1 </fs:ResourceIdentifier>
9    </Header>
10
11   <Body>
12       <wsrp:SetResourceProperties xmlns:wsrp=\"http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd\"
13           xmlns:fs=\"http://ws.apache.org/resource/example/arq\">
14             <wsrp:Update>
15                 <fs: Rst.b Shell Command >
16                   <fs:IP_src> 192.168.20.17 </fs:IP_src>
17                   <fs:IP_dst> 192.168.10.5 </fs:IP_dst>
18                   <fs:Date> 08/01/2006 </fs:Date>
19                   <fs:Time> 16:30 </fs:Time>
20                 </fs: Rst.b Shell Command >
21             </wsrp:Update>
22       </wsrp:SetResourceProperties>
23   </Body>
24
25 </Envelope>
26
```

Figura 4.20 - Formato simplificado de uma notificação para o evento *Exploit*

No exemplo ilustrado na Figura 4.20, é possível observar o formato do conteúdo de uma requisição SetResourceProperties invocada para a propriedade Rst.b Shell Command. Observe que em

cada operação *set* diversas informações, tais como o nome do evento e a data e a hora de realização do mesmo, são repassadas ao serviço de detecção.

4.3.3 Serviço de detecção

Ao receber uma mensagem de notificação, o serviço de detecção (desenvolvido em linguagem PHP) armazena a notificação em uma base de dados local. A Figura 4.21 ilustra uma notificação recebida pelo serviço de detecção referente à observação do evento Rst.b Shell Command. Na notificação é possível observar o valor atualizado (NewValue) para cada uma das informações (entre as linhas 7 e 11) fornecidas para a propriedade em questão (linha 5).

```
1  <wsn:Message>
2    <wsrf:ResourcePropertyValueChangeNotification
3      xmlns:wsrf="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd">
4      <wsrf:NewValue>
5        <fs: Rst.bShellCommand xmlns:wsrp="http://docs.oasis-open.org/wsrf/2004/06/wsrf-WS-ResourceProperties-1.2-draft-01.xsd"
6        xmlns="http://schemas.xmlsoap.org/soap/envelope/" xmlns:fs="http://ws.apache.org/resource/arq">
7          <fil:IP_src> 192.168.20.12 </fil:IP_src>
8          <fil:IP_dst> 192.168.10.5 </fil:IP_dst>
9          <fil:Date> 08/01/2006 </fil:Date>
10         <fil:Time> 16:45 </fil:Time>
11       </fs: Rst.bShellCommand>
12     </wsrf:NewValue>
13   </wsrf:ResourcePropertyValueChangeNotification>
14 </wsn:Message>
15
```

Figura 4.21 - Formato simplificado de uma notificação recebida pelo serviço de detecção

Uma vez que o motor de detecção tenha sido programado para monitorar um cenário, o mesmo é carregado em memória. A cada nova mensagem de notificação recebida, é realizada uma verificação que determina se o evento notificado permite a evolução de algum dos cenários monitorados. Para auxiliar na visualização da monitoração dos cenários foi criada uma interface *web* (vide Figura 5.4).

Caso esteja previsto, na especificação do cenário de ataque, que a observação de um determinado evento deve resultar na geração de um alerta, o motor de detecção envia uma mensagem de notificação à aplicação de gerenciamento. Essa mensagem, uma vez recebida é armazenada em uma base de dados local. A ocorrência de um evento, além da geração de uma mensagem, pode provocar a invocação de um serviço de contenção.

Por serem serviços *web* podem ser desenvolvidos em qualquer linguagem de programação. A atual implementação não contemplou o desenvolvimento de serviços de contenção.

4.4 Análise da linguagem e da arquitetura proposta

Nesta seção é realizada um análise da linguagem proposta e da arquitetura desenvolvida nessa dissertação, frente ao trabalhos relacionados (Capítulo 3).

4.4.1 Análise da linguagem

Conforme já discutido (vide Subseção 3.1.4), os requisitos para avaliação das linguagens empregadas na representação de cenários de ataques são: heterogeneidade, extensibilidade e expressividade.

No que tange ao requisito heterogeneidade, diferentemente das linguagens estudadas, MADL possibilita que sejam especificados cenários de ataque através de eventos de qualquer natureza (ex: sistemas de detecção de intrusão baseados em rede e em *host* independente do padrão em que os alertas são gerados, analizadores de *logs* e aplicações de gerenciamento).

Quanto a extensibilidade. MADL é muito semelhante a linguagem LAMBDA uma vez que ambas possuem uma representação textual baseada em XML e, portanto, toda vez que for inserido um novo elemento ou atributo é necessário atualizar o esquema utilizado. Além disso, o *parser* da linguagem também deverá ser atualizado.

Como pôde ser observado no capítulo anterior o requisito expressividade é avaliado a partir da existência de construtores que permitam representar os seguintes métodos de variação de um ataque: mutação, ordenação aleatória de eventos, substituição e *looping*.

A exemplo das linguagens STATL, CAML e LAMBDA, a linguagem proposta também oferece suporte à mutação. Para tal, utiliza-se os metacaracteres: * (um conjunto qualquer de caracteres), ? (substitui um único caracter), e ! (negação) na definição de valores e atributos das *tags*.

Ao contrário das linguagens STATL e CAML, que não prevêem construtores para contemplar a ordenação aleatória de eventos, MADL através da tag parallel permite definir que um determinado conjunto de eventos pode ocorrer em uma ordem temporal qualquer. No que se refere à especificação de ações alternativas para se alcançar o mesmo objetivo (substituição), diferentemente das demais, a linguagem proposta possui a tag choice que possibilita definir seqüências alternativas de eventos. T-MADL a exemplo das linguagens STATL e CAML não oferece suporte à especificação de eventos em *looping*.

4.4.2 Análise da arquitetura

Para analisar a arquitetura desenvolvida serão utilizados os critérios já descritos na Subseção 3.2.3. Esses critérios são os seguintes: tipo de detecção, ataques de múltiplas etapas, medidas de contenção, fontes de informação e extensibilidade.

Enquanto o M-Correlator e o CIDS são capazes de detectar um ataque somente após a realização do mesmo, a arquitetura desenvolvida nesse trabalho atua de forma pró-ativa, pois através de um determinado conjunto de evidências é possível identificar o ataque antes que o mesmo tenha atingido o seu ponto final.

Nenhuma das arquiteturas apresentadas permite modelar e monitorar ataques de múltiplas etapas. Já a arquitetura desenvolvida não limita-se a monitorar somente eventos individuais, mas também a correlacioná-los com os eventos descritos nos documentos T-MADL, tão logo os mesmos tenham sido notificados ao serviço de detecção.

No que se refere à execução de medidas de contenção, foi previsto um serviço especifico (vide Subseção 4.4.3) para tal finalidade. Quanto à diversidade de fontes de informações utilizadas, enquanto o M-Correlator limita-se aos sistemas de detecção de intrusão e aos *firewalls* a arquitetura desenvolvida ao longo dessa dissertação permite coletar dados de diversas fontes, desde sistemas de detecção de intrusão, plataformas de gerenciamento até serviços de monitoramento de aplicações.

Conforme já discutido, a extensibilidade das arquiteturas apresentadas é bastante limitada. Já a inclusão de recursos à arquitetura proposta é uma tarefa simples. Por exemplo, para incluir um serviço de notificação que informe sobre os eventos observados em um servidor de banco de dados, basta criar um *serviço web* capaz de monitorar os arquivos de log do mesmo.

5 Avaliação Experimental

Este capítulo apresenta uma avaliação experimental da arquitetura implementada. O objetivo dessa avaliação foi realizar a prova de conceito da solução proposta e observar o comportamento dos seus componentes. Para tal, foi modelado um ataque de negação de serviço distribuído. Esse cenário foi utilizado em um ambiente sobre o qual a arquitetura foi instanciada.

5.1 Caracterização do cenário de ataque

Para demonstrar que a linguagem e a arquitetura apresentadas nessa dissertação são adequadas para realizar a especificação e a detecção de ataques distribuídos e de múltiplas etapas, optou-se por modelar um cenário de ataque descrito em um trabalho realizado pelo *MIT Lincoln Laboratory*.

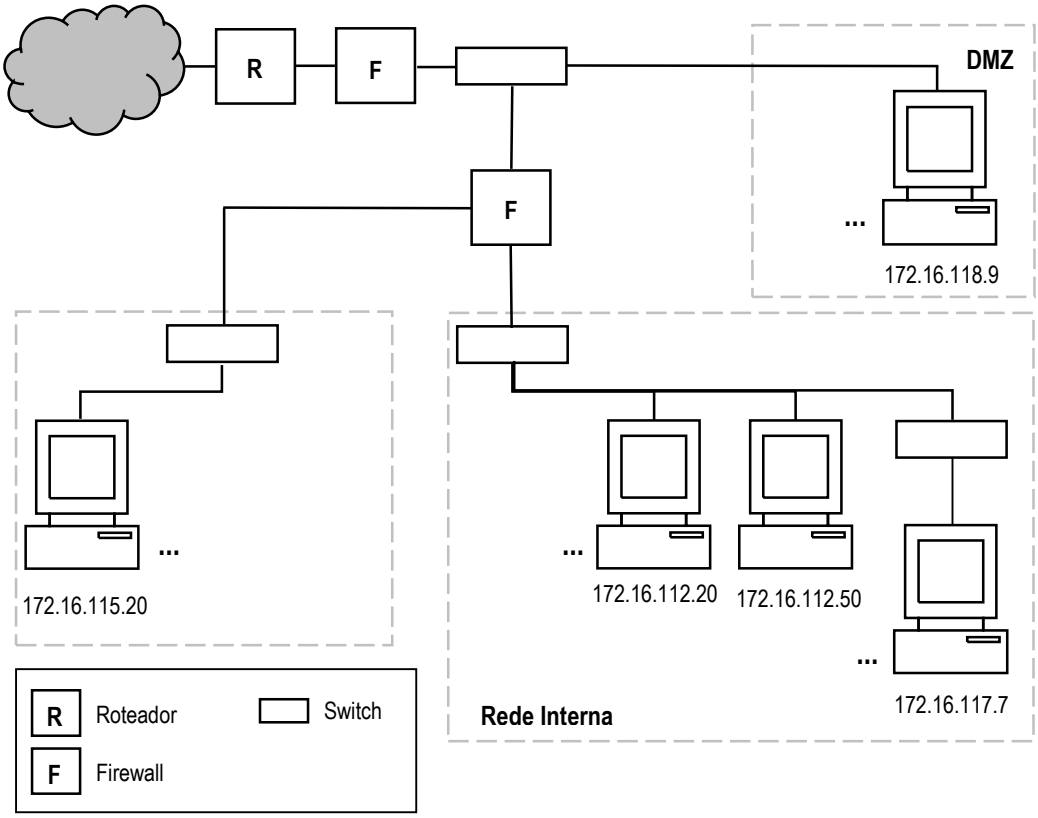

Figura 5.1 – Estrutura de rede

A Figura 5.1 apresenta a estrutura de rede utilizada no MIT para realização do ataque. As estações 172.16.112.20, 172.16.112.50 e 172.16.115.20 serão comprometidas e utilizadas para atacar a máquina 172.16.118.9. Todas as etapas do ataque descrito seguir foram executados a partir do *host* 172.16.117.7.

O cenário, denominado *Lincoln Laboratory Distributed Denial of Service* – LLDOS [Haines, 2000], é agrupado em cinco etapas.

1. sondagem de *hosts* ativos: o objetivo dessa fase é identificar quais os *hosts* que estão ativos na rede. Nessa etapa, as redes 172.16.112.0/24, 172.16.114.0/24 e 172.16.115.0/24 foram alvo de varreduras de portas;
2. sondagem de *hosts* executando *sadmind*: nessa etapa os *hosts* descobertos na fase anterior, são alvos de uma nova sondagem de portas, agora com o objetivo de identificar quais desses *hosts* estão executando a ferramenta de administração remota denominada *sadmind*. Como resultado dessa etapa do ataque, foram identificadas as estações 172.16.115.20, 172.16.112.20 e 172.16.112.50;
3. exploração das vulnerabilidades do serviço *sadmind*: nessa fase o atacante executa o exploit *sadmind Remote-to-Root* com diferente parâmetros contra cada um dos *hosts* identificados na etapa 2. O objetivo é executar remotamente comandos com privilégios do usuário *root*. Esta etapa encerra com a criação de uma nova conta de usuário em cada um dos *hosts* comprometidos.
4. instalação do software *Mstream DDoS*: essa fase tem início no instante em que o atacante conecta-se (via *telnet* ou *ssh*) aos *hosts* nos quais foram criados novos usuários. Em seguida, utiliza o comando *rcp* para copiar os arquivos binários do software *Mstream DDoS* para esses *hosts*. Após realizar a instalação do software em cada um dos *hosts* comprometidos, o atacante utiliza o comando *rsh* para inicializar os serviços instalados.
5. início do ataque: essa é a fase final, na qual o atacante envia um comando para cada um dos *hosts* comprometidos e, então, inicia o ataque. O ataque *Mstream DDoS* consiste do envio de uma grande quantidade de requisições para o *host* alvo (172.16.118.9).

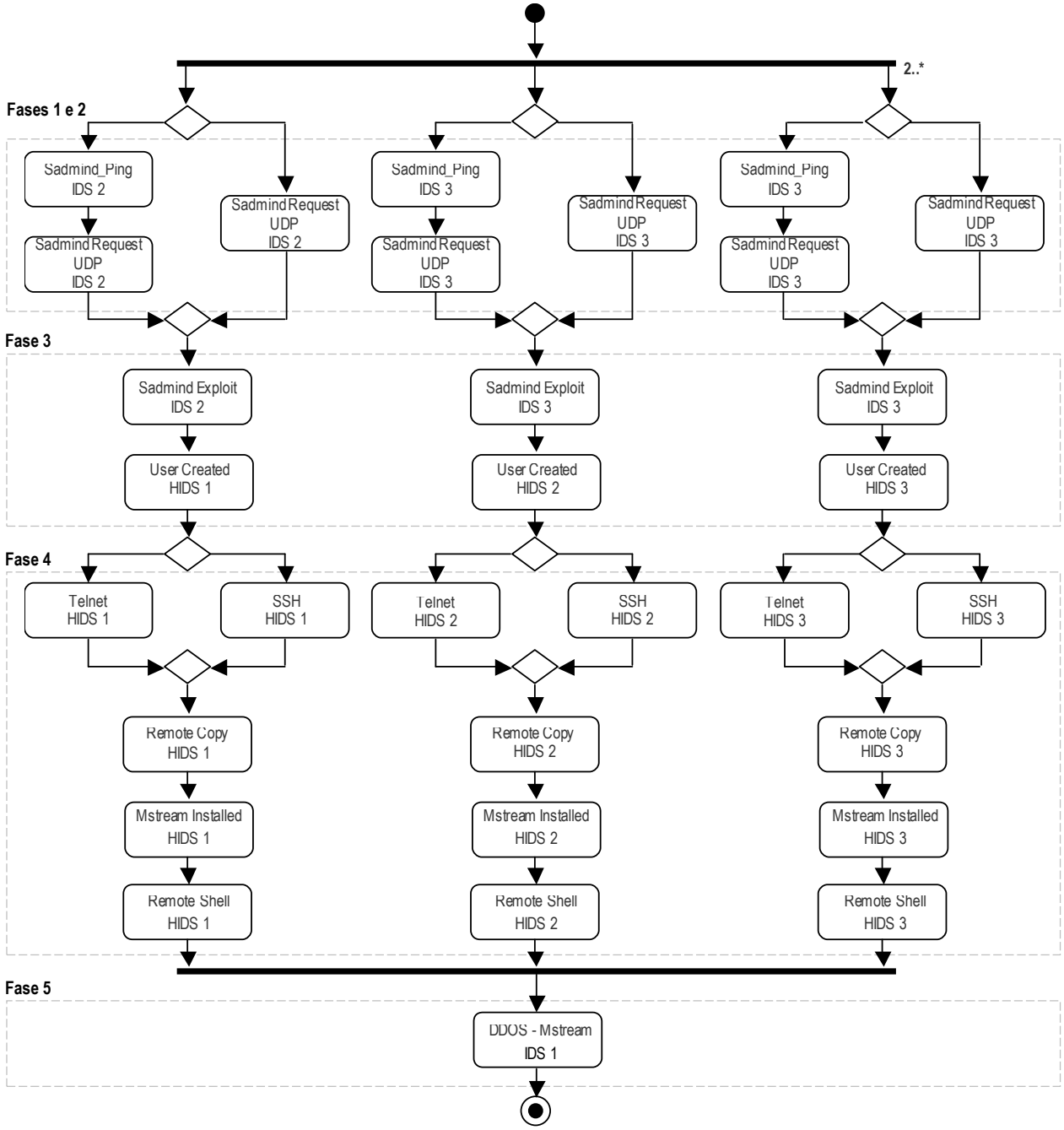

Figura 5.2 – Representação gráfica do cenário LLDOS

A Figura 5.2 ilustra a representação gráfica (G-MADL) do cenário de ataque recém apresentado. A primeira etapa desse ataque (sondagem de *hosts* ativos) é identificada a partir do evento Sadmin_Ping. Já a segunda etapa (sondagem de *hosts* executando *sadmind*) quando identificada gera o evento Sadmin Request UDP. Durante a terceira etapa (exploração das vulnerabilidades do serviço *sadmind*) são observados dois eventos: Sadmin Exploit e User Created A quarta etapa do ataque em questão (instalação do software *Mstream DDoS*) é identificada a partir da ocorrência dos seguintes eventos: Telnet ou SSH, Remote Copy, Mstream Installed e Remote Shell.

A última etapa desse cenário de ataque corresponde ao evento DDOS Mstream. A especificação textual, em T-MADL, correspondente ao cenário ilustrado na Figura 5.1 pode ser observada no Anexo A.

5.2 Instanciação da arquitetura

Foi reproduzido uma infra-estrutura semelhante aquela apresentada anteriormente (Figura 5.1). Sobre esse ambiente foi instanciado a arquitetura, ilustrada na Figura 5.3, através da instalação e configuração da estação de gerenciamento, do serviço de detecção e dos serviços de notificação.

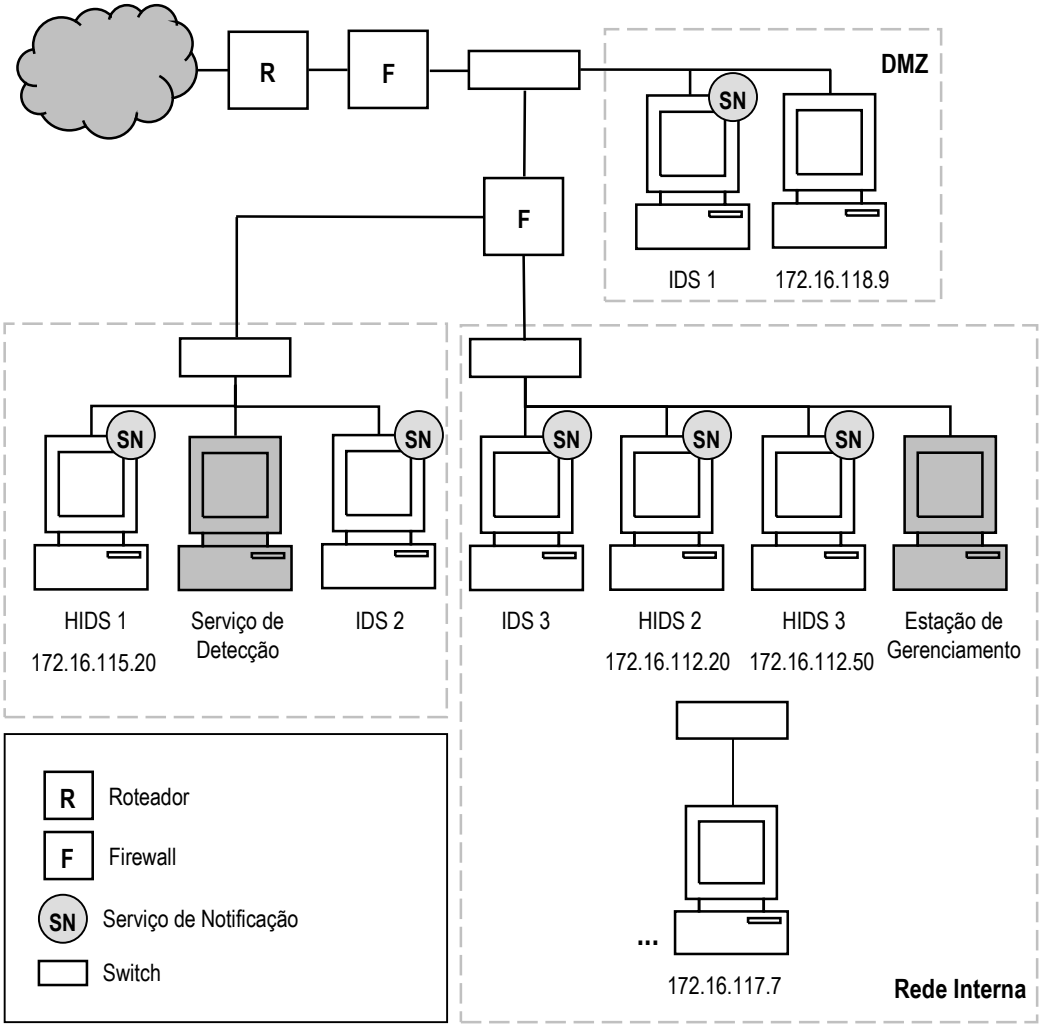

Figura 5.3 – Instanciação da arquitetura para avaliação experimental

A estação de gerenciamento e o serviço de detecção foram instalados em segmentos de rede distintos. Já os serviços de notificação foram posicionados junto aos sistemas de detecção de intrusão (IDS) e monitoração local (HIDS). Posteriormente, cada um desses serviços

foi iniciado, estando assim apto a receber subscrições e monitorar eventos. Em seguida, o cenário de ataque LLDOS foi instanciado.

As subscrições realizadas para detecção de cada uma das fases e a identificação dos serviços responsáveis pelo monitoraramento das mesmas são apresentadas na Tabela 5.1. Por exemplo, para detectar a primeira atividade (Sadmind Exploit) da fase 3 do ataque em questão foram realizadas subscrições para os serviços de notificação IDS 1, IDS 2 e IDS 3.

Tabela 5.1 – Relação das subscrições realizadas e dos respectivos serviços de notificação

Fases	Eventos Observados	Serviços de Notificação
Fase 1	Sadmind_Ping	IDS 1, IDS 2 e IDS 3
Fase 2	Sadmind Request UDP	IDS 1, IDS 2 e IDS 3
Fase 3	Sadmind Exploit	IDS 1, IDS 2 e IDS 3
	User Created	HIDS 1, HIDS 2 e HIDS 3
Fase 4	Telnet	HIDS 1, HIDS 2 e HIDS 3
	SSH	HIDS 1, HIDS 2 e HIDS 3
	Remote Copy	HIDS 1, HIDS 2 e HIDS 3
	Mstream Installed	HIDS 1, HIDS 2 e HIDS 3
	Remote Shell	HIDS 1, HIDS 2 e HIDS 3
Fase 5	DDOS Mstream	IDS 1

Após a instanciação da arquitetura o ataque foi reproduzido nesse ambiente da seguinte forma. A partir da estação 172.16.117.7 foi executado um *script* que se conecta a cada uma das estações envolvidas e gera os alertas esperados em cada uma das cinco etapas previstas. Os alertas referentes aos eventos descritos na Tabela 5.1 foram sinteticamente gerados nos serviços de notificação envolvidos. Esses alertas foram gerados de acordo com a ordem cronológica esperada para esse ataque. Ao mesmo tempo foram inseridos de maneira aleatória alertas para eventos que não foram subscritos e alertas repetidos. Com isso, foi possível observar o comportamento dos serviços de notificação e também do motor de detecção.

Assim que se iniciou a reprodução do ataque, os serviços de notificação passaram a comunicar o serviço de detecção. Para que fosse possível acompanhar visualmente o processo de identificação de cada uma das fases do ataque em questão, foi desenvolvido uma pequena aplicação *web*, cuja interface de monitoração ilustrada na Figura 5.4 permite, por meio da

notação gráfica do cenário monitorado, acompanhar o andamento da seqüência de eventos observada pelo motor de detecção.

Através da Figura 5.4 observa-se que a estação 172.16.112.20 está totalmente comprometida, pois foram realizadas quatro das cinco etapas previstas para esse ataque. Na estação 172.16.112.50 se verifica o inicio da quarta fase do ataque. Já o *host* identificado pelo endereço IP 172.16.115.20 encontra-se sob ataque do exploit sadmind.

Após o término do experimento, todas as etapas do ataque LLDOS foram detectadas conforme o previsto. Aqueles eventos que foram notificados, mas não representavam atividades observadas no cenário monitorado, foram devidamente interpretados pelo motor de detecção. O mesmo ocorreu quando foram notificados eventos em uma ordem diferente daquela definida na especificação desse ataque.

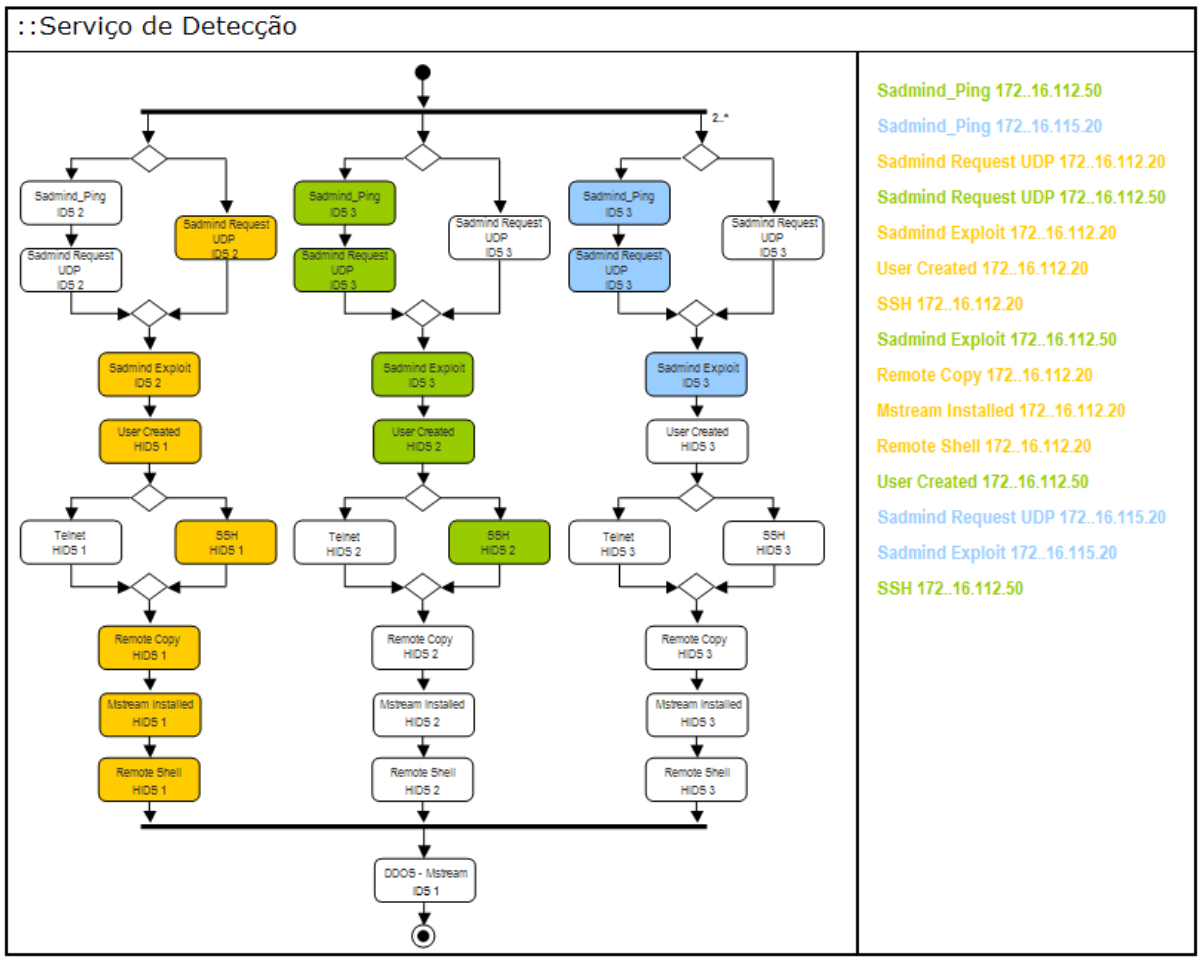

Figura 5.4 – Monitoração do ataque LLDOS

5.3 Considerações parciais

O Capítulo 5 descreveu como foi realizado o experimento de avaliação da linguagem e da arquitetura prospotas. Em particular foram discutidos: as etapas do ataque utilizado, a instanciação da arquitetura de detecção e a forma como cada etapa do ataque foi reproduzida durante os testes.

Ao finalizar o Capítulo demonstrou-se que os resultados obtidos permitem afirmar que a tanto a linguagem MADL quanto a arquitetura proposta (vide Capítulo 4) são adequadas para realizara detecção do tipo de ataque empregado na avaliação.

6 Conclusões e Trabalhos Futuros

O desenvolvimento de arquiteturas de detecção de intrusão que possibilitem a monitoração e a identificação de ataques distribuídos e de múltiplas etapas é de extrema relevância. As soluções existentes, contudo, são limitadas (a) por não oferecerem uma forma adequada que permita representar e descrever cenários de ataques baseados em múltiplas etapas e (b) por não permitirem correlacionar alertas gerados por diferentes mecanismos de segurança, limitando-se a uma quantidade reduzida de sensores.

Esta dissertação propôs uma linguagem para representação e especificação de ataques e uma arquitetura baseada em *web services* para detecção de cenários de ataque distribuídos e de múltiplas etapas. A linguagem proposta, denominada *Multistage Attack Description Language*, possui uma notação gráfica para representação visual e em alto nível do ataque e uma notação textual, baseada em XML, que permite a especificação detalhada do mesmo. Já a arquitetura oferece um mecanismo uniforme para comunicação com diferentes serviços de segurança, que possibilita subscrever pelos eventos que constituem os cenários especificados, detectar a sua ocorrência e acompanhar a evolução desses cenários.

Ao longo do processo de especificação da linguagem MADL, algumas experiências foram realizadas com o objetivo de verificar a possibilidade de modelar diferentes tipos de ataques. Após a realização dessa etapa observou-se que a linguagem proposta demonstrou-se flexível o bastante para a especificação dos fluxos de atividades que constituíram os cenários de ataques modelados.

Outra importante contribuição dessa dissertação diz respeito à incorporação do conceito de *web services*, já disseminada em outras áreas, por exemplo computação em grade e gerência de redes, a uma arquitetura de detecção de intrusão. A utilização de *web services* possibilitou a definição de uma maneira uniforme para se obter informações geradas por diferentes mecanismos de segurança, localizados em domínios administrativos distintos. Além disso, oferece a possibilidade de agregar novos recursos à arquitetura por meio da criação de interfaces *web* para novos serviços de segurança.

A avaliação realizada utilizou como estudo de caso um ataque proposto pelo MIT e que, nos últimos anos, tem sido empregado na avaliação de diversos trabalhos na área de

detecção de intrusão. O resultado obtido nessa avaliação apesar de não ser totalmente conclusivo é significativo.

Como trabalhos futuros pretende-se (a) realizar a avaliação da arquitetura mediante outros tippos de ataques, por exemplo, *worms* e tentativas de roubo de informações armazenadas em servidores *web*, (b) desenvolver uma interface gráfica para a aplicação de gerenciamento que facilite a especificação e instanciação dos cenários de ataque e (c) verificar a viabilidade de incoporar à arquitetura proposta mecanismos que garantam autenticação e integridade das mensagens de notificação e que estejam de acordo com o padrão XML Security [Lautenbach et al., 2005]

Anexo A Especificação T-MADL para o ataque LLDOS

Este anexo apresenta a especificação *Textual* MADL para o ataque utilizado na avaliação experimental (Capítulo 5).

```
1  <?xml version=”1.0” ?>
2  <scenario code="77" xmlns=”http://www.w3.org/2001/XMLSchema-instance” xsi=”schema_madl”>
3      <catalogue>
4        <description> Especificação do ataque LLDOS </description>
5        <author> Leonardo Lemes Fagundes </author>
6        <date> 01/12/2005 </date>
7      <catalogue>
8      <sequence>
9        <parallel min=”2”>
10         <sequence>
11           <choice>
12             <sequence>
13               <event name=”Sadmind_Ping” ip_dst=”172.16.112.20” />
14               <event name=”Sadmind Request” ip_dst=”172.16.112.20” />
15             </sequence>
16               <event name=”Sadmind Request” ip_dst=”172.16.112.20” />
17           </choice>
18           <event name=”Sadmind Exploit” ip_dst=”172.16.112.20” />
19           <event name=”User Created” ip_src=”172.16.112.20” />
20           <choice>
21             <event name=”Telnet” ip_dst=”172.16.112.20” />
22             <event name=”SSH” ip_dst=”172.16.112.20” />
23           </choice>
24           <event name=”Remote Copy” ip_src=”172.16.112.20” />
25           <event name=”Mstream Installed” ip_src=”172.16.112.20” />
26           <event name=”Remote Shell” ip_dst=”172.16.112.20” />
27         </sequence >
28         <sequence>
29           <choice>
30             <sequence>
31               <event name=”Sadmind_Ping” ip_dst=”172.16.112.50” />
32               <event name=”Sadmind Request” ip_dst=”172.16.112.50” />
33             </sequence>
34               <event name=”Sadmind Request” ip_dst=”172.16.112.50” />
35           </choice>
35           <event name=”Sadmind Exploit” ip_dst=”172.16.112.50” />
37           <event name=”User Created” ip_src=”172.16.112.50” />
38           <choice>
39             <event name=”Telnet” ip_dst=”172.16.112.50” />
```

```
40                <event name=”SSH” ip_dst=”172.16.112.50” />
41              </choice>
42              <event name=”Remote Copy” ip_src=”172.16.112.50” />
43              <event name=”Mstream Installed” ip_src=”172.16.112.50” />
44              <event name=”Remote Shell” ip_dst=”172.16.112.50” />
45            </sequence >
46            <sequence>
47              <choice>
48                <sequence>
49                  <event name=”Sadmind_Ping” ip_dst=”172.16.115.20” />
50                  <event name=”Sadmind Request” ip_dst=”172.16.115.20” />
51                </sequence>
52                  <event name=”Sadmind Request” ip_dst=”172.16.115.20” />
53              </choice>
54              <event name=”Sadmind Exploit” ip_dst=”172.16.115.20” />
55              <event name=”User Created” ip_src=”172.16.115.20” />
56              <choice>
57                <event name=”Telnet” ip_dst=”172.16.115.20” />
58                <event name=”SSH” ip_dst=”172.16.115.20” />
59              </choice>
60              <event name=”Remote Copy” ip_src=”172.16.115.20” />
61              <event name=”Mstream Installed” ip_src=”172.16.115.20” />
62              <event name=”Remote Shell” ip_dst=”172.16.115.20” />
63            </sequence >
64          </parallel>
65          <event name=”DDOS Mstream ip_dst=”172.16.118.9” />
66        </sequence>
67  </scenario>
```

Anexo B Esquema XML da linguagem MADL

Este anexo apresenta o esquema criado para a linguagem MADL.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4      <xs:import namespace="http://www.w3.org/2001/XMLSchema-instance" schemaLocation="xsi.tmp\schema_v2" />
5      <xs:element name="scenario">
6          <xs:complexType>
6              <xs:sequence>
7                  <xs:element ref="sequence" />
8              </xs:sequence>
9              <xs:attribute ref="xsi:noNamespaceSchemaLocation" use="required" />
10         </xs:complexType>
11     </xs:element>
12     <xs:element name="sequence">
13         <xs:complexType>
14             <xs:sequence>
15                 <xs:element minOccurs="0" ref="parallel" />
16                 <xs:choice maxOccurs="unbounded">
17                     <xs:element ref="event" />
18                     <xs:element ref="choice" />
19                 </xs:choice>
20             </xs:sequence>
21         </xs:complexType>
22     </xs:element>
23     <xs:element name="parallel">
24         <xs:complexType>
25             <xs:sequence>
26                 <xs:element maxOccurs="unbounded" ref="sequence" />
27             </xs:sequence>
28             <xs:attribute name="min" use="required" type="xs:integer" />
29         </xs:complexType>
30     </xs:element>
31     <xs:element name="choice">
32         <xs:complexType>
33             <xs:sequence>
34                 <xs:element minOccurs="0" ref="sequence" />
35                 <xs:element maxOccurs="unbounded" ref="event" />
35             </xs:sequence>
37         </xs:complexType>
38     </xs:element>
39     <xs:element name="event">
40         <xs:complexType>
```

```
41                <xs:attribute name="ip_dst" />
42                <xs:attribute name="ip_src" type="xs:NMTOKEN" />
43                <xs:attribute name="name" use="required" />
44            </xs:complexType>
45        </xs:element>
46    </xs:schema>
```

Referências Bibliográficas

[Baratloo, 2000] Baratloo, A., Tsai, T. and Singh, N.: **Transparent Run-Time Defense Against Stack Smashing Attacks.** Proceedings of the USENIX Annual Technical Conference (2000).

[Lautenbach et al., 2005] Lautenbach, B., Mattheus, A., Tomic, M., Mullan, S.: **The Apache XML Project**. Disponível em: http://xml.apache.org/security/ (último acesso: janeiro de 2006).

[Booth et al., 2003] Booth, D., Haas, H., McCabe, F., Champion, M., Ferris, C. and Orchard, D.: **Web Service Architecture**. W3C Working Draft. Disponível em: http://www.w3.org/TR/2003/WD-ws-arch-20030808 (último acesso: junho de 2005).

[Booth et al., 2005] Booth, D., Haas, H., McCabe, F., Champion, M., Ferris, C. and Orchard, D.: **Web Services Description Language (WSDL) Version 2.0 Part 0. Working Draft**. Disponível em: http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803 (último acesso: dezembro de 2005).

[Campello, 2001] Campello, R.: **Sistemas de Detecção de Intrusão**. 19º Simpósio Brasileiro de Redes de Computadores. UFSC: Florianópolis (2001).

[Cerami, 2002] Cerami, E. **Web Services Essentials – Distributed Applications with XML-RPC, SOAP, UDDI & WSDL**. O'Reilly (2002).

[Cert, 2001] CERT/CC. **Advisory CA-2001-26 Nimda Worm**. (2001).

[Cuppens, 2002] Cuppens, F. and Miége, A.: **Alert Correlation in a Cooperative Intrusion Detection Framework**. Proceedings of the 2002 IEEE Symposium on Security and Privacy, 187 – 200 (2002).

[Cheung, 2003] Cheung, S., Lindqvist, U. and Fong, W. M.: **Modeling Multistep Cyber Attacks for Scenario Recognition**. DARPA Information

Survivability Conference and Exposition (DISCEX III), 284 – 292 (2003).

[Chung, 2003] Chung, J.: **Web Services Computing: Advancing Software Interoperability**. Publishe by the IEEE Computer Society, vol. 7, n°. 6, 12-15 (2003).

[Clement et al., 2004] Clement, L. et al. (2004) **Universal Description, Discovery and Integration (UDDI). Committee Draft**. http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm (último acesso: dezembro de 2005).

[Debar, 2004] Debar, H., Curry, D. and Feinstein, B.: **The Intrusion Detection Message Exchange**. IETF Intrusion Detection Exchange Format Working Group, Internet Draft. (2004).

[Debar, 2001] Debar, H. and Wespi, A.: **Aggregation and Correlation of Intrusion-Detection Alerts**. Lecture Notes in Computer Science, Proceedings RAID, 85 – 103 (2001).

[Durst et al., 1999] Durst, R., Champion, T., Witten, B., Miller, E. and Spagnuolo, L.: **Testing and Evaluating Computer Intrusion Detection Systems**. Communications of ACM, vol. 42, n°. 7, 53 – 61 (1999).

[Eckmann, 2002] Eckmann, T. S., Vigna, G., Kemmerer, A. R.: **STATL: An Attack Language for State-based Intrusion Detection**. Journal of Computer Security, vol. 10, n°. 2, 71-104 (2002).

[Fagundes, 2004] Fagundes, L. L. and Gaspary, P. L.: **Network-Based Intrusion Detection Systems Evaluation Through a Short Term Experimental Script**. ICETE, vol. 2, n°. 1, 54 - 60 (2004).

[Ferris, 2003] Ferris, C. and Farrel, J.: **What Are Web Services? Communication of the ACM**, vol. 46, no. 6, p. 31 (2003).

[Fowler, 2003] Fowler, M.: **UML Distilled: A Brief Guide to the Standard Object Modeling Language**. Addison-Wesley (2003).

[Fuller, 2003] Fuller, J., Fuecks, H., Egervari, K, Waters, B., Solin, D.,

Stephens, J., Reynolds, L.: **Professional PHP Web Services**. Wrox (2003).

[Gudgin et al., 2003] Gudgin, M. et al. (2003) **Simple Object Access Protocol (SOAP)**. **Version 1.2 Part 1: Messaging Protocol. Committee Specification**. Disponível em: http://www.w3.org/TR/2003/REC-soap12-part1-20030624/ (último acesso: junho de 2005).

[Graham et al., 2004] Graham, S., Nib lett, P., Chappell, D., Lewis, A., Nagaratnam, N., Parikh, J., Patil, S., Samdarshi, S., Tuecke, S., Vambenepe, W. and Weihl, B.: **Web Services Notification**. Disponível em: http://www.oasis-open.org/committees/documents.php (último acesso: junho de 2005).

[Haines, 2000] Haines, J.: **2000 DARPA Intrusion Detection Scenario Specific DataSets** Disponível em: http://www.ll.mit.edu/IST/ideval/data/2000/2000_data_index.html (último acesso: dezembro de 2005).

[Mauch, 1998] Mauch, J.: **Sysmon Documentation.** Disponível em: http://www.sysmon.org/docs.html (último acesso: junho de 2005).

[Meier, 2002] Meier, M; Bischof, N; Holz, T.: **SHEDEL: A Simple Hierarchical Event Description Language for Specifying Attack Signatures**. International Conference on Information Security, 559 – 572 (2002).

[Medeiros, 2004] Medeiros, Ernani. **Desenvolvendo software com UML 2.0: definitivo**. São Paulo: Pearson, 264 p. (2004).

[Michel, 2001] Michel, C.; Me, L.: **ADeLe: an Attack Description Language for Knowledge-based Intrusion Detection**. Proceedings of International Conference on Information Security, Kluwer, (2001).

[Mounji, 1997] Mounji, A.: **Languages and Tools for Rule-Based Distributed Intrusion Detection**, PhD thesis, University of Namur, Belgium, (1997).

[Mutaf, 1999] Mutaf, P.: **Defending against a Denial-of-Service Attack on TCP**. RAID (1999).

[Northcutt, 2000] Northcutt, S.: **Como Detectar Invasão em Rede – Um Guia para Analistas**. Editora Ciência Moderna (2000).

[Ptacek, 1998] Ptacek, T. H. and Newsham, T. N.: **Insertion, Evasion and Deny of Service: Eluding network intrusion detection**. (1998).

[Porras et al., 2002] Porras, A. P., Fong, W. M., Valdes, A.: **A Mission-Impact-Based Approach to INFOSEC Alarm Correlation**. Lecture Notes in Computer Science, Proceedings of RAID, 95 – 114 (2002).

[Roesch, 1999] Roesch, M.: **Snort – Lightweight Intrusion Detection for Networks**. Proceedings of USENIX LISA (1999).

[Templeton, 2001] Templeton, S., Levitt, Karl.: **A Requires/Provides Model for Computer Attacks.** Proceedings of the 2000 workshop on New security paradigms, 31-38 (2001)

[Valdes, 2001] Valdes, A. and Skinner, K.: **Probabilistic Alert Correlation**. Proceedings RAID, 54 – 68 (2001).

[Walsh, 2002] Walsh, A. E.: **UDDI, SOAP and WSDL The Web Services Specification Reference Book.** Prentice Hall Books (2002).

[Wu et al., 2003] Wu, Y., Foo, B., Mey, Y. and Bagchi, S.: **Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS.** Proceeding of 19th Annual Computer Security Applications Conference, (2003).