Paulo Fernando da Silva

EXTENSÃO DO MODELO IDWG PARA DETECÇÃO DE INTRUSÃO EM AMBIENTES COMPUTACIONAIS

Florianópolis – SC
2004

UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Paulo Fernando da Silva

EXTENSÃO DO MODELO IDWG PARA DETECÇÃO DE INTRUSÃO EM AMBIENTES COMPUTACIONAIS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Carlos Becker Westphall
Orientador

Florianópolis, Abril de 2004

EXTENSÃO DO MODELO IDWG PARA DETECÇÃO DE INTRUSÃO EM AMBIENTES COMPUTACIONAIS

Paulo Fernando da Silva

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração Sistemas de Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Raul S. Wazlawick, Dr.
Coordenador

Banca Examinadora:

Prof. Carlos Becker Westphall, Dr.
Orientador

Prof. Carlos Barros Montez, Dr.

Prof. Mário Antonio Ribeiro Dantas, Dr.

Prof. Vitório Bruno Mazzola, Dr.

Dedico este trabalho aos meus pais, Oscar e Diná, e a minha noiva Viviane, pelo incentivo recebido durante o desenvolvimento do mesmo.

Agredeço em especial ao Prof. Dr. Carlos Becker Westphall,
pelas contribuições prestadas como meu orientador
no desenvolvimento deste trabalho.
Agradeço também a todos os professores deste programa
de pós-gradução, por contribuírem na minha formação
durante o período em que estive nesta instituição.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIAÇÕES

ABNF	*Augmented Backus-Naur Form*
ATM	*Asynchronous Transfer Mode*
BEEP	*Block Extensible Exchange Protocol*
CGI	*Common Gateway Interface*
CIDF	*Common Intrusion Detection Framework*
CISL	*Common Intrusion Specification Language*
DOM	*Document Objetc Model*
DoS	*Denial of Service*
DTD	*Document Type Definition*
HTTP	*Hiper Text Transfer Protocol*
IANA	*Internet Assigned Numbers Authority*
ICMP	*Internet Control Message Protocol*
IDMEF	*Intrusion Detection Message Exchange Protocol*
IDREF	*Intrusion Detection Response Exchange Format*
IDS	*Intrusion Detection System*
IDWG	*Intrusion Detection Working Group*
IDXP	*Intrusion Detection Exchange Protocol*
IETF	*Internet Engineering Task Force*
IP	*Internet Protocol*
IPv4	*Internet Protocol version 4*
IPv6	*Internet Protocol version 6*
MAC	*Media Access Control*
MIB	*Management Information Base*
MIME	*Multipurpose Internet Mail Extensions*
SASL	*Simple Authentication and Security Layer*
SMI	*Structured of Management Information*
SNMP	*Simple Network Management Protocol*
SNMPv3	*Simple Network Management Protocol version 3*
TCP	*Transport Control Protocol*
TLS	*Transport Layer Security*
UML	*Unified Modeling Language*
URI	*Uniform Resource Identifier*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

RESUMO

Este trabalho propõe um modelo de ambiente de detecção de intrusão que permita a interoperabilidade entre sistemas de detecção de intrusão de diferentes tipos e fabricantes. O modelo proposto descreve a arquitetura do ambiente, os protocolos de comunicação a serem utilizados e o formato dos dados trocados pelos componentes do ambiente.

O desenvolvimento do modelo proposto tomou por base os trabalhos realizados pelo grupo IDWG relacionados com a interoperabilidade entre os sistemas de detecção de intrusão. No modelo proposto estendeu-se a arquitetura de detecção de intrusão desenvolvida pelo grupo IDWG e utilizou-se os protocolo BEEP e IDXP para a troca de informações, sendo que o protocolo IDXP sofreu pequenas alterações em sua especificação. Na comunicação de alertas utilizou-se o modelo de dados IDMEF.

O modelo IDWG foi extendido de forma a suportar o tratamento de respostas aos alertas, para tanto foi necessário criar novos componentes na arquitetura e desenvolver um novo modelo de dados.

Para validação do modelo proposto foram implementados componentes responsáveis pelo envio de alertas, gerenciamento de alertas e respostas e, recepção e tratamento de respostas. Com estes componentes, juntamente com o IDS Snort, foi montado um ambiente de validação correspondente ao modelo proposto, onde foram realizados testes.

Como resultado deste trabalho temos um ambiente onde é possível gerenciar alertas e respostas de diferentes tipos e fabricantes de IDSs, atuando no auxílio à segurança das redes de computadores.

ABSTRACT

This dissertation proposes an intrusion detection enviroment model which allows interoperability between intrusion detection systems of different kinds and manufactures. The proposes model describes the enviroment architecture, the communication protocols to be used and the data format exchanged by the enviroment's components.

The development of the proposed model took as a basis the works carried out by the IDWG group related to the interoperability between the intrusion detection systems. In the proposed model the intrusion detection architecture developed by IDWG group has been improved. Also, the BEEP and IDXP protocols have been used for information exchange. The IDXP protocol has suffered minor alterations on its specification. The IDMEF data model was used for the communication of alerts.

The IDWG model has been improved in order to support the treatment of responses to the alerts. It was necessary to create new components in the architecture, as well as to develop a new data model to achieve that goal.

To validate the proposed model components responsible for the sending and management of alerts and responses were implemented, as well as components responsible for the reception and treatment of such responses with those, together with the IDS Snort, a validation enviroment equivalent to the proposed model was set up. In such enviroment, tests were carried out.

As a result of the present work, an enviroment where if is posible to manage alertas and responses of different kinds and manufactures of IDSs is made available. Therefore, this model has an iportant role to play in helping improve computer nerwork security.

1 INTRODUÇÃO

1.1 Considerações Iniciais

Detecção de intrusão é o processo de identificar e responder a atividades maliciosas dirigidas a computadores ou recursos de redes de computadores, este conceito é exposto por [AMOROSO 1999]. [PROCTOR 2001] conceitua detecção de intrusão como sendo a tarefa de coletar informações de uma variedade de fontes, sistemas ou redes e analisá-las buscando sinais de intrusão ou mau uso. Diversos são os conceitos de detecção de intrusão encontrados na literatura, alguns deles divergem quanto à abrangência de uma detecção de intrusão. O conceito apresentado por [AMOROSO 1999] considera as respostas a uma intrusão como parte integrante da detecção de intrusão, enquanto o conceito apresentado por [PROCTOR 2001] não considera as respostas.

Sistemas de detecção de intrusão (*Intrusion Detection Systems* – IDSs) são ferramentas utilizadas na segurança de redes de computadores, estas ferramentas são utilizadas na tentativa de identificar e rastrear ataques às redes de computadores [NIST 2001].

O funcionamento de um IDS consiste na monitoração de eventos da rede ou *host* em que está instalado, tentando identificar se estes eventos correspondem a informações que o IDS reconheça como um ataque. Se o IDS encontrar uma seqüência de eventos no sistema que possa ser considerada um ataque, então um alerta será emitido informando sobre o possível ataque em andamento. Em alguns IDSs existe a possibilidade de serem enviadas respostas ou contra-medidas a um ataque em andamento, estas respostas têm o objetivo de conter o ataque. Como exemplos de respostas ou contra-medidas temos o encerramento da conexão ou do processo que está causando o ataque, a alteração de permissões de arquivos ou da rede e a reconfiguração de *firewalls*.

Diversas técnicas e arquiteturas são aplicadas aos IDSs na busca de uma maior eficiência na detecção de ataques. Dentro desta diversidade podemos classificar os IDSs das seguintes maneiras:

- quanto ao método de detecção: análise de assinaturas, sistemas adaptativos ou análise estatística;
- quanto ao alvo: baseado em *host*, baseado em rede ou híbrido;

- quanto à localização: centralizado, distribuído ou hierárquico.

O método de detecção de análise de assinaturas consiste na comparação das informações que são monitoradas pelo IDS com uma base de assinaturas, esta base contém a definição de quais informações são consideradas ataques em potencial. Nos sistemas adaptativos o IDS se adapta ao comportamento do seu ambiente de funcionamento, situações que não estejam de acordo com o funcionamento normal do ambiente podem ser consideradas ataques em potencial. Na análise estatística são montados modelos estatísticos do ambiente em que o IDS está monitorando, informações que não estejam de acordo com o modelo estatístico são consideradas ataques em potencial.

Um IDS baseado em *host* monitora as informações apenas do *host* em que está instalado (CPU, processos, usuários, etc), enquanto que um IDS baseado em rede monitora as informações da rede em que está instalado (protocolos). Os IDSs de alvo híbrido combinam as informações dos IDSs baseados em *host* e rede para realizar a tarefa de detecção de intrusão.

A classificação quanto à localização reflete a distribuição dos módulos do IDS pelo ambiente, sendo assim, uma arquitetura de localização centralizada ocorre quando todos os módulos do IDS estão localizados em um único ponto do ambiente (por exemplo, em um único *host* da rede). Na arquitetura distribuída, os módulos do IDS atuam espalhados pelos diversos *hosts* do ambiente. A distribuição parcial dos componentes do IDS caracteriza a localização hierárquica.

Existem diversos sistemas de detecção de intrusão disponíveis no mercado, desenvolvidos por diversos fabricantes e abrangendo todas as classificações apresentadas acima. O Anexo 1 apresenta uma tabela com diversos IDSs e algumas de suas característica.

Os IDSs existentes no mercado não possuem um padrão quanto à sua implementação. Alguns deles possuem vários módulos e outros possuem apenas um módulo que realiza todo serviço. A forma como as informações são tratadas dentro do IDS (coleta de informações, testes e comparações, geração de alertas) e a comunicação entre os módulos dos IDSs também não são padronizadas. A interoperabilidade entre os componentes de um IDS e entre os IDSs de diferentes fabricantes é uma necessidade que justifica a padronização da implementação e da comunicação dos mesmos.

O CIDF (*Common Intrusion Detection Framework*) [KAHN 1998] foi uma tentativa de padronização dos sistemas de detecção de intrusão que tem por objetivo definir a estrutura de um IDS. Juntamente com o CIDF, foi desenvolvida uma linguagem para especificação de eventos denominada "*Common Intrusion Specification Language*" (CISL), que visa possibilitar a troca de informações entre os componentes sobre os eventos ocorridos no ambiente.

Os esforço empregado no desenvolvimento do CIDF e da linguagem CISL atingem toda a estrutura de um IDS, com a divisão do mesmo em módulos e a padronização tanto da implementação dos módulos quanto da forma como este módulos irão interagir uns com os outros. A Figura 1 apresenta os componentes da arquitetura CIDF e a interação entre os mesmo.

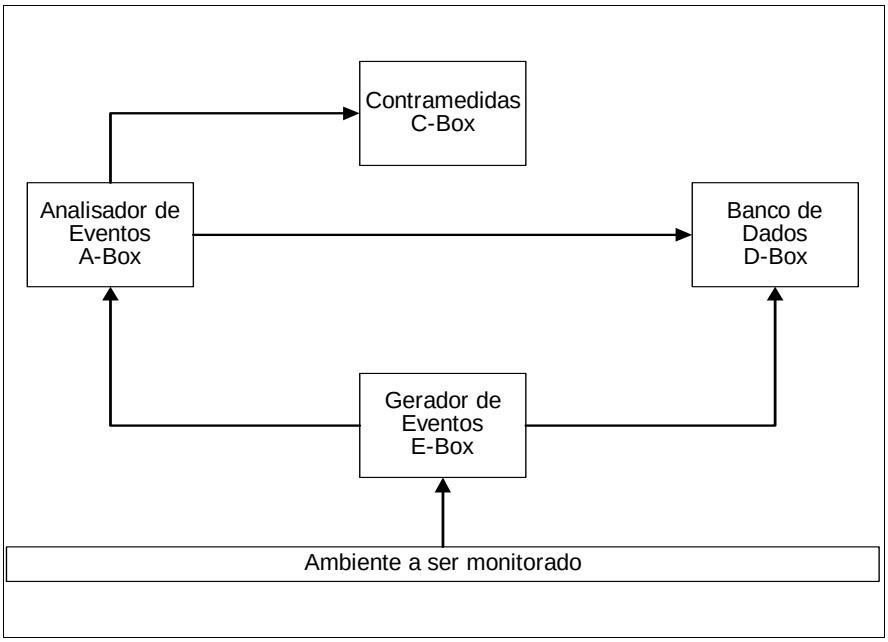

Figura 1 – Componentes do Modelo CIDF.

Conforme apresentado na Figura 1 os componentes da arquitetura CIDF são:

- gerador de eventos (E-Box): responsável por capturar as informações do ambiente e gerar os eventos para o analisador de eventos;
- analisador de eventos (A-Box): é responsável por identificar o que é e o que não é um ataque, de acordo com o método de detecção utilizado;
- banco de dados (D-Box): recebe e armazena informações do gerador de eventos e do analisador de eventos;

- contramedidas (C-Box): é responsável por tomar ações baseadas nos eventos, deve ter a capacidade de comunicar-se com outros IDSs ou até com *firewalls*.

Outro importante trabalho de padronização dos sistemas de detecção de intrusão está sendo desenvolvido pelo grupo IDWG (*Intrusion Detection Working Group*) do IETF (*Internet Engineering Task Force*). O objetivo do IDWG é definir formatos de dados e procedimentos para o compartilhamento de informações em Sistemas de Detecção de Intrusão. O trabalho do IDWG resultou na especificação de um formato para troca de mensagens, o IDMEF (*Intrusion Detection Message Exchange Protocol*). Também foi especificado pelo IDWG um protocolo de comunicação para o transporte das mensagens IDMEF, o IDXP (*Intrusion Detection Exchange Protocol*).

O IDMEF tem a finalidade de ser um formato de dados padrão que sistemas de detecção de intrusão possam utilizar para comunicar alertas. A utilização deste formato permitirá a interoperabilidade entre os sistemas de detecção de intrusão, pois se os diferentes IDSs utilizarem o mesmo padrão de comunicação, poderão trocar informações entre si.

O IDXP é o protocolo que irá prover um canal de comunicação para as trocas de mensagens IDMEF entre os componentes do sistema de detecção de intrusão. Esse protocolo é implementado como um novo perfil do protocolo BEEP (*Block Extensible Exchange Protocol*) e utiliza vários recursos já implementados neste.

A arquitetura proposta pelo IDWG para os sistemas de detecção de intrusão é mais detalhada que a arquitetura proposta pelo CIDF. Tendo em vista que este trabalho foi desenvolvido sob a arquitetura do IDWG, a mesma será detalhada nos próximos capítulos.

Atualmente a maioria dos mecanismos de segurança disponíveis no mercado estão voltados para a prevenção aos ataques. O *firewall* e a criptografia são exemplos deste tipo de mecanismo. Menos representativa é a utilização de mecanismos para detecção de ataques e resposta aos ataques detectados. Como mecanismo para detecção de ataques temos os sistemas de detecção de intrusão, porém grande parte dos sistemas detecção de intrusão não provêm mecanismos de resposta a ataques. Sendo assim, existe uma deficiência de mecanismos de segurança que possibilitem respostas rápidas e eficientes a ataques.

O modelo de sistema de detecção de intrusão proposto pelo IDWG procura definir e padronizar apenas a comunicação dos alertas, este modelo não define a comunicação nem a arquitetura das respostas aos alertas gerados. Para facilitar o gerenciamento e aumentar a segurança do ambiente deve ser possível tomar ações (respostas ou contra-medidas) para um determinado alerta, estas ações também devem seguir um modelo que permita a interoperabilidade dos sistemas, para que em um ambiente distribuído e com vários tipos de IDSs possa ser feito um gerenciamento centralizado dos alertas e de suas respectivas respostas. Isto permitirá o envio de respostas rápidas e eficazes quando o ambiente que está sendo protegido trabalhar com uma considerável diversidade de sistemas de detecção de intrusão.

Com o gerenciamento centralizado dos alertas e suas respectivas contra-medidas, será possível solicitar uma ação a um sistema de um fabricante para um alerta gerado por um sistema de outro fabricante, permitido assim a cooperação total entre os sistemas instalados no ambiente.

Com o modelo de IDS atualmente proposto pelo IDWG apenas o formato dos alertas está especificado, caso o operador do sistema queira tomar ações contra um alertas, terá que fazê-las separadamente em cada sistema de seu ambiente. Isto dificulta o gerenciamento das ações e causa um atraso no envio das mesmas, além de exigir que o operador tenha conhecimentos específicos sobre cada um dos IDSs instalados em seu ambiente.

Este trabalho propõe uma extensão tanto à arquitetura dos IDSs quanto ao protocolo e modelo de comunicação definidos pelo IDWG. Esta extensão deve conter elementos voltados para o gerenciamento, comunicação e tratamento de respostas aos alertas. A proposta apresentada neste trabalho irá adicionar novos componentes à arquitetura do IDS, novas funcionalidades ao protocolo IDXP e novas classes de dados ao modelo IDMEF.

Também será desenvolvido um gerenciador de sistemas de detecção de intrusão que irá trabalhar de acordo com o modelo proposto pelo IDWG e com as extensões propostas neste trabalho. Este gerenciador será utilizado para validar as extensões ao modelo do IDWG propostas neste trabalho.

1.2 Objetivos

1.2.1 Objetivos Gerais

O objetivo geral deste trabalho é propor uma extensão à arquitetura de IDSs, ao protocolo IDXP e ao modelo de dados IDMEF desenvolvidos pelo grupo IDWG do IETF. Esta extensão deve proporcionar o suporte ao envio de respostas aos alertas gerados pelos IDSs.

1.2.2 Objetivos Específicos

Os objetivos específicos deste trabalho são:

- estender a arquitetura de IDSs apresentada pelo IDWG de forma que esta arquitetura suporte respostas aos alertas;
- estender o modelo de dados IDMEF para que o mesmo suporte a comunicação de respostas aos alertas;
- estender o protocolo IDXP para que o mesmo suporte os novos modelos de dados desenvolvidos para a formatação de respostas;
- desenvolver um gerenciador de IDSs que utilize a arquitetura de comunicação apresentada pelo IDWG juntamente com as extensões propostas neste trabalho.

1.3 Metodo*log*ia

O desenvolvimento deste trabalho está dividido em quatro partes:

- fundamentação teórica;
- desenvolvimento do modelo proposto;
- desenvolvimento do gerenciador de IDSs que utiliza o modelo proposto;
- validação do modelo no gerenciador e discussão dos resultados.

A fase de fundamentação teórica proporcionou uma visão global dos sistemas de detecção de intrusão e dos trabalhos desenvolvidos pelo grupo IDWG. Isto proporcionou o desenvolvimento de um modelo que estende os trabalhos deste grupo adicionando novas funcionalidades ao modelo já existente.

Para o desenvolvimento do gerenciador de IDS foi utilizada a linguagem Java. Optou-se por esta linguagem por ser orientada a objetos, como o modelo de dados IDMEF, e por possibilitar a execução em várias plataformas. O desenvolvimento do

gerenciador de IDSs serviu de ferramenta para validação do modelo proposto e para geração dos resultados que são discutidos na última fase do trabalho.

1.4 Trabalhos Correlatos

Tanto [FEINSTEIN 2001] quando em [BUCHHEIM 2001] são apresentados resultados dos trabalhos do grupo IDWG no sentido de desenvolver protocolos para a comunicação entre IDSs.

Em [FEINSTEIN 2001] é apresentada a implementação do GlobalGuard IAP, um protocolo para transmissão de alertas mais simples que o IDXP, neste artigo também é apresentada uma visão geral do formato dos alertas através do modelo IDMEF.

Aspectos da implementação do protocolo IDXP são apresentados em [BUCHHEIM 2001]. No artigo são apresentados os componentes da arquitetura de um IDS e os requisitos que devem ser atendidos por um protocolo para o transporte de informações de intrusão. Em seguida são apresentados os protocolos BEEP e IDXP como uma solução que atende aos requisitos para o transporte de informações de intrusão.

Em [BETSER 2001] é comentado o aprendizado obtido pelo grupo IDWG em cinco anos de pesquisa no desenvolvimento do protocolo IDXP e na comunicação entre sistemas de detecção de intrusão. São resumidamente comentados os problemas relacionados com a detecção de intrusão e os esforços que precederam o IDXP. Posteriormente são apresentadas as lições aprendidas nos cinco anos de pesquisa, que contribuíram para o desenvolvimento do protocolo IDXP.

A importância da abstração em sistemas de detecção de intrusão é comentada em [NING 2001]. Este artigo apresenta um modelo hierárquico que permite a especificação de ataques e de eventos de maneira abstrata em um ambiente de detecção de intrusão distribuído. O modelo utiliza componentes descentralizados, autônomos e cooperativos para proporcionar a abstração. No artigo são citadas das experiências do grupo IDWG com o protocolo IDXP e o modelo IDMEF como exemplos de modelos que proporcionam a abstração em sistemas de detecção de intrusão.

O desenvolvimento de um módulo de detecção de intrusão capaz de correlacionar alertas para simplificar o gerenciamento é apresentado em [CUPPENS 2002]. O módulo *CRIM*, apresentado no artigo, se localiza entre diversos IDSs recebendo e analisando

seus alertas, correlacionando-os e transformando-os em informações mais globais e sintéticas, facilitando assim o gerenciamento do sistema. O módulo de detecção de intrusão apresentado no artigo utiliza o modelo de dados IDMEF como padrão para receber os alertas de diversos tipos de IDSs diferentes. O módulo considera que os IDSs ligados a ele conhecem o modelo IDMEF e lhe enviarão alertas neste formato. Em cima das informações formatadas de acordo com o IDMEF, o módulo realiza análises de forma a correlacionar e simplificar os alertas.

O Artigo [JULISCH 2003] propõe novos métodos de agrupamento de alarmes de detecção de intrusão, com o objetivo de tornar mais fácil por parte do operador, a identificação da causa principal de geração dos alertas. Com isto o operador pode tomar ações mais específicas e eficazes contra o ataque.

Outro trabalho que utiliza o modelo IDMEF é [SUN 2003], este artigo trabalha com IDSs baseados em zonas para redes móveis Ad hoc e propõe mecanismos de colaboração entre os agentes do IDS, como a agregação das informações de alerta de vários agentes. O modelo de alertas apresentado neste artigo está em conformidade com o modelo IDMEF, para facilitar a interoperabilidade entre os agentes.

Ainda relacionado com o modelo IDMEF temos [VIGNA 2001] e [VIGNA 2003], nos quais é apresentado o desenvolvimento do modelo STAT, que permite a criação de novas funcionalidades de detecção de intrusão em forma de módulos. O modelo STAT utiliza o modelo de dados IDMEF na geração de seus alertas. A infra-estrutura de comunicação entre os módulos do modelo STAT também utiliza o modelo IDMEF, pois estes módulos podem ser bastante diferentes entre si com relação aos seus métodos de detecção de intrusão.

Diversos trabalhos apresentam pesquisas e implementações relacionadas com a capacidade dos IDSs responderem aos ataques identificados. O projeto FIDRAN é apresentado em [HESS 2003], trata-se de um modelo flexível para detecção de intrusão e resposta às intrusões detectadas. Este modelo possui mecanismos específicos para execução de respostas capazes de descartar pacotes, reconfigurar *firewalls*, redirecionar o trafego e executar serviços auxiliares.

Em [ZHANG 2003] é apresentada uma arquitetura para detecção de intrusão em redes móveis. A arquitetura apresentada prevê mecanismos de resposta como a

reinicialização do canal de comunicação entre os nós da rede ou a solicitação de re-autenticação de um nó específico.

Os artigos [SCHNACKENBERG 2000], [KOILPILLAI 2000] e [PETKAC 2000] também apresentam propostas de detecção de intrusão com recursos para responder aos ataques detectados.

1.5 Organização do Trabalho

Este trabalho está organizado em 8 capítulos. O capítulo 1 apresenta o contexto em que o trabalho atua, definindo seus objetivos gerais e específicos, metodologia e trabalhos correlatos.

Os capítulos 2, 3 e 4 constituem a fundamentação teórica. O capítulo 2 apresenta a arquitetura de detecção de intrusão desenvolvida pelo grupo IDWG e os requisitos de protocolo de comunicação, formato de mensagens e conteúdo de mensagens que esta arquitetura deve seguir. O capítulo 3 descreve detalhes do funcionamento do protocolo BEEP, que serve de base para o protocolo IDXP. No capítulo 3 também é apresentado o protocolo IDXP desenvolvido pelo grupo IDWG para troca de mensagens entre os sistemas de detecção de intrusão. O capítulo 4 detalha o modelo de dados IDMEF, apresentando a funcionalidade e o relacionamento de suas classes de dados.

Os capítulos 5 e 6 constituem o desenvolvimento do trabalho. No capítulo 5 é proposto um modelo de ambiente de detecção de intrusão capaz de enviar respostas aos alertas recebidos, neste capítulo são apresentados a arquitetura, o protocolo de comunicação e o modelo de dados que compõem o modelo de ambiente proposto. O capítulo 6 descreve a implementação dos componentes do modelo apresentado no capítulo 5, juntamente com a descrição do ambiente de desenvolvimento e das bibliotecas utilizadas. No capítulo 6 também é apresentada a validação do modelo previamente proposto, descrevendo o ambiente montado para a validação e os testes realizados.

O capítulo 7 descreve os resultados obtidos com o desenvolvimento do trabalho e realiza uma discussão sobre cada um destes resultados.

O capítulo 8 contém as conclusões do trabalho, apresentando as principais dificuldades e sugerindo trabalhos futuros.

Ao final, são apresentadas as referências bibliográficas e os anexos.

2 ARQUITETURA E REQUISITOS

O grupo IDWG descreve em [WOOD 2002] a arquitetura de um sistema de detecção de intrusão, os requisitos do protocolo de comunicação e do formato das mensagens necessários para a interoperabilidades destes sistemas. Este capítulo apresenta a arquitetura de um IDS proposta pelo IDWG e os requisitos necessários a esta arquitetura.

Os motivos que justificam a necessidade de se definir um padrão de arquitetura e comunicação para os sistemas de detecção de intrusão, segundo [WOOD 2002], são:

- existem vários sistemas de detecção de intrusão disponíveis no mercado, e esta quantidade tende a crescer cada vez mais. Alguns sistemas utilizam a rede para detectar intrusões, outros utilizam o sistema operacional do *host* e outros ainda utilizam as aplicações que estão instaladas no *host*. Comparando estes diferentes IDSs descobrimos que seus pontos fortes e fracos são muito diferentes. Sendo assim, seria interessante para os usuários utilizarem mais de um tipo de IDS em seu ambiente, e gerenciar estes diferentes IDSs a partir de um ou mais gerenciadores. A padronização da arquitetura e da comunicação dos IDSs simplificaria a tarefa de gerenciamento;
- intrusões geralmente envolvem redes de várias empresas ou várias redes de uma mesma empresa. Provavelmente estas redes terão IDSs de diferentes fabricantes. Seria de grande utilidade fazer o relacionamento de um ataque distribuído por várias redes ou domínios administrativos diferentes. A obtenção de relatórios em um formato comum das várias redes que estão sendo atacadas iria facilitar esta tarefa;
- a existência de um formato e uma arquitetura comum permitiria que componentes de diferentes IDSs fossem integrados mais facilmente. *Log*o, pesquisas sobre detecção de intrusão poderiam migrar para produtos comerciais mais facilmente;
- inicialmente feito para permitir a comunicação entre um analisador e um gerenciador, o sistema de comunicação IDMEF pode também permitir a comunicação entre outros de componentes de detecção de intrusão.

Além destes motivos, a criação de um padrão de arquitetura e comunicação de IDSs pode fazer com que o desenvolvimento destes produtos aumente e as técnicas utilizadas evoluam com mais facilidade e velocidade, resultando em ambientes mais seguros.

2.1 Arquitetura

Os componentes de uma arquitetura IDS, segundo a proposta do grupo IDWG, são apresentados na Figura 2.

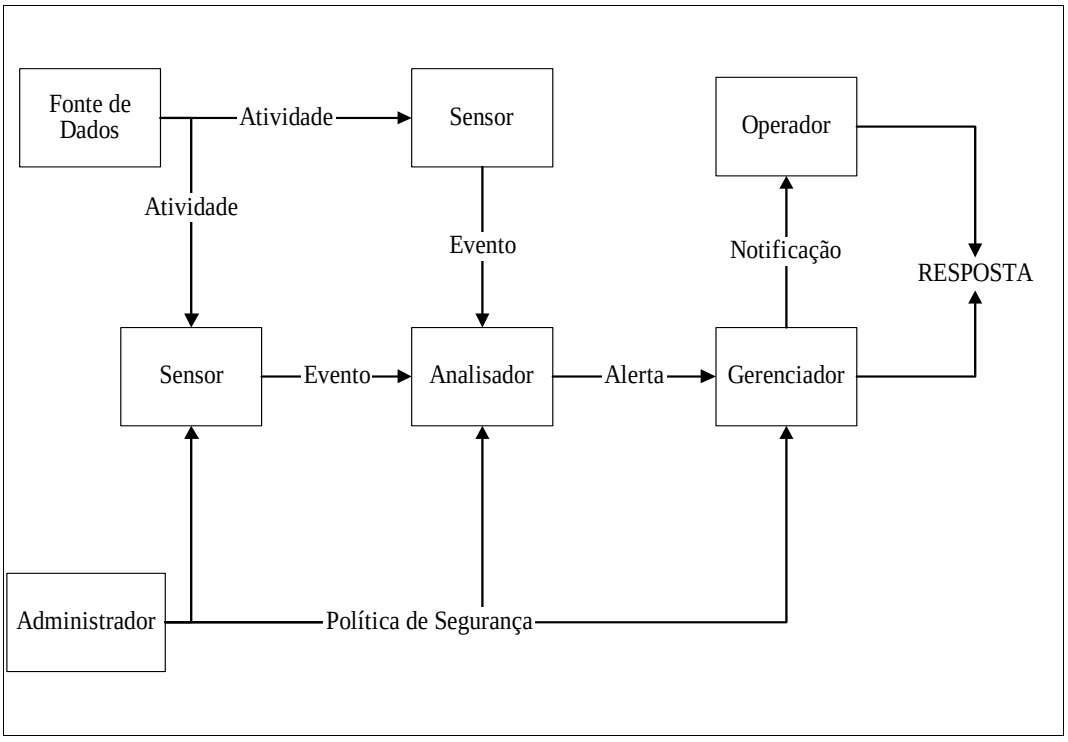

Figura 2 – Arquitetura de um IDS IDWG.

O início do processamento de um sistema de detecção de intrusão está na Fonte de Dados. A Fonte de Dados corresponde às informações que serão analisadas a fim de se detectar uma intrusão. Como exemplos de Fonte de Dados temos os pacotes de rede e os *logs* de sistemas operacionais e de aplicativos.

A Atividade corresponde às ocorrências de Fonte de Dados que são identificadas pelo Sensor ou pelo Analisador como de interesse do Operador. Como exemplos de Atividades temos sessões de Telnet inesperadas ou entradas em *logs* de sistemas indicando que um usuário tentou acessar um recurso não autorizado.

Os Sensores são componentes que coletam as informações da Fonte de Dados e encaminham Eventos para os Analisadores. O Evento é a ocorrência de uma Atividade na qual o Sensor conclui que esta poderá gerar um alerta pelo Analisador, um exemplo de Evento é a identificação de várias falhas de *log*in em um curto espaço de tempo.

O Analisador recebe os Eventos do Sensor e verifica a necessidade de gerar um Alerta. Em muitos sistemas de detecção de intrusão o Sensor e o Analisador fazem parte do mesmo componente.

Um Alerta é uma mensagem para o Gerenciador informando que um Evento de seu interesse foi detectado, geralmente ele contém detalhes da Atividade que gerou o Evento. O Gerenciador é um componente no qual o Operador gerencia todos os outros componentes do IDS, entre as funções desempenhadas por um gerenciador estão a configuração dos Sensores e dos Analisadores, o gerenciamento dos Alertas e das Notificações e a geração de relatórios.

A forma pela qual o Gerenciador sinaliza o Operador sobre os Alertas ocorridos é chamada de Notificação, isto pode ser feito através de ícones coloridos na tela do Gerenciador, envio de E-mail, mensagens de pages, etc.

Um Operador é um usuário que utiliza o Gerenciador, o Operador monitora as saídas do Gerenciador e, quando necessário, executa ações para conter um ataque em andamento. As ações que o Operador executa para conter um ataque em andamento são chamadas de Respostas (ou Contra-Medidas).

As Respostas a um ataque podem ser enviadas automaticamente por alguma entidade da arquitetura do IDS ou pelo Operador. Algumas respostas que podem ser acionadas pelo Operador são: o encerramento de conexões, a finalização de sessões do usuário no sistema operacional ou em aplicativos e a alteração de configurações em *firewalls*.

O Administrador é um usuário que tem a responsabilidade de definir a Política de Segurança do ambiente e também a configuração e organização do IDS, ele pode ou não ser o mesmo usuário que desempenha a função de Operador, em algumas empresas o Administrador e o Operador podem também desempenhar a função de gerência da rede. A Política de Segurança desenvolvida pelo Administrador é um documento que define quais atividades são permitidas na rede ou em um *host* específico da empresa. Os

Sensores, Analisadores e Gerenciadores do IDS trabalham baseados na Política de Segurança desenvolvida pelo Administrador.

Analisando os detalhes da arquitetura de IDSs proposta pelo IDWG, percebemos que ela não define componentes específicos para o tratamento de Respostas. Esta arquitetura também não define como será a interação entre os componentes e nem quais eventos estão envolvidos com uma resposta desde a sua geração até a sua execução.

2.2 Requisitos do Protocolo de Comunicação

Em [WOOD 2002] estão definidos os requisitos que um protocolo precisa ter para transportar mensagens IDMEF entre sistemas de detecção de intrusão. O primeiro requisito especificado determina que o protocolo utilizando garanta a confiabilidade da transmissão das mensagens, ou seja, o protocolo tem que garantir que uma mensagem enviada por um componente do IDS para outro chegará efetivamente ao seu destino.

Outro requisito especificado é a interação com *firewalls*, este requisito determina que as mensagens transmitidas pelo protocolo tenham a capacidade que passar por *firewalls* sem comprometer a segurança do ambiente. Este requisito é importante, pois os Analisadores e os Gerenciadores podem estar separados por um *firewall*, neste caso são necessárias técnicas que permitam a passagem das mensagens IDMEF pelo *firewall*.

O requisito de autenticação mutua determina que os Analisadores e os Gerenciadores devem se autenticar em nível de aplicação, independente da autenticação em nível de transporte existente. Os alertas gerados por Analisadores são utilizado pelos Gerenciadores para geração de respostas ou futuras investigações sobre um ataque ao ambiente da empresa, sendo assim, é importante que um Analisador conheça a identidade de um Gerenciador ao qual ele esteja enviando um alerta. Da mesma forma, é importante para o Gerenciador conhecer a identidade do Analisador que está lhe enviado os alerta.

A confidencialidade das mensagens é um requisito que especifica que as mensagens não podem ser lidas por outros usuários durante a sua transmissão, para atender a este requisito o protocolo precisa utilizar técnicas de criptografia. O requisito ainda especifica que o protocolo escolhido precisa suportar uma grande variedade de algorítmos de criptografia. As mensagens IDMEF geralmente contêm informações de interesse dos invasores, e como estas mensagens poderão trafegar por redes externas ao

ambiente do IDS é importante que a confidencialidade seja garantida. Devido a grande variedade de algorítmos de criptografia, também é importante que o protocolo escolhido suporte esta variedade e permita ao usuário configurar o seu ambiente.

O protocolo escolhido para a troca de mensagens IDMEF também deve suportar o requisito de integridade das mensagens trocadas, ou seja, o protocolo deve garantir que o conteúdo das mensagens não foi alterado durante a transmissão. Uma grande variedade de mecanismos de integridade devem ser suportados pelo protocolo, para permitir que ele se adapte a uma grande variedade de ambientes. As mensagens IDMEF são utilizadas no gerenciamento da segurança do ambiente em que o IDS está instalado, devido a isto, é muito importante que o conteúdo destas mensagens não seja alterado após o seu envio.

Além do requisito de autenticação mutua, também existe o requisito de autenticação por origem, o protocolo deve suportar autenticações separadas para cada componente que originou um alerta. Isto é importante para que um Gerenciador possa conhecer individualmente cada um dos Analisadores com quem está trocando informações.

O protocolo deve ter mecanismos que previnam contra ataques de negação de serviço (DoS – Denial of Service). Um possível ataque contra o sistema de detecção de intrusão será causar a sobrecarga de seus recursos, ocasionando a negação de serviço. Um atacante que invade uma rede protegida por um IDS e percebe a presença do mesmo, irá tentar primeiro atacar o IDS e tirá-lo do ar, caso o IDS resista ao ataque e continue funcionando, provavelmente o atacante vai desistir do seu ataque a outros recursos da rede.

Além de mecanismos para resistir a ataques de DoS, o protocolo também deve possuir mecanismos para resistir a ataques de duplicação de mensagens. Para confundir um Gerenciador e atrapalhar o seu funcionamento, um atacante pode copiar mensagens originais enviadas por um Analisador e enviá-las novamente para o Gerenciador, é importante que o protocolo não aceite as mensagens copiadas.

2.3 Requisitos de Formato das Mensagens

O formato das mensagens IDMEF deve ser independente do protocolo de comunicação que transporta as mensagens, isto permite que sejam utilizados diferentes mecanismos de transporte sem alterar o formato das mensagens IDMEF.

A especificação das mensagens IDMEF deve suportar requisitos de internacionalização e localização. Um IDS pode estar distribuído por uma área geográfica e culturalmente distinta, as mensagens IDMEF precisam estar formatadas de maneira que um operador as entenda em seu local de trabalho. Por exemplo, se uma mensagem IDMEF conter *strings*, ela ser formatada através de um padrão internacional de representação de caracteres.

O formato das mensagens IDMEF também deve suportar filtragem e agregação de dados. Caso um Gerenciador queira fazer uma filtragem ou uma agregação das mensagens que está recebendo, o formato IDMEF deve está organizado de maneira a facilitar esta tarefa. Devido a este requisito, é recomendado que as mensagens IDMEF tenham um formato fixo e claramente definido.

2.4 Requisitos de Conteúdo das Mensagens

Vários requisitos foram especificados com relação ao conteúdo das mensagens IDMEF, estes requisitos têm por objetivo definir as informações mínimas que devem existir nas mensagens IDMEF.

As mensagens IDMEF precisam estar preparadas para receber informações dos diversos tipos de IDSs existentes. Alguns IDSs analisam *logs* do sistema operacional, outros analisam o tráfego de rede e são baseados em assinaturas, e outros ainda podem utilizar métodos estatísticos analisando a rede ou *hosts*. Estes diferentes tipos de IDSs geram diferentes informações, e todas elas devem ser suportadas pelas mensagens IDMEF.

Uma mensagem IDMEF precisa conter um identificador do tipo de evento que a mensagem representa, se esta informação for conhecida. Este identificador e os tipos de eventos existentes devem estar padronizados em uma lista. Segundo [WOOD 2002], ainda não foi definido como esta lista será criada, acessada e atualizada. Estas questões fazem parte dos trabalhos atuais do grupo IDWG.

Um componente que envia uma mensagem IDMEF deve ter a possibilidade de colocar na mensagem informações que auxiliem o componente que recebe a mensagem a encontrar informações complementares sobre o evento que gerou a mensagem. Estas informações podem ser, por exemplo, uma referência a um *site* que auxilie o Operador a encontrar soluções para um possível ataque que esteja ocorrendo em seu ambiente.

Deve existir a possibilidade de se incluir dados adicionais sobre um evento em uma mensagem IDMEF, estes dados adicionais servem para descrever peculiaridades de um evento específico. A informação de dados adicionais não é obrigatória e não existe um formato pré-definido para estes dados.

Outro requisito relacionado ao conteúdo das mensagens IDMEF é a identificação, quando possível, da origem e do destino do ataque. No caso de sistemas baseados em rede esta identificação pode ser feita pelo endereço IP dos componentes de origem e destino.

Além da identificação de origem e destino do ataque, existe um requisito de conteúdo que determina que deve haver uma identificação que permita a localização do componente que originou as mensagens, em IDSs baseados em rede a localização pode ser feita também com o endereço IP de origem.

As mensagens IDMEF devem também permitir a representação de diferentes tipos de endereçamento de dispositivos. Qualquer elemento endereçável de rede é considerado um dispositivo. Os ataques podem envolver dispositivos de diferentes tecno*log*ias de rede e em diferentes níveis da rede e as mensagens IDMEF devem suportar o endereçamento de todos estes dispositivos.

O impacto causado por um determinado evento enviado através de uma mensagem IDMEF é uma informação que deve estar contida dentro da mensagem. Esta informação é utilizada pelo Operador para saber o quão crítico é um evento que está ocorrendo em seu ambiente.

Caso o Analisador que gerou a mensagem IDMEF tenha tomado alguma ação automática contra o evento transmitindo, deve haver na mensagem informações sobre a ação tomada. Estas informações são muito importantes para que o Operador saiba o que já está sendo feito para conter o ataque e possa decidir qual a próxima ação a ser tomada da melhor maneira possível.

Um ambiente pode ter diversos tipos de Analisadores executando em um mesmo *host*, quando isto ocorrer, a identificação de origem e destino das mensagens não será capaz que determinar qual Analisador gerou evento. Devido a isto as mensagens IDMEF devem conter um identificador de analisador.

Um ambiente pode ser configurado para emitir alertas para qualquer suspeita de ataque, por menor que seja. Portanto deve existir um campo na mensagem IDMEF que informe qual a possibilidade do evento reportado na mensagem ser um ataque real.

Cada mensagem IDMEF precisa ter um identificador único, de maneira que uma determinada mensagem não possa ser confundida com outras mensagens. Este identificador pode ser utilizado para fazer o relacionamento entre as mensagens. Além de um identificador único as mensagens devem conter informações sobre a data e hora de sua criação, as informações de data e hora são importantes para a geração de relatórios e a correlação entre as mensagens recebidas.

A semântica do conteúdo das mensagens precisa ser bem definida. Isto é importante porque é baseado nas mensagens IDMEF que o Operador irá tomar decisões para conter um ataque, se a mensagem não for clara, o Operador poderá tomar decisões erradas prejudicando o funcionamento do ambiente ao invés de evitar o ataque.

As mensagens IDMEF precisam suportar mecanismos de extensão utilizados para implementações específicas desenvolverem tipos de mensagens e dados específicos. As informações estendidas deverão ser definidas pela sua respectiva implementação, sendo que estas informações não podem interferir nas informações definidas pelo padrão e nem impedir a interoperabilidade das mensagens IDMEF.

Além das mensagens serem extensíveis, o próprio modelo IDMEF precisa ser extensível. Novas tecno*log*ias de detecção de intrusão são criadas a todo o momento e o modelo de dados IDMEF precisa estar preparado para suportar estas novas tecno*log*ias.

3 PROTOCOLOS UTILIZADOS NO MODELO IDWG

3.1 Protocolo BEEP

Nesta seção será apresentado o protocolo BEEP (*Block Extensible Exchange Protocol*). Sendo o protocolo IDXP a implementação de um novo perfil do protocolo BEEP, este é de grande importância na comunicação entre os sistemas de detecção de intrusão. Alguns requisitos especificados para o protocolo IDXP são atendidos na implementação do protocolo BEEP, sendo assim, estes requisitos não precisaram ser implementados no IDXP.

3.1.1 Introdução

Segundo [ROSE 2001], o protocolo BEEP é um protocolo de aplicação genérico para comunicação orientada à conexão e assíncrona, que permite a troca de mensagens simultâneas e independentes entre pares. As mensagens são formadas por textos estruturados no formato XML [W3C 2000].

As trocas de mensagens ocorrem através de canais de comunicação. Cada canal está associado a um perfil (*profile*) que define a sintaxe e a semântica das informações trocadas. O protocolo BEEP realiza o gerenciamento dos canais, criando novos canais e destruindo-os quando necessário.

O BEEP possuí alguns perfis pré-definidos em sua implementação, como o perfil de transporte seguro (TLS *Profile - Transport Layer Security Profile*) e alguns perfis específicos para autenticação, como a família de perfis para autenticação simples e camada segura (SASL *Profiles - The Simple Authentication and Security Layer Family of Profiles*). Além destes perfis pré-definidos, qualquer aplicação que utilize o BEEP pode desenvolver seus próprios perfis para atender às suas necessidades. Um exemplo disto é o desenvolvimento do perfil IDXP para atender às necessidades de comunicação entre sistemas de detecção de intrusão.

Inicialmente o protocolo BEEP estabelece uma sessão de comunicação, estas são mapeadas diretamente sobre um serviço de transporte (TCP, TLS, etc), no estabelecimento de sessão cada par da comunicação BEEP informa quais perfis são suportados. Na criação dos canais, o par que está criando o canal informa quais perfis

pretende utilizar e recebe a resposta do outro par informando se os perfis solicitados são suportados ou não. Se algum perfil for recusado a criação do canal é cancelada.

Os canais podem ser divididos em duas categorias:

- túnel inicial: este tipo de canal é associado a perfis utilizados para iniciar e estabilizar uma sessão BEEP;
- contínuo: este tipo de canal é associado a perfis utilizados para troca de informação, geralmente estes canais são criados depois do túnel inicial ter sido criado.

Para cada sessão é criado primeiro um único canal de túnel inicial, depois disto são criados vários canais contínuos para a troca de informações.

3.1.2 Regras, Mensagens e *Frame*s

Para um melhor entendimento e uma melhor representação da comunicação realizada pelo protocolo BEEP, algumas regras foram definidas referentes a esta comunicação. Durante o estabelecimento de uma sessão, o par da comunicação que fica aguardando a solicitação de novas sessões é chamada de "*listening*" e o par que solicita novas sessões é chamado de "*initiating*", estes dois comportamentos são representados respectivamente por "L:" e "I:".

O par da comunicação que inicia uma troca de informação é chamada de cliente e o outro par desta comunicação é chamado de servidor, estes dois comportamentos são representados respectivamente por "C:" e "S:".

Normalmente o par que realiza o *listening* será também o servidor e o par que realiza o *initiating* será o cliente, porém este não é um requisito obrigatório na comunicação BEEP.

Quanto ao estilo das trocas de mensagens na comunicação, o protocolo BEEP define três formas:

- MSG/RPY: o cliente envia um "MSG" para o servidor, solicitando que ele execute alguma ação. O servidor executa a ação e retorna "RPY" para o cliente;
- MSG/ERR: o cliente envia um "MSG" para o servidor, porém o servidor não está apto a atender à solicitação e retorna um "ERR" para o cliente;

- MSG/ANS: o cliente envia um “MSG” para o servidor, durante a execução da tarefa solicitada o servidor retorna várias mensagens “ANS” para o cliente. Uma mensagem “NUL” significa o fim do processamento da tarefa solicitada.

As mensagens BEEP são estruturadas de acordo com as regras MIME [FREED 1996]. Normalmente as mensagens são enviadas em um único *frame*, porém pode haver casos em que serão necessários vários *frame*s para enviar uma mensagem. Um *frame* é formado por *header*, *payload* e *trailer*. O *header* está no início do *frame* e contém informações sobre o formato do *frame*. O *payload* contém as informações do perfil que está sendo transmitido. E o *trailer* indica o final do *frame*. O Quadro 1 apresenta um exemplo de *frame*.

```
C: MSG 0 1 . 52 120
C: Content-Type: application/beep+xml
C:
C: <start number='1'>
C:    <profile uri='http://iana.org/beep/SASL/OTP' />
C: </start>
C: END
```

Quadro 1 – Exemplo de *frame* BEEP.

As duas primeiras linhas do exemplo acima formam o *header*, a última linha é o *trailer* e o restante forma o *payload.* A sintaxe de um *frame* BEEP é especificada pela ABNF apresentada no Quadro 2.

```
frame      = data / mapping

data       = header payload trailer

header     = msg / rpy / err / ans / nul

msg        = "MSG" SP common          CR LF
rpy        = "RPY" SP common          CR LF
ans        = "ANS" SP common SP ansno CR LF
err        = "ERR" SP common          CR LF
nul        = "NUL" SP common          CR LF

common     = channel SP msgno SP more SP seqno SP size
channel    = 0..2147483647
msgno      = 0..2147483647
more       = "." / "*"
seqno      = 0..4294967295
size       = 0..2147483647
ansno      = 0..2147483647

payload    = *OCTET

trailer    = "END" CR LF
```

```
mapping    = ;; cada mapeamento de transporte pode definir frames
             ;; adicionais
```

Quadro 2 – Especificação do formato do *frame* BEEP.

A ABNF apresentada no Quadro 2 especifica que um *frame* é formado por data ou mapping. Data por sua vez é formado por *header*, *payload* e *trailer*. O *header* pode ser formado por msg, rpy, err, ans ou nul. *Payload* é definido apenas como um conjunto de octetos, pois cada perfil deverá definir o seu *payload.* E *trailer* é formado pelos caracteres “END” e CR LF que indicam uma quebra de linha. Caso o *frame* seja um mapping, o formato do *frame* será definido pelo mapeamento realizado com o serviço de transporte.

No *header*, os identificadores msg, rpy, err, ans e nul são formados pelas letras “MSG”, “RPY”, “ERR”, “ANS” E “NUL” respectivamente, seguidas por um espaço, pelo identificador *common* e por CF LF, exceto o *header* ans, que possui um campo a mais (ansno) que serve para indicar o número de uma determinada resposta a uma solicitação feita.

Dentro do identificador *common,* o campo *channel* indica o número do canal a que a mensagem pertence. Msgno é um número seqüencial que identifica a mensagem, este número deve ser diferente para cada nova mensagem (“MSG”) dentro de um mesmo canal. O campo *more* indica se a mensagem possui mais *frame*s, quando *more* for “.” a mensagem não possui mais *frame*s, quando *more* for “*” existem outros *frame*s para a mensagem. Seqno é um número seqüencial obtido através do deslocamento do primeiro caracter de cada *payload* transmitido em um canal. O campo size indica o tamanho do *payload* em octetos.

Analisando novamente o *frame* apresentado no Quadro 1 podemos agora identificar que trata-se de uma mensagem de solicitação (“MSG”) no canal 0 com número de seqüência da mensagem igual a 1. Esta mensagem não possui mais *frame*s (*more* = “.”), seu número de seqüência é 52 e o *payload* possui 120 octetos.

O Quadro 3 apresenta um exemplo de mensagem com vários *frame*s. Podemos observar que a resposta (“RPY”) de número de seqüência 287 da mensagem 1 possui o campo *more* igual a “*”. A mensagem de número de seqüência 307 é a continuação da mensagem anterior, pois o seu número seqüencial de mensagem também é 1. A mensagem do *frame* 307 não possui mais *frame*s, pois o seu campo *more* é igual a “.”.

```
S: RPY 0 1 * 287 20
```

```
S:     ...
S:     ...
S: END
S: RPY 0 1 . 307 0
S: END
```

Quadro 3 – Mensagem com vários *frame*s.

Observamos no Quadro 3 que o campo número da mensagem realiza a correlação de uma mesma mensagem em vários *frame*s diferentes. Quando são necessárias várias respostas diferentes para uma determinada mensagem (mensagens do tipo "ANS"), e estas respostas são quebradas em vários *frame*s, não é possível fazer a identificação das mensagens somente pelo campo número da mensagem. Neste caso é utilizado também o campo ansno, conforme é visto no Quadro 4.

```
S: ANS 1 0 * 0 20 0
S:     ...
S:     ...
S: END
S: ANS 1 0 * 20 20 1
S:     ...
S:     ...
S: END
S: ANS 1 0 . 40 10 0
S:     ...
S: END
```

Quadro 4 – Mensagem do tipo "ANS".

No Quadro 4 todas as mensagens estão no canal 1. Existe uma mensagem com número de seqüência igual a 0, que possui mais *frame*s e tem tamanho 20. A resposta seguinte, com número de seqüência igual a 20, não é a continuação da primeira, pois o seu número de resposta é igual a 1. A continuação da primeira resposta está na terceira resposta, com número de seqüência igual a 40 e número de resposta igual a 0.

A semântica das mensagens trocadas durante uma comunicação depende do perfil utilizado na comunicação. Cada perfil BEEP deve especificar as seguintes informações:

- quais as mensagens de inicialização, que serão trocadas durante a criação do canal;
- quais mensagens que podem ser utilizadas durante a troca de informações do canal;
- a semântica de todas estas mensagens.

Todas estas informações são cadastradas de maneira organizada quando um novo perfil BEEP é registrado.

3.1.3 Gerenciamento de Canais

Quando uma sessão BEEP é iniciada, apenas o canal zero é definido, este canal é utilizado para o gerenciamento dos outros canais da sessão. Existe um perfil BEEP específico para o gerenciamento de canais, este perfil é utilizado no canal zero. O gerenciamento de canais permite que os pares de uma comunicação informem quais perfis são suportados e qual perfil será utilizando no novo canal que será criado para fazer a troca de informações entre os pares. O gerenciamento de canais também é responsável pela comunicação de encerramento de um canal.

Uma sessão BEEP está estabilizada quando cada par da comunicação envia uma resposta "*greeting*" pelo canal zero com número de seqüência da mensagem igual a zero. Isto indica que os pares estão prontos para a comunicação de gerenciamento de canais. Um exemplo de mensagem *greeting* do perfil de gerenciamento de canais do protocolo BEEP é apresentando no Quadro 5.

```
L: <wait for incoming connection>
I: <open connection>
L: RPY 0 0 . 0 110
L: Content-Type: application/beep+xml
L:
L: <greeting>
L:    <profile uri='http://iana.org/beep/TLS' />
L: </greeting>
L: END
I: RPY 0 0 . 0 52
I: Content-Type: application/beep+xml
I:
I: <greeting />
I: END
```

Quadro 5 – Mensagem *greeting*.

As respostas *greeting* enviadas pelo *listening* e pelo *initiating* apresentadas no Quadro 5 são independentes uma da outra.

O elemento "*greeting*" possui dois atributos opcionais: *features* e localize. E zero ou mais elementos "*profile*".

O atributo *features*, quando presente, indica uma ou mais características opcionais suportadas pelo canal que está sendo gerenciado. O atributo localize, quando presente, indica a linguagem que deve ser utilizada em elementos textuais da comunicação, como por exemplo, em mensagens de erro. Cada elemento *profile* da mensagem *greeting* identifica um perfil que poderá ser associado a um canal da sessão BEEP corrente.

Para criar um novo canal é enviada uma mensagem "*start*" pelo canal zero. O Quadro 6 apresenta um exemplo de mensagem *start*.

```
C: MSG 0 1 . 52 120
C: Content-Type: application/beep+xml
C: <start number='1'>
C:    <profile uri='http://iana.org/beep/SASL/OTP' />
C: </start>
C: END
```

Quadro 6 – Mensagem *start*.

No Quadro 6, o atributo number do elemento *start* informa o número do canal que deverá ser criado. Enquanto que o elemento *profile*, através de seu atributo "uri", especifica quais perfis podem deve ser utilizados neste novo canal.

Quando um par BEEP recebe uma mensagem *start* através do canal zero, ele verifica todos os perfis propostos e decide qual destes será utilizado no canal que será criado. Se um par BEEP recebe uma mensagem *start* e aceita algum dos perfis propostos, então ele deve responder com o perfil que foi aceito. O Quadro 7 apresenta um exemplo deste caso.

```
C: MSG 0 1 . 52 178
C: Content-Type: application/beep+xml
C:
C: <start number='1'>
C:    <profile uri='http://iana.org/beep/SASL/OTP' />
C:    <profile uri='http://iana.org/beep/SASL/ANONYMOUS' />
C: </start>
C: END
S: RPY 0 1 . 221 87
S: Content-Type: application/beep+xml
S:
S: <profile uri='http://iana.org/beep/SASL/OTP' />
S: END
```

Quadro 7 – Resposta positiva à mensagem *start*.

Através do *frame* 52 foram propostos dois perfis para a criação do canal 1 ("http://iana.org/beep/SASL/OTP" ou "http://iana.org/beep/SASL/ANONYMOUS"). A resposta (número 221) indica que o primeiro perfil proposto foi aceito.

Caso o par que recebeu a mensagem *start* não aceite criar um canal com nenhum dos perfis propostos, então uma mensagem de erro deve ser retornada. O Quadro 8 apresenta um exemplo deste caso.

```
C: MSG 0 1 . 52 120
C: Content-Type: application/beep+xml
C:
C: <start number='2'>
C:    <profile uri='http://iana.org/beep/SASL/OTP' />
C: </start>
```

```
C: END
S: ERR 0 1 . 221 127
S: Content-Type: application/beep+xml
S:
S: <error code='501'>number attribute
S: in &lt;start&gt; element must be odd-valued</error>
S: END
```

Quadro 8 – Resposta negativa à mensagem *start*.

No exemplo do Quadro 8, através do *frame* 52, é feita uma solicitação para criar o canal 2 com o perfil "http://iana.org/beep/SASL/OTP". A resposta de erro ("ERR") com número de seqüência 221 indica que a criação do canal com o perfil solicitado foi recusada. Um estudo das mensagens de erro será feito no final desta seção.

Para realizar o encerramento de um canal o perfil de gerenciamento de canais BEEP envia uma mensagem "close" através do canal zero, conforme demonstrado no Quadro 9.

```
C: MSG 0 2 . 235 71
C: Content-Type: application/beep+xml
C:
C: <close number='1' code='200' />
C: END
```

Quadro 9 – Mensagem close.

O atributo number do elemento close indica qual canal deverá ser fechado e o atributo code é um número de três dígitos que indica o motivo pelo qual o canal está sendo fechado. Podem também ser adicionados elementos textuais à mensagem close, indicando os motivos pelo qual o canal foi encerrado.

Além dos atributos mostrados no Quadro 9, a mensagem close também possui um atributo chamado "xml:lang", este atributo é utilizado para informar a linguagem utilizada nos elementos textuais. O atributo localize da mensagem "*greeting*", quando definido, é dominante sobre atributo "xml:lang".

Se o atributo number for igual a zero então o par BEEP deseja encerrar a sessão corrente. Sendo diferente de zero o atributo number deve corresponder ao número de um canal existente que será encerrado.

Quando um par BEEP recebe uma mensagem close para um canal ele pode rejeitar o encerramento do canal enviando uma mensagem de erro. Um exemplo de solicitação de encerramento de canal com uma resposta de erro é apresentado no Quadro 11.

```
C: MSG 0 2 . 235 71
C: Content-Type: application/beep+xml
```

```
C:
C: <close number=’1’ code=’200’ />
C: END
S: ERR 0 2 . 392 79
S: Content-Type: application/beep+xml
S:
S: <error code=’550’>still working</error>
S: END
```

Quadro 10 – Encerramento de canal com erro.

Caso a solicitação de encerramento do canal seja aceita, o par BEEP que receber tal solicitação envia uma mensagem “ok” como resposta. Antes de enviar uma mensagem de “ok” para confirmar o encerramento do canal, as seguintes ações devem ser tomadas:

- enviar quaisquer mensagens (“MSG”) que estejam na fila de mensagens do canal;
- aguardar as respostas (“RPY”, “ERR”, “ANS”, “NUL”) das mensagens (“MSG”) que já foram enviadas;
- enviar as respostas (“RPY”, “ERR”, “ANS”, “NUL”) das mensagens (“MSG”) que foram recebidas.

Um exemplo solicitação de encerramento de canal com uma resposta positiva (“ok”) é apresentado no Quadro 11.

```
C: MSG 0 2 . 235 71
C: Content-Type: application/beep+xml
C:
C: <close number=’1’ code=’200’ />
C: END
S: RPY 0 2 . 392 46
S: Content-Type: application/beep+xml
S:
S: <ok />
S: END
```

Quadro 11 – Encerramento de canal com sucesso.

A mensagem “ok” é utilizada para indicar uma resposta positiva. Conforme vimos no exemplo acima esta mensagem não possui atributos.

A mensagem “error” é utilizada para indicar uma resposta negativa. Conforme vimos no Quadro 10 o elemento “error” possui o atributo “code” e também elementos textuais que descrevem o erro ocorrido, estes elementos textuais são opcionais. O atributo “code” é utilizado para informar um código de erro. A lista com os códigos de erro suportados está definida em [ROSE 2001]. Além do atributo “code” existe também um atributo chamado “xml:lang”, este atributo tem a mesma função e se comporta da

mesma maneira que o atributo “xml:lang” apresentado na mensagem close, porém é aplicado aos elementos textuais da mensagem “error”.

Mensagens BEEP de erro podem ser enviadas nos seguintes casos:

- quando é recebida uma mensagem fora do formato definido pelo protocolo BEEP ou quando uma mensagem possui elementos desconhecidos;
- quando é recebida uma solicitação de encerramento de um canal e o par BEEP que recebeu a solicitação não deseja que o canal seja encerrado;
- quando durante o estabelecimento da sessão BEEP o par *listening* for solicitado e não estiver pronto para enviar uma mensagem *greeting*.

No último caso citado acima, ambos os pares cancelam o estabelecimento da sessão.

3.1.4 Estabelecimento e Encerramento de Sessões

Observamos na seção anterior, no Quadro 5, o estabelecimento de uma sessão BEEP com sucesso através da mensagem *greeting*. Porém nem sempre o estabelecimento de uma sessão BEEP ocorrerá com sucesso, o Quadro 12 apresenta um exemplo de estabelecimento de sessão BEEP com uma resposta de erro.

```
L: <wait for incoming connection>
I: <open connection>
L: ERR 0 0 . 0 60
L: Content-Type: application/beep+xml
L:
L: <error code=’421’ />
L: END
I: RPY 0 0 . 0 52
I: Content-Type: application/beep+xml
I:
I: <greeting />
I: END
I: <close connection>
L: <close connection>
L: <wait for next connection>
```

Quadro 12 – Estabelecimento de sessão BEEP com erro.

No exemplo do Quadro 12, após o *initiating* ter iniciado uma conexão através do mapeamento de transporte, o *listening* retorna uma mensagem de erro. Neste caso, mesmo que o *initiating* tenha enviado uma mensagem *greeting* confirmado que está pronto para o estabelecimento da sessão, a sessão não será estabelecida, ou seja, a mensagem *greeting* do *initiating* é ignorada.

Quando um par BEEP deseja encerrar uma sessão, ele envia uma mensagem close com o valor do atributo number igual a zero. Caso a solicitação de encerramento de sessão seja aceita, ela terá uma resposta positiva (“ok”). Se o encerramento da sessão não for aceito será enviada uma mensagem de erro informando o porque da recusa.

O Quadro 13 apresenta um exemplo de encerramento de sessão BEEP com resposta positiva. Podemos observar na mensagem close (número de seqüência 52) a ausência do atributo number, isto indica que o seu valor será igual a zero.

```
C: MSG 0 1 . 52 60
C: Content-Type: application/beep+xml
C:
C: <close code=’200’ />
C: END
S: RPY 0 1 . 264 46
S: Content-Type: application/beep+xml
S:
S: <ok />
S: END
I: <close connection>
L: <close connection>
L: <wait for next connection>
```

Quadro 13 – Encerramento de sessão BEEP.

Logo após a resposta positiva à solicitação de encerramento da sessão, a mesma é encerrada de acordo com o mapeamento do serviço de transporte.

3.1.5 Mapeamento do Serviço de Transporte

A comunicação BEEP ocorre dentro de um contento de sessões, estas sessões são mapeadas em um serviço de transporte específico. O mapeamento do protocolo BEEP em um serviço de transporte deve estar definido em um documento específico para este fim. Este documento precisa conter várias informações que expliquem como os requisitos de comunicação do protocolo BEEP serão atendidos pela camada de transporte.

Alguns exemplos de informações que devem conter no documento de mapeamento do protocolo BEEP em um serviço de transporte são:

- como as sessões BEEP serão estabelecidas;
- como se identifica um par BEEP *listening*;
- como se identifica um par BEEP *initiating*;
- como uma sessão BEEP será encerrada.

Como um exemplo deste tipo de documento temos [ROSE 2001a], que especifica o mapeamento do protocolo BEEP sobre o protocolo de transporte TCP.

3.2 Protocolo IDXP

Nesta seção será apresentado o protocolo IDXP (*Intrusion Detection Exchange Protocol*). A definição do protocolo IDXP desenvolvida pelo IDWG é encontrada em [FEINSTEIN 2002].

3.2.1 Introdução

O IDXP é um protocolo de aplicação que tem por objetivo realizar a troca de informações entre sistemas de detecção de intrusão, este protocolo está de acordo com os requisitos especificados na seção 2.2. Através do protocolo IDXP podem ser trocadas mensagens IDMEF, textos sem uma estrutura definida ou dados em formato binário.

Conforme apresentado na seção 3.1, o protocolo IDXP é um perfil do protocolo BEEP, portanto muitas das funcionalidades e características do IDXP vêm do protocolo BEEP, assim como também alguns requisitos especificados para o protocolo de comunicação entre sistemas de detecção de intrusão já são atendidos pelo BEEP.

Além do perfil IDXP, vários outros perfis BEEP podem estar envolvido na troca de informações entre IDSs. Os perfis “TLS *Profile*” e “SASL Family of *Profiles*” são responsáveis por requisitos de segurança como autenticação e confiabilidade. O perfil “Tunnel *Profile*” pode ser utilizado para criar um túnel de comunicação no qual o perfil IDXP irá realizar a sua comunicação.

3.2.2 Comunicação

Componentes de um sistema de detecção de intrusão inicialmente estabelecem uma sessão BEEP para se comunicarem, então um ou mais canais de comunicação são abertos com o perfil IDXP para a troca de mensagens IDMEF, textos sem uma estrutura definida ou dados em formato binário. A comunicação de mensagens IDMEF é feita com componentes Analisadores e Gerenciadores, transmitindo informações sobre alertas ocorridos no ambiente.

Analisadores não podem realizar conexões IDXP com outros Analisadores, porém um Analisador pode ser conectado a um ou vários Gerenciadores, da mesma forma, um Gerenciador também pode ser conectado a um ou vários Analisadores. Os

Gerenciadores podem ser conectados entre si, isto possibilita a um Gerenciador receber os alertas de vários outros Gerenciadores e também repassar os alertas gerados pelos seus Analisadores para vários outros Gerenciadores. Isto permite que um Gerenciador receba os alertas de uma grande quantidade de Analisadores, através de Gerenciadores intermediários.

Quando um Analisador pretende transmitir um alerta para um Gerenciador, ele primeiro irá estabelecer uma sessão BEEP, após isto os perfis de segurança BEEP (TLS e SASL) podem ser negociados a fim de determinar a segurança que esta comunicação necessitará. Estabelecida a sessão e determinado o nível de segurança, é então criado um canal com o perfil IDXP para a troca de mensagens de alerta.

O Figura 3 apresenta uma visão geral da seqüência de comunicação necessária para o início das trocas de mensagens IDMEF pelo protocolo IDXP.

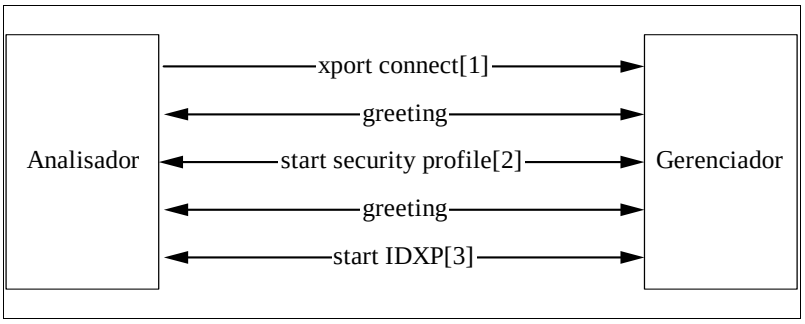

Figura 3 – Comunicação IDXP.

Na transmissão identificada por [1], o Analisador inicia a conexão de transporte e *log*o após são transmitidas as mensagens de estabelecimento de sessão BEEP (*greeting*). Em [2] o Gerenciador negocia com o Analisador os perfis de segurança que são necessários para esta comunicação. E em [3], é feita a negociação da utilização do protocolo IDXP, após esta negociação iniciam as trocas de mensagens IDMEF pelo protocolo IDXP.

Entre os Gerenciadores e os Analisadores podem existir um ou mais proxies. Estes podem ter sido criados por motivos de segurança ou por diversos motivos administrativos do ambiente. A instalação de *firewall* para controlar a comunicação em uma rede de computadores é um exemplo de criação de proxy.

O perfil BEEP chamado “Tunnel *Profile*” pode ser utilizado para criar um túnel de comunicação em nível de aplicação que torna transparente a existência de proxies.

Maiores detalhes sobre a especificação do perfil “Tunnel *Profile*” podem ser encontrados na RFC 3620 – “The TUNNEL *Profile*”.

A Figura 4 apresenta a seqüência utilizada para formar um túnel de comunicação antes do início das trocas de mensagens IDXP. Nesta figura o Analisador pretende trocar informações de detecção de intrusão com o Gerenciador, porém entre eles existe o Proxy1 e o Proxy2. Neste caso o Analisador inicia a criação de uma sessão BEEP com o Gerenciador entrando em contato primeiro o Proxy1. Estabelecida a sessão BEEP entre o Analisador e o Proxy1, o perfil Tunnel é selecionado para criar o túnel de comunicação, e esta seqüência é seguida até que o túnel chegue ao Gerenciador.

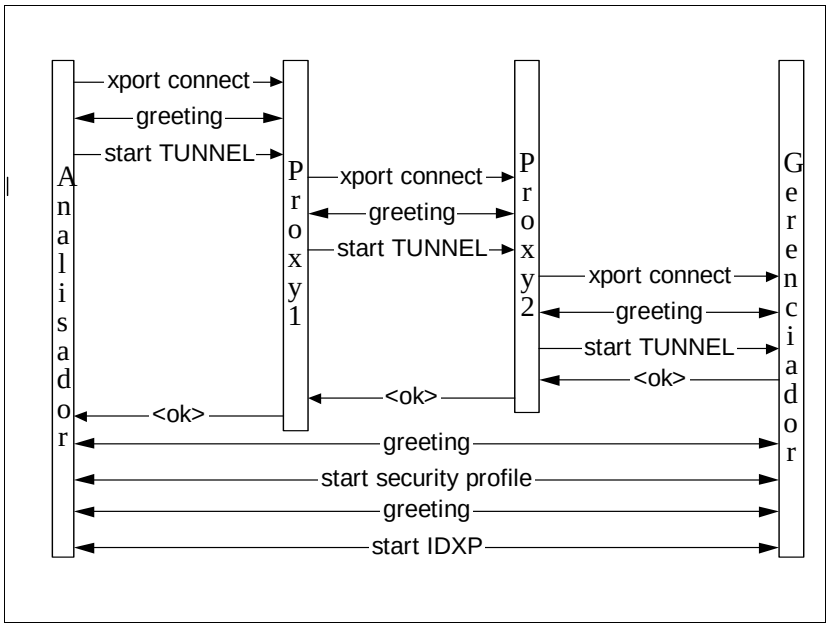

Figura 4 – Comunicação IDXP através de um túnel.

Quando o Gerenciador recebe a solicitação de túnel do Proxy2 ele envia uma resposta confirmando o estabelecimento do túnel, esta resposta se propaga até o Analisador e com isto o túnel está formado. Após o estabelecimento do túnel é enviada uma mensagem BEEP (*greeting*) confirmando a estabelecimento da sessão entre o Analisador e o Gerenciador, daí seguem a negociação dos perfis de segurança e o início da comunicação com o perfil IDXP.

Em uma mesma sessão BEEP podem ser abertos vários canais para comunicação com o perfil IDXP, opcionalmente podem ser abertas várias sessões BEEP diferentes entre dois componentes com canais utilizando o perfil IDXP. A criação de vários canais para realizar a comunicação entre dois componentes de um sistema de detecção de

intrusão pode ser utilizada para organizar esta comunicação em categorias ou definir prioridades para as informações de detecção de intrusão.

A Figura 5 apresenta uma comunicação entre dois componentes Gerenciadores na qual o Gerenciador cliente envia mensagens de alerta para o Gerenciador servidor. Nesta comunicação as mensagens de alerta estão sendo transmitidas por vários canais BEEP, os quais representam diferentes tipos de mensagens de alerta. As mensagens de um canal são alertas baseados em rede, as mensagens de outros canal são alertas baseados em *host* e no terceiro canal estão todos os outros tipos de alertas. Esta diferenciação dos alertas em canais permite ao Gerenciador processar de maneira diferente os alertas vindos de canais diferentes.

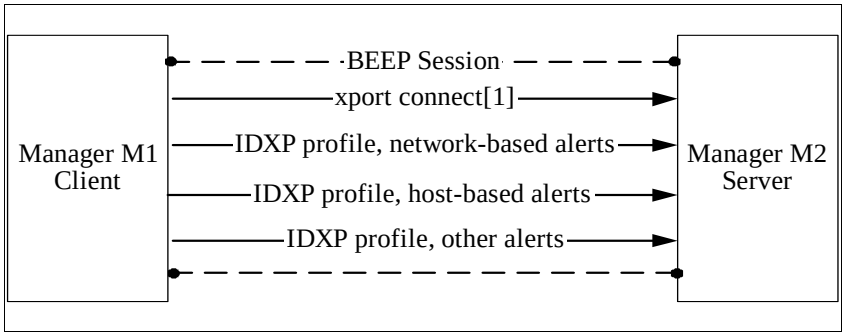

Figura 5 – Comunicação com vários canais IDXP.

O encerramento da comunicação pelo protocolo IDXP é feito quando um dos membros da comunicação envia uma mensagem BEEP “close” pelo canal zero informando o número do canal a ser fechado. A sessão BEEP pode ser fechada enviando-se a mensagem close com número de canal igual a zero, conforme apresentado na seção 3.1.4.

3.2.3 Perfil IDXP

Os principais elementos do perfil IDXP são:

- *IDXP-Greeting*: elemento que faz a identificação de um Analisador ou Gerenciador de um lado do canal BEEP para o Analisador ou Gerenciador do outro lado do canal BEEP;
- *Option*: elemento usado para transportar opções de configuração do canal durante a troca dos elementos *IDXP-Greeting*;
- *IDMEF-Message*: elemento usado para transportar informações de detecção de intrusão entre Analisadores e Gerenciadores.

Um perfil IDXP é identificado quando o elemento "*profile*" do protocolo BEEP conter o valor "*http://iana.org/beep/transient/idwg/idxp*".

Durante a criação de um canal com o perfil IDXP, elementos *IDXP-Greeting* são trocados entre os pares da comunicação BEEP. A primeira mensagem "MSG" do canal será uma mensagem com o elemento "*start*", sendo que dentro do elemento *profile* do elemento *start* haverá um elemento *IDXP-Greeting*. Se o canal for criado com sucesso, uma resposta "RPY" será retornada com o elemento "ok", caso contrário, esta será retornada com o elemento "error".

Cada par da comunicação BEEP envia elementos *IDXP-Greeting* utilizando mensagens MSG para se identificar. O elemento *IDXP-Greeting* deve informar se o par da comunicação está trabalhando como cliente ou como servidor, além de conter uma identificação do par que está enviando a mensagem, na forma de um URI (Uniform Resource Identifier). Opcionalmente uma mensagem *IDXP-Greeting* pode conter informações de domínio e um ou mais sub-elementos "Options" contendo opções de comunicação do canal.

Uma mensagem *IDXP-Greeting* pode ser enviada a qualquer momento, sendo que o par que receber esta mensagem deve responder com um elemento "ok", indicando que concorda com a comunicação ou com um elemento "error" caso não concorde com a comunicação. A regra de comunicação (Cliente ou Servidor) e qualquer opção de comunicação do canal sempre são definidas pela última mensagem *IDXP-Greeting* que foi confirmada por uma resposta "ok".

```
I: MSG 0 10 . 1592 187
I: Content-Type: text/xml
I: <start number='1'>
I:   <profile uri='http://iana.org/beep/transient/idwg/idxp'>
I:     <![CDATA[ <IDXP-Greeting uri='http://example.com/alice'
I:       role='client' /> ]]>
I:   </profile>
I: </start>
I: END
L: RPY 0 10 . 1865 91
L: Content-Type: text/xml
L:
L: <profile uri='http://iana.org/beep/transient/idwg/idxp'>
L:   <![CDATA[ <ok /> ]]>
L: </profile>
L: END
L: MSG 0 11 . 1956 61
L: Content-Type: text/xml
L:
L: <IDXP-Greeting uri='http://example.com/bob' role='server' />
```

```
L: END
I: RPY 0 11 . 1779 7
I: Content-Type: text/xml
I:
I: <ok />
I: END
```

Quadro 14 – Mensagem *IDXP-Greeting* com sucesso.

O Quadro 14 apresenta o envio de uma mensagem *IDXP-Greeting* com resposta positiva (RPY com elemento "ok"). No exemplo acima o *initiating* solicita a criação do canal 1 com o perfil IDXP (*http://iana.org/beep/transient/idwg/idxp*). Dentro do elemento *profile* ele está enviando um elemento *IDXP-Greeting* identificando-se como "*http://example.com/alice*" e informando que irá trabalha como cliente (*role=client*). O *listening* responde à solicitação do *initiating* com o elemento "ok", informando que concorda em abrir um canal com este perfil.

Em seguida, a *listening* também envia uma mensagem com o elemento *IDXP-Greeting*, identificando-se como "*http://example.com/bob*" e informando que irá trabalhar como servidor (*role=server*). O *initiating* responde à mensagem *IDXP-Greeting* do *listening* e a partir de então podem ser feitas trocas de mensagens IDMEF sob este canal pelo protocolo IDXP.

Uma mensagem *IDXP-Greeting* pode ser recusada (elemento "error") por vários motivos, por exemplo, quando uma autenticação falha, quando os algorítmos de criptografia são incompatíveis entre os pares ou quando são enviadas opções que não são conhecidas pelo par que recebeu a mensagem. O Quadro 15 apresenta um exemplo de resposta de erro à uma solicitação *IDXP-Greeting*.

```
I: MSG 0 10 . 1776 185
I: Content-Type: text/xml
I: <start number='1'>
I:   <profile uri='http://iana.org/beep/transient/idwg/idxp'>
I:     <![CDATA[ <IDXP-Greeting uri='http://example.com/eve'
I:       role='client' /> ]]>
I:   </profile>
I: </start>
I: END
L: RPY 0 10 . 1592 182
L: Content-Type: text/xml
L:
L: <profile uri='http://iana.org/beep/transient/idwg/idxp'>
L:   <![CDATA[
L:     <error code='530'>'http://example.com/eve' must first
L:       negotiate the TLS profile</error> ]]>
L: </profile>
L: END
```

Quadro 15 – Mensagem *IDXP-Greeting* com erro.

No Quadro 15 o *initiating* envia uma mensagem de criação de canal para o perfil IDXP, identificando-se como "*http://example.com/eve*" e informando que deseja trabalhar como cliente. O *listening* responde à mensagem do *initiating* com um elemento "error" informando que "*http://example.com/eve*" deve primeiro negociar o perfil de transporte seguro, para somente depois abrir um canal com o perfil IDXP, ou seja, foi configurado no *listening* que "*http://example.com/eve*" somente pode se comunicar através de uma sessão BEEP segura.

Uma mensagem *IDXP-Greeting* pode conter um ou mais elementos *Option*. O elemento *Option* indica que um dos pares da comunicação deseja habilitar opções específicas para o canal que está sendo criado.

A definição de uma opção IDXP deve conter as seguintes informações:

- a identificação da opção;
- o conteúdo das opções, se for necessário;
- as regras para o processamento da opção.

Estas informações estão contidas no documento de registro das opções, um exemplo deste tipo de documento pode ser encontrado em [FEINSTEIN 2002].

O elemento *Option* contém os seguintes atributos:

- internal ou external: estes atributos são utilizados para identificar a opção, é um atributo obrigatório;
- mustUnderstand: indica que a opção deve obrigatoriamente ser reconhecida pelo destinatário, o valor default deste atributo é "false". Este é um atributo opcional;
- localize: especifica a linguagem que deve ser utilizada em mensagem textuais, este atributo é opcional.

As opções com o atributo internal são registradas pela IANA, o valor especificado neste atributo é o nome da opção. Opções com o atributo external possuem como valor uma URI que indica a localização do registro da opção.

Quando o atributo mustUnderstand estiver com o valor verdadeiro "true" e o destinatário não souber como processa a respectiva opção, uma mensagem de erro será retornada informando que a opções não é suportada.

Uma das opções definidas pela IANA é a opção "*channelPriority*". Esta opção permite a definição de prioridades entre os canais BEEP quando dois pares utilizam

vários canais de comunicação com o perfil IDXP. Quando um elemento *IDXP-Greeting* é enviado para a criação de um canal com o perfil IDXP, o *initiating* pode solicitar ao *listening* que atribua uma prioridade específica para o canal, através da opção *channelPriority*.

O opção *channelPriority* possui o atributo *priority*, o valor "0" para este atributo indica a prioridade mais alta. O atributo *priority* é obrigatório na opção *channelPriority*.

O Quadro 16 apresenta uma mensagem com o elemento *IDXP-Greeting* contendo a opção *channelPriority*.

```
C: MSG 1 17 . 1984 165
C: Content-Type: text/xml
C:
C: <IDXP-Greeting uri='http://example.com/alice' role='client'>
C:   <Option internal='channelPriority'>
C:     <channelPriority priority='0' />
C:   </Option>
C: </IDXP-Greeting>
C: END
S: RPY 1 17 . 2001 7
S: Content-Type: text/xml
S:
S: <ok />
S: END
```

Quadro 16 – Opção *channelPriority*.

No Quadro 16 o cliente está solicitando que seja atribuída a prioridade 0 para o canal 1, e o servidor responde aceitando a opção solicitada.

A opção "*streamType*", também registrada pela IANA, permite que os canais de comunicação sejam divididos em categorias de acordo com o tipo de informação que está sendo transmitida. Quando um elemento *IDXP-Greeting* é transmitido, um par da comunicação pode solicitar que seja associado ao canal um tipo de informação específica, esta solicitação é feita adicionando-se um elemento *streamType* dentro de um elemento *Option.*

Um elemento *streamType* possui um atributo obrigatório chamado "*type*", este atributo pode conter os valores "alert", "heartbeat" e "config". O valor do atributo *type* indica o tipo de informação que está sendo transmitida pelo canal. Se o valor do atributo *type* do elemento *streamType* for "*alert*", o canal estará trocando informações sobre alertas de intrusões detectadas. O valor "heartbeat" indica que o canal está transmitindo mensagens de controle, maiores detalhes sobre este tipo de mensagem serão vistos no

Capítulo 4. O valor “config” classifica o canal como destinado à transmissão de mensagens de configuração.

O Quadro 17 apresenta a transmissão de uma opção *streamType* em um elemento *IDXP-Greeting* de uma mensagem. Neste exemplo está sendo solicitado que o canal 1 seja classificado como um canal exclusivo para a transmissão de alertas. Logo abaixo à mensagem *IDXP-Greeting*, é retornada uma mensagem de resposta contendo o elemento “ok”, informando que a opção foi aceita pelo servidor.

```
C: MSG 1 21 . 1963 155
C: Content-Type: text/xml
C:
C: <IDXP-Greeting uri='http://example.com/alice' role='client'>
C:   <Option internal='streamType'>
C:     <streamType type='alert' />
C:   </Option>
C: </IDXP-Greeting>
C: END
S: RPY 1 21 . 1117 7
S: Content-Type: text/xml
S:
S: <ok />
S: END
```

Quadro 17 – Opção *streamType*.

Eventualmente uma solicitação de opção pode ter uma resposta negativa, isto ocorre quando o destinatário da opção não aceita as características especificadas na opção ou quando o destinatário não reconhece uma opção e esta possui o atributo mustUnderstand com valor igual a true. O Quadro 18 apresenta a transmissão de uma opção que teve como resposta uma mensagem de erro. No exemplo abaixo o servidor envia uma mensagem com um elemento *IDXP-Greeting* contendo uma opção solicitando que o canal 1 seja classificado como um canal exclusivo para a transmissão de informações de configuração e o cliente responde à esta solicitação com uma mensagem de erro informando que a opção *streamType* não é reconhecida por ele, ou seja, o cliente não sabe como processar tal opção.

```
S: MSG 1 21 . 1969 176
S: Content-Type: text/xml
S:
S: <IDXP-Greeting uri='http://example.com/bob' role='server'>
S:   <Option internal='streamType' mustUnderstand='true'>
S:     <streamType type='config' />
S:   </Option>
S: </IDXP-Greeting>
S: END
C: ERR 1 21 . 1292 63
C: Content-Type: text/xml
```

```
C:
C: <error code='504'>'streamType' option was unrecognized</error>
C: END
```

Quadro 18 – Opção recusada.

A troca de qualquer tipo de informação (alerta, controle ou configuração) entre os dois pares da comunicação é feita através de mensagem contendo o elemento *IDMEF-Message*. Esta troca pode ser feita somente depois de concluída a abertura de um canal BEEP com o perfil IDXP. O detalhamento do conteúdo dos elementos *IDMEF-Message* será apresentado no Capítulo 4.

3.2.4 Conformidade com os Requisitos

Para realizar a troca de informações entre sistemas de detecção de intrusão, o protocolo IDXP deve atender a cada um dos requisitos especificados na seção 2.2.

O protocolo IDXP trabalha sobre o protocolo BEEP, isto possibilita que muitos requisitos necessários ao protocolo IDXP sejam atendidos pelo protocolo BEEP.

A confiabilidade na transmissão das mensagens é atendida pelo fato de que o protocolo BEEP utiliza protocolos de transporte orientados à conexão. Além de que o próprio BEEP utiliza um sistema de solicitações e respostas na sua comunicação.

O requisito de interação com *firewall* é implementado através da utilização de perfil de túnel BEEP (*Tunnel Profile*), que cria um canal de comunicação transparente entre dois pares que desejam se comunicar.

A autenticação mutua, a confidencialidade, a integridade e a autenticação individual dos Analisadores que se comunicam com um Gerenciador, é feita através do perfil de transporte seguro (TLS *Profile* - *Transport Layer Security Profile*) e alguns perfis específicos para autenticação, como a família de perfis para autenticação simples e camada segura (SASL *Profiles* - *The Simple Authentication and Security Layer Family of Profiles*) do protocolo BEEP.

A resistência a ataques de negação de serviço (DoS) também é garantida pelos perfils de segurança do protocolo BEEP. Para resistir a ataques de DoS o protocolo BEEP pode utilizar mecanismos de autenticação e tunelamento para criar um túnel de comunicação autenticada, todo o tráfego que não estiver autenticado é rapidamente descartado, evitando assim o desgaste dos recursos da comunicação e a conseqüente negação do serviço.

Os requisitos também especificam que o protocolo deve possuir mecanismos de resistência a ataques de duplicação de mensagens. No caso do protocolo IDXP este requisito é garantido pelo perfil de transporte seguro do protocolo BEEP, que já possui mecanismos de resistência a este tipo de ataque.

4 O MODELO IDMEF

O modelo IDMEF (*Intrusion Detection Message Exchange Protocol*) tem por objetivo definir o formato e o significado dos dados sobre detecção de intrusão que são compartilhados por IDSs. A especificação completa do modelo IDMEF é encontrada em [CURRY 2003].

4.1 Introdução

O modelo de dados (IDMEF) apresentado neste capítulo padroniza o formato dos dados trocados entre os componentes de um sistema de detecção de intrusão. O desenvolvimento deste modelo de dados possibilitará a interoperabilidade entre IDSs comerciais, IDSs de código aberto e pesquisas sobre IDSs.

O local mais indicado para se utilizar o modelo IDMEF é entre um Analisador e um Gerenciador, pois o envio de um alerta gerado por um Analisador para um Gerenciador poderá ser implementado através do modelo IDMEF. Porém existem outros lugares onde o IDMEF pode ser utilizado:

- em uma base de dados que armazena informações de detecção de intrusão de vários produtos diferentes, o IDMEF ajudaria na análise dos dados e na geração de relatórios unificados;
- em sistemas de correlação de eventos que recebem alertas de vários IDSs e realizam cálculos para tentar descobrir e informar a relação existente entre estes alertas;
- em uma interface gráfica que mostre ao usuário os alertas gerados por diversos IDSs, com isto o usuário pode gerenciar vários IDSs de um único monitor e não precisa aprender a sintaxe / semântica de todos os IDSs que está monitorando, apenas do IDMEF;
- poderá ser utilizado entre as organizações para facilitar a comunicação sobre intrusões ocorridas, sendo um formato pré-definido para estes tipo de comunicação.

O IDMEF é um modelo orientado a objetos que contém as informações de um alerta gerado por um Analisador para um Gerenciador de um sistema de detecção de intrusão. É importante ressaltar que o modelo não define como o alerta deve ser criado, ele define apenas o formato das informações do alerta, cada Analisador deve decidir

como criar os seus alertas utilizando os recursos do modelo de dados IDMEF da melhor maneira possível.

Para produzir um formato de dados único a partir de vários IDSs diferentes, alguns problemas associados à representação de alertas de intrusão precisaram ser analisados e resolvidos. Um deles é o fato de que os diferentes IDSs geram diferentes quantidades de informação para identificarem os seus alertas, alguns alertas são identificados por uma pequena quantidade de informações, como origem, destino, nome e hora em que o alerta ocorreu. Outros são identificados por informações mais detalhadas como portas, serviços, processos, usuários, etc. Para suportar esta diversidade o modelo precisa representar as informações de alerta com o máximo de flexibilidade. A orientação a objetos provê a flexibilidade necessária ao modelo IDMEF, através dos mecanismos de herança e agregação.

Outro problema a ser analisado está relacionado com a forma como os IDSs capturam informações. Alguns IDSs analisam o tráfego de rede, outros trabalham com *logs* do sistema operacional ou de aplicativos. Com isto, alertas sobre o mesmo ataque poderão conter diferentes informações de origem e destino, se vierem de diferentes tipos de Analisadores. Para suportar este tipo de situação o modelo define classes para acomodar as diferentes informações de origem e destino.

As extensões que podem ser feitas ao modelo IDMEF original permitem que sejam transportadas desde informações simples até complexas informações sobre alertas, estas extensões são feitas através dos mecanismos de herança e agregação da orientação a objetos. Através das extensões, pode ser feita a herança de uma classe existente no modelo IDMEF original para acomodar informações de alerta de uma rede ou sistema operacional específico, ou IDSs comerciais podem definir novas classes para acomodar informações mais detalhadas em seus produtos sobre certos tipos de ataques.

As informações contidas dentro do modelo de dados IDMEF não devem possuir sentido ambíguo e devem permitir a inserção de informações extras para determinados alertas, em campos específicos para este fim.

Um analisador pode gerar alertas simples, que são gerados a partir de uma única ação não permitida, por exemplo, a digitação de uma senha incorreta. Ou alertas complexos, que são gerados a partir de um conjunto de eventos detectados, por exemplo, a tentativa de conexão de diversas portas diferentes em um curto espaço de

tempo. O modelo deve prover uma forma de se identificar um alerta simples dentro de um alerta complexo, quando este for formado por vários alertas simples.

Inicialmente o IDWG possuía duas propostas para a implementação do modelo IDMEF, uma através de SMI (*Structured of Management Information*) com o desenvolvimento de uma MIB, e outra através de XML com o desenvolvimento de um DTD (*Document Type Definition*). Após várias reuniões, em Fevereiro de 2000 o grupo IDWG escolheu o formado XML. Este formato foi escolhido por ser mais flexível, estar presente em um grande número de implementações comerciais e livres de diversas naturezas e melhor se adequar aos requisitos especificados em [WOOD 2002].

4.2 As Classes do Modelo

A seguir serão apresentadas as classes que compõe o modelo IDMEF, através de diagramas UML. O DTD que especifica o modelo de dados IDMEF para o formato XML é apresentado no Anexo 2, neste documento estão especificadas todas as classes do modelo com seus atributos e valores possíveis.

A classe base de todo o modelo IDMEF é a classe *IDMEF-Message*, todas as classes do modelo derivam desta classe. A classe *IDMEF-Message* possui um atributo chamado “version”, que armazena a versão do modelo, possui também duas classes subordinadas, as classes Alert e Heartbeat.

A Figura 6 apresenta um diagrama UML com o relacionamento entre as classes *IDMEF-Message*, Alert e Heartbeats, nesta figura também constam as classes Analyser, CreateTime e AdditionalData, que são classes agregadas e comuns às classes Alert e Heartbeats.

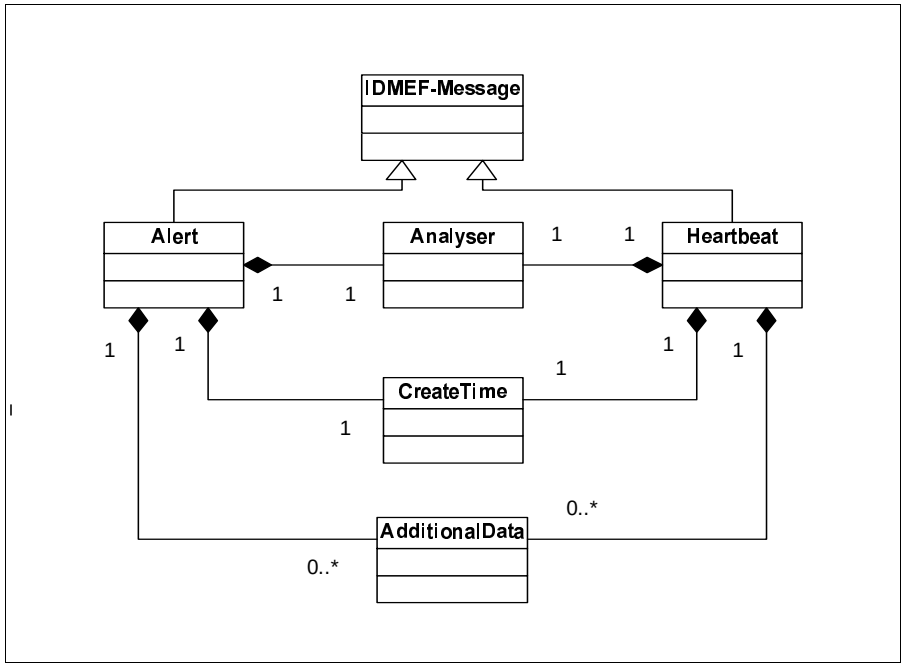

Figura 6 – Visão geral das classes IDMEF.

As classes Alert e Heartbeats representam os dois tipos básicos de mensagem que o IDMEF suporta. A classe Alert é utilizada para representar um alerta e a classe Heartbeats é utilizada para um Analisador informar a um Gerenciador as suas condições de status.

4.2.1 A Classe Alert

Quando um Analisador deseja enviar um alerta para um Gerenciador ele irá criar uma classe do tipo Alert, preencher as informações contidas nesta classe e enviar estas informações ao seu Gerenciador.

A classe Alert possuí um atributo opcional chamada “ident”, este atributo contém um identificador único para o alerta, esta classe também possui várias classes agregadas e três classes subordinadas, conforme apresentado na Figura 7.

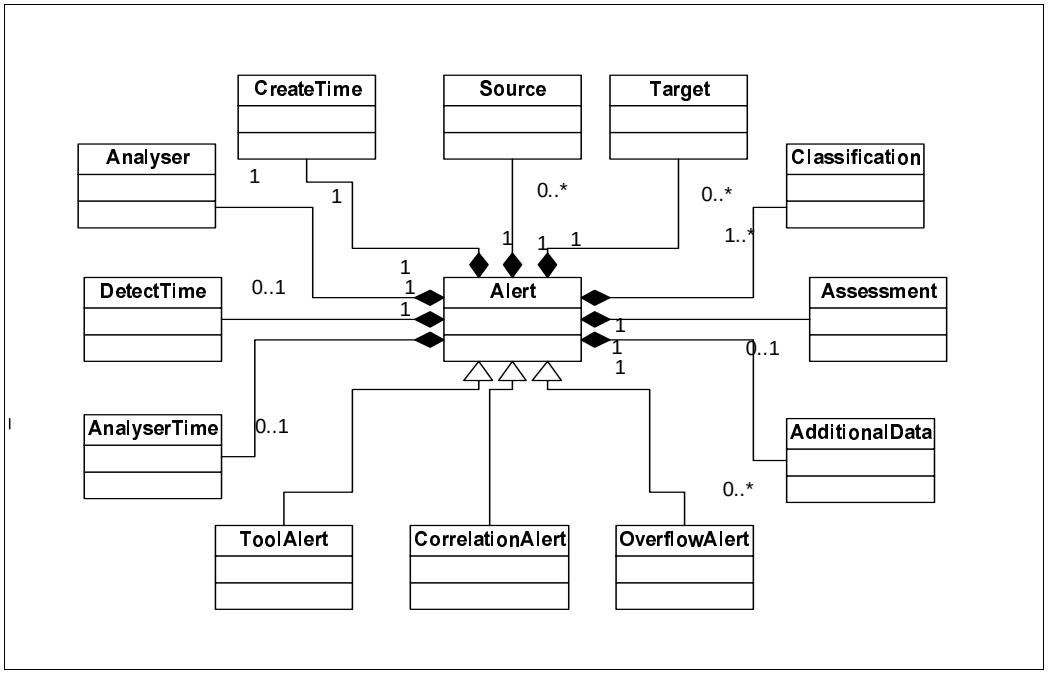

Figura 7 – Relacionamentos da classe Alert.

ToolAlert é uma especialização da classe Alert que adiciona informações específicas para ataques realizados através de ferramentas ou programas como por exemplo os cavalos de tróia. Um Analisador pode enviar um alerta deste tipo quando conseguir identificar o tipo de ferramenta ou programa que está realizando o ataque. A classe ToolAlert possui três classes agregadas, conforme apresentado na Figura 8.

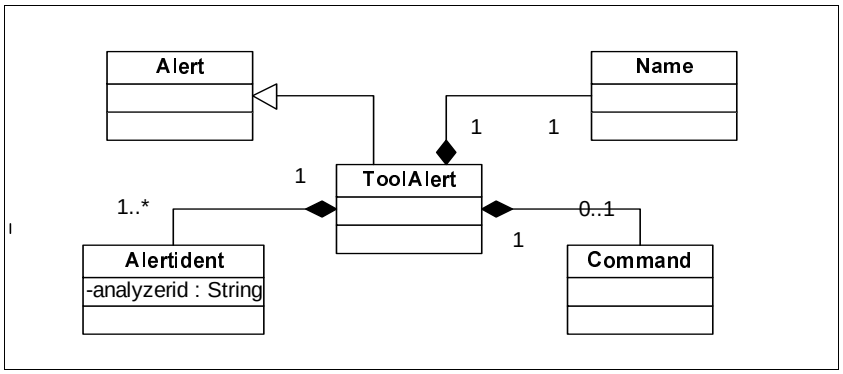

Figura 8 – Relacionamentos da classe ToolAlert.

A classe Name armazena uma *string* utilizada para agrupar os ataques realizados por ferramentas, normalmente esta classe contém o nome da ferramenta que está realizando o ataque. A classe command armazena uma *string* que contém o comando que a ferramenta executou para realizar o ataque.

Ataques realizados com ferramentas podem gerar vários alertas, devido a isto a classe ToolAlert possui uma subclasse chamada alertident, esta é uma classe do tipo *string* que contém um identificador de outro alerta que está relacionado com o alerta atual. A classe Alertident possui o atributo analyzerid para distinguir os alertas gerados por diferentes Analisadores. Um ToolAlert pode conter várias instâncias da classe alertident, formando assim uma lista de alertas relacionados com o alerta atual.

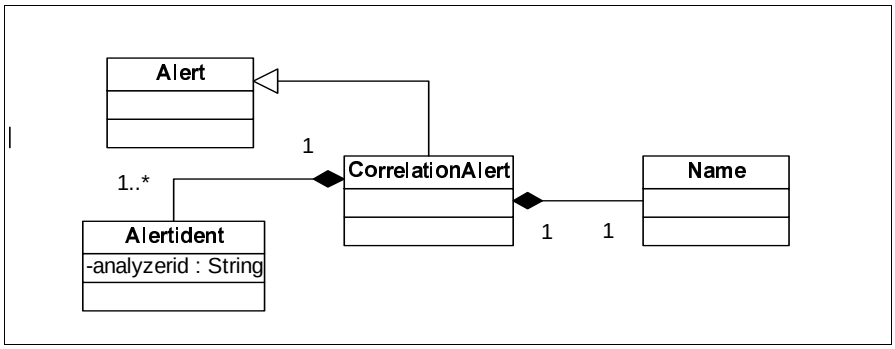

Figura 9 – Relacionamentos da classe CorrelationAlert.

CorrelationAlert é uma especialização da classe Alert que adiciona informações que permitem a correlação entre mensagens de alerta. Este tipo de alerta é enviado quando um Analisador precisa criar vários alertas para um mesmo ataque, neste caso os alertas precisa ser correlacionados pelo Gerenciador. A classe CorrelationAlert possui um relacionamento de agregação com as classes Name e Alertident, conforme apresentado na Figura 9.

OverflowAlert é uma especialização da classe Alert que adiciona informações específicas para ataques do tipo *overflow*, esta classe possui três classes agregadas, conforme apresentado na Figura 10.

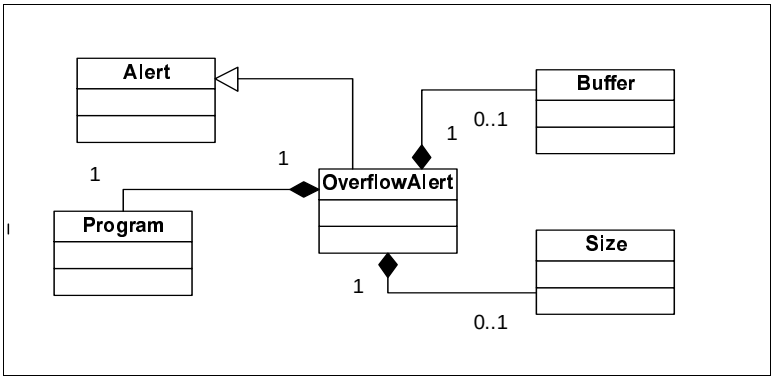

Figura 10– Relacionamentos da classe OverflowAlert.

A classe Program é uma *string* que armazena o nome do programa que está sendo utilizado para realizar o ataque de *overflow*. A classe size armazena a quantidade, em bytes, de informações que o atacante enviou para causar o *overflow*. A classe buffer é um conjunto de bytes que armazena os dados contidos no *overflow*.

4.2.2 A Classe Heartbeat

Os Analisadores utilizam as classes Heartbeat para informar aos Gerenciadores sobre suas condições de funcionamento. As mensagens Heartbeat devem ser enviadas em períodos regulares de tempo, estas indicam ao Gerenciador que o Analisador está ativo, quando uma mensagem Heartbeat não é enviada dentro do período de tempo configurado, o Gerenciador entende que o Analisador não está ativo.

Todos os Gerenciadores devem suportar o recebimento de mensagens Heartbeat, porém o envio destas mensagens por parte dos Analisadores é opcional. Um Gerenciador deve possuir a possibilidade de configuração de utilização ou não de mensagens Heartbeat, para cada um de seus Analisadores.

A classe Heartbeat possui quatro classes agregadas, relacionadas a ela conforme apresentado na Figura 11.

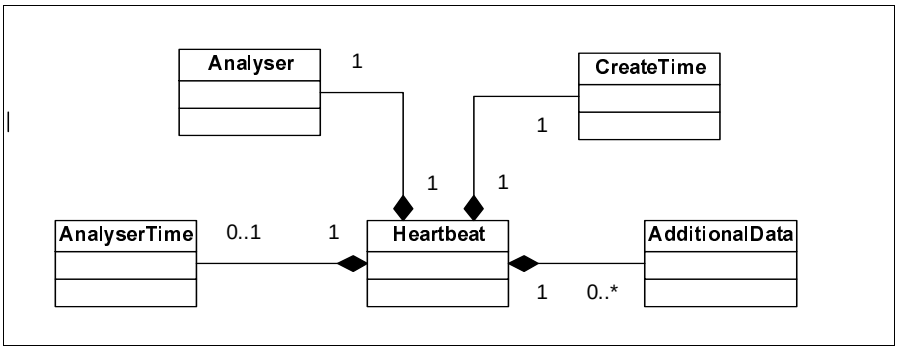

Figura 11 – Relacionamentos da classe Heartbeat.

4.2.3 As Classes Auxiliares

As classes auxiliares, que são utilizadas diretamente pelas classes Alert e Heartbeat, são as classes Analyser, Classification, Source, Target, Assessment, AdditionalData, CreateTime, DetectTime e AnalyserTime.

A classe Analyser identifica o Analisador que gerou a mensagem (Alert ou Heartbeat). Somente um Analisador pode ser identificado pela classe Analyser, o modelo de dados IDMEF não permite a identificação de hierarquias de Analisadores.

Conforme apresentado na Figura 12, a classe Analyser possui duas classes agregadas e sete atributos.

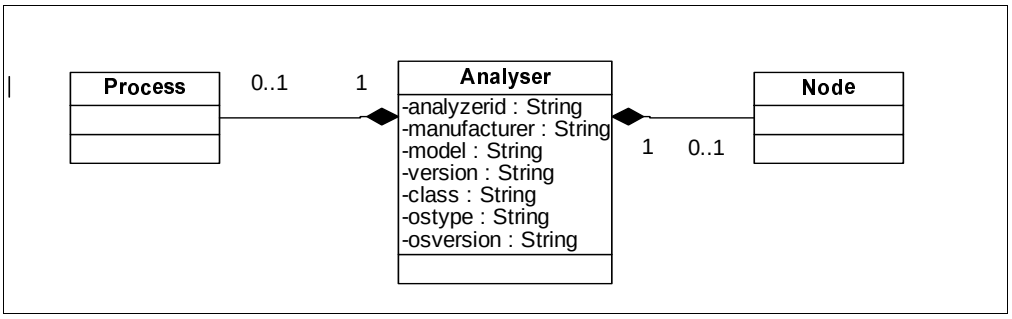

Figura 12 – Relacionamentos da classe Analyser.

Os atributos da classe Analyser apresentados na Figura 12 contém respectivamente: um identificador único, uma descrição do fabricante, uma descrição do modelo, a versão e a classificação do Analisador, além do tipo e versão do sistema operacional em que o Analisador atua.

A classe Classification prove informações que permitem ao Gerenciador classificar o tipo de alerta que está sendo recebido. A classe Classification possui um atributo e duas classes agregadas.

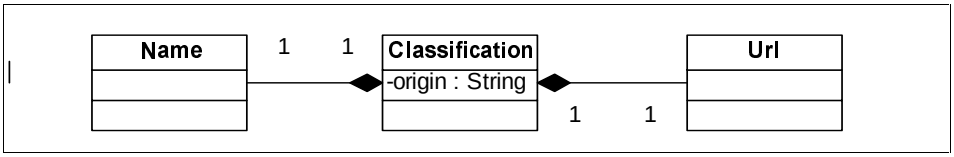

Figura 13 – Relacionamentos da classe Classification.

A classe Name é uma *string* que contém o nome do alerta. O atributo origin da classe Classification informa de onde se origina o nome do alerta que estará descrito na classe Name, os valores para o atributo origin são:

- unknown: origem do nome do alerta desconhecida;
- bugtraqid: o nome do alerta se origina de SecurityFocus.com;
- cve: o nome do alerta se origina de cve.mitre.org;
- vendor-specific: o nome do alerta se origina do endereço especificado na classe url.

A classe auxiliar Source contém informações sobre a origem do evento que gerou o alerta, esta classe possui quatro classes agregadas e três atributos, conforme mostra a Figura 14.

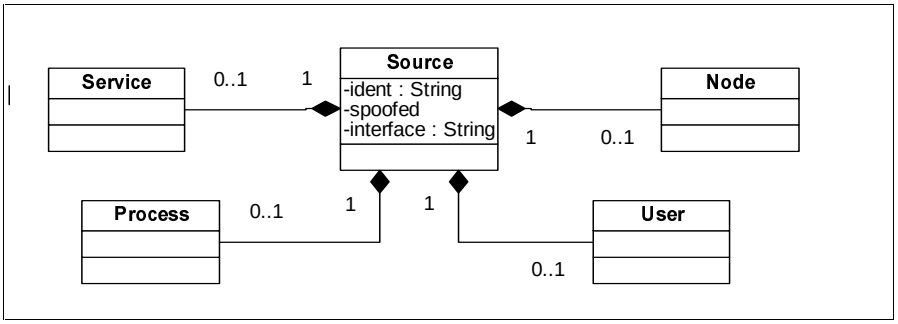

Figura 14 – Relacionamentos da classe Source.

O atributo ident é um atributo opcional que identifica de maneira única cada instância da classe Source. O atributo spoofed é utilizado para o Analisador determinar se as informações de origem do ataque são falsas ou verdadeiras, este atributo pode conter valores “unknown”, “yes” e “no”, que especificam respectivamente que o Analisador não conseguiu determinar se as informações são falsas ou verdadeiras, que o Analisador acredita que as informações são falsas e que o Analisador acredita que as informações são verdadeiras.

O atributo interface é utilizado para que um Analisador baseado em rede informe qual das suas interfaces originou o alerta.

Semelhante à classe Source, a classe Target contém informações sobre o destino do evento que originou a geração do alerta, esta classe possui três atributos com as mesmas funções e valores dos atributos da classe Source, e cinco classes agregadas conforme mostrado na Figura 15.

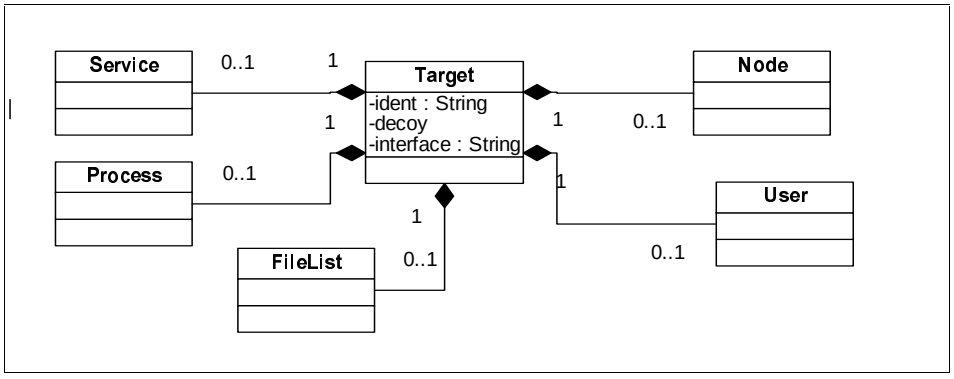

Figura 15 – Relacionamentos da classe Target.

A classe Assessment, apresentada na Figura 16, provê informações que permitem ao Gerenciador a avaliação do evento que gerou o alerta. Esta classe não possui atributos, porém possui três classes agregadas: Impact, Action e Confidence.

Através da classe Impact é possível determinar o impacto do evento sobre o sistema, o atributo severity desta classe pode conter valores de impacto baixo, médio e alto sobre a segurança do sistema. Outro atributo da classe Impact é o Completion, que pode conter os valores *failed* e *succeeded*, informando se o evento falhou ou foi completado com sucesso. O terceiro atributo desta classe é o *type*, que pode conter os seguintes valores:

- admin: tentativa ou obtenção de privilégios administrativos;
- dos: tentativa ou realização de ataque de negação de serviços;
- file: tentativa ou realização de ações sobre arquivos;
- recon: tentativa ou realização de ações de reconhecimento do sistema;
- user: tentativa ou obtenção de privilégios de usuários;
- other: nenhuma das categorias acima foi identificada.

Também agregada à classe Assessment, a classe Action descreve as ações que já foram tomadas pelo Analisador em resposta ao evento. A classe Action possui somente o atributo categoty, que pode conter os seguintes valores:

- block-installed: foi feito um bloqueio (porta, endereço, conta, etc) no sistema para impedir o ataque;
- notification-sent: foi enviada uma notificação (pager, E-mail, etc), o envio do alerta não é considerado notificação;
- taken-offline: o sistema, computador ou usuário envolvido com o ataque foi tirado de funcionamento;
- other: foi tomada uma ação que não se enquadra nas categorias anteriores.

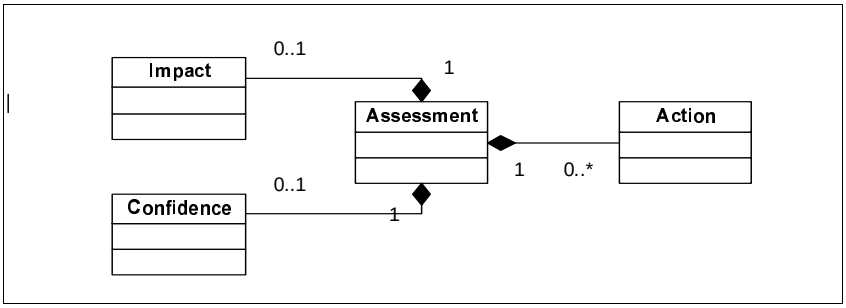

Figura 16 – Relacionamento da classe Assessment.

A classe Confidence possui somente o atributo rating, este atributo informa o grau de confiança das informações prestadas pelo Analisador. O atributo rating pode conter

os valores baixo, médio e alto de confiança das informações, ou um valor numérico que especifica o percentual de confiança.

Para um Analisador acrescentar informações que não estão previstas no modelo IDMEF, é utilizada a classe AdditionalData. Esta classe possui dois atributos: *type* e meaning. O primeiro informa o tipo das informações adicionais (byte, integer, real, character, etc). Segundo contém uma *string* que descreve o significa das informações adicionais.

Estão definidas no modelo de dados IDMEF três classes relacionadas com informações de data e hora. A classe CreateTime informa o momento em que o alerta foi criado pelo Analisador. A classe DetectTime informa o momento em que o evento que originou o alerta foi detectado, o valor desta classe pode ou não ser o mesmo valor da classe CreateTime, pois o Analisador não precisa necessariamente criar o alerta no momento em que o evento foi detectado. A classe AnalyzerTime contém a data e a hora atual do Analisador, isto pode ser utilizado para realizar uma sincronização entre o Analisador e o Gerenciador.

As três classes relacionadas com informações de data e hora possuem apenas um atributo chamado ntpstamp, este atributo especifica o formato da data e hora que estará contida nos elementos XML da respectiva classe.

4.2.4 As Classes de Suporte

As classes de suporte são as classes mais relevantes utilizadas pelas classes auxiliares e compartilhadas entre elas.

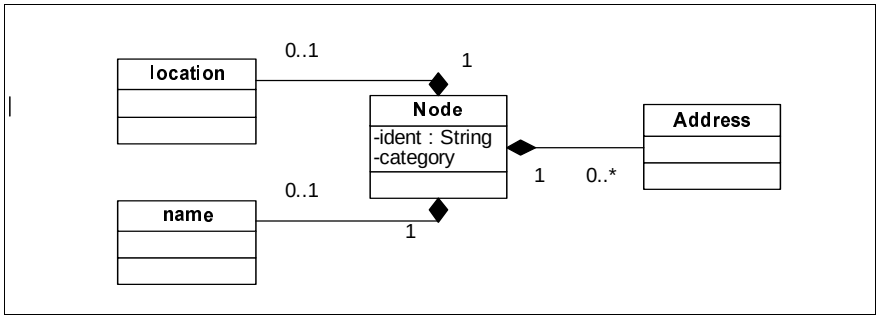

Figura 17 – Relacionamentos da classe Node.

A classe Node é uma classe de suporte agregada às classes Analyser, Target e Source, esta classe contém informações que identificam dispositivos de rede como

hosts, roteadores ou comutadores. A classe Node possui dois atributos e três classes agregadas, conforme apresentado na Figura 17.

O atributo ident é um identificador único para cada instância da classe Node. O atributo category identifica o ambiente em que o dispositivo de rede atua.

Dentre as classes agregadas à classe Node, a classe Location é uma *string* que contém a localização do dispositivo, e a classe Name contém uma *string* com o nome do equipamento. A classe Address é utilizada para representar a rede em que o dispositivo se encontra, esta classe possui quatro atributos e duas classes agregadas conforme demonstrados no diagrama UML da Figura 18.

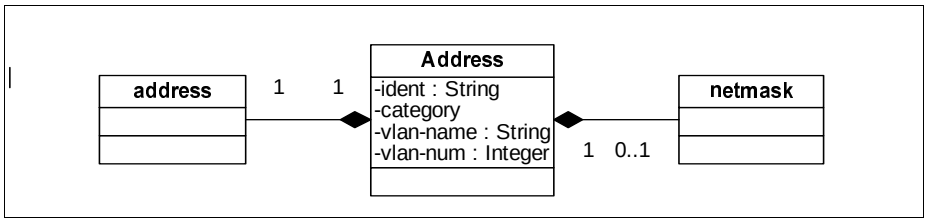

Figura 18 – Relacionamentos da classe Address.

O atributo ident trata-se de um identificador único semelhantes aos já apresentados em outras classes. O atributo category identifica o tipo de endereço de rede que está sendo utilizado, este atributo pode conter valores que identificam endereços de rede IPv4, IPv6, ATM, MAC, etc. Os atributos vlan-name e vlan-num contêm um nome e número que identifica a rede ao qual o endereço pertence.

A classe address, agregada à classe Address, é uma *string* que contém um endereço, o formato do endereço contido nesta classe é determinado pelo atributo category da classe Address. A classe netmask é uma *string* que contém a máscara de rede, quando esta informação for apropriada para o endereço utilizado.

Outra importante classe de suporte é a classe User, utilizada pelas classes Source e Target. A classe User é utilizada para descrever um usuário do ambiente, esta classe possui dois atributos e uma classe agregada, conforme apresentado pela Figura 19.

O primeiro atributo da classe User é um identificador (ident). O segundo atributo, category, indica se o usuário é de um aplicativo, do sistema operacional ou desconhecido.

A classe UserId, além do atributo ident, contém um atributo *type*, que informa o tipo de usuário que está sendo tratado, esta classe também possui duas classe agregadas,

a classe Name e a classe Number, que armazenam respectivamente o nome e o número do usuário.

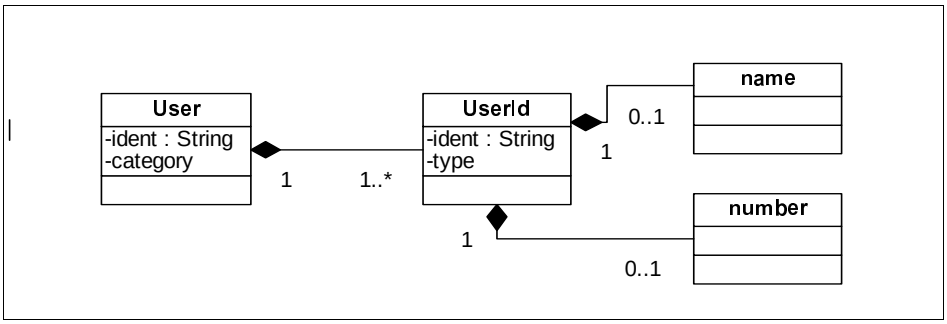

Figura 19 – Relacionamentos da classe User e UserId.

A classe de suporte Process é utilizada pelas classes auxiliares Analyser, Source e Target. As informações contidas na classe Process representam um processo que está sendo executado, esta classe possui um atributo ident e cinco classes agregadas, conforme apresentado na Figura 20.

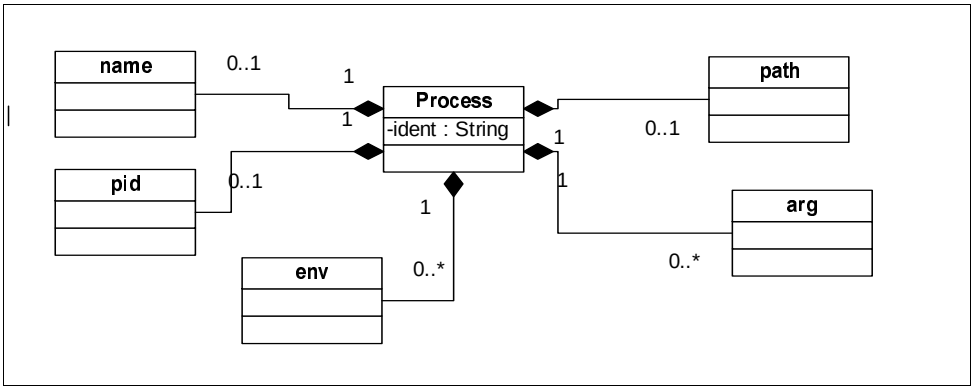

Figura 20 – Relacionamentos da classe Process.

As classes name, path, arg e env são *string* que contém respectivamente o nome do programa que está executando, o seu caminho, as linhas de comando envolvidas com a sua execução e as variáveis de ambiente relacionadas com o programa em questão. A classe pid contém o identificador de processo do programa.

Service é uma classe de suporte responsável por descrever os serviços de rede da origem e do destino (Source e Target). Conforme demonstrado na Figura 21, a classe Service possui um atributo identificador, quatro classes agregadas e duas classes especializadas. As classes agregadas name, port, portlist e protocol armazenam respectivamente o nome, o número da porta utilizada, uma lista de número de portas auxiliares utilizadas e o protocolo referentes ao serviço em questão.

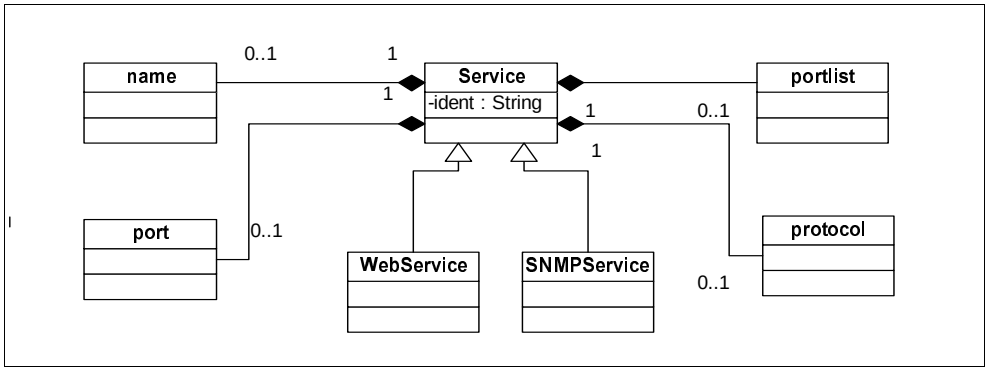

Figura 21 – Relacionamentos da classe Service.

A classe WebService adiciona informações específicas para serviços web, como: descrição de requisições URL e CGI, comandos HTTP e argumentos de scripts CGI.

A classe SNMPService adiciona informações específicas para serviços SNMP, como identificador de objetos, comunidade e comando SNMP enviado ao servidor, além de informações específicas do SNMPv3, como: nome de segurança do objeto, contexto do nome do objeto e identificador do contexto do objeto.

Para descrever os arquivos relacionados com o destino (Target) do evento, é utilizada a classe de suporte FileList.

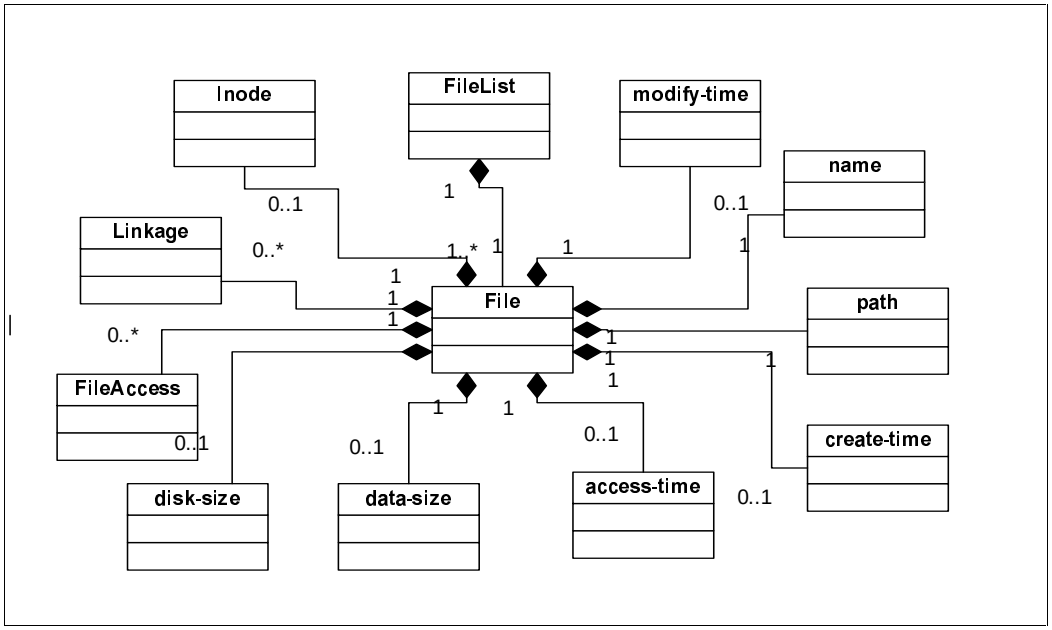

Figura 22 – Relacionamentos das classes FileList e File.

Conforme apresentado no diagrama UML da Figura 22, a classe FileList possui agregada a si apenas a classe File. A classe File possui dez classes agregadas, as quais provêm diversas informações sobre um arquivo, como nome, localização, data de

criação, modificação e último acesso, tamanho dos dados e tamanho em disco, e permissões.

4.3 Mensagens IDMEF em XML

Para serem enviados alertas através do protocolo IDXP, o modelo de dados IDMEF é mapeado para o formato XML conforme o DTD exposto no Anexo 2.

O Quadro 19 apresenta o exemplo de um alerta no formato XML e especificado de acordo com o modelo IDMEF. No exemplo, um Analisador baseado em rede detectou um ataque de “ping da morte”, este é um ataque de negação de serviço caracterizado pelo recebimento de uma solicitação de ping com endereço de origem falso, com isto o servidor tentará responder ao ping sem sucesso, consumindo os seus recursos em uma solicitação falsa. Caso vários destes pings sejam recebidos, o servidor poderá ficar sem recursos para atender às solicitações legítimas, causando assim a negação de seus serviços.

```
   <?xml version="1.0" encoding="UTF-8"?>

   <!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFC XXXX IDMEF
v1.0//EN" "idmef-message.dtd">

   <IDMEF-Message version="1.0">
     <Alert ident="abc123456789">
       <Analyzer analyzerid="bc-sensor01">
         <Node category="dns">
           <name>sensor.example.com</name>
         </Node>
       </Analyzer>
       <CreateTime ntpstamp="0xbc71f4f5.0xef449129">
         2000-03-09T10:01:25.93464Z
       </CreateTime>
       <Source ident="a1a2" spoofed="yes">
         <Node ident="a1a2-1">
           <Address ident="a1a2-2" category="ipv4-addr">
             <address>192.0.2.200</address>
           </Address>
         </Node>
       </Source>
       <Target ident="b3b4">
         <Node>
           <Address ident="b3b4-1" category="ipv4-addr">
             <address>192.0.2.50</address>
           </Address>
         </Node>
       </Target>
       <Target ident="c5c6">
         <Node ident="c5c6-1" category="nisplus">
           <name>lollipop</name>
         </Node>
```

```
      </Target>
      <Target ident="d7d8">
        <Node ident="d7d8-1">
          <location>Cabinet B10</location>
          <name>Cisco.router.b10</name>
        </Node>
      </Target>
      <Classification origin="cve">
        <name>CVE-1999-128</name>
        <url>http://www.cve.mitre.org/</url>
      </Classification>
    </Alert>
  </IDMEF-Message>
```

Quadro 19 – Exemplo de mensagem IDMEF em XML.

No Quadro 19 observamos que o primeiro elemento definido é o *IDMEF-Message*, que é a classe base de todo o modelo, este contém um elemento chamada Alert, indicando que esta mensagem representa um alerta.

O elemento Alert contém os elementos (classes) Analyser, CreateTime, Source, Target e Classification, o elemento Target aparece três vezes, isto significa que o Analisador identificou que este ataque possui três destinos.

No elemento Source, é importante observar que o atributo spoofed possui o valor “yes”, indicando que as informações de origem são falsas.

O elemento Classification especificou o nome do ataque e o local onde este está classificado. Consultando o ataque “CVE-1999-128” em “http://www.cve.mitre.org” observamos que este está cadastrado como um ataque de “ping da morte”.

5 O MODELO PROPOSTO: IDREF

Neste capítulo será apresentando um modelo de ambiente de detecção de intrusão que permitirá aos sistemas de detecção de intrusão enviar respostas aos alertas detectados, este modelo será chamado de IDREF - *Intrusion Detection Response Exchange Format*, ou seja, formato para troca de respostas de detecção de intrusão. O modelo IDREF está baseado nos trabalhos desenvolvidos pelo grupo IDWG relacionados com a comunicação e interoperabilidade entre sistemas de detecção de intrusão, sendo assim, o modelo segue a arquitetura de IDS proposta pelo IDWG, utiliza o protocolo IDXP para comunicação entre os sistemas, e está fortemente relacionado com o modelo de dados IDMEF.

5.1 Introdução

Conforme visto nos capítulos anteriores, a arquitetura de IDS, o protocolo IDXP e o modelo IDMEF propostos pelo IDWG trabalham voltados para a transmissão de alertas de um Analisador para um Gerenciador, eles não prevêem mecanismos para que um Operador tome atitudes com relação a um alerta. O Operador toma conhecimento do alerta, mas se desejar enviar qualquer tipo de resposta contra o alerta deverá fazê-lo por fora do modelo.

O modelo proposto tem por objetivo estender os trabalhos do grupo IDWG de forma a implementar mecanismos para que o Operador possa enviar respostas aos alertas detectados.

Alguns IDSs permitem a configuração de respostas automáticas aos ataques, para estes IDSs o modelo IDMEF possibilita ao Analisador informar o Gerenciador sobre respostas automáticas que foram disparadas, porém nestes casos o Operador apenas recebe informações do que já está sendo feito, o modelo não lhe permite interagir com o ataque.

Um dos motivos que justifica a necessidade das respostas partirem de um Operador, ao invés de serem enviadas automaticamente pelo Analisador, é o fato de que nem todos os alertas representam ataques reais, alguns alertas podem se configurar como falsos positivos (alertas falsos), por isto é necessário que o Operador analise cada alerta e decida quais ações deve tomar.

A geração de respostas por parte do Operador (ao contrário da automática) também permite a especificação de regras de alerta mais restritas, aumentando a segurança do ambiente. Caso um alerta seja um falso positivo, nenhuma resposta será tomada pelo Operador e a operação que gerou o alerta continuará sendo executada normalmente. Porém a mesma operação que em um dado momento foi considerada um falso positivo, em outro momento poderá ser considerada um ataque real, exigindo uma resposta por parte do operador. A decisão de quando um alerta é um falso positivo ou um ataque real é tomada pelo Operador através da análise de cada alerta. Isto justifica a necessidade do Operador disparar respostas a partir de sua estação de gerenciamento.

As respostas não devem ser executadas pela estação de gerenciamento, elas devem ser transmitidas para outro componente da arquitetura do IDS, pois em algumas arquiteturas a estação de gerenciamento pode estar remotamente separada do ambiente que está sendo gerenciado, nestes casos a estação de gerenciamento não tem acesso direto aos recursos da rede gerenciada. Por isto é necessário solicitar a execução da resposta a outro componente, que está localizado dentro da rede gerenciada.

A implementação de um modelo de respostas semelhante ao modelo de alertas IDMEF possibilitará vantagens no gerenciamento dos IDSs, como:

- gerenciamento centralizado das respostas que são enviadas no ambiente, mesmo sendo IDS de fabricantes diferentes;
- cadastramento das respostas dos diferentes IDSs em um formato único, com isto o Operador não precisa conhecer todos os tipos de IDS, basta apenas conhecer o modelo;
- interoperabilidade entre os IDSs do ambiente, o alerta de um IDS pode ser respondido por um componente de outro IDS;
- configuração de respostas automáticas no gerenciador, para todos os Analisadores ligados a ele, independente do tipo de IDS.

A necessidade de transmissão de respostas gerou a adição de novos componentes na arquitetura de IDS proposta pelo IDWG. Também foram feitas alterações no protocolo IDXP para suportar a transmissão de respostas. Para completar o suporte à transmissão de respostas foi desenvolvido o modelo de dados IDREF, este modelo de dados contém as informações referentes às respostas que são trocadas entre os componentes do sistema de detecção de intrusão.

5.2 Arquitetura

Para o desenvolvimento de uma arquitetura de IDS que suporte o envio de respostas, foram feitas as devidas alterações a partir da arquitetura apresentada na seção 2.1. Os relacionamentos e as funcionalidades dos elementos da arquitetura original do IDWG (seção 2.1) foram mantidos inalterados com exceção do elemento Resposta, que possui relacionamentos e funcionalidades mais específicas nesta nova arquitetura.

A Figura 23 apresenta a nova arquitetura de IDS proposta por este trabalho, com suporte ao envio de respostas. Comparando esta arquitetura com a arquitetura original do IDWG (Figura 2) observamos a inclusão de três novos elementos: Contra-Medidas, Ação e Recurso. Além da alteração no elemento Resposta, relacionando-se agora com o Operador, com o Gerenciador e com o novo componente de Contra-Medidas.

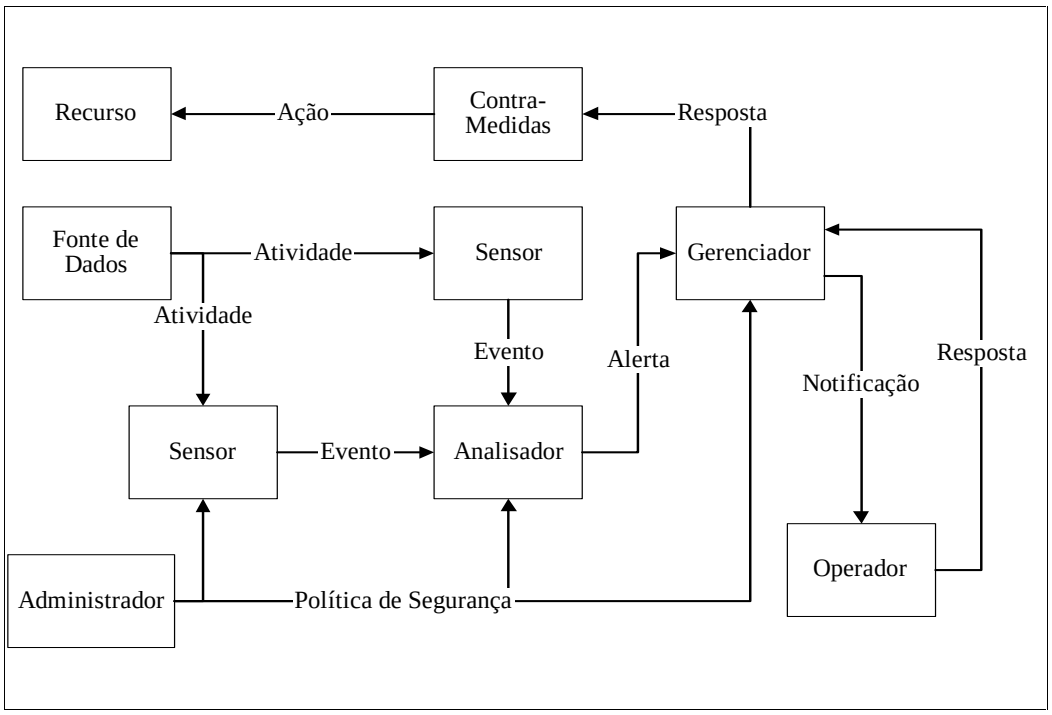

Figura 23 – Arquitetura de IDS com suporte à respostas.

Nesta nova arquitetura, quando o Operador recebe uma notificação do Gerenciador, ele tem a opção de enviar uma resposta de volta ao Gerenciador, esta resposta informa ao gerenciador medidas que devem ser adotadas no ambiente para conter um ataque. O Operador pode acionar uma resposta através de itens de menu, telas ou ícones implementados no gerenciador para este fim.

Quando o Gerenciador recebe uma resposta do Operador, ele codifica esta resposta de acordo com o modelo IDREF e a transmite através do protocolo IDXP para o componente de Contra-Medidas do IDS.

O componente de Contra-Medidas sabe receber e interpretar as respostas enviadas pelo Gerenciador, pois ele conhece o protocolo IDXP e o modelo IDREF. A função do componente de Contra-Medidas é aplicar ações a Recursos do ambiente, outra função deste componente é armazenar em um *log* todas as respostas que foram executadas, para que possam ser feitas auditorias posteriores no ambiente. Em alguns IDSs o componente de Contra-Medidas pode estar implementado junto com o componente Analisador ou com o componente Sensor.

Um Recurso é qualquer elemento do ambiente, seja este hardware ou software, que possa estar relacionado com um ataque ou que precise sofrer ações para que um ataque seja controlado. Exemplos de Recursos podem ser contas de usuário, sessões de usuário, roteadores, *firewalls*, processos do sistema operacionais ou arquivos.

Uma ação é algo que precisa ser feito no ambiente para que um ataque seja controlado. Exemplos de ações podem ser o bloqueio ou fechamento de algum recurso, ou o envio de pacotes através da rede para conter a origem do ataque.

Uma resposta sempre especifica uma ou mais ações que devem ser aplicadas a Recursos do ambiente. Quando o componente de Contra-Medidas recebe uma resposta do Gerenciador, ele deve interpretar as informações contidas nesta resposta e aplicar as ações em seus respectivos recursos.

5.3 Protocolo de Comunicação

Para a transmissão de respostas entre o Gerenciador e o componente de Contra-Medidas é utilizado o protocolo IDXP, este protocolo possui mecanismos para prover toda a segurança necessária para a transmissão das respostas, além de atender aos requisitos necessários para comunicações entre sistemas de detecção de intrusão especificados pelo IDWG, conforme apresentado na seção 2.2.

A única alteração no protocolo IDXP feita para melhor se adequar à transmissão respostas está relacionada com a opção *streamType* das mensagens *IDXP-Greeting* (seção 3.2.2). Foi adicionado um novo valor ao atributo *type* para que os canais de comunicação possam ser classificados como canais de transmissão de respostas.

Quando for informado o valor “response” no atributo *type* da opção *streamType*, então o canal será classificado como um canal exclusivo para o trafego de respostas.

5.4 O Modelo de Dados

O modelo de dados IDREF, desenvolvido neste trabalho para possibilitar o envio de respostas em sistemas de detecção de intrusão, está fortemente relacionado com o modelo IDMEF desenvolvido pelo IDWG.

As classes do modelo IDREF foram desenvolvidas baseadas nas informações que um Gerenciador obtém quando recebe um alerta formatado dentro do modelo IDMEF, deste fato resulta no forte relacionamento dentre os dois modelos.

O IDREF possui classes de nome igual às classes do modelo IDMEF, porém o conteúdo destas classes pode ser diferente entre os dois modelos, pois o propósito dos dois modelos é diferente.

O modelo IDMEF tem por propósito reunir o maior conjunto possível de informações úteis sobre o ataque que está ocorrendo, para permitir ao Analisador e ao Operador uma análise mais completa sobre a situação do ambiente. Já o modelo IDREF não possui esta necessidade.

O modelo IDREF deve conter o mínimo de informações possível capazes de possibilitar a execução da resposta desejada pelo Operador, estas informações se resumem à especificação da ação a ser executada e ao endereçamento do recurso ao qual a ação será aplicada.

Além das informações necessárias para a execução da resposta, uma mensagem IDREF também deve conter informações que identifiquem de maneira exclusiva o Gerenciador que enviou a mensagem e informações que sejam relevantes para futuras auditorias.

O DTD que especifica o modelo de dados IDREF para o formato XML é apresentado no Anexo 3, neste documento estão especificadas todas as classes do modelo com seus atributo e valores possíveis.

5.4.1 As Classes Principais

A classe base do modelo IDREF é a classe IDREF-Message, todas as classes do modelo derivam desta classe. A classe IDREF-Message possui um atributo chamado

version que indica a versão do modelo de dados e um atributo chamado ident, que é utilizado como identificação única para todas as mensagens de resposta geradas.

A Figura 24 apresenta os relacionamentos da classe IDREF-Message, com suas cinco classes agregadas: alertident, CreateTime, description, AddicionalData e Manager; e três classes derivadas: Response, React e Config.

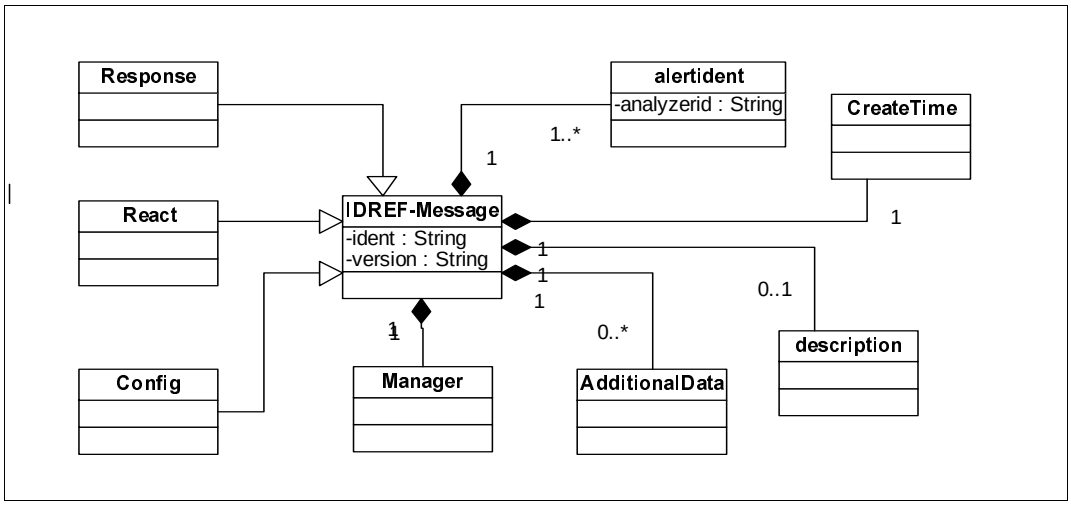

Figura 24 – Visão geral do modelo IDREF.

As classes alertident, CreateTime e AdditionaData possuem a mesma estrutura das classes de mesmo nome do modelo IDMEF, diferindo apenas quanto ao significado de duas informações.

A classe alertident armazena a identificação do alerta do modelo IDMEF que causou a geração da resposta, esta classe possui o atributo analyserid para distinguir entre alertas de diferentes Analisadores, semelhante à funcionalidade empregada no modelo IDMEF. As informações contidas na classe alertident são importantes para permitir futuros relacionamentos entre alertas e respostas.

A classe CreateTime armazena o momento em que a resposta foi criada e a classe AddicionalData pode ser utilizada para transportar informações que não estão previstas no modelo IDREF, os dois atributos desta classe (*type* e meaning) possuem o mesmo significado já apresentado no modelo IDMEF.

A classe description contém uma *string* com uma descrição simples da resposta que está sendo aplicada ao sistema. Observações e detalhes específicos da resposta ou informações que sejam úteis para futuras análises do *log* de respostas gerado pelo componente de Contra-Medidas também podem ser colocadas nesta *string*.

A classe Manager identifica o gerenciador que gerou a resposta e será vista em detalhes na seção 5.4.2.

Uma resposta pode ser de três tipos, correspondendo às três classes derivadas da classe IDREF-Message: Response, React e Config.

A primeira classe derivada da classe IDREF-Message é a classe Response, esta classe contém informações que devem ser enviadas para controlar ou avisar sobre o ataque. A classe Response possui três classes derivadas e uma classe agregada, conforme apresentado na Figura 25. As classes TCP e ICMP são utilizadas para enviar informações através da rede, geralmente para a origem do ataque. E a classe notify é utilizada quando se deseja avisar alguém externo à arquitetura do IDS sobre a ocorrência de um ataque.

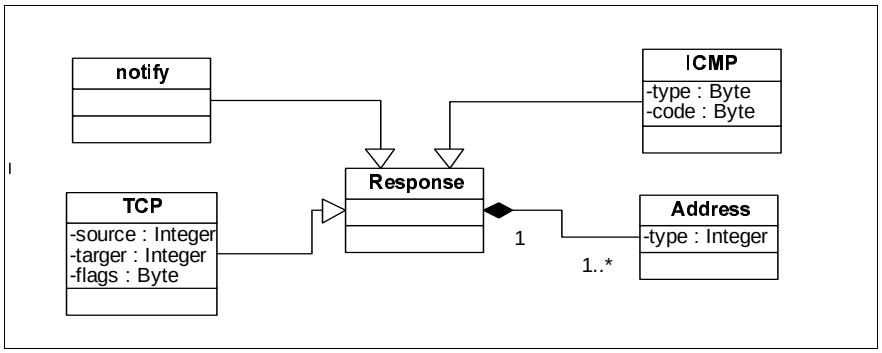

Figura 25 – Relacionamentos da classe Response.

A classe TCP indica que deve ser enviado um pacote TCP pela rede como resposta a um alerta ocorrido. Esta classe possui os atributos source, target e flags, que correspondem às informações que devem ser colocadas no pacote TCP que será enviado. Como exemplos de respostas utilizando a classe TCP, podem ser enviados pacotes com flags para resetar ou para fechar conexões de transporte, na origem ou destino da conexão.

A classe ICMP indica que deve ser enviada uma mensagem ICMP pela rede como resposta a um alerta ocorrido. Esta classe possui os atributos *type* e code, que correspondem às informações que devem ser colocadas na mensagem ICMP e definem o significado da mensagem que será enviada. Como exemplos de respostas utilizando a classe ICMP, podem ser enviadas mensagens ICMP de rede, *host* ou porta não encontrada para a origem do ataque.

A classe notify é do tipo *string* e contém um texto com informações sobre o ataque, estas informações podem ser inclusive o próprio alerta IDMEF que gerou a resposta. Este tipo de resposta pode ser configurado no Gerenciador como um aviso para o Operador quando ele não estiver presente, neste caso esta seria uma resposta automática disparada pelo Gerenciador.

Agregada à classe Response, a classe Address é utilizada para endereçar o destino da resposta TCP, ICMP ou notify. O atributo *type* indica o tipo de endereço utilizado para enviar a resposta, para as classes TCP e ICMP este atributo pode conter os mesmos valores especificados no atributo category da classe Address do modelo IDMEF, para a classe notify este atributo pode conter os valores “fone”, “E-mail” e “page”. A classe Address é do tipo *string*, sendo que o seu conteúdo corresponde ao endereço de destino da resposta, de acordo com o tipo especificado no atributo *type*.

A segunda classe derivada da classe IDREF-Message é a classe React, esta classe representa uma reação do ambiente contra o ataque, ou seja, alguma atitude será tomada no ambiente para conter o ataque. A classe React possui duas classe agregadas, conforme apresentado na Figura 26.

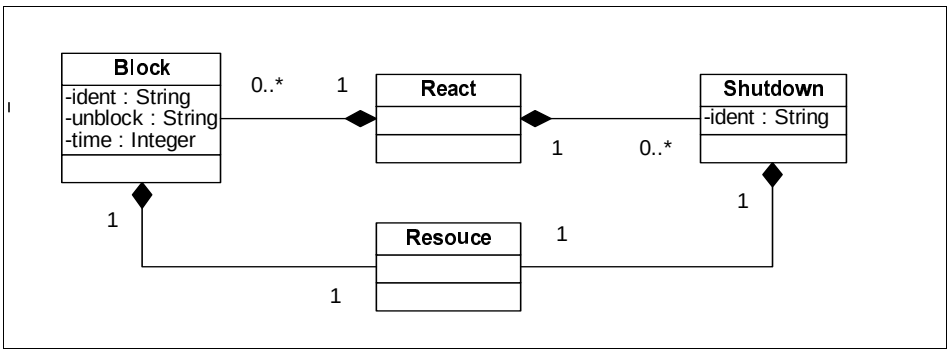

Figura 26 – Relacionamentos da classe React.

Uma reação pode ser o bloqueio ou o fechamento de um recurso. Quando um recurso é bloqueado ele não pode ser utilizado pelo ambiente por um determinado espaço de tempo. Quando um recurso é fechado, ele pode ser aberto novamente a qualquer momento sem nenhuma restrição.

Em uma reação a responsabilidade da execução da resposta é do componente de Contra-Medidas, ou seja, ele deve saber como bloquear ou fechar os recursos do ambiente.

As classes Block e Shutdown representam respectivamente o bloqueio e o fechamento de algum recurso. Estas duas classes possuem um atributo chamado ident que é utilizado para identificar de maneira única cada bloqueio ou fechamento, elas também possuem agregadas a si a classe Resource, que representa um recurso.

Além do atributo ident, a classe Block contém os atributos unblock e time. O atributo unblock indica quando o recurso deve ser desbloqueado, este atributo pode conter os valores “reset”, indicando que o recurso deve ser reinicializado para ser desbloqueado, e “time”, indicando que o recurso deve ficar bloqueado pelo período de tempo especificado no atributo time. O atributo time especifica em minutos o período de tempo que o recurso deve ficar bloqueado.

Uma resposta React pode conter uma ou mais solicitações de bloqueio ou fechamento, porém cada solicitação de bloqueio ou fechamento está relacionada com um único recurso, pois a identificação e os atributos de bloqueio ou fechamento devem estar relacionados com um recurso específico.

Como exemplo de respostas do tipo React temos o bloqueio de um arquivo do sistema operacional, o bloqueio de um equipamento da rede, o fechamento de uma sessão de usuário ou o fechamento de um processo do sistema operacional.

O terceiro tipo de resposta é enviado através da classe Config, esta classe representa uma alteração na configuração de algum recurso do ambiente. A classe Config possui duas classe agregadas, conforme apresentado na Figura 27.

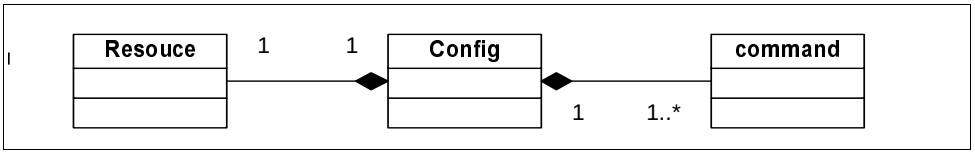

Figura 27 – Relacionamento da classe Config.

A classe command é do tipo *string* e contém um comando a ser executado pelo recurso que será configurado. Uma classe Config pode possuir uma ou várias instâncias da classe command. A classe Resource representa o recurso a ser configurado.

Na resposta de configuração o componente de contra-medida não precisa saber como executar o comando que lhe foi passado, ele precisa apenas saber como solicitar ao recurso, ou ao responsável pelo recurso, a execução do comando contido na resposta.

Como exemplo de respostas de configuração temos a alteração de permissões de usuários ou arquivos, a reconfiguração de *firewalls* ou serviços e a ativação de dispositivos auxiliares de segurança.

5.4.2 As Classes Auxiliares

Semelhante à classe Analyser do modelo IDMEF, a classe Manager identifica o gerenciador que enviou uma determinada resposta. Para fazer esta identificação são utilizados o atributo "managerid" e as classes agregadas Node e Process. A Figura 28 apresenta os relacionamentos da classe Manager.

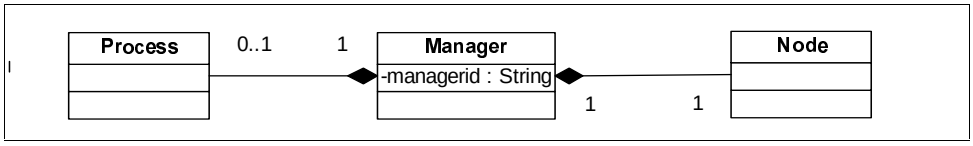

Figura 28 – Relacionamentos da classe Manager.

Para identificar o gerenciador, sempre é enviado o seu ID e as suas informações de rede (classe Node), eventualmente podem ser enviadas informações complementares relacionadas com o processo do gerenciador (classe Process). As informações da classe Manager são armazenadas em um *log* pelo componente de contra-medidas para futuras consultas.

A classe Resource representa um recurso ao qual será aplicada a resposta. Nas respostas tipo Response considera-se que o recurso ao qual se aplica a resposta é a própria rede de origem do alerta, por isto este tipo de resposta não possui relacionamento com a classe Resource. Nas respostas do tipo React a classe Resource está agregada às classes Block e Shudown. E nas respostas tipo Config a classe Resource está agregada diretamente à classe Config.

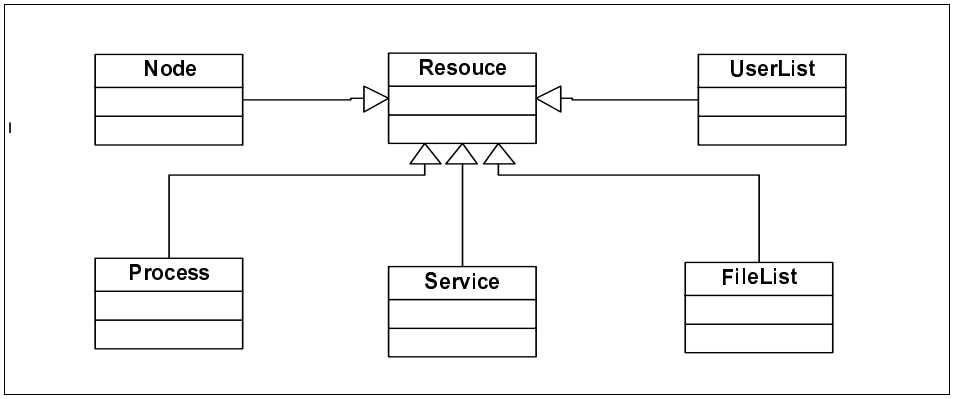

Figura 29 – Relacionamentos da classe Resource.

Na Figura 29 podemos observar que a classe Resource possui cinco classes derivadas: Node, Process, Service, UserList e FileList. Isto demonstra que um recurso pode ser nó da rede, um processo do sistema operacional, um serviço de rede, uma lista de usuários ou uma lista de arquivos.

5.4.3 As Classes de Recurso

As classes derivadas da classe Resource contêm sempre informações que identifiquem de maneira única o recurso ao qual a resposta deverá ser aplicada, além disto estas classes também contém informações complementares que são armazenadas em um *log* pelo componente de contra-medidas e permitem a identificação do recurso após a execução da resposta no mesmo.

Embora a classe Node do modelo IDREF tenha o mesmo nome e sirva ao mesmo propósito da classe Node do modelo IDMEF, ela possui uma estrutura diferente. Conforme podemos observar na Figura 30, a classe Node do modelo IDREF possui uma única instância da classe Address, esta instância contém o endereço do equipamento de rede em que a resposta será aplicada.

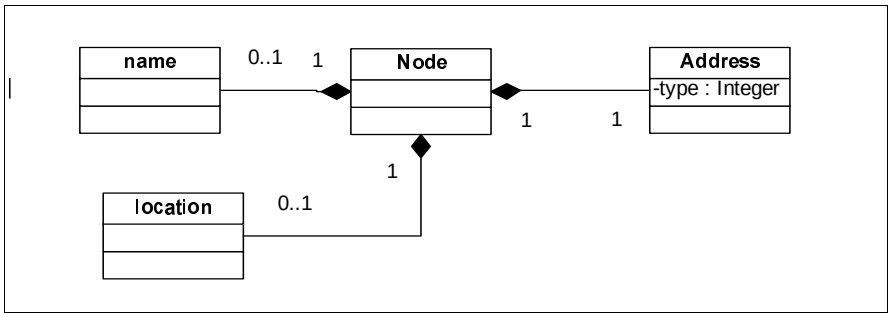

Figura 30 – Relacionamentos da classe Node.

Além do endereço do equipamento de rede, armazenado na classe Address, a classe Node também contém o nome e a localização do equipamento no momento em que a resposta foi recebida, estas informações complementares estão nas classes name e location e tem por objetivo permitir a identificação do recurso através do *log*, visto que o endereço do equipamento pode ser alterado.

O recurso Process é representado de maneira única pelo atributo pid, que contém o número do processo ao qual a resposta será aplicada. Informações complementares ao recurso Process estão nas classes agregadas name e path, que contém o nome e o caminho do processo que estava executando no momento em que a resposta foi aplicada, estas informações são armazenadas em um *log* para futuras consultas e identificações do processo que recebeu a resposta. A Figura 31 apresenta os relacionamentos da classe Process.

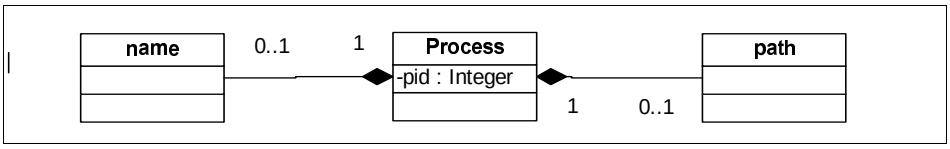

Figura 31 – Relacionamentos da classe Process.

Um serviço de rede é outro recurso que pode sofrer uma resposta, ele é representado pela classe Service. Durante a execução do serviço são utilizadas as portas que o mesmo trabalha para fazer a sua identificação de maneira única. Para identificar posteriormente no *log* o serviço que sofreu uma resposta, são utilizadas as informações complementares de nome e protocolo do serviço.

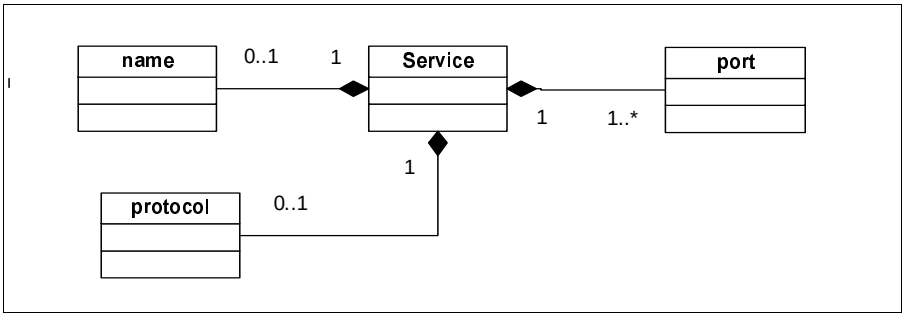

Figura 32 – Relacionamentos da classe Service.

Na Figura 32 podemos observar que a classe Service possui uma ou mais instâncias da classe port agregadas a si. A classe port contém um número de porta utilizado pelo serviço. As informações complementares estão nas classes agregadas name e protocol.

Uma resposta também pode ser aplicada a uma lista de usuários ou a uma lista de arquivos. Tanto a lista de usuários quando a lista de arquivos são consideradas um único recurso do modelo IDREF, neste caso a resposta é aplicada a cada um dos elementos, usuário ou arquivo, da lista especificada como recurso.

A lista de usuários é representada pela classe UserList e seus relacionamentos são apresentados na Figura 33. A classe User representa um elemento da lista de usuários UserList, cada usuário é identificado pelo seu número ou pelo seu nome. Os atributos category e *type* possuem as mesmas funcionalidades dos atributos encontrados nas classes User e UserId do modelo IDMEF. No *log* a identificação posterior de um usuário também é feita pelo seu nome ou número.

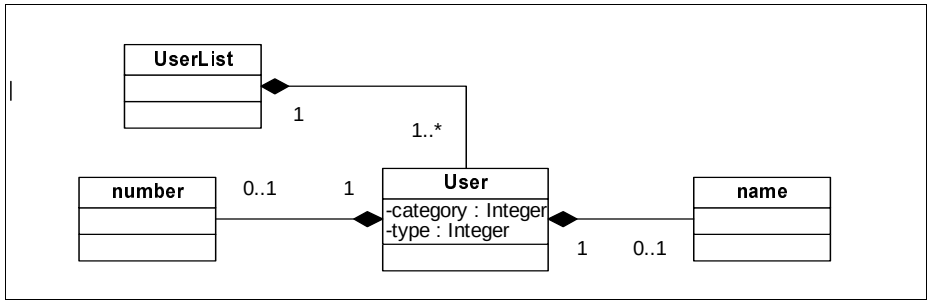

Figura 33 – Relacionamentos da classe UserList.

A lista de arquivos é representada pela classe FileList e seus relacionamentos são apresentados na Figura 34.

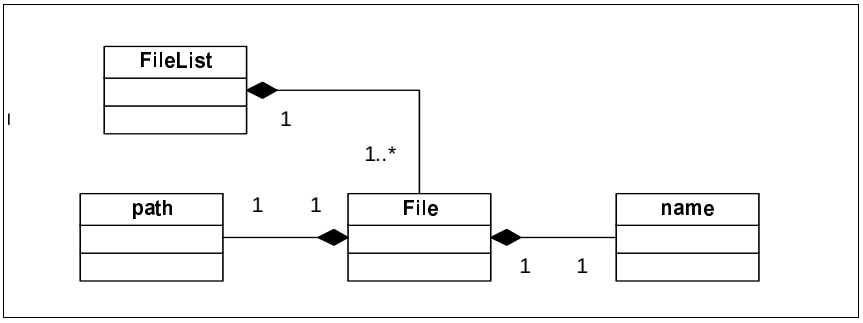

Figura 34 – Relacionamentos da classe FileList.

A classe File representa um elemento da lista de arquivos FileList, cada arquivo é identificado pelo seu nome e pelo seu caminho, informados através das classes name e path respectivamente. No *log* a identificação posterior de um arquivo também é feita pelo seu nome e pelo seu caminho.

5.4.4 Relacionamento dos modelos IDMEF e IDREF

Conforme dito anteriormente o modelo IDREF possui um forte relacionamento com o modelo IDMEF, pois as informações contidas em uma resposta dependem das informações prestadas pelo alerta. Sendo assim, muitas das informações existentes no modelo IDREF podem ter como origem informações do alerta IDMEF recebido. Outras informações utilizadas na resposta podem ser geradas internamente pelo Gerenciador ou serem informadas pelo usuário.

Na classe IDREF-Message as informações de identificação e versão, além das informações contidas nas classes Manager e CreateTime, podem ser geradas internamente pelo Gerenciador. As informações da classe alertident são obtidas através do atributo ident da classe Alert e do atributo analyserid da classe Analyser do modelo IDMEF. AdditionalData e description devem ser informados pelo usuário quando necessário.

Uma resposta tipo Response pode ser enviada para conter a origem de um ataque quando o Alerta contém informações sobre o endereço de origem do ataque. Neste caso a informação da classe Address agregada à classe Response é obtida através do relacionamento das classes Alert, Source, Node até chegar à classe Address do modelo IDMEF. Observe que um Alerta pode ter vários endereços de origem, com base nisto a classe Response também aceita vários endereços de destino para a resposta. O conteúdo das classes TCP, ICMP ou notify devem ser informado pelo usuário no Gerenciador.

Na resposta do tipo React o conteúdo da classe Block deve ser informado pelo usuário no Gerenciador. O conteúdo de Shutdown é gerado internamente, pois esta classe possui apenas um identificador. E na resposta do tipo Config os diversos comandos que são aplicados ao recurso devem ser informados pelo usuário.

As respostas do tipo React e Config utilizam a classe Resource para identificar quem será o destino da resposta. Quando estes tipos de resposta visam atuar sobre o recurso que está sendo atacado o conteúdo da classe Resource poderá ser obtido através das informações contidas no Alerta.

Para obter o recurso que está sendo atacado devem ser acessadas as informações da classe Target agregada à classe Alert do modelo IDMEF. Todas as informações sobre o destino de um ataque podem ser convertidas em respostas que atuam sobre o recurso que está sendo atacado.

A classe Target pode conter as classes Node, Process, Service, User e FileList. Em uma resposta, as informações destas classes poderão ser convertidas em recursos do tipo Node, Process, Service, UserList e FileList respectivamente no modelo IDREF.

Caso o gerenciador receba um alerta indicando que um determinado equipamento do ambiente está sendo atacado, ou seja, um alerta com informações na classe Node agregada à classe Target, o operador pode utilizar esta informação para enviar uma resposta que bloqueie, feche ou reconfigure o equipamento que está sendo atacado.

O mesmo procedimento pode ser adotado quando o alerta contém informações sobre um processo, um serviço, uma lista de usuários ou uma lista de arquivos que estejam sendo atacados. Nestes casos o modelo IDREF sempre permitirá que se converta a informação de destino do ataque em um recurso que sofrerá algum tipo de alteração de forma que o ataque seja contido.

5.4.5 Exemplos de Respostas

A seguir serão apresentados três exemplos de respostas IDREF em formato XML, estes exemplos foram validados e estão de acordo com o DTD apresentado no Anexo 3.

Os exemplos apresentados nesta seção foram gerados pelo "IDSMan - Gerenciador de IDSs IDMEF/IDREF", que foi desenvolvido no presente trabalho e será detalhado no próximo capítulo.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE IDREF-Message PUBLIC "-//UFSC//DTD IDREF v1//EN"
                               "./idref-message.dtd">
<IDREF-Message ident="1" version="1">
    <description>Encerrar conexao de origem</description>
    <Manager managerid="MAN1">
        <Node>
            <name>IDSMan</name>
            <Address type="ipv4-addr">192.168.1.3</Address>
        </Node>
        <Process pid="5486">
            <name>IDMan</name>
            <path>c:\IDSMan</path>
        </Process>
    </Manager>
    <CreateTime ntpstamp="0xc3ccb8ac.0x24200000">2004-02-
05T10:42:20Z</CreateTime>
    <alertident>123abc</alertident>
    <Response>
        <Address type="ipv4-addr">192.168.1.50</Address>
        <TCP flags="000001" source="80" target="1053"/>
    </Response>
</IDREF-Message>
```

Quadro 20 – Resposta IDREF tipo Response.

O Quadro 20 apresenta um exemplo de resposta do tipo Response. Observando este exemplo verificamos que o tipo de Response utilizado foi TCP e a resposta tem como objetivo encerrar uma conexão TCP com o endereço 192.168.1.50 na porta 1053.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE IDREF-Message PUBLIC "-//UFSC//DTD IDREF v1//EN"
                               "./idref-message.dtd">
<IDREF-Message ident="1" version="1">
    <description>Bloquear usuarios e fechar processo</description>
    <Manager managerid="MAN1">
        <Node>
            <name>IDSMan</name>
            <Address type="ipv4-addr">192.168.1.3</Address>
        </Node>
        <Process pid="5486">
            <name>IDMan</name>
            <path>c:\IDSMan</path>
        </Process>
    </Manager>
    <CreateTime ntpstamp="0xc3ccba2d.0x36600000">2004-02-
05T10:48:45Z</CreateTime>
    <alertident>123abc</alertident>
    <React>
        <Block ident="1" time="20" unblock="time">
            <UserList>
                <User category="os-device" type="current-user">
                    <name>Joao</name>
                    <number>10</number>
                </User>
                <User category="os-device" type="current-user">
                    <name>Maria</name>
                    <number>14</number>
                </User>
            </UserList>
        </Block>
        <Shutdown ident="1">
            <Process pid="5475">
                <name>Backdoor.exe</name>
                <path>c:\temp</path>
            </Process>
        </Shutdown>
    </React>
</IDREF-Message>
```

Quadro 21 – Resposta IDREF tipo React.

O Quadro 21 apresenta um exemplo de resposta IDREF do tipo React. Esta resposta contém uma solicitação de bloqueio e uma solicitação de fechamento de recurso. No bloqueio é utilizado o recurso UserList, contendo os usuários de número 10 e 14. E no fechamento é utilizado o recurso Process, contendo o processo de número 5475. Sendo assim, esta resposta resultaria no bloqueio dos usuários 10 e 14 por 20 minutos e no fechamento do processo 5475.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE IDREF-Message PUBLIC "-//UFSC//DTD IDREF v1//EN"
                               "./idref-message.dtd">
<IDREF-Message ident="1" version="1">
    <description>Tirar permissao de arquivos</description>
    <Manager>
        <Node>
            <name>IDSMan</name>
            <Address type="IPv4-addr">192.168.1.3</Address>
        </Node>
        <Process pid="896">
            <name>IDMan</name>
            <path>c:\man</path>
        </Process>
    </Manager>
    <CreateTime ntpstamp="0xc3b7ad15.0x72600000">2004-01-
20T11:35:17Z</CreateTime>
    <alertident>12345abcde</alertident>
    <Config>
        <FileList>
            <File>
                <name>arq1.txt</name>
                <path>c:\</path>
            </File>
            <File>
                <name>arq2.txt</name>
                <path>c:\</path>
            </File>
        </FileList>
        <command>chmod 744</command>
    </Config>
</IDREF-Message>
```

Quadro 22 – Resposta IDREF tipo Config.

O Quadro 22 apresenta um exemplo de resposta IDREF do tipo Config em que é aplicado um comando ao recurso FileList. No exemplo acima o recurso FileList contém dois arquivos (arq1.txt e arq2.txt) aos quais é aplicado um comando que altera as suas permissões de acesso.

6 DESENVOLVIMENTO E VALIDAÇÃO DO MODELO IDREF

6.1 Desenvolvimento do Modelo

Para possibilitar a validação do modelo proposto no capítulo anterior foram desenvolvidos os componentes da arquitetura de IDS necessários para a formação de um ambiente de IDS com suporte ao envio de respostas, conforme apresentado anteriormente na Figura 23.

Os componentes desenvolvidos para a validação do modelo são:

- IDSMan – é um Gerenciador de alertas de IDSs, este gerenciador tem a capacidade de receber mensagens IDMEF e enviar mensagens IDREF através do protocolo IDXP;
- IDSAna – faz a ponte entre o Analisador e o Gerenciador, que tem a capacidade de ler mensagens IDMEF de um arquivo texto e enviá-las ao Gerenciador IDSMan;
- IDSRes – é um componente de contra-medidas que tem a capacidade de receber mensagens IDREF do Gerenciador IDSMan, armazená-las em um arquivo de *log* e aplicá-las ao recurso.

Além do desenvolvimento destes três componentes também foi necessário desenvolver uma biblioteca que implemente o modelo IDREF, a qual foi utilizada pelos componentes IDSMan e IDSRes.

6.1.1 Ambiente de Desenvolvimento

No desenvolvimento dos componentes e da biblioteca IDREF foi utilizada a linguagem Java e a ferramenta de desenvolvimento Eclipse.

A liguagem Java foi escolhida por permitir a execução em vários ambientes (multiplataforma), ser orientada a objetos, da mesma forma que os modelos a serem implementados, e por já existirem bibliotecas BEEP, IDXP e IDMEF implementadas nesta linguagem.

A ferramenta de desenvolvimento Eclipse foi escolhida por ser de livre distribuição (“http://www.eclipse.org/”) e apresentar uma grande quantidade de recursos que facilitam o desenvolvimento na linguagem Java, dentre estes recursos temos a

facilidade de organização dos projetos em desenvolvimento, a depuração do código e o Code Complete[1].

6.1.2 Bibliotecas Utilizadas no Desenvolvimento

Para que os componentes se comunicassem trocando mensagens IDMEF através do protocolo IDXP, como prevê a arquitetura IDWG, também foram necessárias bibliotecas que implementassem os protocolos BEEP e IDXP, e o modelo de dados IDMEF.

A biblioteca BEEP utilizada foi desenvolvida por "*SourceForge*" e está disponível em "http://sourceforge.net/projects/beepcore-java". Esta biblioteca implementa a criação de sessões BEEP e o gerenciamento de canais conforme as RFC's 3080 e 3081. Para o desenvolvimento dos componentes foi utilizada a versão 0.9.07. Esta biblioteca foi utilizada nos componentes IDSMan, IDSAna e IDSRes.

A biblioteca IDXP não possuía nenhuma versão liberada, apenas estavam disponíveis os arquivos fonte implementados até o momento, portanto foi necessário baixar cada um destes arquivos e gerar a biblioteca. Os arquivos fonte utilizados também foram desenvolvidos por "*SourceForge*" e estão disponíveis através de "http://sourceforge.net/projects/idxp-java". Esta biblioteca utiliza funções implementadas na biblioteca BEEP para implementar a criação de canais IDXP e a troca de mensagens IDXP de acordo com [FEINSTEIN 2003], e foi utilizada neste trabalho pelos componentes IDSMan, IDSAna e IDSRes.

A biblioteca IDMEF utilizada foi desenvolvida por "*Silicon Defense*" e está disponível em "http://www.silicondefense.com/idwg/javaidmef/javaidmef.html". Esta biblioteca implementa todas as classes do modelo IDMEF de acordo com [CURRY 03].

A biblioteca IDMEF permite a criação das classes IDMEF automaticamente a partir de um arquivo XML devidamente formatado, também é possível gerar o arquivo XML com base em uma classe *IDMEF-Message* previamente criada. Para isto existem duas funções principais na biblioteca, uma que converte uma *string* no formato XML em objetos Java que representam as classes IDMEF, e outra que converte os objetos Java do modelo IDMEF em uma *string* no formato XML.

---

[1] Recurso no qual o desenvolvedor digita o início das chamadas de funções e procedimentos e a ferramenta de desenvolvimento sugere o nome completo da chamada.

A versão 0.92 da biblioteca IDMEF foi utilizada pelos componentes IDSMan e IDSAna.

As três bibliotecas apresentadas até o momento e a biblioteca IDREF que foi desenvolvida neste trabalho utilizam funções da biblioteca Xerces. A biblioteca Xerces foi desenvolvida por "*The Apache Software Foundation*" e está disponível em "http://xml.apache.org/xerces2-j/index.html". Esta biblioteca possui várias funções para geração, leitura e manipulação de documentos XML, implementando inclusive as funções DOM - *Document Object Model*. Neste trabalho foi utilizada a versão 1.3.1 da biblioteca Xerces.

Exclusivamente no componente IDSRes foi necessário utilizar uma biblioteca que viabilizasse a aplicação de algumas respostas IDREF. Para tanto foi utilizada a biblioteca JPcap disponível em "http://netresearch.ics.uci.edu/kfujii/jpcap/" na sua versão 0.4. A biblioteca JPCap possui funções para captura, geração e envio de pacotes de rede.

Exemplos da utilização destas bibliotecas serão apresentados nas seções seguintes.

6.1.3 Implementação da Biblioteca IDREF

Com base na implementação IDMEF do grupo *Silicon Defense*, foi desenvolvida neste trabalho uma biblioteca que implementa todas as classes do modelo IDREF. Para a implementação da biblioteca foi criado um pacote Java chamado "*org.idref*", este pacote possui doze arquivos ".java" que implementam as doze classes do modelo IDREF. Além dos doze arquivos das classes, existem mais dois arquivos implementados no pacote, um é uma interface que define um método para converter uma instância de uma classe IDREF em um elemento de arquivo XML, todas as classes IDREF implementam esta interface. O outro arquivo contém funções auxiliares à geração do arquivo XML, utilizando funções da biblioteca Xerces.

Para criar uma mensagem Java IDREF a classe base do modelo (classe *IDREF-Message*) possui dois métodos estáticos de nome *createMessage*, o primeiro recebe como parâmetro uma *string* em formato XML e o segundo recebe uma classe *Document*, esta classe está definida na biblioteca Xerces e representa um documento XML. Com estes dois metidos é possível realizar a conversão de uma *string* ou um *Document* XML em objetos Java IDREF. Além destes dois métodos estáticos, cada

objeto pode ser criado separadamente e agregado a outros objetos, formando a mensagem IDREF completa.

De posse dos objetos Java IDREF criados, é possível alterar seus atributos ou objetos agregados, alterando assim a resposta IDREF resultante.

É possível converter os objetos IDREF criados em um *Document* ou uma *string* XML, a método *toString* da classe *Objetc* é sobrescrito pela classe *IDREF-Message* para implementar a geração da *string* XML, quando for chamado este método em qualquer uma das classes derivadas da classe IDREF-Message (Response, React, Config) o resultado será o conteúdo da classe convertido em uma *string* XML.

Para converter os objetos Java IDREF em um *Document*, é utilizado o método *toXML* da classe *IDREF-Message*, este método retorna um *Document* com o conteúdo do objeto Java IDREF.

Exemplos da utilização desta biblioteca serão apresentados nas seções seguintes, juntamente com a apresentação da implementação de cada módulo que utiliza a biblioteca.

6.1.4 Implementação do Componente IDSMan

O componente IDSMan desenvolvido neste trabalho é um componente que se enquadra com conceito de Gerenciador definido pela arquitetura de IDS IDWG (seção 2.1), ou seja, ele consegue receber alertas de um Analisador no formato IDMEF através do protocolo IDXP e apresentar estes alertas a um Operador. Porém este componente também tem a capacidade de enviar respostas aos alertas IDMEF recebidos, estas respostas são enviadas através do protocolo IDXP para um componente de Contra-Medidas e estão formatadas de acordo com o modelo de dados IDREF, sendo assim, este Gerenciador também se enquadra no conceito de Gerenciador do modelo proposto no presente trabalho (seção 5.2).

O projeto IDSMan possui doze classes conforme apresentado no diagrama de classes da Figura 35. A classe IDSMan é a classe principal que contém o método main, este método cria a classe WinMan, que contém a tela principal do Gerenciador.

A classe WinMan possui agregadas à si as classes WinConfig e WinDetails que correspondem respectivamente às telas de configuração do Gerenciador e Detalhes de um Alerta selecionado. A classe Manager contém os métodos *Start* e Stop, responsáveis

por iniciar e finalizar a classe ManServer, a qual é uma Thread cuja função é ficar aguardando solicitações de novas seções BEEP por parte de algum Analisador.

As classes WinResponse, WinBlock, WinResource, WinFile e WinUser corresponde às telas em que usuário irá preencher as informações da resposta IDREF que será enviada. A classe SendResponse cria uma sessão BEEP e um canal IDXP e envia a mensagem IDREF inserida pelo usuário para o componente de Contra-Medidas.

O projeto também inclui um arquivo de configuração no formato XML que especifica quais perfis BEEP o gerenciador suporta e qual classe implementa cada perfil suportado. Neste projeto o arquivo iconfig.xml especifica que o gerenciador suporta apenas o perfil IDXP, porém novos perfis podem ser incluídos alterando-se o arquivo de configuração.

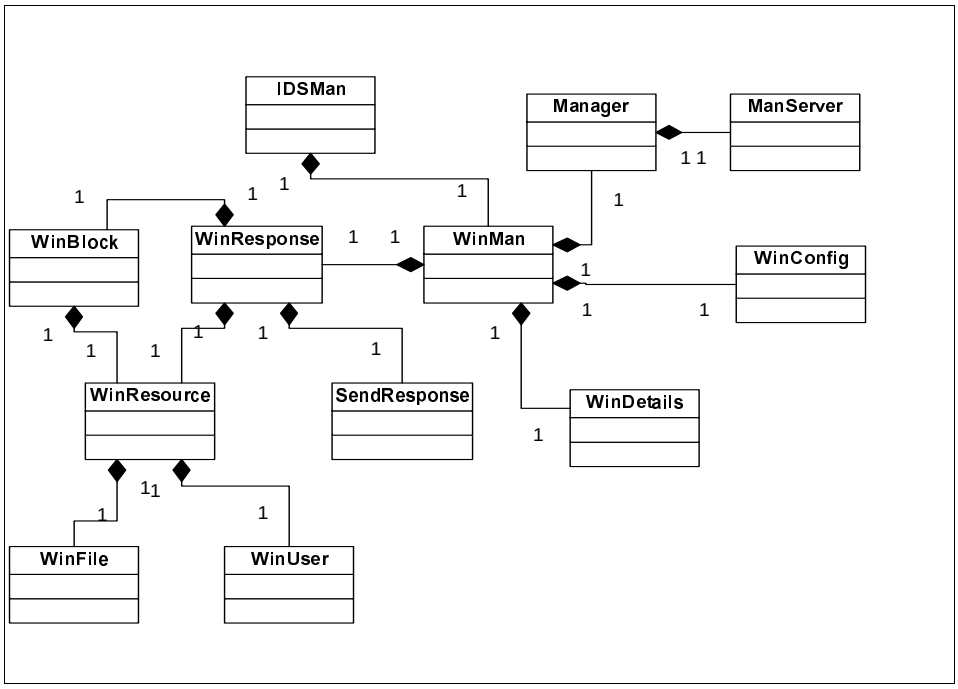

Figura 35 – Diagrama de classes do Gerenciador IDSMan.

A Figura 36 apresenta a tela principal do Gerenciador IDSMan, implementada na classe WinMan. Os botões superiores atuam na operação e configuração do Gerenciador. O painel abaixo dos botões superiores contém as informações mais relevantes sobre os alertas recebidos. E os botões inferiores atuam sobre o alerta selecionado no painel de alertas.

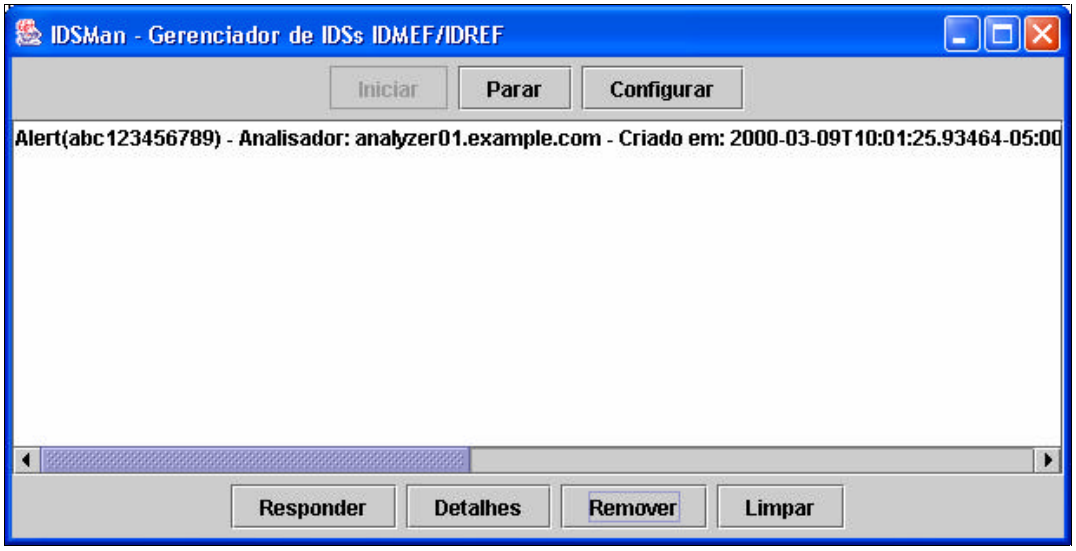

Figura 36 – Tela principal do Gerenciador IDSMan.

Através do botão Iniciar é executado o método *Start* da classe Manager, o qual lê o arquivo de configuração iconfig.xml, cria e inicializa uma instância da classe ManServer para cada perfil suportado. Quando a classe ManServer é criada ela configura e registra os perfis suportados, o Quadro 23 apresenta o código que faz a configuração do perfil IDXP, no Gerenciador o perfil é configurado como Servidor e *Listening* (Initiator igual a false).

O método setReceiveMessages foi implementado na biblioteca IDXP pelo presente trabalho, para que fosse possível ao Gerenciador receber uma cópia das mensagens IDXP que o perfil recebia, este método registra no perfil o callback ReceiveMessages da classe passada como parâmetro, no caso a própria classe ManServer. O método ReceiveMessages da classe ManServer passa a mensagem recebida para a classe WinMan que apresenta a mensagem ao usuário.

```
ProfileConfiguration profileConfig = new ProfileConfiguration();
if (className.indexOf("idxp") > 0) {
      profileConfig.setProperty(IdxpConfiguration.PROPERTY_LOCAL_URI,
                                localURI);
      profileConfig.setProperty(IdxpConfiguration.PROPERTY_VALID_LOCAL
                                _ROLE, IdxpProfile.SERVER_ROLE);
      profileConfig.setProperty(IdxpConfiguration.PROPERTY_IS_INITIATO
                                R, "false");
      //registra callback
      ((IdxpProfile) p).setReceiveMessages(this);
}
// Inicializa o perfil e adiciona à lista de perfis públicos
profileRegistry.addStartChannelListener(uri, p.init(uri,
                                        profileConfig), null);
```

Quadro 23 – Configuração do perfil IDXP.

Quando a classe Manager inicializa a Thread ManServer é executado um código que fica aguardando a solicitação de novas seções BEEP para os perfis registrados. O Quadro 24 apresenta o método run que contém este código.

```
public void run() {
      try {
            Session session;
            //Aguarda sessões para a porta e os perfis especificados
            while (true)
                  session = TCPSessionCreator.listen(port,
                                          profileRegistry);
      } catch (Exception e) {
            System.out.print("Erro: Listener já existente.");
      }
}
```

Quadro 24 – Servidor BEEP aguardando novas sessões.

Quando uma nova sessão é solicitada, devido ao *profile*Registry que foi passado como parâmetro no método TCPSessionCreate.listen, o método ReceiveMessages da classe ManServer começa a receber as mensagens que chegam aos canais da sessão, exceto mensagens de controle.

O botão Parar executa o método Stop da classe Manager, este método interrompe a execução de todas as Threads ManServer previamente inicializadas.

O botão Configurar cria a tela da classe WinConfig, onde o usuário informa a identificação do Gerenciador que será utilizada posteriormente na geração das respostas IDREF.

O botão Responder cria uma instância da classe WinResponse, que apresenta uma tela onde o usuário informa o conteúdo da resposta IDREF a ser enviada para o Alerta selecionado no painel de Alertas. A tela da classe WinResponse permite ao usuário informar todos os dados de uma mensagem IDREF, a Figura 37 apresenta a tela WinResponse.

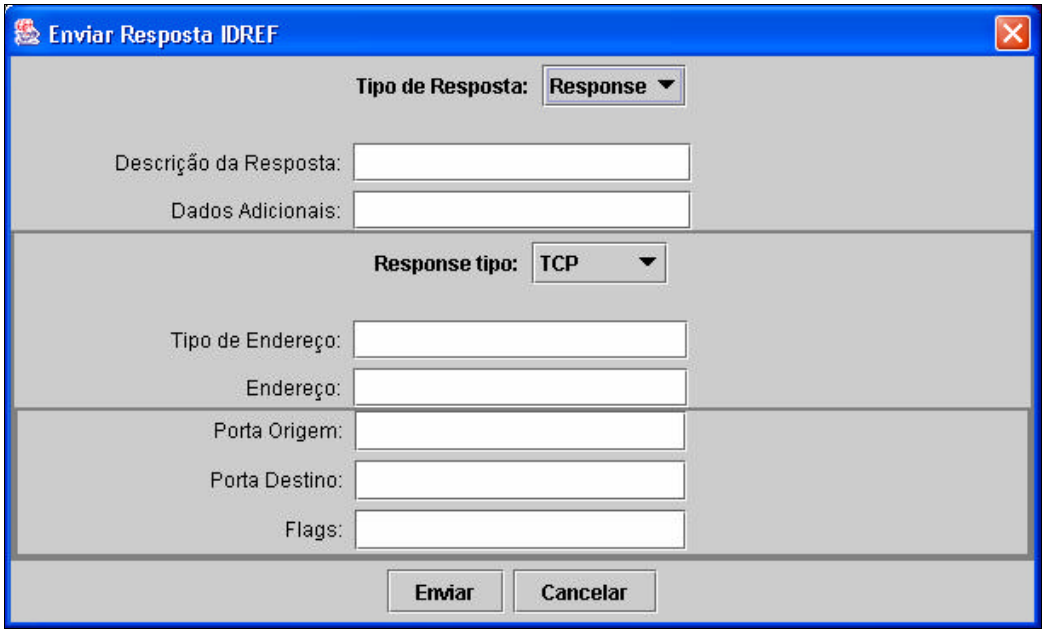

Figura 37 – Tela de envio de respostas IDREF tipo Response.

No campo “Tipo de Resposta” o usuário escolhe entre Response, React ou Config o tipo de resposta IDREF a ser enviado, conforme o usuário altera este campo os campos do painel central são alterados. Na Figura 37 o tipo de resposta selecionado é Response, e as informações do painel central correspondem a este tipo de resposta IDREF.

Para uma resposta do tipo Response, o usuário escolhe no campo “Response Tipo” o tipo de mensagem Response a ser enviada, as opções são TCP, ICMP ou Notify. Conforme o usuário altera este campo também são alterados os campos do painel de dados inferior, exibindo os campos necessários para cada tipo de resposta. O conteúdo da classe Address associada à classe Response é informado nos campos “Tipo de Endereço” e “Endereço”.

A Figura 38 apresenta a tela WinResponse configurada para enviar uma resposta do tipo React. Este tipo de resposta aceita uma lista de solicitações de bloqueio e de fechamento de recursos, para adicionar um elemento na resposta basta utilizar os botões “Adicionar Bloqueio de Recurso” e “Adicionar Fechamento de Recurso”.

Figura 38 - Tela de envio de respostas IDREF tipo React.

Através do botão "Adicionar Bloqueio de Recurso" será criada a tela WinBlock, esta tela permite a informação do conteúdo da classe Block no modelo IDREF. A classe Block contém informações sobre o desbloqueio do recurso e possui associada a si uma classe do tipo Resource. A tela WinBlock, apresentada na Figura 39, possui os campos correspondentes aos atributos da classe Block e também o botão "Adicionar Recurso", o qual cria uma instância da classe WinRecource, onde é apresentada uma tela para o usuário inserir as informações correspondentes à classe Resource associada ao bloqueio.

Figura 39 – Tela para registrar informações da classe Block.

A classe Shutdown associada a uma resposta do tipo React possui apenas um atributo e um recurso associado. O atributo (ident) é gerado internamente pelo gerenciador, portando ao pressionar o botão "Adicionar Fechamento de Recurso" é criado diretamente uma instância da classe WinResource, para que seja feita a inclusão das informações do recurso associado ao fechamento.

Uma resposta IDREF do tipo config possui um recurso e vários comandos, a Figura 40 apresenta a tela WinResponse configurada para respostas do tipo Config. O botão “Adicionar Recurso” cria uma instância da classe WinResource onde são incluídas as informações do recurso a ser configurado. O botão “Adicionar Comando” permite ao usuário informar uma *string* com o comando a ser aplicado no recurso.

Figura 40 - Tela de envio de respostas IDREF tipo Config.

A classe WinResource é utilzada pela classe WinBlock para adicionar um recurso à classe Block e pela classe WinResponse para adicionar um recurso às classes Shutdown ou Config.

A Figura 41 apresenta a tela da classe WinResource, o campo “Tipo de Recurso” contém as opções Node, Process, Service, FileList e UserList, onde o usuário define o tipo de recurso a ser inserido. O painel central apresenta os campos referentes a cada tipo de recurso.

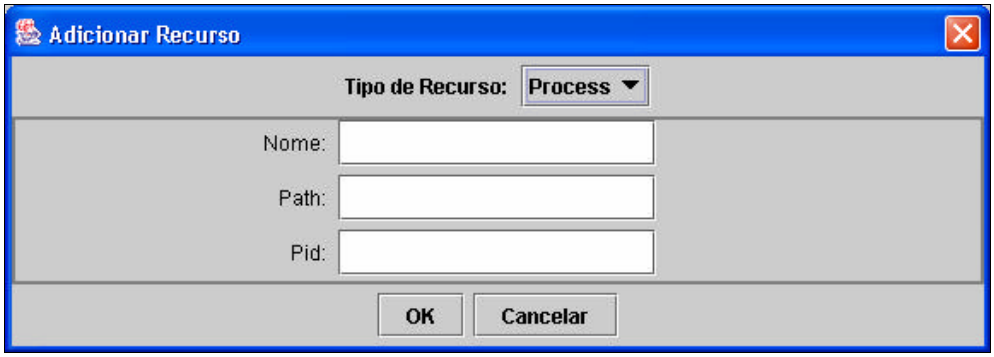

Figura 41 – Tela para adicionar informações de um recurso.

Os recursos FileList e UserList são formados por uma ou várias instâncias das classes File e User respectivamente, para estes recursos a tela WinResponse apresenta um botão onde é aberta uma tela para cadastrar cada um dos elementos das listas. No recurso FileList é apresentado o botão “Adicionar Arquivo”, o qual cria uma instância da classe WinFile com uma tela para se registrar as informações da classe File. No recurso UserList é apresentado o botão “Adicionar Usuário”, onde é criada uma instância da classe WinUser com uma tela para se registrar as informações da classe User.

Após preencher todas as informações da mensagem IDREF a ser enviada, utiliza-se no botão Enviar da tela WinResponse para que a mensagem seja enviada ao componente de Contra-Medidas apropriado.

Cada uma das telas apresentadas cria a sua respectiva classe do modelo IDREF com as informações inseridas e repassa esta à classe WinResponse, o botão Enviar reúne todas estas informações e cria a classe do tipo da resposta (Response, React ou Config), após isto é criada uma instância da classe SendResponse, que envia a resposta para o componente de Contra-Medidas. O Quadro 25 apresenta o método Send da classe WinResponse, que é executado quando o botão Enviar é pressionado.

```
private void Send() {
  IDREF_Message idref_message = null;

  //parâmentros de IDREF_Message
  String ident = String.valueOf(countIdent++);
  String description = null;

  if (!mainFields.getValue("Descrição da Resposta: ").equals(""))
    description = mainFields.getValue("Descrição da Resposta: ");
  Vector alertIdents = new Vector();
  alertIdents.add(new Alertident(analyserID, alertID));
  IDREF_Manager manager = CreateManager();
  CreateTime ct = new CreateTime();
```

```
  Vector ad = null;

  if (!mainFields.getValue("Dados Adicionais: ").equals("")) {
    ad = new Vector();
    ad.add(new AdditionalData("string", null,
           mainFields.getValue("Dados Adicionais: ")));
    }

  //tipo da resposta
  switch (cType.getSelectedIndex()) {
  case 0 : { //RESPONSE
    Address address = new Address(addrFields.getValue("Endereço: "),
                          addrFields.getValue("Tipo de Endereço: "));

    //response tipo
    switch (cresponseType.getSelectedIndex()) {
    case 0: { //TCPResponse
      idref_message = new TCPResponse(ident, description, alertIdents,
                              manager, ct, ad, address,
                              tcpFields.getValue("Porta Origem: "),
                              tcpFields.getValue("Porta Destino: "),
                              tcpFields.getValue("Flags: "));
      break;
      }
    case 1: { //ICMPResponse
      idref_message = new ICMPResponse(ident, description,
                             alertIdents,
                              manager, ct, ad, address,
                              icmpFields.getValue("Tipo: "),
                              icmpFields.getValue("Código: "));
      break;
      }
    case 2: { //NotifyResponse
      idref_message = new NotifyResponse(ident, description,
                             alertIdents,
                              manager, ct, ad, address,
                              messageFields.getValue("Mensagem: "));
      }
    }
    break;
  }

  case 1 : { //REACT
    idref_message = new React(ident, description, alertIdents,
                    manager, ct, ad, blocks, shutdowns);
    break;
    }

  case 2 : { //CONFIG
    idref_message = new Config(ident, description, alertIdents,
                     manager, ct, ad, commands, resource);
    }
  }

  //converte mensagem para string
  String s = idref_message.toString();

  //envia string
  SendResponse send = new SendResponse(host, port, s);
```

```
  JOptionPane.showMessageDialog(null, "Resposta Enviada");
}
```

Quadro 25 – Método Send da classe WinResponse.

No método acima podemos observar a criação da resposta IDREF com base nas informações das telas, a conversão da resposta IDREF em *string* e a criação da classe SendResponse onde é passado como parâmetro o destino da mensagem e a mensagem propriamente dita.

O Quadro 26 apresenta o construtor da classe SendResponse, observando este método podemos ver a solicitação da sessão BEEP através do método TCPSessionCreator.initiate da biblioteca BEEP. Depois de criada a sessão, é feita a criação do canal IDXP através da função IdxpChannelFactory.createIdxpChannel da biblioteca IDXP. Com a sessão BEEP e o canal IDXP criados, é enviada a mensagem pelo canal IDXP através do método sendMSG (idxpChannel.sendMSG). Para finalizar o envio da mensagem são fechados o canal IDXP e a sessão BEEP.

```
public SendResponse(String host, int port, String resp)
{
  Session session = null;
  IdxpChannel idxpChannel = null;

  try {
    //Solicita uma sessão ao servidor
    try     {
      session = TCPSessionCreator.initiate(host, port);
    } catch (BEEPException e) {
      System.out.println("Erro ao iniciar sessão.");
    }

    //cria o canal IDXP
    idxpChannel = IdxpChannelFactory.createIdxpChannel(session,
                            LOCAL_URI, LOCAL_ROLE, true, null);

    while (!idxpChannel.isHandshakeComplete())
      Thread.sleep(1000);

    try     {
      StringOutputDataStream os =   new StringOutputDataStream(
                              IdxpProfile.TEXT_PLAIN_CONTENT, resp);
      os.setComplete();

      //envia a mensagem pelo canal
      idxpChannel.sendMSG(os);
    } catch (BEEPException e) {
      System.out.println("Erro ao enviar mensagem.");
    }
  } catch (Exception e) {
    System.out.println("Erro na conexão/transmissão da mensagem.");
  }
  finally {
```

```
    try     {
      idxpChannel.close();
    } catch (BEEPException e) {
      System.out.println("Erro ao fechar o canal IDXP.");
    }

    try     {
      session.close();
    } catch (BEEPException e) {
      System.out.println("Erro ao fechar a sessão BEEP.");
    }
  }
}
```

Quadro 26 – Conexão e envio de resposta IDREF.

Depois de enviada a mensagem IDREF para o componente de Contra-Medidas a tela WinResponse é fechada e o usuário volta para a tela principal do Gerenciador.

O painel de alertas recebidos da tela principal contém algumas informações sobre cada alerta recebido, informações completas podem ser obtidas através do botão Detalhes, este botão cria a tela da classe WinDetails que apresenta a mensagem IDMEF completa referente ao Alerta. A Figura 42 apresenta a tela de detalhes do Alerta mostrado na Figura 36.

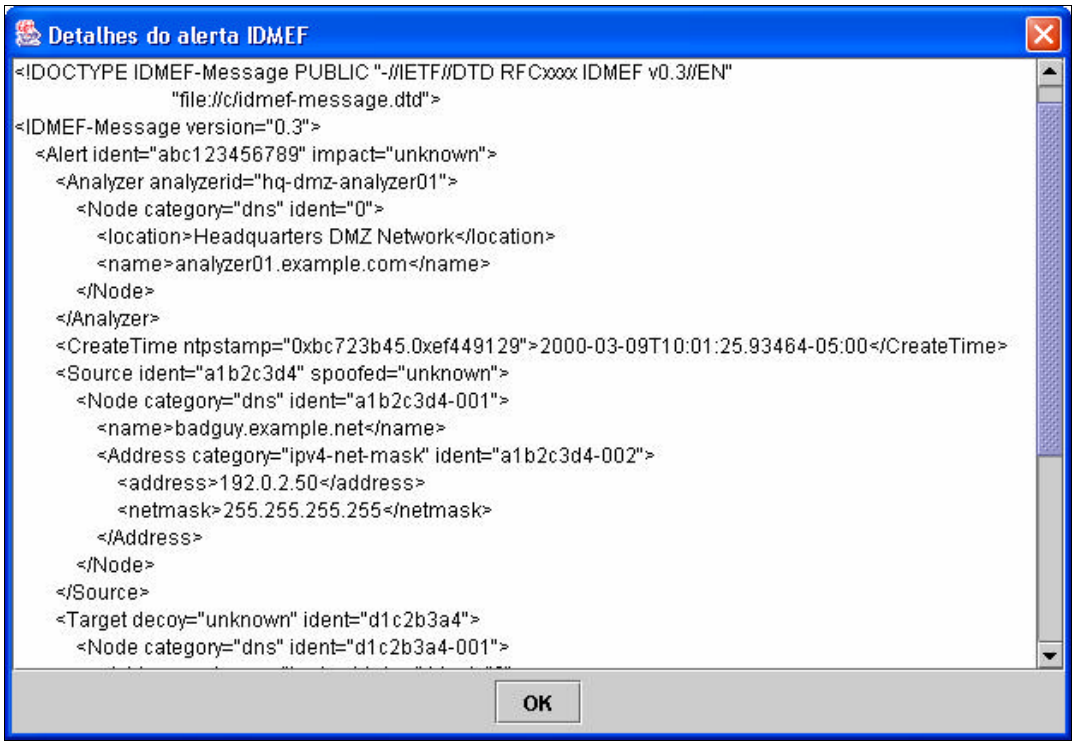

Figura 42 – Detalhes de um alerta IDMEF.

Os botões Remover e Limpar da tela principal são utilizados para remover um Alerta do painel de alertas ou limpar o painel respectivamente.

6.1.5 Implementação do Componente IDSAna

O componente IDSAna desenvolvido neste trabalho não se caracteriza como um Analisador dentro do conceito IDWG, pois ele não tem a responsabilidade de analisar dados e decidir se são ou não Alertas. O IDSAna tem a função de ler um arquivo texto e enviar, através do protocolo IDXP, para o Gerenciador os Alertas IDMEF contidos neste arquivo. Sendo assim o IDSAna caracteriza-se mais como sendo um programa auxiliar de um Analisador, do que um Analisador propriamente dito.

Para que o IDSAna funcione é necessário que exista um arquivo texto com Alertas no formato IDMEF, este arquivo texto deve ser alimentado por um IDS. Quando um Alerta é colocado no arquivo texto pelo IDS, o IDSAna automaticamente envia o Alerta para o Gerenciador.

A implementação do componente IDSAna possui três classes: IDSAna, WinAna e Client. A classe IDSAna é a classe principal do componente, a qual possui o método main. O método main cria uma instância da classe WinAna, que possui a tela principal do componente. A classe Client é uma thread que inicialmente realiza a conexão BEEP/IDXP com o Gerenciador, e depois de inicializada faz a transmissão dos Alertas IDMEF. Esta classe utiliza funções das bibliotecas Xerces, BEEP, IDXP e IDMEF.

A Figura 43 apresenta a tela da classe WinAna, onde é possível informar no campo *Host* o endereço do Gerenciador, se este campo ficar em branco será utilizado o endereço de LoopBack (127.0.0.1). No campo Arquivo é informado o arquivo texto que contém os Alertas IDMEF a serem enviados.

Figura 43 – Tela principal do componente IDSAna.

O botão Iniciar solicita à classe Client que estabeleça uma conexão com o Gerenciador através de seu método StartClient, este método cria uma sessão BEEP e um canal IDXP, e retorna uma *string* com o resultado da conexão, que pode ou não ser um

erro. Caso a conexão tenha ocorrido com sucesso, o código do botão Iniciar inicializa a thread da classe Client, e apartir deste momento os Alertas do arquivo texto são enviados para o Gerenciador.

O botão Parar interrompe a thread e chama o método CloseClient da classe Client, este método fecha o canal IDXP e a sessão BEEP encerrando a conexão com o Gerenciador.

O Quadro 27 apresenta um trecho do código do método StartClient. Podemos observar a criação da sessão BEEP através do método TCPSessionCreater.initiate da biblioteca BEEP. Se não ocorrer erro na criação da sessão, é feita a criação do canal IDXP, através do método IdxpChannelFactory.createIdxpChannel da biblioteca IDXP. Após a criação do canal IDXP, aguarda-se o processamento do handshake do protocolo, e com isto a conexão com o Gerenciador está estabelecida.

```
// Inicializa a sessão com o servidor
try {
      session = TCPSessionCreator.initiate(host, port);
} catch (BEEPException e) {
      return "Erro ao conectar com  " + host + ":" + port + "\n\t" +
               e.getMessage();
}

// Inicia um canal com o perfil echo e com o perfil IDXP
try {
      idxpChannel = IdxpChannelFactory.createIdxpChannel(session,
                                    LOCAL_URI, LOCAL_ROLE, true, null);

      while (!idxpChannel.isHandshakeComplete())
            Thread.sleep(1000);

} catch (BEEPError e) {
      if (e.getCode() == 550) {
            return "Erro: perfil não suportado pelo servidor.";
      } else {
            return "Erro ao criar canal (" + e.getCode() + ": " +
                     e.getMessage() + ")";
      }
} catch (BEEPException e) {
      return "Erro ao criar canal (" + e.getMessage() + ")";
} catch (Exception e) {
      return "Erro (" + e.getMessage() + ")";
}
```

Quadro 27 – Inicialização do componente IDSAna.

Após o estabelecimento da conexão é inicializada a thread. O Quadro 28 apresenta o método run da classe Client, o qual possui o código executado pela thread que faz o envio dos Alertas IDMEF para o Gerenciador.

```
public void run() {
```

```
  String linha;
  String msg = "";

  while (true) {
    try {
      while ((linha = arq.readLine()) != null) {
        msg = msg.concat(linha);
        if (linha.equalsIgnoreCase("</IDMEF-Message>"))
          break;
      }

      if (msg.indexOf("</IDMEF-Message>") > 0) {
        IDMEF_Message idmef_message = IDMEF_Message.createMessage(
                                      msg);
        IDMEF_Message.setDtdFileLocation(
                                      "file://c/idmef-message.dtd");
        StringOutputDataStream os = new StringOutputDataStream(
                                        IdxpProfile.TEXT_XML_CONTENT,
                                        idmef_message.toString());
        os.setComplete();
        idxpChannel.sendMSG(os);
        msg = "";
      }
    }
    catch (BEEPException e) {
      System.out.println("Erro ao enviar msg IDXP " + e.getMessage());
    }
    catch (IOException e) {
      System.out.println("Erro desconhecido ao enviar msg IDXP " +
                         e.getMessage());
    }
  }
}
```

Quadro 28 – Envio de Alertas do IDSAna.

Neste método, as novas linhas do arquivo texto são lidas e montadas na variável "msg" até que seja encontrada a linha final da mensagem IDMEF, representada pelo conteúdo "*</IDMEF-Message>*".

Com a mensagem IDMEF completa em forma de *string* na variável "msg", são criadas as classes Java da mensagem IDMEF através da função "IDMEF_Message.createMessage" da biblioteca IDMEF. O método createMessage cria todas as classes do modelo IDMEF especificadas na *string* passado como parâmetro. Isto é feito para fazer a validação do conteúdo de "msg" e garantir que a mensagem IDMEF está corretamente formatada.

Após isto é criada a classe *String*OutputDataStream da biblioteca BEEP, esta classe recebe o tipo de conteúdo e a mensagem a ser enviada. Com as classes do modelo IDMEF totalmente criadas, é chamado o método to*String* da classe IDMEF_Message

para que a mensagem seja convertida em *string* e o seu resultado seja passado como a mensagem a ser enviada na criação da classe *String*OutputDataStream.

Depois de criada a instância da classe *String*OutputDataStream, esta é passada como parâmetro no método “idxpChannel.sendMSG” da biblioteca IDXP e com isto é enviado o Alerta IDMEF pelo canal IDXP para o destino.

Após o envio da mensgem o conteúdo da variável “msg” é limpo, para que uma nova mensagem IDMEF possa ser montada e enviada.

6.1.6 Implementação do Componente IDSRes

O componente IDSRes foi desenvolvido para desempenhar a função de um componente de Contra-medidas do modelo IDREF. Ele tem a capacidade de receber respostas IDREF do Gerenciador através do protocolo IDXP, aplicar no ambiente as instruções contidas nestas respostas, e armazenar o conteúdo das respostas em um arquivo de *log*.

Para desempenhar as funções de recebimento de mensagens IDREF do Gerenciador, o IDSRes faz uso das bibliotecas BEEP, IDXP, e da biblioteca IDREF desenvolvida no presente trabalho.

O modelo IDREF proposto no capítulo anterior prevê diversas combinações de respostas diferentes a serem aplicadas no ambiente. Para exemplificar a aplicação das respostas IDREF definidas no modelo, foi implementado no componente IDSRes a aplicação de três tipos específicos de respostas: as respostas ICMP do tipo Response, as respostas TCP do tipo Response e as respostas de bloqueio de Node do tipo React.

Para implementar a aplicação das respostas escolhidas para exemplificar o modelo foram utilizadas funções da biblioteca JPCap.

A implementação do componente IDSRes possui cinco classes: IDSRes, WinRes, ResServer, ApplyIDREFMessage e BlockNode. A classe IDSRes é a classe principal, que possui o método main. Quando este método é executado ele cria uma instância da classe WinRes. A classe WinRes implementa a tela principal do componente IDSRes. A classe ResServer implementa o servidor BEEP/IDXP que aguarda a solicitação de novas sessões e o recebimento de novas mensagens por parte do Gerenciador. A classe ApplyIDREFMessage implementa a interpretação e aplicação das mensagens IDREF no ambiente sob domínio do IDS. E a classe BlockNode é uma thread que implementa o

bloqueio de um Node do ambiente, através do monitoramento da comunicação do respectivo Node.

A Figura 44 apresenta a tela principal do componente IDSRes, implementada pela classe WinRes. No campo "Arquivo *Log*" deve ser informado o arquivo de destino das respostas IDREF recebidas.

Figura 44 – Tela principal do componente de Contra-Medidas IDSRes.

O botão Iniciar cria uma instância da classe ResServer. A classe ResServer é uma thread que inicialmente realiza a configuração e faz o registro do perfil que o servidor de respostas aceita receber, e posteriormente fica aguardando a solicitação de novas sessões BEEP e o recebimento de novas mensagens IDREF.

O botão Parar chama o método StopServer da classe ResServer, este método fecha o arquivo de *log* e interrompe a thread ResServer que estava em execução, encerrando o recebimento de mensage IDREF.

O Quadro 29 apresenta o método construtor da classe ResServer, onde podemos observar a criação de um perfil IDXP através da classe Idxp*Profile* e a criação de uma configuração para o perfil IDXP através da classe *Profile*Configuration. Tanto o perfil quanto a configuração são registradas através da classe *profile*Registry.

No construtor da classe ResServer também podemos observar que no perfil IDXP é chamado o método setReceiveMessages passando como parâmetro a referência da própria classe ResServer que está sendo criada. O método setReceiveMessages registra dentro do perfil uma classe que implementa a interface IdxpReceiveMessages. Esta interface possui um único método chamado ReceiveMessages. Quando o método setReceiveMessages registra uma classe dentro do perfil, o método ReceiveMessages desta classe é chamado cada vez que uma mensagem chega ao canal.

```
public ResServer(String file, int port) throws Exception {

  this.port = port;
  this.file = file;

  profileRegistry = new ProfileRegistry();

  IdxpProfile p = new IdxpProfile();
```

```
  ProfileConfiguration profileConfig = new ProfileConfiguration();
  profileConfig.setProperty(IdxpConfiguration.PROPERTY_LOCAL_URI,
                            localURI);
  profileConfig.setProperty(
                           IdxpConfiguration.PROPERTY_VALID_LOCAL_ROLE,
                           IdxpProfile.SERVER_ROLE);
  profileConfig.setProperty(IdxpConfiguration.PROPERTY_IS_INITIATOR,
                            "false");

  //registra esta classe para receber mensagens IDREF
  p.setReceiveMessages(this);
  // Inicializa o perfil e adiciona à lista de perfis públicos
  profileRegistry.addStartChannelListener(IdxpProfile.getURI(),
                                          p.init(IdxpProfile.getURI(),
                                         profileConfig), null);

  arqLog = new DataOutputStream(new FileOutputStream(file));
}
```

Quadro 29 – Inicialização do componente IDSRes.

O método setReceieMessages e a interface IdxpReceiveMessages foram implementadas na biblioteca IDXP pelo presente trabalho para permitir que a classe ResServer receba uma cópia das mensagens recebidas pelo canal IDXP.

Depois de criada a classe ResServer, o código do botão Iniciar inicializa a thread desta classe. A thread da classe ResServer fica aguardando a solicitação de novas sessões BEEP por parte do Gerenciador, o código executado pela thread é igual ao código apresentado no Quadro 24.

Conforme dito anteriormente, devido ao registro feito pelo método setReceiveMessages, cada vez que o canal IDXP receber uma mensagem, uma cópia desta será enviada para o método ReceiveMessages. O Quadro 30 apresenta o método ReceiveMessages implementado pela classe ResServer.

```
public void ReceiveMessages(String s) {
  //executa a resposta no ambiente
  ApplyIDREFMessage apply = new ApplyIDREFMessage(s);

  try {
     arqLog.writeBytes("Resposta recebida: " + s + "\n");
  } catch (IOException e) {
     System.out.print("Erro ao gravar arquivo " + file);
  }
}
```

Quadro 30 – Recebimendo de uma resposta IDREF.

No Quadro 30 podemos observar que quando o componente de Contra-Medidas recebe uma mensagem IDREF ele primeiro cria uma instância da classe

ApplyIDREFMessage passando a mensagem como parâmetro, e depois grava a mensagem no arquivo de *log* especificado anteriormente na tela principal.

O constructor da classe ApplyIDREFMessage cria as classes do modelo IDREF com base na *String* recebida utilizando funções da biblioteca IDREF. Com as classes do modelo IDREF criadas, a classe ApllyIDREFMessage inicia a interpretação das informações da mensagen IDREF. Primeiro descobre-se qual o tipo de mensagem IDREF que está sendo tratada e em seguida chama-se o respectivo método de tratamento da mensagem.

O Quadro 31 apresenta o método que será chamado para tratar as mensagems TCP do tipo Response.

```
private void ApplyTCP() {
  //Enviar pacote pela rede
  TCP resp = (TCP)msg;
  TCPPacket packet = new TCPPacket(
            new Integer(resp.getSource()).intValue(),
            new Integer(resp.getTarget()).intValue(),
            0, 0,
            //Flags
            resp.getFlags().substring(1, 1).equals("1"),//URG
            resp.getFlags().substring(2, 2).equals("1"),//ACK
            resp.getFlags().substring(3, 3).equals("1"),//PSH
            resp.getFlags().substring(4, 4).equals("1"),//RST
            resp.getFlags().substring(5, 5).equals("1"),//SYN
            resp.getFlags().substring(6, 6).equals("1"),//FIN
            false, false, 512, 0);

  Vector v = resp.getAddress();
  SendPacket(packet, (Address)v.get(0));
}
```

Quadro 31 – Execução da resposta TCP do tipo Response.

No Quadro 31 é criada a classe TCPPacket da biblioteca JPCap, que representa um pacote TCP. As informações passada para o pacote TCP são obtitas através das classes do modelo IDREF, que contém as informações da mensagem IDREF recebida pelo IDSRes.

Semelhante às respostas TCP, para as respostas ICMP é chamado o método ApplyICMP, que cria uma instância da classe ICMPPacket da biblioteca JPCap e preenche suas informações com base nas classes do modelo IDREF.

Tanto em ApplyTCP quanto em ApplyICMP, depois do pacote ser preenchido é chamado o método SendPacket da classe ApplyIDREFMessages para que o pacote seja transmitido pela rede.

O método SendPacket é apresentado no Quadro 32, neste são setadas as informações relacionadas com o datagrama IP, sempre com base nas informações das classes IDREF.

```
private void SendPacket(IPPacket p, Address addr) {
   try {
      p.data = "".getBytes();

      //BUG1 - A biblioteca JPcap está invertendo os endereços IP
      //na montagem dos endereços MAC, por isto os endereços IP
      //estão invertidos aqui.
      //Seta Destino
      p.src_ip = new IPAddress(addr.getAddr());
      //Seta Origem
      p.dst_ip = new IPAddress(InetAddress.getLocalHost().
                               getAddress());
      if (addr.getType().equalsIgnoreCase("ipv6-addr"))
         p.version = 6;
      else
         p.version = 4;

      JpcapSender send = JpcapSender.openDevice(
                                            Jpcap.getDeviceList()[0]);
      try {
         send.sendPacket(p);
      } finally {
         //BUG2 - close não está fechando o RawSocket
         send.close();
      }
   } catch (UnknownHostException e) {
      System.out.print("Endereço de destino desconhecido. " +
                       e.getMessage());
   } catch (IOException e) {
      System.out.print("Erro ao abrir dispositivo. " +
                       e.getMessage());
   }
}
```

Quadro 32 – Envio de um pacote pela rede.

Ainda no método SendPacket são utilizadas funções da biblioteca JPCap para obter uma interface de rede (JpcapSender.openDevice) e enviar um pacote por esta interface (send.sendPacket). Desta forma são aplicadas as respostas IDREF de TCP e ICMP Response, conforme no figuradas no IDSMan e recebidas pelo IDSRes.

Caso a resposta recebida pelo IDSRes seja uma resposta do tipo React, o constructor da classe ApplyIDREFMessage chamará o método ApplyReact, este método percorrerá as listas de bloqueio (Block) e fechamento (Shutdown) da mensagem IDREF chamando os métodos ApplyBlock e ApplyShutdown respectivamente para cada elemento das listas.

No método ApplyBlock é feita a identificação do recurso a qual o bloqueio se refere, caso o recurso seja um Node e o desbloqueio seja por tempo (unBlock igual a time) será criada e inicializada uma instância da thread BlockNode.

A classe BlockNode tem por objetivo aplicar um bloqueio a um Node por determinado espaço de tempo, para realizar esta tarefa ela monitora o tráfego de rede em busca de qualquer comunicação endereçada ao Node que está bloqueado, encontrando um datagrama endereçado ao Node bloqueado a classe BlockNode gera uma mensagem ICMP de *host* inacessível, interrompendo a comunicação de rede com este Node.

Para realizar o monitoramento do tráfego de rede são utilizadas funções da biblioteca JPCap, o Quadro 33 apresenta o método run executado pela thread da classe BlockNode, onde é obtida a interface de rede que será monitorada (Jpcap.openDevice).

```
public void run() {
  try {
    for (int i=0; i<=(time*60); i++) {
      //fica 1 segundo capturando pacotes
      Jpcap jpcap = Jpcap.openDevice(Jpcap.getDeviceList()[0], 1000,
                                     true, 1000);
      try {
         jpcap.processPacket(-1, this);
      } finally {
         //BUG3 - close não está fechando o Device
         jpcap.close();
      }
    }
    System.out.println("Fim do monitoramento.");
  } catch (java.io.IOException e) {
        System.out.println("Erro abrindo dispositivo." +
                           e.getMessage());
  }
}
```

Quadro 33 – Monitoração de rede da classe BlockNode.

Para que ocorra a captura dos pacotes é necessário chamar o método processPacket da interface de rede obtida, passando como parâmetro uma classe que implemente a interface JpcapHandle. Esta interface possui um único método chamado handlePacket, onde são entregues os pacotes capturados.

A classe BlockNode implementa a interface JpcapHandle e é registrada para receber os pacotes conforme podemos observar no Quadro 33 (jpcap.processPacket com parâmetro this), *log*o, os pacotes monitorados serão entregues em seu método handlePacket.

O método handlePacket da classe BlockNode verifica se o endereço de destino do pacote capturado corresponde ao endereço do Node bloqueado, caso os endereços sejam o mesmo, é chamado o método SendInaccessible*Host*Packet da classe BlockNode.

```
private void SendInaccessibleHostPacket(IPAddress ipAddress) {
  ICMPPacket packet = new ICMPPacket();

  packet.type = 3; //destino inacessível
  packet.code = 1; //Host inacessível

  try {
      packet.data = "".getBytes();
      packet.version = 4;

      //BUG1 - A biblioteca JPcap está invertendo os endereços IP
      //na montagem dos endereços MAC, por isto os endereços IP
      //estão invertidos aqui.
      packet.src_ip = ipAddress; //Seta Destino
      packet.dst_ip = new IPAddress(addr); //Seta Origem

      JpcapSender send = JpcapSender.openDevice(
                                          Jpcap.getDeviceList()[0]);
      try {
          send.sendPacket(packet);
      } finally {
          //BUG2 - close não está fechando o RawSocket
          send.close();
      }
  } catch (UnknownHostException e) {
      System.out.print("Endereço de destino desconhecido. " +
                       e.getMessage());
  } catch (IOException e) {
      System.out.print("Erro ao abrir dispositivo. " +
                       e.getMessage());
  }
}
```

Quadro 34 – Envio de pacote ICMP para bloquear o Node.

O Quadro 34 apresenta o método SendInaccessible*Host*Packet, onde é criado um pacote ICMP com tipo igual a 3 (destino inacessível) e código igual a 1 (*host* inacessível). O endereço IP de origem do pacote criado é o endereço do Node bloqueiado e o endereço IP de destino é o endereço do pacote capturado.

Depois de configurado, o pacote ICMP é enviado pela rede conforme visto anteriormente, interferindo na comunicação com o Node bloqueado.

6.2 Validação do Modelo

Nesta seção serão apresentados os testes realizados para a validação da arquitetura desenvolvida. Os testes consistem em executar um ambiente de funcionamento real, onde um IDS irá gerar alertas em um arquivo texto no formato IDMEF e o componente

IDSAna irá enviar estes alertas ao Gerenciador IDSMan. No Gerenciador IDSMan será feita a configuração da resposta IDREF para o alerta recebido e o envio desta resposta ao componente de Contra-Medidas IDSRes. Finalmente, o IDSRes aplica no ambiente as ações definidas na resposta IDREF recebida e interrompe o ataque que estava em andamento.

6.2.1 Ambiente de Validação

O ambiente de validação é formado por um ambiente de rede, pelo IDS Snort, por um arquivo texto e pelos componentes IDSAna, IDSRes e IDSMan. O IDS Snort foi escolhido por se tratar de um IDS bastante utilizado e por já existir um *plugin* que o torna capaz de gerar alertas no formato IDMEF.

A Figura 45 apresenta o relacionamento entre os componentes do ambiente de validação, onde podemos observar o caminho percorrido pela mensagem IDMEF do IDS ao Gerenciador, e o caminho percorrido pela mensagem IDREF do Gerenciador ao componente de Contra-Medidas.

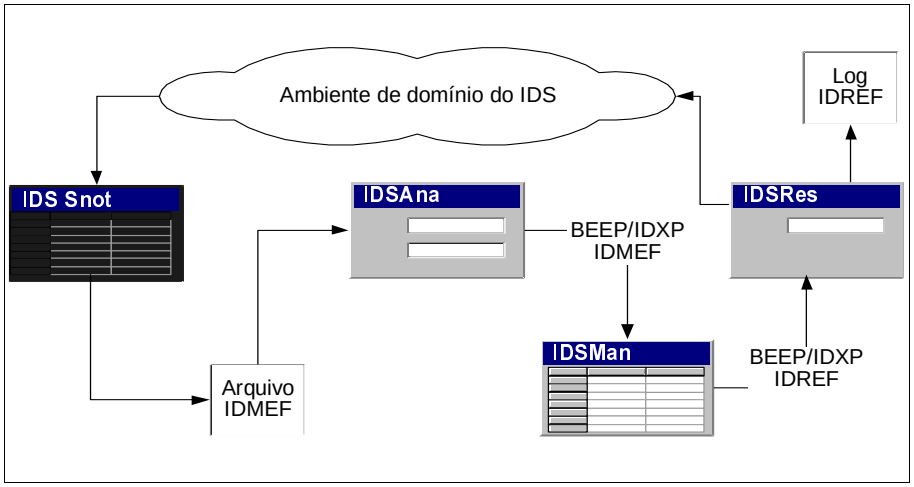

Figura 45 – Visão geral da validação da arquitetura.

De acordo com a Figura 45, o IDS Snort fica monitorando o seu ambiente de domínio em busca de alertas, quando um alerta é detectado o IDS Snort armazena este alerta no formato IDMEF em um arquivo texto. O componente IDSAna, quando ativado, fica monitorando o arquivo texto em busca de novos alertas, quando um novo alerta é gravado no arquivo texto, o IDSAna obtém este alerta e o envia através do protocolo IDXP para o Gerenciador, o qual apresenta o alerta para o Operador.

De posse do alerta, o Operador tem no Gerenciador a opção de enviar uma resposta para conter um possível ataque que esteja em andamento. Após configurar a resposta no Gerenciador, esta é enviada ao componente de Contra-Medidas IDSRes no formato IDREF através do protocolo IDXP. No IDSRes a resposta é inicialmente armazenada em um *log* para posteriores consultas, e após isto é aplicada no ambiente de acordo com as características de cada tipo de resposta (Response, React ou Config).

Vários softwares e bibliotecas foram utilizados na construção e configuração do ambiente acima apresentado. O primeiro software obtido para a construção do ambiente de validação foi o Snort, obtido em "http://www.snort.org", foi utilizada neste trabalho a versão 1.9.0.

Porém somente o Snort não foi suficiente para completar o funcionamento do ambiente, pois o este IDS não gera nativamente mensagens IDMEF. Apesar de não gerar mensgens IDMEF nativamente, o Snort foi escolhido por existir um *plugin* que habilita para tal função.

O *plugin* "*Snort IDMEF XML plugin*", desenvolvido por "*SiliconDefense*" e obtido em "http://www.silicondefense.com/idwg/snort-idmef/", foi aplicado ao Snort na construção do ambiente de validação. Foi utilizada a versão 0.2.2 deste *plugin.*

Para que o *plugin* fosse aplicado era necessário adicionar seus dois arquivos (spo_idmef.c e spo_idmef.h) no projeto do snort, incluir chamadas da inicialização do *plugin* e recompilar o snort. A recompilação do Snort foi realizada na ferramenta Microsoft Visual C++ 6.0 Enterprise Edition.

Foi necessária a utilização de funções de outras quatro bibliotecas para que o *plugin* compilasse com sucesso, estas bibliotecas são: LibIDMEF, LibXML2, LibNtp e LibIsc.

A biblioteca LibIDMEF é requerida pelo *plugin* e implementa funções relacionadas com a geração e leitura de mensagens IDMEF. Foi utilizada a versão 0.7.2 desta biblioteca, que foi obtida em "http://www.silicondefense.com/idwg/libidmef/" e desenvolvida por "*Silicon Defense*".

As funções da biblioteca LibXML2 são utilizadas pela biblioteca LibIDMEF na manipulação do formato XML das mensagens IDMEF. Esta biblioteca pode ser obtida em "http://www.xmlsoft.org/" e foi desenvolvida por "*World Wide Web Consortium (W3C)*". Na construção do ambiente foi utilizada a versão 2.4.12.

Outra biblioteca também utilizada por LibIDMEF é a LibNtp. Esta biblioteca implementa funções relacionadas com o *Network Time Protocol* e foi obtida em "http://www.ntp.org". Neste trabalho foi utilizada a versão 4.2.0.

Uma função da biblioteca LibIsc é requerida pelo *plugin* (spo_idmef.c), portanto também foi necessário incluir um fonte desta biblioteca no projeto do Snort. Foi adicionado no projeto o arquivo "*string.c*" versão 1.5.4.1, obtido em "http://www.mit.edu/afs/net.mit.edu/project/bind/9.1.0/lib/isc/" e implementado por "*Internet Software Consortium*".

Após reunir todas estas bibliotecas e resolver diversos problemas de compilação e de compatibilidade entre as bibliotecas, conseguiu-se obter um executável do Snort capaz de gerar alertas no formato IDMEF. Neste momento foram iniciadas as configurações para habilitar os alertas IDMEF no Snort.

A primeira alteração deve ser feita no arquivo de configuração do Snort, onde o *plugin* deve ser habiblitado. Foi adicionada a seguinte linha no arquivo Snort.conf: "*output idmef: 192.168.1.0/24 output=log analyzerid=IDS1 dtd=file://c/IDMEF-Message.dtd alert_id=c:\snort\bin\nextAlertId.log facility_=file|c:\snort\bin\idmef.ids|*".

A linha acima ativa o *plugin* IDMEF e informa o endereço da rede local, além da identificação do analisador (IDS1), a localização do arquivo DTD do formato IDMEF, a localização de um arquivo com a seqüência do próximo alerta ID a ser enviado e a localização do arquivo onde os alertas IDMEF devem ser gravados. De acordo com a linha apresentada acima os alertas IDMEF serão gravados no arquivo "c:\snort\bin\idmef.ids".

Depois de ativar o *plugin* no arquivo de configuração, devem ser especificadas quais regras devem gerar alertas no formato IDMEF, isto é feito adicionando-se a palavra chave "idmef" no final da regra. A palavra chave "idmef" deve ser seguida de uma palavra que indica o tipo de alerta, que pode ser "default", "web" ou "*overflow*".

Para efeito do teste de validação foi criada e adicionada ao conjunto de regras do Snort a seguinte regra: "*alert icmp $HOME_NET any -> $HOME_NET any (msg: "Alerta de Validacao - Qualquer tipo de trafego ICMP foi gerado."; idmef: default;)*".

Esta regra especifica que deve ser gerado um alerta caso ocorra uma comunicação através do protocolo ICMP de um endereço da rede local em qualquer porta para outro endereço da rede local em qualquer porta. A mensagem que estará contida no Alerta

está especificada no campo “msg”. Também podemos observar a ativação do *plugin* IDMEF ao final da regra.

6.2.2 Transmissão de Mensagens IDMEF

Com o ambiente completamente configurado, basta iniciar a execução dos componentes que fazem parte do ambiente e gerar algum tipo de comunicação ICMP para que a arquitetura seja testada.

O primeiro componente a ser iniciado é o Gerenciador IDSMan, ele deve ser ativado conforme apresentado na Figura 36. Uma vez iniciado este componente ficará aguardando mensagens IDMEF do componente IDSAna.

Depois de iniciado o Gerenciador, pode ser iniciado a componente IDSAna. Este componente foi previamente configurado para monitorar as mensagens IDMEF gravadas no arquivo “c:\snort\bin\idmef.ids”, o qual é o mesmo arquivo em que o Snort foi configurado para gravar os seus Alertas IDMEF. A tela de inicialização do componente IDSAna é apresentada na Figura 43.

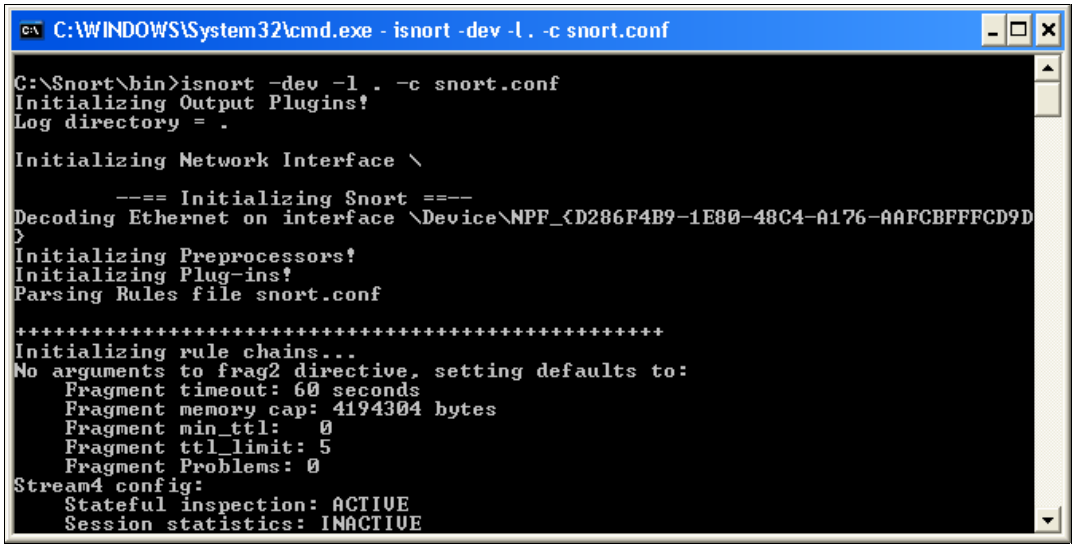

Figura 46 – Inicialização do Snort.

Com o IDSMan e o IDSAna ativados, inicia-se o Snort, conforme apresentado na Figura 46. Os parâmetros de inicialização do Snort indicam como as informações devem ser apresentadas no console (“-dev”), onde será feito o *log* dos Alertas (“-l .”), e qual arquivo contém as regras e configurações (“-c snort.conf”) a serem aplicadas neste sistema de detecção de intrusão. O arquivo de configuração passado como parâmetro

nesta inicialização do Snort é o mesmo em que foram configurados o *plugin* IDMEF e a regra para emissão de Alerta caso ocorra qualquer comunicação ICMP.

A inicialização deste três componentes já é suficiente para testar a transmissão das mensgens IDMEF geradas pelo Snort, passando pelo arquivo texto e pelo IDSAna até chegar ao IDSMan.

Conforme vimos anteriormente, a regra cadastrada para o teste de validação acusa qualquer tipo de comunicação ICMP na rede local, sendo assim, para gerar um alerta foi executado o comando "P*ing 192.168.1.1*" na rede local.

Após a execução do ping o Snort gerou vários alertas em seu arquivo e formato padrão, além disto, os alertas também foram gravados em formato IDMEF no arquivo configurado pelo *plugin.* O Quadro 35 apresenta um dos alertas IDMEF gravados pelo Snort no arquivo "c:\snort\bin\idmef.ids".

```
<?xml version="1.0"?>
<!DOCTYPE IDMEF-Message PUBLIC "-//IETF//DTD RFC XXXX IDMEF v1.0//EN"
"file://c/idmef-message.dtd">
<IDMEF-Message version="1.0">
      <Alert ident="0">
            <Analyzer analyzerid="IDS1"/>
            <CreateTime ntpstamp="0xc3c9000a.0x51a9fbe7">2004-02-
02T16:57:46Z</CreateTime>
            <Source>
                  <Node>
                        <Address category="ipv4-addr">
                              <address>192.168.1.3</address>
                        </Address>
                  </Node>
            </Source>
            <Target>
                  <Node>
                        <Address category="ipv4-addr">
                              <address>192.168.1.1</address>
                        </Address>
                  </Node>
            </Target>
            <Classification>
                  <name>Alerta de Validacao - Qualquer tipo de trafego
ICMP foi gerado.</name>
                  <url>No URL available</url>
            </Classification>
      </Alert>
</IDMEF-Message>
```

Quadro 35 – Mensagem IDMEF gerada pelo Snort.

Podemos observar no Quadro 35 algumas informações que foram cadastradas na configuração do *plugin*, como o analyzerid igual a IDS1. Outras informações foram obtidas através da regra Snort que gerou o alerta, como o nome do alerta apresentado

em Classification. E ainda outras informações foram obtidas no momento em que ataque é detectado, como o endereço de origem e destino do ataque.

Quando o alerta é gravado no arquivo texto, como o IDSAna está ativado e constantemente monitorando este arquivo, o alerta é capturado pelo IDSAna e enviado ao Gerenciador através do canal IDXP previamente estabelecido.

O Gerenciador, ao receber uma mensagem IDMEF de um IDSAna, apresenta esta mensagem para o Operador em seu painel de Alertas.

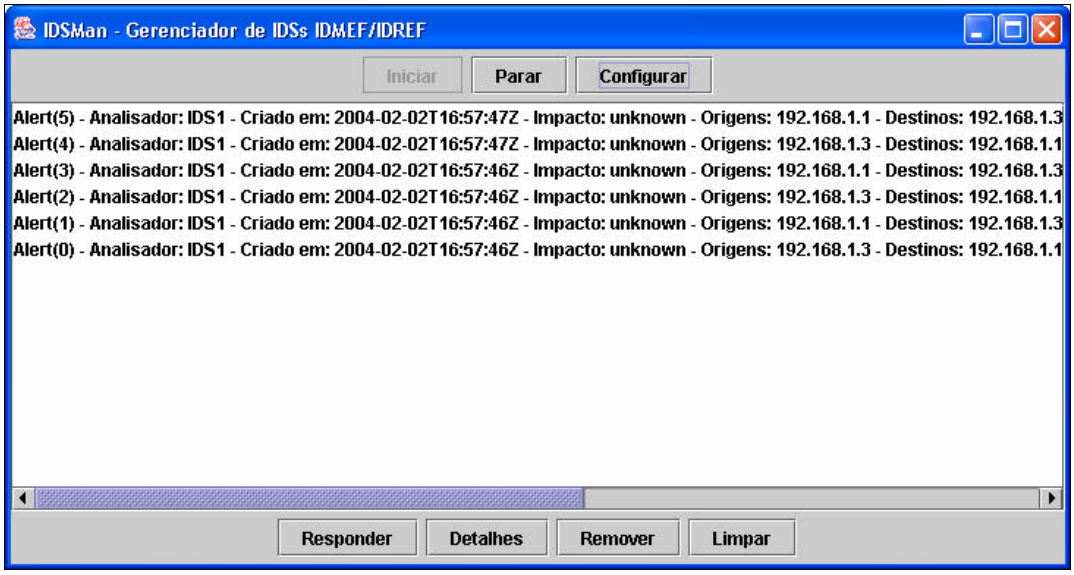

Figura 47 – Alertas recebidos pelo Gerenciador IDSMan.

A Figura 47 apresenta seis alertas recebidos pelo Gerenciador, estes alertas foram originado pelo ping executado anteriormente. O ping foi executado em 192.168.1.3 e enviou três datagramas para 192.168.1.1, gerando assim os seis alertas apresentados no Gerenciador. Analisando os campos Origens e Destinos do painel de alertas do Gerenciador, podemos observar as solicitações e respostas de eco transmitidas pela rede local.

De posse deste Alertas o Operador pode optar por observar a mensagem IDMEF completa, utilizando o botão Detalhes apresentado na Figura 42. Também pode optar por conter o ataque, enviando uma resposta para o componente IDSRes.

6.2.3 Geração da Resposta IDREF

Para realizar a validação da geração e transmissão das mensagens IDREF é necessário inicialmente executar o componente de Contra-Medidas IDSRes no mesmo

host em que se encontra o IDSAna que gerou o Alerta a ser respondido, pois é para este endereço que o IDSMan irá encaminhar a resposta IDREF.

A inicialização do IDSRes é feita conforme apresentado na Figura 44. Depois de iniciado, o IDSRes aguarda solicitações de conexão BEEP do IDSMan, para a recepção de mensagens IDREF.

Com o IDSRes iniciado, o Operador pode configurar e enviar respostas aos Alertas recebidos. Neste teste de validação será gerada uma resposta para o primeiro Alerta recebido pelo Gerenciador IDSMan.

No exemplo aqui apresentado o Operador considera que o *host* 192.168.1.1 está sendo atacado e determina na resposta gerada que este *host* deve ficar bloqueado por 30 minutos.

A geração da resposta inicia-se com o Operador selecionando o Alerta cujo ID é zero e pressionando o botão Responder (ver Figura 47). Como o Operador deseja bloquear um recurso, na tela de envio de respostas IDREF ele seleciona a resposta do tipo React e pressiona o botão “Adicionar Bloqueio de Recurso” (ver Figura 38).

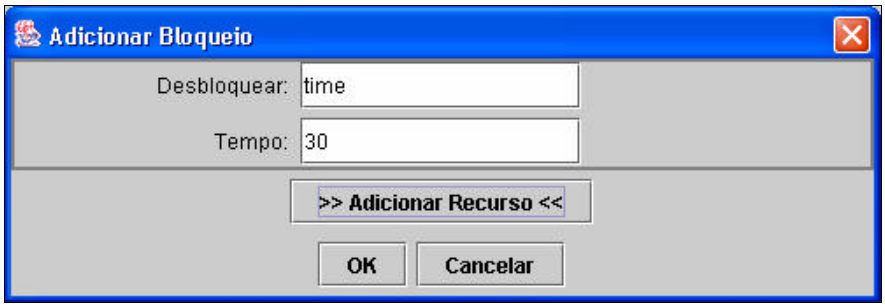

Figura 48 – Configuração da resposta de bloqueio.

A Figura 48 apresenta a tela que aparece para o Operador quando ele pressiona o botão “Adicionar Bloqueio de Recurso”. Podemos observar na figura que o Operador definiu que o recurso deve ser desbloqueado em 30 minutos. Em seguida o Operador utiliza o botão “Adicionar Recurso” para especificar qual recurso deve ser bloqueado.

Na tela de “Adicionar Recurso”, apresentada na Figura 49, o Operador especifica que deseja bloquear um recurso do tipo Node, e em seguida informa o tipo de endereço e o endereço do mesmo. Observe que as informações de tipo de endereço e de endereço foram obtidas pelo Operador no Alerta que está sendo respondido, portanto os campos Nome e Localização do Node ficam em branco, pois o Alerta não possui estas informações.

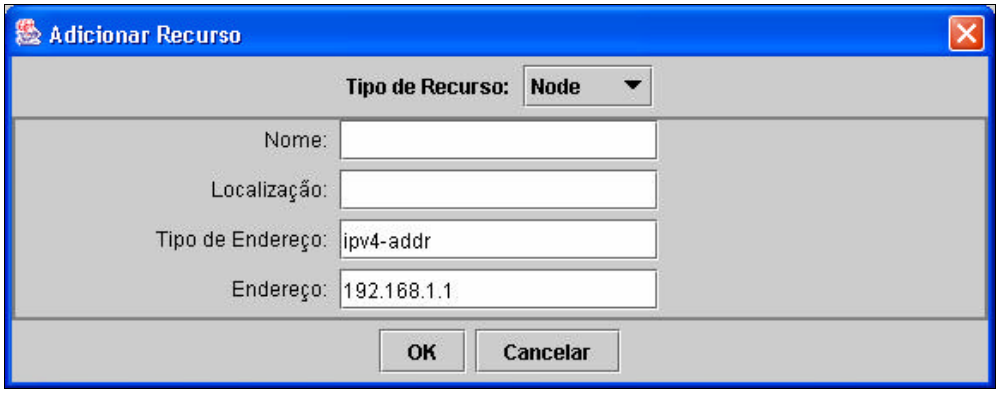

Figura 49 – Configuração do recurso da resposta.

Pressionando o botão “OK” nas duas telas que lhe foram abertas (“Adicionar Recurso” e “Adicionar Bloqueio”) o Operador volta para a tela de envio de respostas, agora com as informações de bloqueio e recurso já especificadas.

O estado da tela de envio de respostas no momento em que o Operador retorna é apresentado na Figura 50, nesta figura podemos observar que o tipo de resposta escolhida foi React e que foi informada uma descrição para a resposta.

Como o Operador já especificou um bloqueio, podemos também observar na tela que já existe uma linha na lista de bloqueios, esta linha apresenta para o Operador todas as informações que foram inseridas no cadastro do bloqueio, temos então na tela as informações das classes Block, Node e Address, que fazem parte da mensagem do tipo React. Note que a classe Node possui dois campos em branco, que correspondem aos campos Nome e Localização que não foram informados.

Neste exemplo não foram feitas solicitações de fechamento de recurso, devido a isto a lista de fechamento de recurso aparece vazia na Figura 50.

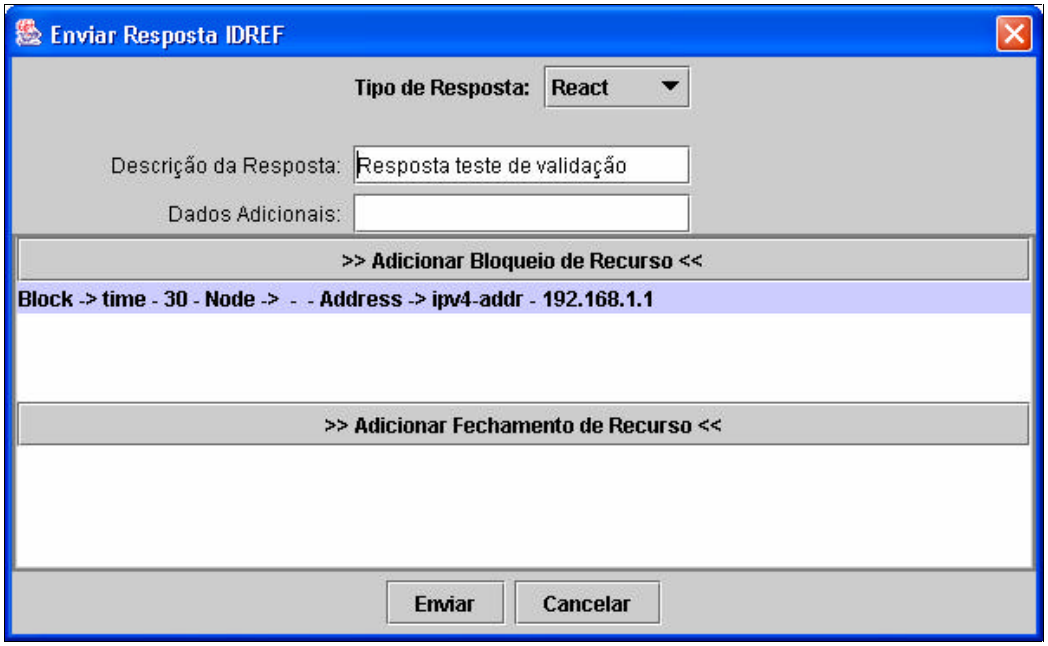

Figura 50 – Envio da resposta do teste de validação.

Se mais bloqueios forem cadastrados mais linhas serão inseridas na lista de bloqueios, o mesmo ocorre na lista de fechamentos, quando forem cadastradas informações para a classe Shutdown.

6.2.4 Transmissão de Mensagens IDREF

Com a resposta IDREF completamente configurada, basta o Operador pressionar o botão Enviar, neste momento o Gerenciador irá estabelecer uma sessão BEEP com o componente IDSRes, que deve ter sido previamente iniciado, e irá enviar a mensagem IDREF através do protocolo IDXP.

O componente IDSRes recebe a mensagem IDREF, aplica a mesma no ambiente conforme apresentado na seção 6.1.6, e a grava em um arquivo de *log* previamente configurado em sua tela, conforme apresentado na Figura 44 o arquivo de *log* configurado é "*c:\respostas.log*".

A mensagem recebida e gravada no *log* pelo IDSRes neste teste de validação é apresentada no Quadro 36. Podemos observar no Quadro 36 a descrição da resposta no elemento description, as informações sobre o Gerenciador no elemento Manager e a identificação do alerta que gerou a resposta no elemento alertident.

O elemento React contém um único elemento Block, cujos atributos especificam que o recurso deve ficar bloqueado por 30 minutos. O elemento Node contém as

informações do recurso a ser bloqueado, neste elemento as informações de nome e localização não contém valor, e o elemento Address contém o endereço 192.168.1.1 do tipo IPv4, que é o endereço do *host* que estava sendo atacado.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE IDREF-Message PUBLIC "-//UFSC//DTD IDREF v1//EN"
                               "./idref-message.dtd">
<IDREF-Message ident="1" version="1">
    <description>Resposta teste de validação</description>
    <Manager managerid="MAN1">
        <Node>
            <name>IDSMan</name>
            <Address type="ipv4-addr">192.168.1.3</Address>
        </Node>
        <Process pid="5486">
            <name>IDMan</name>
            <path>c:\IDSMan</path>
        </Process>
    </Manager>
    <CreateTime ntpstamp="0xc3ccc189.0x01400000">2004-02-
05T11:20:09Z</CreateTime>
    <alertident>0</alertident>
    <React>
        <Block ident="1" time="30" unblock="time">
            <Node>
                <location/>
                <name/>
                <Address type="ipv4-addr">192.168.1.1</Address>
            </Node>
        </Block>
    </React>
</IDREF-Message>
```

Quadro 36 – Resposta recebida pelo IDSRes.

Com isto verificamos que todas as informações que foram configuradas na resposta IDREF do Gerenciador chegaram ao componente IDSRes corretamente e foram aplicadas no ambiente de domínio do IDS.

Enquanto o bloqueio estiver ativo, caso seja executado outro ping para 192.168.1.1, o IDSRes irá interceptar esta comunicação e enviar para a origem do ping uma mensagem ICMP de *host* não acessível, impedindo a comunicação com o *host* bloqueado.

7 RESULTADOS E DISCUSSÃO

Neste capítulo serão apresentados os resultados obtidos no desenvolvimento do presente trabalho. Em seguida serão discutidos detalhes e características de cada um dos resultados.

7.1 Resultados

O presente trabalho tem como principal resultado um modelo de ambiente de detecção de intrusão que, ao contrário do modelo originalmente apresentado pelo IDWG, permite o tratamento dos Alertas IDMEF gerando respostas em um formato de dados pré-definido, o IDREF.

Outro resultado do trabalho surge da implementação do modelo proposto, desta temos como resultado a implementação de um Gerenciador de Alertas e a implementação de um componente que envia os Alertas ao Gerenciador, fazendo uma ponte entre o Analisador implementado por um IDS específico e o Gerenciador que trabalha sob os conceitos do grupo IDWG.

O resultado da implementação do modelo proposto segue além da arquitetura IDWG, implementando também o componente de Contra-Medidas e o modelo IDREF propostos no modelo do presente trabalho.

Como resultado final da implementação do modelo proposto temos um ambiente gerenciamento de detecção de intrusão que segue a arquitetura IDWG no que diz respeito ao envio de alertas, e também é capaz de gerar e enviar respostas aos alertas recebidos. Sendo que todo o ambiente implementado utiliza os protocolos BEEP/IDXP para comunicação, o modelo de dados IDMEF para formatar os Alertas e o modelo de dados IDREF para formatar as respostas.

Após a implementação, a construção do ambiente de validação juntamente com os componentes implementados criou um ambiente completo de detecção de instrusão para o IDS Snort.

O ambiente de validação apresenta-se como um resultado deste trabalho, que viabiliza o efetivo funcionamento, em um ambiente completo e real de detecção de intrusão, do modelo previamente proposto e implementado. Para o efetivo funcionamento do modelo proposto era necessário que um IDS gerasse em um arquivo texto mensagens IDMEF, e isto foi obtido na construção do ambiente de validação

recompilando-se o IDS Snort com um *plugin* específico para geração de mensagens IDMEF, resultando no ambiente completo de validação. O ambiente de validação é dito completo porque contempla todos os elementos previstos na arquitetura de IDS do modelo proposto (ver Figura 23).

Em [BUCHHEIM 2001] são sugeridos trabalhos futuros relacionados com a especificação e implementação do IDXP, também é sugerido o desenvolvimento de aplicações que utilizem o protocolo IDXP e o desenvolvimento de extensões tanto do protocolo IDXP quanto do modelo de dados IDMEF para que estes incorporem novas funcionalidades. A sugestão de extensão do protocolo IDXP e do modelo de dados IDMEF vem de encontro aos resultados obtidos pelo presente trabalho, que estende o protocolo e o modelo para suportarem a comunicação de respostas.

Uma das propostas de trabalho futuro de [JULISCH 2003] é a incorporação de características do modelo IDMEF no modelo de agrupamento desenvolvido no artigo. Outra possível sugestão de trabalho futuro, que facilitaria o trabalho do operador, é fazer com que ele possa enviar respostas ao ataque através do mesmo gerenciador que lhe auxiliou na identificação da causa principal do ataque. Para isto poderia ser utilizado o modelo proposto no presente trabalho.

O modelo STAT, descrito em [VIGNA 2001] e [VIGNA 2003], possibilita a criação de módulos de resposta, nos quais é possível enviar uma resposta a um alerta detectado. O modelo de respostas de sistemas de detecção de intrusão desenvolvido no presente trabalho poderia ser incorporado ao modelo STAT para formatar a comunicação de respostas entre os seus módulos.

7.2 Discussão Sobre o Modelo Proposto

O modelo proposto no capítulo 5, que envolve aspectos de arquitetura, protocolo de comunicação e modelo de dados, apresenta-se como um modelo compatível com os projetos desenvolvidos pelo grupo IDWG, pois tomando por base os trabalho deste grupo, o modelo proposto procurou alterar ao mínimo a arquitetura e o protocolo de comunicação desenvolvidos pelo IDWG, modificando apenas o necessário para a inclusão da funcionalidade de responder aos alertas.

O modelo de dados IDREF criado para responder aos alertas é orientado a objetos e implementado em XML, da mesma forma que o modelo de dados IDMEF desenvolvido pelo IDWG.

Conforme apresentado na seção 5.4.4, o modelo IDREF foi projetado para aproveitar ao máximo as informações contidas nos alertas IDMEF, ou seja, as informações fornecidas no alerta são transformadas em respostas para conter o próprio alerta. Esta característica faz com que o modelo IDREF torne-se uma extensão do modelo IDMEF, não no sentido de estender as classes do modelo IDMEF, mas sim no sentido de permitir que as informações do modelo IDMEF não parem no Gerenciador, mas continuem sendo utilizadas em um fluxo inverso ao dos alertas.

A forma como as classes e os atributos do modelo IDREF estão dispostos visa melhorar a legibilidade de suas informações quando apresentadas em XML, ou seja, desenvolveu-se o modelo de dados prevendo a sua implementação em XML.

De forma a melhorar a legibilidade em XML e manter a compatibilidade com o modelo IDMEF, informações que normalmente seriam implementadas como atributos foram implementadas como classes agregadas. Exemplos de classes que poderiam ser atributos e foram implementadas como classes agregadas são: name, patch, protocol e location.

Para definir se uma informação seria uma nova classe agregada ou apenas um atributo, na definição das classes e atributos do modelo IDREF optou-se pela seguinte regra: informações de conteúdo potencialmente grande (como *strings*, por exemplo) seriam novas classes agregadas e as informações de conteúdo menor (como números e enumerações, por exemplo) seriam atributos.

Uma exceção a esta regra ocorre na classe User, onde existe a classe agregada number que contém uma informação de tamanho pequeno. Isto ocorreu para se manter a compatibilidade com o modelo IDMEF, onde number também é uma classe, e porque na classe User a informação de number é opcional junto com name, e como name deve ser uma classe e não um atributo, pois seu conteúdo é uma *string*, number também foi implementado como uma classe.

Analisando os exemplos de resposta IDREF apresentados ao longo do trabalho (Quadro 20, Quadro 21, Quadro 22 e Quadro 36) verificamos que a classe Resource nunca é convertida em um elemento XML. No arquivo DTD das mensagens IDREF

(Anexo 3) também verificamos que a classe Resource foi especificada como uma entidade e não como um elemento, sendo substituída por uma de suas cinco classes agregadas quando invocada. Isto foi implementado desta forma para otimizar a representação das mensagens IDREF em XML, pois a classe Resource não possui atributos nem classes agregadas diretamente a si.

7.3 Discussão Sobre a Implementação do Modelo

O protocolo IDXP ainda não é muito utilizado pelos sistemas de detecção de intrusão existentes no mercado. Nas pesquisas realizadas neste trabalho encontrou-se apenas uma implementação do protocolo IDXP, descrita na seção 6.1.2, onde existem alguns exemplos básicos de como utilizar o protocolo.

No *plugin* de geração de mensagens IDMEF para Snort existe a intenção de se transportar os alertas gerados através de um protocolo próprio para isto, em versões futuras.

A implementação do modelo proposto neste trabalho aparece como uma ferramenta que permite diversos testes sobre os trabalhos que estão sendo desenvolvidos pelo grupo IDWG e por outros grupos com relação à comunicação, geração de alertas e respostas em sistemas de detecção de intrusão. Pois esta é uma das primeiras implementações que une o formato de mensagens IDMEF com os protocolos BEEP/IDXP em uma arquitetura compatível com a proposta do IDWG.

Também foi implementado neste trabalho uma biblioteca para manipulação das mensagens IDREF e um componente de Contra-Medidas, sendo assim todo o modelo proposto no capítulo 5 já está implementado em uma ferramenta, onde diversos testes futuros podem ser aplicados. Além disto, a biblioteca IDREF implementada e utilizada neste trabalho já está pronta para ser utilizada também no desenvolvimento de novas ferramentas.

A funcionalidade desempenhada pelo componente IDSAna não aparece na arquitetura de um IDS, pois o IDSAna não é um Analisador propriamente dito. Na arquitetura o IDSAna se localizaria dentro do Analisador, como parte de suas funcionalidades.

Neste trabalho foi desenvolvido um componente a parte, o IDSAna, para fazer a transmissão dos Alertas para que não fosse necessário alterar o IDS Snort internamente.

A alteração interna do IDS Snort, ou a criação de um *plugin* para transmissão das mensagens, seria um grande complicador devido à falta de uma biblioteca IDXP para a linguagem C.

Caso um IDS existente no mercado venha a suportar a arquitetura proposta pelo IDWG, ou ainda a arquitetura proposta por este trabalho, a funcionalidade de transmitir os Alertas para o Gerenciador pode ser implementada dentro de seu Analizador, e não necessariamente em um componente separado.

A implementação do Gerenciador IDSMan é capaz de gerar e transmitir todas as combinações de respostas que o modelo IDREF prevê, porém o componente de Contra-Medidas IDSRes apresentado neste trabalho implementa a aplicação efetiva de apenas três combinações de respostas: as respostas TCP do tipo Response, as respostas ICMP do tipo Response e as respostas de bloqueio de Node do tipo React, somente ainda quando estas forem configuradas para desbloquear o Node em um determinado espaço de tempo.

Estas três combinações de respostas foram implementadas apenas para exemplificar o modelo IDREF proposto, pois as combinações de respostas permitidas pelo modelo IDREF vão muito além das respostas implementadas no componente IDSRes.

A implementação de todas as respostas do modelo IDREF demanda um estudo individual sobre como implementar cada tipo de resposta. Para aplicar respostas em um recurso como User, por exemplo, é necessário pesquisar os mecanismos que cada sistema operacional oferece para tal tarefa.

As respostas do tipo Config são específicas para cada recurso a qual se pretende configurar, sendo assim é necessário estudar como cada tipo de recurso irá receber as suas configurações do componente de Contra-Medidas.

Para a implementação completa de um componente de Contra-Medidas seria interessante o desenvolvimento de um suporte a *plugins*, semelhante ao que existe no Snort. No sistema de *plugins* do componente de Contra-Medidas cada recurso instalaria o seu *plugin* no componente, informando como as respostas devem ser aplicadas a si, e o componente de Contra-Medidas apenas passaria a respostas ao *plugin* correspondente a cada recurso.

7.4 Discussão Sobre a Validação do Modelo

A validação do modelo apresentada no capítulo 6 demonstrou o funcionamento da implementação dos componentes. Além disto serviu para validar, através de um estudo de caso, a correta implementação do modelo previamente proposto.

O estudo de caso apresentado, bem como as respostas implementadas e escolhidas como exemplo, estão intimamente relacionadas com o tipo de IDS utilizado na validação.

Certas combinações de respostas estão mais relacionadas com certos tipos de IDS, devido ao tipo de informação que cada tipo de IDS obtém. Os IDSs baseados em rede relacionam-se principalmente respostas que envolvem o tráfego, serviços ou equipamentos da rede, pois este é o tipo de informação que os IDSs baseados em rede monitoram. Os IDSs baseados em *host* relacionam-se principalmente com respostas que envolvem recursos como processos, usuários ou arquivos do sistema operacional. E os IDSs baseados em aplicativos, quando desejarem realizar alguma ação dentro do aplicativo, irão relacionar-se principalmente com respostas de configuração, pois o conteúdo da resposta será específico para cada aplicativo.

O fato de trabalhar com dados e suportar respostas de diversos tipos de IDS demonstram toda a flexibilidade e capacidade tanto do modelo de Alertas IDMEF quanto do modelo de respostas IDREF.

Na validação apresentada neste trabalho foi utilizado o IDS Snort. O Snort é um IDS de rede, onde as informações analisadas são obtidas através do monitoramento do tráfego da rede. Devido a isto, optou-se pela implementação das respostas TCP e ICMP do tipo Response, e das respostas tipo React de bloqueio de Node.

8 CONCLUSÕES

Este trabalho apresentou um modelo de ambiente de detecção de intrusão, descrevendo sua arquitetura, seus protocolos de comunicação e os modelos de dados utilizados pelo ambiente para troca de informações entre seus componentes.

O modelo de ambiente proposto tem como característica particular a capacidade de enviar respostas aos alertas recebidos, sendo que estas respostas estão formatadas de acordo com um modelo de dados específico desenvolvido neste trabalho, o IDREF. Este modelo visa aproveitar ao máximo as informações fornecidas pelo modelo de Alertas IDMEF desenvolvido pelo IDWG e também utilizado no modelo de ambiente detecção de intrusão proposto.

Desenvolvido de acordo com os trabalhos já realizados pelo grupo IDWG, o modelo de ambiente de detecção de intrusão proposto visa a interoperabilidade entre os IDSs de fabricantes diferentes. Porém a interoperabilidade suportada pelo modelo proposto vai além da comunicação de Alertas apresentada pelo modelo do grupo IDWG, chegando também à comunicação das respostas aos Alertas, através do modelo de dados IDREF.

Conforme verificado através da pesquisa de trabalhos correlatos, apresentada na seção 1.4, diversos trabalhos relacionados com IDS tratam do envio de alertas e suas respectivas respostas. Com relação aos alertas, o trabalho do grupo IDWG trata de reuni-los em um modelo de forma a permitir a interoperabilidade entre os IDSs. Porém com relação às respostas, não se tem conhecimento de um trabalho que vise reuni-las em um modelo único a fim de permitir uma interoperabilidade ainda maior entre diferentes IDSs. Assim, o modelo visto neste trabalho se apresenta como uma nova opção a ser incorporada no desenvolvimento de sistemas de detecção de intrusão.

Utilizando o modelo proposto e implementado neste trabalho é possível gerenciar Alertas originados em IDSs de diversos tipo e fabricantes, enviando respostas para um componente de Contra-Medidas de um fabricante sobre um alerta gerado por um IDS de outro fabricante. Obtendo assim a cooperação total entre os IDS, tanto no recebimento de alertas quanto no envio de respostas.

O modelo proposto foi desenvolvido tomando-se por base o modelo de detecção de intrusão desenvolvido pelo grupo IDWG. Inicialmente obteve-se a arquitetura de IDS (Figura 2), o protocolo de comunicação (IDXP) e o modelo de dados (IDMEF)

utilizados pelo IDWG. Em seguida foram feitas as alterações necessárias para o suporte ao envio de respostas.

A arquitetura IDWG foi estendida, adicionando-se o caminho a ser percorrido pelas respostas (Figura 23). O protocolo de comunicação também foi estendido, de forma a possibilitar a configuração de canais específicos para o envio de respostas (ver seção 5.3). E foi desenvolvido um novo modelo de dados, o IDREF, específico para a formatação de respostas. O modelo IDREF foi desenvolvido baseado nas informações obtidas através dos Alertas IDMEF.

Com isto atigimos os objetivos gerais inicialmente propostos por este trabalho na seção 1.2.1.

Após desenvolver as extensões do modelo IDWG, foram implementados os componentes do novo modelo resultante. Os componentes implementados foram testados e validados em um ambiente de validação, verificando-se que estavam de acordo com o modelo previamente especificado.

As extensões propostas ao modelo IDWG, juntamente com a implementação de seus componentes e de suas extensões propostas, demonstram que foram atingidos os objetivos específicos inicialmente propostos por este trabalho na seção 1.2.2.

A apresentação das funcionalidades contidas nos componentes desenvolvidos e a sua posterior validação demonstram que o modelo especificado e implementado neste trabalho atua como um mecanismo de segurança para o envio rápido e eficiente de respostas a ataques. Pois através dos componentes desenvolvidos é possível gerenciar os Alertas de diversos tipos e fabricantes de IDS diferentes de maneira centralizada e enviar respostas em um mesmo formato independente de qual IDS gerou o alerta. Com isto não é nessário que o operador conheça detalhes de todos os tipos de IDSs que fazem parte de seu ambiente, basta conhecer o formato utilizado para o envio das respostas.

Sendo assim, o presente trabalho contribui para o desenvolvimento da segurança das redes de computadores, no sentido de prover um modelo de ambiente de detecção de intrusão que possibilita a interoperabilidade entre diferentes tipos e fabricantes de IDSs, sendo que a interoperabilidade apresentada por este modelo vai além do escopo de alertas, suportando também o tratamento de respostas aos alertas.

8.1 Dificuldades Encontradas

No decorrer do trabalho foram encontradas algumas dificuldades, dentre as quais as mais relevantes foram:

- dificuldade em se obter uma biblioteca que implementasse o protocolo IDXP. Na linguagem C não foi possível encontrar tal biblioteca. A única biblioteca encontrada, escrita na na liguagem Java, ainda não estava liberada para utilização. Foi necessário baixar todos os arquivos do repositório de fontes e compilá-los separadamente para se obter a biblioteca.
- pouca disponibilidade de IDSs que têm a capacidade de gerar mensagens IDMEF. O Snort é um dos IDSs mais populares, devido a isto já foi desenvolvido um *plugin* para que o mesmo gere mensagens IDMEF. Uma maior quantidade de IDSs com esta característica poderia enriquecer a validação do modelo proposto;
- dificuldade em se aplicar o *plugin* IDMEF ao Snort. Existiam incompatibilidades entre as bibliotecas informadas na documentação, foi necessário baixar várias versões das bibliotecas para verificar quais compilavam entre si e com o *plugin.* Também foi necessário corrigir erros de compilação e realizar alterações que não estavam descritas na documentação do *plugin.*

8.2 Trabalhos Futuros

Vários outros trabalhos podem ser desenvolvidos de forma dar continuidade ao que foi exposto no presente trabalho, entre eles:

- analisar e validar o modelo IDREF junto à uma grande quantidade de IDSs existentes no mercado, de forma a propor melhorias ou novas funcionalidades à si;
- analisar e implementar no Gerenciador de IDSs o preenchimento automático das informações de resposta, baseado nas informações do alerta que está sendo respondido. Este preenchimento seria feito em forma de sugestão ao Operador, tornando a geração das respostas mais simples, rápida e eficiente;

- analisar e implementar a aplicação de outros tipos de respostas IDREF no componente de Contra-Medidas, além das que foram implementados no presente trabalho;
- desenvolver uma estrutura de *plugin* no componente de contra-medidas, de forma que não seja necessário alterá-lo internamente para adicionar a aplicação de novos tipos de respostas IDREF;
- avaliar o funcionamento da arquitetura proposta em um ambiente com outros tipos de IDSs, diferentes do Snort, como IDSs baseados em *host*, por exemplo.

REFERÊNCIAS BIBLIOGRÁFICAS

AMOROSO, Edward. **Intrusion Detection: An intruduction to Internet Surveillance, Correlation, Trace Back, Traps, and Response.** Intrusion.Net Books, New Jersey, 1999.

BETSER, J. **The Challenge of Creating Productive Collaborating Information Assurance Communities via Internet Research and Standards.** 20th IEEE Symposium on Reliable Distributed Systems (SRDS'01), Louisiana, Outubro de 2001.

BUCHHEIM, Tim; ERLINGER, Michael; et al. **Implementing the intrusion detection exchange protocol.** 17th Annual Computer Security Applications Conference (ACSAC'01), Louisiana, Dezembro de 2001.

CUPPENS, Frédéric; MIÈGE, Alexandre. **Alert Correlation in a Cooperative Intrusion Detection *Frame*work.** Proceedings of the 2002 IEEE Symposium on Security and Privacy (S&P'02), California, Maio de 2002.

CURRY, D.; DEBAR, H. **Intrusion Detection Message exchange format data model and Extensible Markup Language (XML) Document *Type* Definition.** Draft-ietf-idwg-idmef-xml-10, Janeiro de 2003. Disponível por http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-10.txt. Acesso em 22 dez 2003.

FEINSTEIN, Ben; BUCHHEIM, Tim; et al. **GlobalGuard: Creating the IETF-IDWG Intrusion Alert Protocol.** DARPA Information Survivability Conference and Exposition (DISCEX II'01) Volume I-Volume 1, California, Junho de 2001.

FEINSTEIN, Ben; MATTHEWS, G.; WHITE, J. **The Intrusion Detection Exchange Protocol (IDXP).** Draft-ietf-idwg-beep-idxp-07, Outubro de 2002. Disponível por http://www.ietf.org/internet-drafts/draft-ietf-idwg-beep-idxp-07.txt. Acesso em 22 dez 2003.

FREED, N.; BORENSTEIN, N. **Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.** RFC 2045, Novembro de 1996. Disponível por http://www.ietf.org/rfc/rfc2045.txt. Acesso em 22 dez. 2003.

FUJII, Keita. **JPcap Project**. Início das atividades em Junho de 2000. Disponível por http://netresearch.ics.uci.edu/kfujii/jpcap/doc/index.html/. Acesso em 05 Dez. 2003.

HESS, A.; JUNG, M.; SCHÄFER, G. **FIDRAN: A Flexible Intrusion Detection and Response *Frame*work for Active Networks.** Proceedings of the Eighth IEEE International Symposium on Computers and Communication (ISCC'03), Antalya, Julho de 2003.

JULISCH, Klaus. **Clustering intrusion detection alarms to support root cause analysisn.** ACM Transactions on Information and System Security (TISSEC), Novembro de 2003.

KAHN, Cliffordn; PORRAS, Phillip; et al. **A *common* intrusion detection *frame*work.** Journal of Computer Security, Julho de 1998.

KOILPILLAI, Juanita; BEAVERS, John B.; SWINTON, Paul. **Recon - A Tool for Incident Detection, Tracking and Response.** DARPA Information Survivability Conference & Exposition - Volume 1, South Carolina, Janeiro de 2000.

LibNTP Project. Disponível por http://www.ntp.org. Acesso em 05 Dez. 2003.

NING, Peng; JAJODIA, Sushil; WANG, XiaoYang Sean. **Abstraction-based intrusion detection in distributed environments.** ACM Transactions on Information and System Security (TISSEC), Novembro de 2001.

NIST - National Institute for Standards and Techno*log*y. **Special Publication on Intrusion Detection Systems**. Novembro de 2001. Disponível por http://csrc.nist.gov/publications/nistpubs/800-31/sp800-31.pdf. Acesso em 15 Nov. 2003.

PETKAC, Mike; BADGER, Lee. **Security Agility in Response to Intrusion Detection.** 16th Annual Computer Security Applications Conference (ACSAC'00), Louisiana, Dezembro de 2000.

PROCTOR, Paul. **Practical Intrusion Detection Handbook.** Prentice Hall, New Jersey, 2001.

ROSE, M. **The Blocks Extensible Exchange Protocol Core.** RFC 3080, Março de 2001. Disponível por http://www.ietf.org/rfc/rfc3080.txt. Acesso em 22 dez 2003.

ROSE, M. **Mapping de BEEP core onto TCP.** RFC 3081, Março de 2001a. Disponível por http://www.ietf.org/rfc/rfc3081.txt. Acesso em 22 dez. 2003.

SCHNACKENBERG, Dan; DJAHANDARI, Kelly; STERNE, Dan. **Infrastructure for Intrusion Detection and Response.** DARPA Information Survivability Conference & Exposition - Volume 2, South Carolina, Janeiro de 2000.

Silicon Defense. **JavaIDMEF Project**. Início das atividades em Maio de 2001. Disponível por http://www.silicondefense.com/idwg/javaidmef/javaidmef.html. Acesso em 20 Nov. 2003.

Silicon Defense. **Snort IDMEF XML *plugin* Project.** Início das atividades em Julho de 2001. Disponível por http://www.silicondefense.com/idwg/snort-idmef/. Acesso em 10 Jan. 2004.

Silicon Defense. **LibIDMEF Project.** Início das atividades em Janeiro de 2001. Disponível por http://www.silicondefense.com/idwg/libidmef/. Acesso em 10 Jan. 2004.

SourceFire. **Snort Project**. Disponível por http://www.snort.org. Acesso em 05 Dez. 2003.

SourceForge. **Beepcore - Java Project**. Início das atividades em Março de 2001. Disponível por http://sourceforge.net/projects/beepcore-java. Acesso em 20 Nov. 2003.

SourceForge. **IDXP - Java Project**. Início das atividades em Abril de 2001. Disponível por http://sourceforge.net/projects/idxp-java. Acesso em 20 Nov. 2003.

SUN, Bo; WU, Kui; POOCH, Udo W. **Alert aggregation in mobile ad hoc networks.** Proceedings of the 2003 ACM workshop on Wireless security, San Diego, Setembro de 2003.

The Apache Software Foundation. **Xerces2 Java Parser Project.** Início das atividades em 1999. Disponível por http://xml.apache.org/xerces2-j/index.html. Acesso em 20 Nov. 2003.

VIGNA, Giovanni; KEMMERER, Richard A.; BLIX, Per. **Designing a web of highly-configurable intrusion detection sensors.** Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection, Outubro de 2001.

VIGNA, Giovanni; VALEUR, Fredrik; KEMMERER Richard A. **Designing and implementing a family of intrusion detection systems.** Proceedings of the 9th European software engineering conference held jointly with 10th ACM SIGSOFT international symposium on Foundations of software engineering, Helsinki, Setembro de 2003.

WOOD, M. **Intrusion Detection message exchange requirements.** Draft-ietf-idwg-requirements-10, Outubro de 2002. Disponível por http://www.ietf.org/internet-drafts/draft-ietf-idwg-requirements-10.txt. Acesso em 22 dez 2003.

W3C - World Wide Web Consortium. **LibXML2 Project.** Início das atividades em 1999. Disponível por http://www.xmlsoft.org/. Acesso em 10 Jan. 2004.

W3C - World Wide Web Consortium.**Extensible Markup Language XML.** Outubro de 2000. Disponível por http://www.w3.org/TR/2000/REC-xml-20001006. Acesso em 22 dez 2003.

ZHANG, Yongguang; LEE, Wenke; HUANG, Yi-An. **Intrusion detection techniques for mobile wireless networks.** ACM Wireless Networks Journal, Setembro de 2003.

ANEXO 1

Alguns Sistemas de Detecção de Intrusão existentes no mercado e suas características:

IDS	Características
Intruder Alert	- Baseado em *Host* - Desenvolvido por Axent - *Site* www.axent.com
NetProwler	- Baseado em Rede - Desenvolvido por Axent - *Site* www.axent.com
RealSecure	- Baseado em *Host* e Rede - Desenvolvido por ISS - *Site* www.iss.net
NetRanger	- Baseado em Rede - Desenvolvido por Cisco - *Site* www.cisco.com
SessionWall	- Baseado em Rede - Desenvolvido por C. Associates - *Site* www.ca.com
Snort	- Baseado em Rede - Desenvolvido por SourceFire - *Site* www.snort.org
Abacus	- Baseado em *Host* - Desenvolvido por Psionic - *Site* www.psionic.com
Bro	- Baseado em Rede - Desenvolvido por Lawrence Berkeley National Laboratory - *Site* http://ee.lbl.gov/bro.html
AAFID	- Baseado em *Host* e Rede - Desenvolvido por CERIAS - *Site* http://www.cerias.purdue.edu/homes/aafid/
Emerald	- Baseado em Rede - Desenvolvido por SRI International - *Site* http://www.sdl.sri.com/projects/emerald/
NFR	- Baseado em Rede - Desenvolvido por NFR - *Site* www.nfr.net

ANEXO 2

Arquivos "*IDMEF-Message*.dtd" desenvolvido pelo grupo IDWG, que contém a definição de tipo de documento utilizada para formatar as mensagens IDMEF em XML.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ************************************************************
**********************************************************************
*** Intrusion Detection Message Exchange Format (IDMEF) XML DTD ***
***                Version 1.0, 30 January 2003                    ***
***                                                                ***
*** The use and extension of the IDMEF XML DTD are described in ***
*** RFC XXXX, "Intrusion Detection Message Exchange Format Data ***
*** Model and Extensible Markup Language (XML) Document Type    ***
*** Definition," D. Curry and H. Debar.                          ***
**********************************************************************
************************************************************ -->

<!-- ====================================================================
======================================================================
=== SECTION 1. Attribute list declarations.
======================================================================
==================================================================== -->

<!--
| Attributes of the IDMEF element.  In general, the fixed values of
| these attributes will change each time a new version of the DTD
| is released.
-->
<!ENTITY % attlist.idmef                "
version             CDATA                   #FIXED    '1.0'
">

<!--
| Attributes of all elements.  These are the "XML" attributes that
| every element should have.  Space handling, language, and name
| space.
-->
<!ENTITY % attlist.global               "
xmlns:idmef         CDATA                   #FIXED
    'urn:iana:xml:ns:idmef'
xmlns               CDATA                   #FIXED
    'urn:iana:xml:ns:idmef'
xml:space           (default | preserve)    'default'
xml:lang            NMTOKEN                 #IMPLIED
">

<!-- ====================================================================
======================================================================
=== SECTION 2. Attribute value declarations.  Enumerated values for
===            many of the element-specific attribute lists.
======================================================================
==================================================================== -->

<!--
| Values for the Action.category attribute.
-->
<!ENTITY % attvals.actioncat            "
( block-installed | notification-sent | taken-offline | other )
">

<!--
| Values for the Address.category attribute.
-->
<!ENTITY % attvals.addrcat              "
( unknown | atm | e-mail | lotus-notes | mac | sna | vm |
  ipv4-addr | ipv4-addr-hex | ipv4-net | ipv4-net-mask |
```

```
  ipv6-addr | ipv6-addr-hex | ipv6-net | ipv6-net-mask )
">

<!--
| Values for the AdditionalData.type attribute.
-->
<!ENTITY % attvals.adtype               "
( boolean | byte | character | date-time | integer | ntpstamp |
  portlist | real | string | xml )
">

<!--
| Values for the Impact.completion attribute.
-->
<!ENTITY % attvals.completion           "
( failed | succeeded )
">

<!--
| Values for the File.category attribute.
-->
<!ENTITY % attvals.filecat              "
( current | original )
">

<!--
| Values for the Id.type attribute.
-->
<!ENTITY % attvals.idtype               "
( current-user | original-user | target-user | user-privs |
  current-group | group-privs | other-privs )
">

<!--
| Values for the Impact.type attribute.
-->
<!ENTITY % attvals.impacttype           "
( admin | dos | file | recon | user | other )
">

<!--
| Values for the Linkage.category attribute.
-->
<!ENTITY % attvals.linkcat              "
( hard-link | mount-point | reparse-point | shortcut | stream |
  symbolic-link )
">

<!--
| Values for the Node.category attribute.
-->
<!ENTITY % attvals.nodecat              "
( unknown | ads | afs | coda | dfs | dns | hosts | kerberos |
  nds | nis | nisplus | nt | wfw )
">

<!--
| Values for the Classification.origin attribute.
-->
<!ENTITY % attvals.origin               "
( unknown | bugtraqid | cve | vendor-specific )
">

<!--
| Values for the Confidence.rating attribute.
-->
<!ENTITY % attvals.rating               "
( low | medium | high | numeric )
">

<!--
| Values for the Impact.severity attribute.
-->
```

```
<!ENTITY % attvals.severity             "
( low | medium | high )
">

<!--
| Values for the User.category attribute.
-->
<!ENTITY % attvals.usercat              "
( unknown | application | os-device )
">

<!--
| Values for yes/no attributes such as Source.spoofed and
| Target.decoy.
-->
<!ENTITY % attvals.yesno                "
( unknown | yes | no )
">


<!-- ====================================================================
======================================================================
=== SECTION 3. Top-level element declarations.  The IDMEF-Message
===            element and the types of messages it can include.
======================================================================
===================================================================== -->

<!ELEMENT IDMEF-Message                 (
(Alert | Heartbeat)*
)>
<!ATTLIST IDMEF-Message
%attlist.global;
%attlist.idmef;
>

<!ELEMENT Alert                         (
Analyzer, CreateTime, DetectTime?, AnalyzerTime?, Source*,
Target*, Classification+, Assessment?, (ToolAlert |
OverflowAlert | CorrelationAlert)?, AdditionalData*
)>
<!ATTLIST Alert
ident               CDATA                   '0'
%attlist.global;
>

<!ELEMENT Heartbeat                     (
Analyzer, CreateTime, AnalyzerTime?, AdditionalData*
)>
<!ATTLIST Heartbeat
ident               CDATA                   '0'
%attlist.global;
>

<!-- ====================================================================
======================================================================
=== SECTION 4. Subclasses of the Alert element that provide more
===            data for specific types of alerts.
======================================================================
===================================================================== -->

<!ELEMENT CorrelationAlert              (
name, alertident+
)>
<!ATTLIST CorrelationAlert
%attlist.global;
>

<!ELEMENT OverflowAlert                 (
program, size?, buffer?
)>
<!ATTLIST OverflowAlert
%attlist.global;
>
```

```
<!ELEMENT ToolAlert                     (
name, command?, alertident+
)>
<!ATTLIST ToolAlert
%attlist.global;
>

<!-- ====================================================================
======================================================================
=== SECTION 5.  The AdditionalData element.  This element allows an
===             alert to include additional information that cannot
===             be encoded elsewhere in the data model.
======================================================================
==================================================================== -->

<!ELEMENT AdditionalData            ANY >
<!ATTLIST AdditionalData
type                %attvals.adtype;        'string'
meaning             CDATA                   #IMPLIED
%attlist.global;
>

<!-- ====================================================================
======================================================================
=== SECTION 6. Elements related to identifying entities - analyzers
===            (the senders of these messages), sources (of
===            attacks), and targets (of attacks).
======================================================================
==================================================================== -->

<!ELEMENT Analyzer                      (
Node?, Process?
)>
<!ATTLIST Analyzer
analyzerid          CDATA                   '0'
manufacturer        CDATA                   #IMPLIED
model               CDATA                   #IMPLIED
version             CDATA                   #IMPLIED
class               CDATA                   #IMPLIED
ostype              CDATA                   #IMPLIED
osversion           CDATA                   #IMPLIED
%attlist.global;
>

<!ELEMENT Source                        (
Node?, User?, Process?, Service?
)>
<!ATTLIST Source
ident               CDATA                   '0'
spoofed             %attvals.yesno;         'unknown'
interface           CDATA                   #IMPLIED
%attlist.global;
>

<!ELEMENT Target                        (
Node?, User?, Process?, Service?, FileList?
)>
<!ATTLIST Target
ident               CDATA                   '0'
decoy               %attvals.yesno;         'unknown'
interface           CDATA                   #IMPLIED
%attlist.global;
>

<!-- ====================================================================
======================================================================
=== SECTION 7. Support elements used for providing detailed info
===            about entities - addresses, names, etc.
======================================================================
==================================================================== -->

<!ELEMENT Address                       (
```

```
address, netmask?
)>
<!ATTLIST Address
ident               CDATA                   '0'
category            %attvals.addrcat;       'unknown'
vlan-name           CDATA                   #IMPLIED
vlan-num            CDATA                   #IMPLIED
%attlist.global;
>

<!ELEMENT Assessment                    (
Impact?, Action*, Confidence?
)>
<!ATTLIST Assessment
%attlist.global;
>

<!ELEMENT Classification                (
name, url
)>
<!ATTLIST Classification
origin              %attvals.origin;        'unknown'
%attlist.global;
>

<!ELEMENT File                          (
name, path, create-time?, modify-time?, access-time?,
data-size?, disk-size?, FileAccess*, Linkage*, Inode?
)>
<!ATTLIST File
ident               CDATA                   '0'
category            %attvals.filecat;       #REQUIRED
fstype              CDATA                   #REQUIRED
%attlist.global;
>

<!ELEMENT FileAccess                    (
UserId, permission+
)>
<!ATTLIST FileAccess
%attlist.global;
>

<!ELEMENT FileList                      (
File+
)>
<!ATTLIST FileList
%attlist.global;
>

<!ELEMENT Inode                         (
change-time?, (number, major-device, minor-device)?,
(c-major-device, c-minor-device)?
)>
<!ATTLIST Inode
%attlist.global;
>

<!ELEMENT Linkage                       (
(name, path) | File
)>
<!ATTLIST Linkage
category            %attvals.linkcat;       #REQUIRED
%attlist.global;
>
<!ELEMENT Node                          (
location?, (name | Address), Address*
)>
<!ATTLIST Node
ident               CDATA                   '0'
category            %attvals.nodecat;       'unknown'
%attlist.global;
>
```

```
<!ELEMENT Process                       (
name, pid?, path?, arg*, env*
)>
<!ATTLIST Process
ident               CDATA                   '0'
%attlist.global;
>

<!ELEMENT Service                       (
(((name, port?) | (port, name?)) | portlist), protocol?,
SNMPService?, WebService?
)>
<!ATTLIST Service
ident               CDATA                   '0'
%attlist.global;
>

<!ELEMENT SNMPService                   (
oid?, (community | (securityName, contextName,
contextEngineID))?, command?
)>
<!ATTLIST SNMPService
%attlist.global;
>

<!ELEMENT User                          (
UserId+
)>
<!ATTLIST User
ident               CDATA                   '0'
category            %attvals.usercat;       'unknown'
%attlist.global;
>

<!ELEMENT UserId                        (
(name, number?) | (number, name?)
)>
<!ATTLIST UserId
ident               CDATA                   '0'
type                %attvals.idtype;        'original-user'
%attlist.global;
>

<!ELEMENT WebService                    (
url, cgi?, http-method?, arg*
)>
<!ATTLIST WebService
%attlist.global;
>

<!-- ====================================================================
======================================================================
=== SECTION 8. Simple elements with sub-elements or attributes of a
===            special nature.
======================================================================
==================================================================== -->

<!ELEMENT Action              (#PCDATA) >
<!ATTLIST Action
category            %attvals.actioncat;     'other'
%attlist.global;
>

<!ELEMENT AnalyzerTime        (#PCDATA) >
<!ATTLIST AnalyzerTime
ntpstamp            CDATA                   #REQUIRED
%attlist.global;
>

<!ELEMENT Confidence          (#PCDATA) >
<!ATTLIST Confidence
rating              %attvals.rating;        'numeric'
```

```
%attlist.global;
>

<!ELEMENT CreateTime          (#PCDATA) >
<!ATTLIST CreateTime
ntpstamp            CDATA                   #REQUIRED
%attlist.global;
>

<!ELEMENT DetectTime          (#PCDATA) >
<!ATTLIST DetectTime
ntpstamp            CDATA                   #REQUIRED
%attlist.global;

>

<!ELEMENT Impact              (#PCDATA) >
<!ATTLIST Impact
severity            %attvals.severity;      #IMPLIED
completion          %attvals.completion;    #IMPLIED
type                %attvals.impacttype;    'other'
%attlist.global;
>

<!ELEMENT alertident          (#PCDATA) >
<!ATTLIST alertident
analyzerid          CDATA                   #IMPLIED
%attlist.global;
>

<!-- ====================================================================
======================================================================
=== SECTION 9. Simple elements with no sub-elements and no special
===            attributes.
======================================================================
==================================================================== -->

<!ELEMENT access-time         (#PCDATA) >
<!ATTLIST access-time
%attlist.global;
>

<!ELEMENT address             (#PCDATA) >
<!ATTLIST address
%attlist.global;
>

<!ELEMENT arg                 (#PCDATA) >
<!ATTLIST arg
%attlist.global;
>

<!ELEMENT buffer              (#PCDATA) >
<!ATTLIST buffer
%attlist.global;
>

<!ELEMENT c-major-device      (#PCDATA) >
<!ATTLIST c-major-device
%attlist.global;
>

<!ELEMENT c-minor-device      (#PCDATA) >
<!ATTLIST c-minor-device
%attlist.global;
>

<!ELEMENT cgi                 (#PCDATA) >
<!ATTLIST cgi
%attlist.global;
>

<!ELEMENT change-time         (#PCDATA) >
```

```
<!ATTLIST change-time
%attlist.global;
>

<!ELEMENT command            (#PCDATA) >
<!ATTLIST command
%attlist.global;
>

<!ELEMENT community          (#PCDATA) >
<!ATTLIST community
%attlist.global;
>

<!ELEMENT create-time        (#PCDATA) >
<!ATTLIST create-time
%attlist.global;
>

<!ELEMENT data-size          (#PCDATA) >
<!ATTLIST data-size
%attlist.global;
>

<!ELEMENT disk-size          (#PCDATA) >
<!ATTLIST disk-size
%attlist.global;
>

<!ELEMENT env                (#PCDATA) >
<!ATTLIST env
%attlist.global;
>

<!ELEMENT http-method        (#PCDATA) >
<!ATTLIST http-method
%attlist.global;
>

<!ELEMENT location           (#PCDATA) >
<!ATTLIST location
%attlist.global;
>

<!ELEMENT major-device       (#PCDATA) >
<!ATTLIST major-device
%attlist.global;
>

<!ELEMENT minor-device       (#PCDATA) >
<!ATTLIST minor-device
%attlist.global;
>

<!ELEMENT modify-time        (#PCDATA) >
<!ATTLIST modify-time
%attlist.global;
>

<!ELEMENT name               (#PCDATA) >
<!ATTLIST name
%attlist.global;
>

<!ELEMENT netmask            (#PCDATA) >
<!ATTLIST netmask
%attlist.global;
>

<!ELEMENT number             (#PCDATA) >
<!ATTLIST number
%attlist.global;
>
```

```
<!ELEMENT oid                (#PCDATA) >
<!ATTLIST oid
%attlist.global;
>

<!ELEMENT path               (#PCDATA) >
<!ATTLIST path
%attlist.global;
>

<!ELEMENT permission         (#PCDATA) >
<!ATTLIST permission
%attlist.global;
>

<!ELEMENT pid                (#PCDATA) >
<!ATTLIST pid
%attlist.global;
>

<!ELEMENT port               (#PCDATA) >
<!ATTLIST port
%attlist.global;
>

<!ELEMENT portlist           (#PCDATA) >
<!ATTLIST portlist
%attlist.global;
>

<!ELEMENT program            (#PCDATA) >
<!ATTLIST program
%attlist.global;
>

<!ELEMENT protocol           (#PCDATA) >
<!ATTLIST protocol
%attlist.global;
>

<!ELEMENT size               (#PCDATA) >
<!ATTLIST size
%attlist.global;
>

<!ELEMENT url                (#PCDATA) >
<!ATTLIST url
%attlist.global;
>
```

ANEXO 3

Arquivos “idref-message.dtd” desenvolvido no presente trabalho, que contém a definição de tipo de documento utilizada para formatar as mensagens IDREF em XML.

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- ********************************************************************
 ********************************************************************
 *** UFSC - Universidade Federal de Santa Catarina                  ***
 *** CPGCC - Curso de Pós-Graduação em Ciências da Computação      ***
 *** LRG - Laboratório de Gerência de Redes                        ***
 *** Carlos Becker Westphall                                       ***
 *** Paulo Fernando da Silva                                       ***
 ***                                                              ***
 *** Intrusion Detection Response Exchange Format (IDREF) XML DTD ***
 ***                 Versão 1.0, 15 Janeiro 2004                  ***
 ***                                                              ***
 ********************************************************************
 ******************************************************************** -->

<!-- ====================================================================
 === Secao 1. Declaracao de Entidades. ==============================
 ==================================================================== -->


<!ENTITY % Resource "Node | Process | Service | UserList |
    FileList">

<!-- ====================================================================
 === Secao 2. Declaracao de valores de atributos. ===================
 ==================================================================== -->

<!ENTITY % attvals.usercat              "
    ( unknown | application | os-device )
  ">

<!ENTITY % attvals.usertype               "
    ( current-user | original-user | target-user | user-privs |
      current-group | group-privs )
  ">

<!ENTITY % attvals.additype               "
    ( boolean | byte | character | date-time | integer | ntpstamp |
      portlist | real | string | xml )
  ">

<!ENTITY % attvals.addrtype              "
    ( unknown | atm | e-mail | lotus-notes | mac | sna | vm |
      ipv4-addr | ipv4-addr-hex | ipv4-net | ipv4-net-mask |
      ipv6-addr | ipv6-addr-hex | ipv6-net | ipv6-net-mask | fone |
      page )
  ">

<!ENTITY % attvals.unblock              "
    ( reset | time )
  ">


<!-- ====================================================================
 === Secao 3. Declaracao das classes principais. ====================
 ==================================================================== -->

<!ELEMENT IDREF-Message                 (
    description?,
    Manager,
    CreateTime,
    alertident+,
    AdditionalData*,
    (Response | React | Config)
```

```
  )>
<!ATTLIST IDREF-Message
    ident             CDATA                  #REQUIRED
    version           CDATA                  #FIXED    '1'
  >

<!ELEMENT Response                      (
    Address+, (Notify | TCP | ICMP)
  )>

<!ELEMENT Notify                  (#PCDATA) >

<!ELEMENT TCP EMPTY>
<!ATTLIST TCP
    source            CDATA                  #REQUIRED
    target            CDATA                  #REQUIRED
    flags             CDATA                  #REQUIRED
  >

<!ELEMENT ICMP EMPTY>
<!ATTLIST ICMP
    type            CDATA                  #REQUIRED
    code            CDATA                  #REQUIRED
  >

<!ELEMENT React                         (
    Block*, Shutdown*
  )>

<!ELEMENT Config                        (
    (%Resource;), command+
  )>


<!-- ================================================================
 === Secao 4. Declaracao das classes auxiliares. ===================
 ================================================================ -->

<!ELEMENT Manager                      (
    Node, Process?
  )>
<!ATTLIST Manager
    managerid          CDATA               #REQUIRED
  >


<!-- ================================================================
 === Secao 5. Declaracao das classes de recurso. ===================
 ================================================================ -->

<!ELEMENT Node                          (
    location?, name?, Address
  )>

<!ELEMENT Process                       (
    name?, path?
  )>
<!ATTLIST Process
    pid               CDATA                  #REQUIRED
  >

<!ELEMENT Service                       (
    name?, protocol?, port+
  )>

<!ELEMENT UserList                      (
    User+
  )>

<!ELEMENT User                          (
    name | number | (name, number)
  )>
<!ATTLIST User
```

```
    category            %attvals.usercat;       'unknown'
    type                %attvals.usertype;      'current-user'
  >

<!ELEMENT FileList                      (
    File+
  )>

<!ELEMENT File                          (
    name, path
  )>

<!-- ============================================================
 === Secao 6. Declaracao das classes de atributos simples. =========
 ============================================================ -->

<!ELEMENT alertident                (#PCDATA) >

<!ELEMENT CreateTime                (#PCDATA) >
<!ATTLIST CreateTime
    ntpstamp            CDATA                   #REQUIRED
  >

<!ELEMENT description               (#PCDATA) >

<!ELEMENT AdditionalData            ANY >
<!ATTLIST AdditionalData
    type                %attvals.additype;        'string'
    meaning             CDATA                   #IMPLIED
  >

<!ELEMENT Address                  (#PCDATA)>
<!ATTLIST Address
    type                %attvals.addrtype;        'unknown'
  >

<!ELEMENT Block (%Resource;)>
<!ATTLIST Block
    ident               CDATA                   #REQUIRED
    unblock             %attvals.unblock;       'reset'
    time                CDATA                   #IMPLIED
  >

<!ELEMENT Shutdown (%Resource;)>
<!ATTLIST Shutdown
    ident               CDATA                   #REQUIRED
  >

<!ELEMENT command               (#PCDATA) >
<!ELEMENT name                  (#PCDATA) >
<!ELEMENT location              (#PCDATA) >
<!ELEMENT path                  (#PCDATA) >
<!ELEMENT port                  (#PCDATA) >
<!ELEMENT protocol              (#PCDATA) >
<!ELEMENT number                (#PCDATA) >
```