CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO "EURÍPIDES SOARES DA ROCHA"
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

API JAVA PARA MENSAGENS DE IDENTIFICAÇÃO DE ANOMALIAS (IDMEF) EM IDSs

João Victor Passareli Galvão

Marília, 2014

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
FUNDAÇÃO DE ENSINO “EURÍPIDES SOARES DA ROCHA”
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

API JAVA PARA MENSAGENS DE IDENTIFICAÇÃO DE ANOMALIAS (IDMEF) EM IDSs

Trabalho de Conclusão de Curso de Bacharelado em Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", mantenedora do Centro Universitário Eurípedes de Marília - UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador:
Prof. Dr. Fábio Dacêncio Pereira

Marília, 2014

CENTRO UNIVERSITÁRIO EURÍPIDES DE MARÍLIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO – AVALIAÇÃO FINAL

João Víctor Passareli Galvão

API JAVA PARA MENSAGENS DE IDENTIFICAÇÃO DE ANOMALIAS (IDMEF) EM IDSs

Banca examinadora da monografia apresentada ao Curso de Bacharelado em Ciência da Computação do UNIVEM/F.E.E.S.R., para obtenção do Título de Bacharel em Ciência da Computação.

Nota: 9,0 (nove)

Orientador: Fábio Dacêncio Pereira

1°. Examinador: Rodolfo Barros Chiaramonte

2°. Examinador: Fabio Piola Navarro

Marília, 01 de dezembro de 2014.

DEDICATÓRIA

Dedico este trabalho primeiramente a Deus, pois sem ele eu não teria forças para continuar nesta jornada.

E em segundo lugar, mas não menos importante, gostaria de dedicar este trabalho ao meu pai, que infelizmente faleceu no início deste ano, não podendo contemplar minha formatura, pois sempre me disse que ver me formar, seria uma realização pessoal, e um sonho, então pai, onde quer que esteja, quero que saiba que dedico este trabalho ao senhor também.

Dedico também a minha mãe, Marisa Passareli, pois sem seu apoio e dedicação para que eu continuasse os estudos, sem seus conselhos, e sem sua companhia, eu não estaria aqui, após quatro árduos anos de estudo e dedicação, a senhora sempre foi minha maior inspiração.

Aos meus avós, João Passareli e Rosa Ferreira Passareli, por todo apoio, dedicação e conselhos, ambos são para mim grandes exemplos, que seguirei pelo resto de minha vida.

AGRADECIMENTOS

Agradeço primeiramente a Deus por me dar forças e me ajudar durante toda esta caminhada.

Devo agradecer também a todos meus familiares que me ajudaram e apoiaram durante toda duração desta formação, principalmente à minha mãe e meu pai, que sempre me apoiaram quando eu mais precisava e me deram conselhos que seguirei para o resto da vida.

A minha namorada Flávia Castelucci, por me apoiar em todos os momentos de dificuldade, sempre dizendo que eu ia conseguir. Muito obrigado por todo seu apoio e companhia, durante todos estes anos.

Agradeço também a meu grande amigo Rafael Gasperetti, que se tornou quase um irmão ao longos destes quatro anos, e sem sua ajuda e apoio e amizade, talvez eu nem teria conseguido chegar tão longe, muito obrigado mesmo “Curuja”.

Ao meu orientador Fábio Dacêncio, deixo os meus mais sinceros agradecimentos, pois sempre esteve presente quando foi preciso, dando dicas, conselhos e ajuda, se tornando um grande mentor para mim, e um modelo a ser seguido.

Agradeço a todos que me ajudaram e apoiaram durante todo este trajeto, tanto os citados aqui quanto aos que não foram, todos desempenharam um papel importante para esta minha formação, deixo o meu mais sincero obrigado, pois sem vocês, eu não teria chegado tão longe.

"Os que se encantam com a prática sem a ciência são como os timoneiros que entram no navio sem timão nem bússola, nunca tendo certeza do seu destino".

Leonardo da Vinci

GALVÃO, João Victor Passareli. **API Java para mensagens de identificação de anomalias em IDS**. 2014. 117 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

RESUMO

Sistemas computacionais são arranjos compostos por usuários, hardware, software e redes de comunicação, que podem atingir escalas robustas e um alto nível de complexidade. Neste cenário, a segurança de informações em sistemas computacionais tornar-se um desafio para desenvolvedores e administradores.

Deste modo, sistemas de detecção intrusões (IDS – *Intrusion Detection System*), devem ser aliados a outras ferramentas e técnicas de segurança, para que assim possa ter um ambiente mais seguro e confiável. Ainda que alguns problemas possam ser encontrados durante o processo, como por exemplo, falsos positivos, que o IDS pode considerar como uma intrusão ou uma atividade não anômala.

Para notificar uma anomalia encontrada, o IDS envia uma mensagem, informando que possivelmente o sistema computacional está com um comportamento anormal, ou que algo está errado.

Neste contexto, o projeto propõe o estudo e a implementação de um protocolo para a tentativa de padronização do envio de mensagens de identificação de anomalias, e para tal, foi escolhido o modelo de dados IDMEF (*Intrusion Detection Message Exchange Format*), fora aceito pelo IETF (*Internet Engineering Task Force*), que será descrito durante toda esta monografia, analisando suas características e conceitos e apresentando os resultados de sua implementação e aplicação.**Palavras-chave:** IDS, Intrusion Detection System, anomalia, segurança, Mensagem de Informação de Intrusão, IDMEF.

GALVÃO, João Victor Passareli. **API Java para mensagens de identificação de anomalias em IDS**. 2014. 117 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", Marília, 2014.

ABSTRACT

Computational systems are arrangements composed of users, hardware, software and communication networks, which can achieve robust scales and a high level of complexity. In this scenario, the security of information in computer systems became a challenge for developers and administrators.

Therefore, intrusion detection systems (IDS) should be combined with other tools and security techniques, so that we can have a more secure and reliable environment. While some problems may be encountered during the process, such as false positives, the IDS may regarded as an intrusion or non- anomalous activity.

To report a problem found, the IDS sends a message stating that potentially the computational system is abnormal behavior, or that something is wrong.

In this context, the project proposes the study and implementation of a protocol to attempt to standardize the messaging identification of anomalies, and to this end, was chosen the IDMEF (Intrusion Detection Message Exchange Format) data model, was accepted by IETF (Internet Engineering Task Force), which will be described throughout this monograph, analyzing its characteristics and concepts and presenting the results of their implementation and enforcement.

Keywords: IDS, Intrusion Detection System, anomaly, security, Information Message Intrusion, IDMEF.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

A.

API: *Application Programming Interface.*

ASCII: *American Standard Code for Information Interchange.*

C.

CERT: *Comissão Especial de Regimes de Trabalho.*

CGI: Common Gateway Interface.

CPU: *Central Processing Unit.*

D.

DMZ: *Demilitarized Zone.*

DTD: *Document Type Definition.*

H.

HIDS: *Host-Based Intrusion Detection System.*

HTML: *Hyper Text Markup Language.*

I.

IANA: *Internet Assigned Numbers Authority.*

IDMEF: *Intrusion Detection Message Exchange Format.*

IDS: *Intrusion Detection System.*

IDWG: *Intrusion Detection Working Group.*

IEC: *International Electrotechnical Commission.*

IP: *Internet Protocol.*

ISO: *International Organization for Standardization.*

IETF: *Internet Engineering Task Force.*

L.

LAN: *Local Area Network.*

M.

MII: *Mensagem de Informação de Intrusão.*

N.

NIDS: *Network-Based Intrusion Detection System.*

NTP: *Network Time Protocol.*

S.

SGML: *Standard Generalized Markup Language.*

SIEM: *Security Information and Event Management.*

SIM: *Security information management.*

SMI: *Structure of Management Information.*

SNMP: *Simple Management Network Protocol.*

T.

TCP: *Transmission Control Protocol.*

TTY: *Tele Typewriter.*

U.

UML: *Unified Modeling Language.*

URI: *Uniform Resource Identifier.*

URL: *Uniform Resource Locator.*

W.

WWWC: *Word Wide Web Consortium.*

X.

XML: *Extensible Markup Language.*

XSL: *Extensible Stylesheet Language.*

SUMÁRIO

INTRUDUÇÃO

A complexidade das redes de computadores e sistemas computacionais está aumentando consideravelmente, assim como o número de informações armazenadas ou em tráfego. A cada dia, novos serviços são criados, fazendo deste aglomerado uma ferramenta importante para diversas áreas como negócios, entretenimento, saúde, pesquisa, entre outras.

O anonimato, a fragilidade e outros fatores muitas vezes estimulam indivíduos mal intencionados a criar ferramentas e técnicas de ataques a informações e a sistemas computacionais. Isto pode gerar desde pequenas inconveniências até mesmo prejuízos financeiros e morais. (PEREIRA, Fábio Dacêncio, 2009)

A detecção de intrusão aliada a outras ferramentas de segurança pode proteger e evitar ataques maliciosos e anomalias em sistemas computacionais. Porém, considerada a complexidade e robustez de tais sistemas, os serviços de segurança muitas vezes não são capazes de analisar e auditar todo fluxo de informações, gerando pontos falhos de segurança que podem ser descobertos e explorados. (ANDROULIDAKIS, Papa Vassiliou, 2008)

Segundo Hermano Pereira (2011), os Sistemas de Detecção de Intrusão (IDS – *Intrusion Detection Systems*) são sistemas que atuam junto ao sistema operacional ou a uma rede de computadores para detectar atividades maliciosas. Para identificar um ataque em potencial, os IDSs são implementados com métodos que geralmente são baseados em mau uso (*misuse-based*) ou baseados em anomalia (*anomaly-based*).

Criar um novo IDS não é uma tarefa trivial, pois devem ser levados em conta vários aspectos, e também o tipo de intrusão a ser analisada, já que não existe apenas um único tipo, tornando assim a criação mais complexa. Quando o IDS encontra alguma anomalia, ela emite ao sistema uma mensagem contendo algumas informações sobre a anomalia encontrada, estas mensagens são chamadas de MIIs (Mensagens de Informação de intrusão), elas facilitam o serviço do IDS, pois podem conter informações como: o provável tipo de anomalia que fora encontrada, seus dados e características, hora do ocorrido e também alguns outros itens que serão analisados para uma possível solução do problema.

PROBLEMÁTICA E JUSTIFICATIVA

O problema das MIIs é a não existência de um padrão, sendo assim, cada IDS envia uma mensagem diferente, em um formato diferente e com dados diferentes, tornando a mensagem utilizada em um IDS provavelmente inoperante em outro. Mesmo que um sistema de segurança usando IDS e MIIs implementado em um local A esteja funcionando perfeitamente, este mesmo sistema pode não funcionar da mesma maneira em um local B, pois poderá haver algumas incompatibilidades na troca de informações.

Visando este problema e com o enorme crescimento das informações a serem transmitidas pelas redes de computadores, também pode crescer o número de vulnerabilidades dos dados trafegados. Neste contexto, a justificativa do trabalho proposto é contribuir com IDSs na padronização das MIIs, com a criação de uma API (*Application Programming Interface*), utilizando o modelo de dados IDMEF, pois já sabemos que estes são essenciais, e como nenhum método de detecção soluciona todos os problemas, este esforço para ajudar na melhoria da segurança entre as redes de computadores se faz de grande utilidade. Pois caso seja possível a padronização da MII, será possível criar uma interoperabilidade entre IDSs distintos utilizando o IDMEF, garantindo que a confiabilidade do sistema continue.

OBJETIVOS

O projeto tem como objetivo o estudo do tema abordado, a análise do modo de funcionamento de um IDS, a pesquisa e estudo sobre o modelo de MII IDMEF que fora selecionado, e sua implementação na plataforma Java. Para isso, pode-se definir alguns objetivos específicos que serão apresentados na sequência:

- Estudo aprofundado do tema a ser abordado.
- Estudo de pesquisas e artigos correlatos.
- Pesquisa de propostas de padrões de MII.

- Escolha de uma proposta para a implementação e testes, utilizando os modelos de ataque *Ping of Death* e *Teardrop Attack*.
- Como o modelo IDMEF fora o escolhido, deve-se descrever e estudar o modelo, como seus modos de funcionamento, suas características entre outros aspectos
- Estudo da plataforma Java para desenvolvimento do projeto.
- Criar um ambiente artificial para simular o IDMEF e mostrar relatórios de anomalias.

METODOLOGIA

A metodologia para pesquisa e desenvolvimento deste projeto foi dividida em quatro etapas principais. Estas que serão descritas a seguir:

1. Pesquisa

1.1. Estudo aprofundado sobre redes de computadores, intrusões e formas de detecção.

1.2. Estudo refinado sobre tipos de intrusões e/ou anomalias.

1.3. Pesquisa de trabalhos correlatos.

1.4. Pesquisa e escolha de uma proposta de padronização de MII.

1.5. Modelagem de classes em Java para o padrão IDMEF (escolhido após tópico 1.4).

2. Desenvolvimento e implementação do projeto.

2.1. Desenvolver e implementar o modelo IDMEF escolhido na etapa anterior, na plataforma Java.

2.2. Análise e testes de desempenho, confiabilidade e eficiência do software desenvolvido por meio de geração de relatórios a partir dos ataques citados nos objetivos.

3. Criar um ambiente artificial para simular o IDMEF, mostrando os relatórios gerados pela API a partir de dados fornecidos pelo documento do IDMEF proposto ao RFC.

4. Escrita da monografia abordando resultados obtidos.

ORGANIZAÇÃO DA MONOGRAFIA

Durante o primeiro capítulo deste documento, são abordados os conceitos e aplicações de um sistema de detecção de intrusão, as IDS mais populares e seus modos de funcionamento, como também suas vantagens e desvantagens. Também é explicado os tipos, classificações, e principais problemas encontrados ao utilizar deste sistema de segurança.

Já no segundo capítulo desta monografia são abordados todos os conceitos do modelo de dados IDMEF. Apresentando sua estrutura, funcionamento, implementação entre outros aspectos.

Durante a terceira etapa do projeto, foram abordados alguns trabalhos correlatos utilizados durante a implementação de todo o projeto, onde serão descritas as principais características de cada um e também sua ligação com este trabalho.

O quarto capítulo descreve e exemplifica o programa implementado, contendo suas especificações, funcionalidades e estrutura.

No quinto e último capítulo, é apresentado as conclusões finais sobre o projeto, como também seus resultados obtidos, resultados alcançados e projetos para o futuro.

1. Conceitos e aplicações de IDS

Ao longo deste capítulo, serão apresentados os conceitos e aplicações de um sistema de detecção de intrusão, abordando suas funções, modos de funcionamentos, classificações e outros aspectos necessários para o entendimento de seu funcionamento e da relação deste com o projeto em si.

A preocupação com segurança em redes e sistemas de computadores ocasionou o surgimento de várias técnicas voltadas à prevenção de ataques, porém, prevenir todas as possíveis quebras de segurança em um ambiente computacional é uma tarefa impossível de ser realizada, pois novas vulnerabilidades são descobertas constantemente. No entanto, é possível identificar tentativas ou violações ocorridas e então gerar respostas que possam minimizar os danos sofridos. Nesse cenário é que se destaca a detecção de intrusão, que é o processo de identificar, relatar e possivelmente reagir a qualquer atividade maliciosa agindo em computadores e recursos da rede.

Essa detecção é realizada através de conjuntos de software e hardware que cooperam de forma a executar análises sobre todas as atividades realizadas no sistema monitorado, identificando os eventos considerados como sendo ataques e posteriormente relatando que uma atividade maliciosa aconteceu, está acontecendo ou irá acontecer.

Na maioria das abordagens de detecção de intrusão utilizadas atualmente a identificação dos eventos maliciosos é realizada através da comparação das atividades correntes com ações esperadas de um intruso.

Os sistemas de detecção de intrusão possuem componentes que desempenham funções específicas como sensores, analisadores de eventos e unidades de respostas, que juntos permitem a capacidade de detectar, analisar e responder a cada evento de acordo com a gravidade do evento ocorrido. (LIMA, Igor Vinícius Mussoi. 2005).

1.1. Breve histórico sobre IDS

Originalmente, administradores de sistemas realizavam as detecções de intrusões sentando em frente a um console e monitoravam as atividades dos usuários. Eles conseguiam detectar intrusões notando, por exemplo, que um usuário de férias estava *logado* localmente, ou que uma impressora que raramente está ativa é utilizada de forma incomum. Embora fosse efetiva, esta forma primitiva era provisória e não escalável.

O próximo passo na detecção de intrusão envolvia a auditoria de logs, que os administradores do sistema reviam em busca de evidências de comportamentos anormais ou maliciosos. No fim dos anos 70 e início dos anos 80, os administradores geralmente imprimiam os logs de auditoria em papel formulário contínuo, que muitas vezes formavam pilhas de quatro a cinco metros de altura até o final de uma semana comum. Realizando uma busca em uma pilha deste tamanho era, obviamente, muito demorado.Com este excesso de informação e sua análise sendo feita manualmente, os administradores usavam os logs de auditoria principalmente logs como uma ferramenta forense para determinar a causa de um incidente de segurança particular, após o ataque já ter ocorrido. Havia pouca esperança de detectar um ataque em andamento.

Conforme as formas de armazenamento se tornaram mais baratas, os logs de auditoria começaram a ser armazenados online, assim os pesquisadores desenvolveram programas para analisar estes dados. (SALTZER, J.; SCHROEDER, M. 1975)

Entretanto, a análise era lenta e quase sempre computacionalmente intensa, e assim, os programas de detecção eram usualmente executados durante a noite, quando o uso dos usuários do sistema era menor. Entretanto, a maioria das intrusões era detectada após o ocorrido.

No começo da década de 90, pesquisadores desenvolveram um sistema de detecção de intrusão de tempo real, que analisada os logs de audição enquanto eram produzidos. Desta maneira, foi possível a detecção de ataques e tentativas no momento em que ocorreram, sendo assim, permitiu que houvesse uma resposta em tempo real, e em alguns casos, evitar o ataque.

Os esforços mais recentes para a detecção de intrusão têm-se centrado no desenvolvimento de um produto em que os usuários possam efetivamente implantar em grandes redes. Porém, não é uma tarefa fácil, dado o aumento na preocupação com a segurança, as inúmeras novas técnicas de ataque, e as mudanças contínuas no ambiente computacional. (KEMMERER, Richard A.; VIGNA, Giovanni. 2002)

1.2. Tipos e classificações de IDS

Nos tópicos a seguir, será explicado mais detalhadamente as formas de classificação e tipos de IDSs, como também técnicas e métodos utilizados para a procura de comportamentos suspeitos ou anômalos no sistema, que influenciam na classificação destas ferramentas de segurança.

1.2.1. Classificação

As técnicas utilizadas para procurar evidências de comportamento suspeito nos registros de utilização dos sistemas podem variar de acordo com a maneira que são implantadas, frequência de operação, pelo tipo dos dados que analisam ou pelas análises que utilizam sobre esses dados. Estes métodos possuem determinadas aplicações, limitações e peculiaridades e podem ser classificadas de várias formas conforme ilustrado na Figura 1.

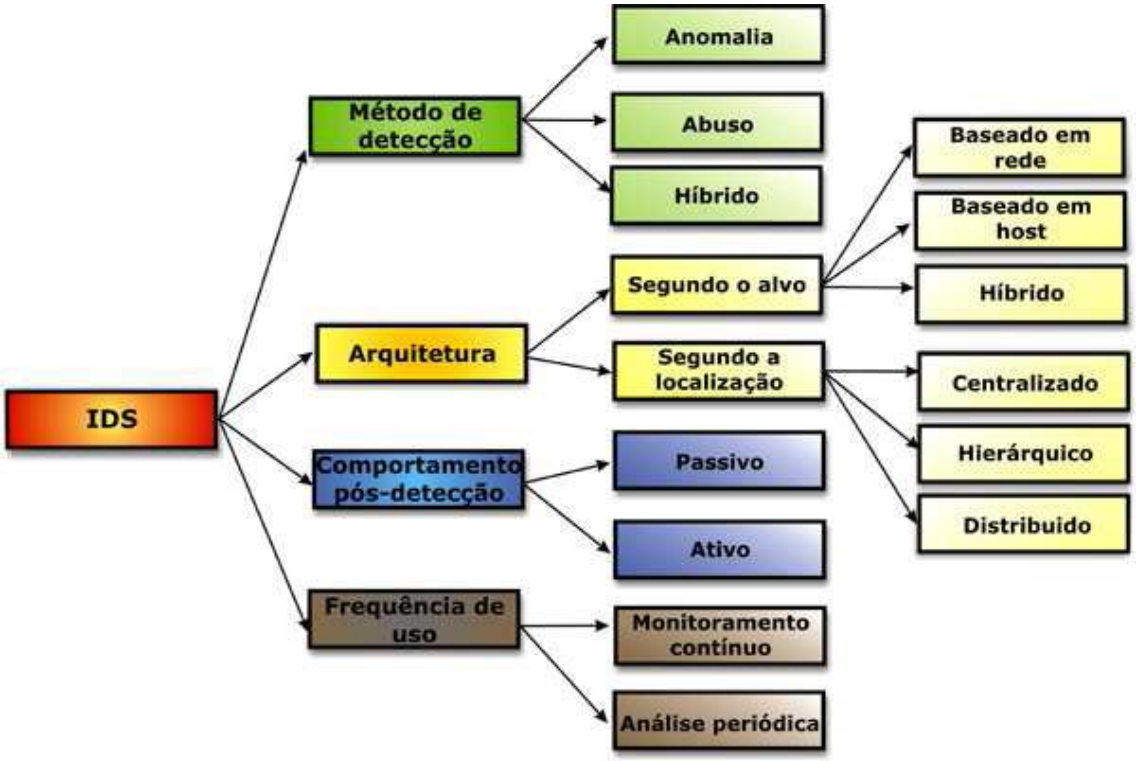

Figura 1: Classificação de IDS. Fonte: (CAMPELLO, Weber, 2001)

Esta classificação permite identificar critérios ou características que são consideradas ao se projetar um sistema de detecção de intrusão. Em função disso, diferentes ferramentas aplicam distintos mecanismos para obtenção, análise e tratamento dos dados, até mesmo utilizando arquiteturas híbridas para otimizar o modelo final. (LIMA, Igor Vinícius Mussoi, 2005)

1.2.2. Tipos de IDS

Várias abordagens e técnicas podem ser utilizadas e combinadas para prover a detecção dos eventos maliciosos ocorridos em determinado ambiente. A forma como estas técnicas são aplicadas na construção de sistemas de detecção de intrusão impactam diretamente sobre sua eficiência e desempenho.

As diferenças vão desde a lógica adotada para analisar os eventos até a abrangência do sistema dentro da infraestrutura em que está inserido.

Podemos classificar os sistemas de detecção de intrusão em duas categorias principais, sendo IDS baseado em host e IDS baseado em rede. Frequentemente os sistemas podem ser híbridos, incorporando características das duas abordagens (CROSBIE; SPAFFORD, 1995).

1.2.2.1. IDS baseado em rede (NIDS).

Esta categoria de IDS denominada NIDS (*Network Intrusion Detection System*), possui um mecanismo de detecção que analisa o tráfego da rede, e que implementa a detecção utilizando dois componentes principais, sendo eles os sensores e estação de gerenciamento.

Os sensores são componentes que são alocados estrategicamente em determinados segmentos de rede, passando a monitorar não somente a máquina onde estão instalados, mas todo o tráfego no seguimento em questão, pois a interface de rede é configurada para trabalhar em modo promíscuo, capturando não apenas os pacotes de dados destinados à interface de rede da estação onde está instalado, mas toda e qualquer informação que trafegue pela rede.

Os dados capturados podem ser previamente selecionados, para que posteriormente sejam analisados de acordo com o mecanismo de detecção adotado, que irá categorizar os eventos e determinar se o mesmo é ou não intrusivo, reportando os eventos de interesse para a estação de gerenciamento, que gerencia os eventos e realiza o tratamento adequado para cada situação como, por exemplo, enviar um alerta ou um *e-mail* para os responsáveis, para que sejam tomadas as devidas providências.

O posicionamento dos sensores na rede pode impactar diretamente sobre a ocorrência dos eventos e os resultados esperados. Em uma situação onde os sensores estejam posicionados antes de um *firewall*, estes estariam expostos a todas as atividades maliciosas da rede. Por outro lado, caso os sensores estejam posicionados na parte interna da rede, o número e a severidade

dos eventos a serem analisados é consideravelmente menor, uma vez que o firewall realizou determinada filtragem bloqueando grande parte das tentativas de ataque (MIIKA, 2000).

Este tipo de implementação permite a descoberta de falhas na configuração dos mecanismos de segurança, e baseado em eventos reportados pelo sistema poderiam ser realizados ajustes nas regras de filtragem do *firewall*.

Existem algumas razões relacionadas ao ambiente e a infraestrutura de rede que inviabilizam a utilização de um IDS baseado em rede. Em redes de alta velocidade, por exemplo, o custo computacional exigido para realizar a captura e a análise do tráfego seria muito alto, podendo acarretar em impactos sobre o tempo de resposta do sistema.

A utilização de *switches* como equipamentos de interconexão em redes de computadores limita a abrangência do sensor, devido à ausência de *broadcast*, pois o tráfego é comutado entre os elementos da rede. Portanto, mesmo em modo promíscuo o sensor não conseguiria capturar todos os pacotes de dados do segmento.

Existe também uma dificuldade relacionada à utilização de criptografia e aplicações de rede, pois em seções criptografadas o IDS estaria incapacitado de realizar análises, já que não seria possível reconhecer assinaturas ou padrões de ataque baseado em dados cifrados. (LIMA, Igor Vinícius Mussoi. 2005)

1.2.2.2. IDS baseado em Host (HIDS).

Os sistemas de detecção de intrusão baseados em host possuem mecanismos e procedimentos de análise que verificam sinais de intrusão exclusivamente nas máquinas onde estão instalados, geralmente tomando como base para esta análise recursos locais, como registro de *logs* do sistema operacional e os registros de aplicações existentes, além de indícios de atividades não usuais, como tentativas de *login* sem sucesso, tentativas de acesso a arquivos, alterações de privilégios do sistema, entre outros.

Esta categoria de IDS possui algumas dificuldades relacionadas à captura e análise frequente em um sistema específico, que pode acarretar em problemas de degradação do desempenho do *host*, podendo ainda ser atacado e ter seu mecanismo de controle de *logs* comprometido, afetando a confiabilidade do mecanismo de detecção (MIIKA, 2000).

Além da análise dos registros de atividades do sistema, podem ser realizadas verificações de integridade de arquivos, que checam se determinados arquivos do sistema foram alterados desde a última verificação. Este processo é realizado gerando um resumo de cada

arquivo através de funções de *hash*. Estes resumos são armazenados e servirão como base em comparações futuras com novos resumos, determinando se houve ou não alteração nos arquivos em questão.

Independente do mecanismo utilizado, alguns tipos de ataques podem passar despercebidos pela maioria das ferramentas disponíveis, principalmente quando são utilizadas de maneira independente. Em função disso, é interessante possuir diversidade de mecanismos de detecção integrados no IDS, de forma a aproveitar as melhores características de cada técnica e tipo de análise. (LIMA, Igor Vinícius Mussoi. 2005)

1.2.2.3. Sistemas Híbridos.

Um sistema IDS híbrido tem como objetivo combinar as vantagens do HIDS e do NIDS, a fim de proporcionar uma melhor capacidade de detecção de intrusões. O IDS híbrido funciona como o NIDS, coletando o tráfego de pacotes de rede, processando as informações, detectando e respondendo a ataques do mesmo modo como ocorre no HIDS.

Com relação ao gerenciamento, alguns sistemas podem ter uma centralização dos IDS, pois alguns sensores, baseados em rede, são localizados em diversos segmentos de rede e outros IDS, baseados em *host*, são usados em servidores. O gerenciador pode controlar as regras dos dois tipos, formando o IDS híbrido. (PINHEIRO, José Maurício dos Santos. 2007).

Segundo Strebe e Perkins (2002), sistemas IDS sempre requerem recursos da rede para funcionarem corretamente. Sistemas NIDS usualmente funcionam em *firewalls* ou computadores dedicados; isso normalmente não é problema porque estes recursos estão disponíveis. Porém, sistemas HIDS destinados a proteger servidores podem ter sérias restrições para funcionamento.

No caso dos servidores da zona neutra, a DMZ (*Demilitarized Zone*), o uso de IDS híbrido é vantajoso uma vez que ataques específicos a cada servidor podem ser identificados com maior precisão. (PINHEIRO, José Maurício dos Santos. 2007).

1.2.3. Métodos de Detecção.

De acordo com Igor Vinícius M. de Lima (2005), o método de detecção empregado é um componente importante na construção de sistemas de detecção de intrusão, pois define a implementação do principal processo em soluções do gênero. Esses métodos definem formas

de detecção por anomalia aplicando métodos estatísticos sobre a utilização do sistema, detecção por abuso desenvolvendo o monitoramento baseado em padrões de ataques conhecidos, e detecção híbrida aplicando ambas as abordagens.

1.2.3.1. Detecção por Anomalia.

Esta técnica considera que todo comportamento intrusivo é necessariamente anômalo, o sistema reage a todo comportamento que não se enquadre nos comportamentos ditos como normais.

Em um tipo de análise denominada estatística estes comportamentos normais são previamente relacionados, baseado na observação do conjunto de atividades que caracterizam operações normais de um sistema, gerando estatísticas de uso dos recursos (CPU, unidades de armazenamento, memória, e outros periféricos) e das atividades dos usuários (tentativas de *login*, aplicativos utilizados, e outras ações). (LIMA, Igor Vinícius Mussoi, 2005)

Estas estatísticas podem ser constantemente atualizadas refletindo o estado atual do sistema (CROSBIE; SPAFFORD, 1995), o desvio de comportamento pode ser observado comparando o padrão de comportamento atual do sistema com as estatísticas geradas. Caso existam divergências abruptas nos parâmetros de comparação, é considerado que este é um comportamento anômalo e que, portanto pode vir a ser intrusivo.

Sendo assim, um atacante sabendo que está sob monitoramento de um sistema com estas características pode subverter o mecanismo de detecção alterando gradativamente seu comportamento, de forma que o sistema os considere normais, e em dado momento efetivar o ataque sem que o sistema perceba um comportamento anômalo, ou ainda executando um conjunto de operações que individualmente não representam nenhuma ameaça, porém se caracterizando um ataque quando consideradas em conjunto.

A detecção por anomalia leva em conta que qualquer comportamento anômalo é considerado como intrusivo, no entanto algumas atividades anômalas podem não ser intrusivas. Esta situação gera quatro estados de detecção (KUMAR, 1995):

a) Intrusivo e anômalo: a atividade é intrusiva e é apontada como tal por ser também anômala; são conhecidos como os verdadeiros positivos;

b) Não intrusivo e não anômalo: a atividade não é anômala e não é apontada como intrusiva; são denominados como verdadeiros negativos;

c) Intrusivo mas não anômalo: a atividade é intrusiva mas, como não é anômala não é reportada como tal, gerando uma falha em sua detecção; são consideradas como falso negativos;

d) Não intrusivo mas anômalo: atividade não é intrusiva, porém como é anômala, o sistema entende que se trata de uma atividade intrusiva, reportando de forma incorreta tal fato; estes são denominados falso positivos.

Altos índices de detecções sendo falso-positivos podem comprometer a eficiência do sistema de detecção de intrusão, pois uma grande quantidade de alertas pode ser gerada reportando como intrusivas as atividades normais dos usuários. Para contornar este problema é preciso redefinir os parâmetros que apontam tal comportamento como anômalo, porém ajustando-os de forma que não ocorram detecções falsas negativas.

Segundo (AXELSSON, 2000) os detectores por anomalia tendem a ser mais caros computacionalmente, pois muitos parâmetros e estatísticas precisam ser ajustados com frequência, dependendo do tipo de atividade do sistema.

1.2.3.2. Detecção por Abuso.

A abordagem de detecção de intrusão por abuso baseia-se na observação de eventos que se assemelham a comportamentos intrusivos já conhecidos, comumente chamados de assinaturas de intrusão. Estas assinaturas são relacionadas e todos os eventos do sistema são comparados com as mesmas, a fim de identificar um padrão de comportamento que se encaixe nas especificações da assinatura.

Podem existir várias formas de se executar um mesmo ataque, sendo necessário que os aspectos, condições, posicionamento e inter-relações entre os eventos que levam a uma intrusão sejam transcritos para sua assinatura de intrusão, a fim de detectar um mesmo ataque iniciado a partir de padrões diferentes.

Alguns ataques são desenvolvidos através de evoluções realizadas a partir de ataques já conhecidos. Sendo assim o uso de assinaturas na detecção por abuso colabora na localização de tentativas de quebra de segurança, de forma que a confirmação parcial da ocorrência de uma assinatura pode indicar uma tentativa de intrusão.

A principal limitação deste tipo de sistema é que a detecção dos eventos baseia-se em ataques já conhecidos, sendo, portanto ineficiente na detecção de comportamentos intrusivos que ainda não foram descobertos e/ou publicamente divulgados. A eficiência deste método

depende da frequente atualização das assinaturas de intrusão do sistema. (LIMA, Igor Vinícius Mussoi, 2005)

1.2.3.3. Detecção Híbrida.

Abordagens de detecção por anomalia ou por abuso podem ser adequadas para casos distintos de formas de ataques, por isso algumas abordagens tratam sobre um método de detecção híbrido que incorpora os dois métodos, compondo uma solução de detecção mais eficiente, levando-se em conta a grande variedade de ataques existentes (BERNARDES, 1999).

1.3. Problemas de segurança com IDS

Quando se fala de sistemas de detecção de intrusão, existem duas situações indesejáveis que frequentemente nos deparamos. Uma é o que chamamos de falsos positivos (ou falsos alertas) e a outra são os falsos negativos, ou seja, os ataques reais, porém não detectados.

A primeira situação representa inconsistência na detecção de eventos. Ocorre geralmente quando um ataque é efetuado na rede e o IDS o detecta, gerando alertas deste e de outros supostos ataques que possuem características semelhantes, mas que na realidade não aconteceram. Falsos positivos podem ser explorados de forma maliciosa.

A segunda situação implica num comprometimento mais imediato caso o ataque seja bem sucedido, tendo em vista que se o administrador não sabe que um ataque está acontecendo na rede, obviamente uma atitude imediata não será tomada por ele nem por algum mecanismo de defesa (FAGUNDES, LEONARDO L., 2002).

1.4. IDS populares

Nos tópicos seguintes serão listadas algumas das IDSs mais populares encontradas, contendo uma breve descrição de cada uma delas.

1.4.1. SNORT

O SNORT é uma ferramenta "*open-source*" NIDS desenvolvido por Martin Roesch bastante popular por sua flexibilidade nas configurações de regras e constante atualização frente às novas ferramentas de invasão. Outro ponto forte desta ferramenta é o fato de ter o maior cadastro de assinaturas, ser leve, pequeno, fazer escaneamento do computador e verificar anomalias dentro de toda a rede ao qual este pertence.

O código fonte é otimizado, desenvolvido em módulos utilizando linguagem de programação C e, junto, com a documentação, são de domínio público.

O Snort conta ainda, com o permanente desenvolvimento e atualização, que são feitos diariamente, tanto em relação ao código propriamente dito, como das regras de detecção. Os padrões utilizados na construção das regras de detecção das subversões são introduzidos no sistema de configuração, tão rápido quando são enviados os alertas originados pelos órgãos responsáveis, como por exemplo o CERT, *Bugtraq* (lista de discussão), entre outros.

Por ser uma ferramenta leve, a utilização do Snort é indicada para monitorar redes TCP/IP pequenas, onde pode detectar uma grande variedade do tráfego suspeito, assim como ataques externos e então fornecer argumentos para as decisões dos administradores.

Os módulos que compõe o Snort são ferramentas poderosas, capazes de produzir uma grande quantidade de informação sobre os ataques monitorados, dado que é possível avaliar tanto o cabeçalho quanto o conteúdo dos pacotes, além de disponibilizar, por exemplo, a opção de capturar uma sessão inteira.

O Snort monitora o tráfego de pacotes em redes IP, realizando análises em tempo real sobre diversos protocolos (nível de rede e aplicação) e sobre o conteúdo (hexa e ASCII). Outro ponto positivo desse software é o grande número de possibilidades de tratamento dos alertas gerados. O subsistema de registro e alerta é selecionado em tempo de execução através de argumentos na linha de comando, são três opções de registro e cinco de alerta. O registro pode ser configurado para armazenar pacotes decodificados e legíveis em uma estrutura de diretório baseada em IP, ou no formato binário do *tcpdump* (*sniffer* para sistemas GNU/Linux) em um único arquivo. Para um incremento de desempenho, o registro pode ser desligado completamente, permanecendo os alertas. Já os alertas podem, ser enviados ao *syslog* (logs do sistema), registrados num arquivo de texto puro em dois formatos diferentes, ou ser enviados como mensagens WinPopup usando o smbclient.

Os alertas podem ser enviados para arquivo texto de forma completa e o alerta rápido. O alerta completo escreve a mensagem de alerta associada à regra e a informação do cabeçalho do pacote até o protocolo de camada de transporte. A opção de alerta rápido escreve um subconjunto condensado de informação do cabeçalho alerta.

Por fim, uma última opção desabilita os alertas completamente. Existe também, a possibilidade de utilizar métodos como o *Database Plug-in*, por exemplo, para registrar pacotes em uma variedade de bases de dados diferentes (*MySQL*, *PostgreSQL*, entre outros), as quais contam com recursos próprios para efetuar consultas, correlações e dispõem de mecanismos de visualização para analisar dados. Abaixo serão apresentados as vantagens e desvantagens do SNORT. (SNORT Brasil, 2014)

1.4.1.1. Vantagens:

Extremamente Flexível:

- Algoritmos sem Inspeção baseado em Regras.
- Sem falsos positivos inerentes.
- Controle total do refinamento das regras.

Metodologias de detecção Multi-dimensional:

- Assinaturas (Impressão Digital) do ataque.
- Anomalias no protocolo.
- Anomalias no comportamento.

Imensa Adoção (Comunidade SNORT):

- Dezenas de Milhares de Instalações (42 mil).
- Algumas das maiores empresas do mundo utilizam. (Microsoft, Intel, PWC, entre outos)
- Milhares de contribuidores fazendo regras para novas vulnerabilidades.

Infraestrutura de Suporte da Comunidade Open Source:

- Rápida resposta a ameaças.
- Velocidade de inovação.
- Velocidade de refinamento.

1.4.1.2. Desvantagens

Interface gráfica limitada:

- Configuração do sensor.
- Gerenciamento de regras.

Implementação lenta e cansativa, duração de pelo menos 10 dias.

Capacidade analítica limitada.

Sem suporte comercial:

- Dependência de pessoas capacitadas, porém, nem sempre estáveis.
- Gastos significativos com recursos humanos.

1.4.2. OSSEC

O OSSEC é um HIDS (*Host-Based Intrusion Detection System*) open source que permite a análise de logs, checagem da integridade dos dados, monitoramento de políticas, detecção de *rootkit*, e também alerta e resposta em tempo real. Esta plataforma mixa todos os aspectos de um HIDS, permitindo o monitoramento de log e SIM/SIEM juntos, em uma solução open source e completa. (OSSEC, 2014)

Este sistema é composto por múltiplos componentes, que serão brevemente explicados na sequência:

- Gerente: É o componente central desta HIDS. O gerente armazena todos os dados necessários, como *logs*, regras, decodificadores em configurações principais, assim facilitando o serviço do administrador mesmo com um grande número de agentes.
- Agentes: É um pequeno programa instalado no sistema a ser monitorado. Os agentes coletam informações em tempo real e enviam ao Gerente para análise e correlações. Por ser um sistema leve, este não afeta a usabilidade do sistema.
- Sem-Agentes: Em sistemas em que não é possível a instalação dos agentes, o OSSEC permite o monitoramento da integridade dos dados sem a necessidade do Agente. Esta ferramenta pode ser muito útil para monitorar o firewall, roteadores e até sistemas baseados em UNIX, onde não são permitidos instalar o Agente.

- Virtualização: Permite a instalação do Agente em um sistema operacional "visitante" ou dentro do host (VMware ESX). Com o agente instalado dentro da VMware ESX, é possível monitorar alertas sobre quando uma VM (*Virtual Machine*) é instalada, removida, iniciada, etc. Também monitora *logins*, *logouts* e erros dentro do servidor da ESX, possui ainda a função de CIS *check* para a VMware, verificando se existe alguma configuração insegura ativa ou outros problemas.

1.4.3. TRIPWIRE

Tripwire é uma HIDS (*Host-Based Intrusion Detection System*) para computadores Linux. Esta ferramenta monitora o sistema com a finalidade de detectar e reportar qualquer atividade não autorizada que poderá fazer alguma alteração nos diretórios ou arquivos do sistema. Assim que uma linha base é criada, ela detecta qualquer mudança feita nos arquivos, como a criação, exclusão ou alteração, informando o usuário responsável pela mudança e a data do ocorrido. Caso alguma alteração anormal deve ser possibilitada aos usuários, pode-se ativar esta função no Tripwire para que não ocorra mais erros ao efetuá-la.

1.5. Considerações finais do capitulo

Neste primeiro capítulo foi abordado desde a história do surgimento do IDS até sua atual importância no cenário computacional, também foi descrito seu modo de funcionamento e suas principais características, uma breve apresentação das versões mais populares com os aspectos que as diferem, os tipos de IDS, suas classificações e modos de operação. Também foram abordados os possíveis estados de detecção que uma anomalia pode apresentar para o IDS e os problemas que esta pode encontrar.

Para que este trabalho de conclusão possa ser bem elaborado, faz-se necessário o estudo aprofundado deste mecanismo de segurança e suas peculiaridades, pois, para ser possível a implementação de um padrão proposto para as mensagens de informação de intrusão, que são o tema desta proposta, é preciso conhecer as mecânicas e elementos presentes em um IDS para que não haja erros por falta de conceitos ou de conhecimento sobre o assunto.

2. Padrão IDMEF

O objetivo do Formato de Detecção de Intrusão por Troca de Mensagens (IDMEF – *Intrusion Detection Message Exchange Format*) é definir formatos de dados e procedimentos de intercâmbio para compartilhar informações de interesse para os sistemas de resposta de detecção de intrusão e os sistemas de gestão que possam precisar interagir com eles. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

Durante todo este capítulo, será apresentado o modelo de dados IDMEF, contendo suas características, suas classes principais e toda a documentação necessária para seu entendimento.

2.1. Introdução ao IDMEF

O modelo de dados IDMEF é uma representação orientada a objetos dos dados de alertas enviados para os gerentes de detecção de intrusão por analisadores.

O IDMEF visa ser um padrão de formato de dados no qual IDSs automatizados possam utilizar para reportar alertas considerados suspeitos. O desenvolvimento deste formato padrão irá permitir a interoperabilidade entre sistemas comerciais, *Open Source* e Sistemas Investigativos, permitindo aos usuários misturar e combinar a implantação destes sistemas de acordo com seus pontos fortes e fracos em uma implementação ideal.

O ambiente mais óbvio para a implementação deste padrão seria no canal de dados entre o Sensor (IDS) e o *Console* (Gerente) para que ele envie os alarmes. Porém, há ainda outros lugares onde o IDMEF pode ser útil:

- Um sistema único de Banco de Dados que pode armazenar os resultados de uma variedade de produtos de detecções de intrusões tornaria possível realizar a análise de dados e relatório de atividades sobre todo o cenário, ao invés de uma pequena parte.
- Um sistema de correlação de eventos que pode aceitar alertas a partir de uma variedade de produtos de detecção de intrusão seria capaz de realizar correlações cruzadas e cálculos de confirmação cruzados mais sofisticados do que aquele que é limitada a um único produto.
- Uma interface gráfica de usuário que pode exibir alertas a partir de uma variedade de produtos de detecção de intrusão, que permitiria ao usuário monitorar todos os produtos a partir de uma única tela, e exigir que ele ou ela aprenda apenas uma interface, em vez de várias.

- Um formato comum de troca de dados tornaria mais simples para as diferentes organizações (usuários, fornecedores, equipes de resposta), não só para a troca de dados, mas também comunicar sobre isso.

A diversidade de usos para o IDMEF precisa ser considerada quando for escolher um método de implementação. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.1.1. Problemas tratados pelo modelo de dados.

O IDMEF aborda vários problemas associados com a representação de alerta de dados de detecção de intrusão:

- Informações de alerta são inerentemente heterogêneos. Alguns alertas são definidos com pouca informação, tais como origem, destino, nome e hora do evento. Outros alertas podem fornecer muito mais informações, como portas ou serviços, processos, informações do usuário, e assim por diante. O modelo de dados que representa esta informação deve ser flexível para acomodar diferentes necessidades.
- Um modelo orientado a objeto é naturalmente extensível via agregação e subclasse. Se uma aplicação do modelo de dados estende com novas classes, seja por agregação ou subclasses, uma implementação que não entende essas extensões ainda será capaz de entender o subconjunto de informações que é definida pelo modelo de dados. Subclasses e agregação fornecem extensibilidade, preservando a consistência do modelo.
- Os ambientes de detecção de intrusão são distintos. Alguns analisadores detectam ataques analisando o tráfego de rede, enquanto outros usam os *logs* do sistema operacional ou informações de auditoria de aplicação. Alertas para o mesmo ataque, enviados por analisadores com diferentes fontes de informação, não irão conter as mesmas informações.
- O modelo de dados define classes de suporte, que podem acomodar as diferenças nas fontes de dados entre os analisadores. Em particular, as noções de origem e de destino para o alerta são representadas pela combinação de nós, processo, serviço e classes do usuário.
- Analisadores de capacidade são diferentes. Dependendo do ambiente, pode-se instalar um analisador leve que fornece pouca informação em seus alertas, ou um analisador mais complexo, que terá um impacto maior sobre o sistema em funcionamento, mas oferece informações mais detalhadas. O modelo de dados deve permitir a conversão de formatos

utilizados por outras ferramentas além dos analisadores de detecção de intrusão, com o objetivo de processamento da informação de alerta.

O modelo de dados define as extensões para a Definição do Tipo de Documento básico (DTD - *Document Type Definition*) que permitem transportar os alertas simples e complexos. Extensões são realizadas através de subclasses ou associação de novas classes.

- Ambientes operacionais são diferentes. Dependendo do tipo de rede ou sistema operacional utilizado, ataques serão observados e relatados com características diferentes. O modelo de dados deve acomodar estas diferenças.

Flexibilidade significativa na comunicação é fornecida pelas classes Nós e Suporte de Serviços. Se informações adicionais devem ser comunicadas, subclasses podem ser definidas para estender o modelo de dados com atributos adicionais.

- Objetivos de vendedores comerciais são diferentes. Por várias razões, os fornecedores podem querer entregar mais ou menos informações sobre determinados tipos de ataques.

A abordagem orientada a objetos permite essa flexibilidade, enquanto as regras de subclassificação preservam a integridade do modelo. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.1.2. Design do modelo de dados.

O modelo de dados IDMEF foi projetado para fornecer uma representação padrão de alertas de uma forma inequívoca, e para permitir o relacionamento e descrição entre os alertas simples e complexos. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.1.3. Representação de Eventos.

O objetivo do modelo de dados IDMEF é fornecer uma representação padrão das informações que um analisador de detecção de intrusão irá reportar quando detectar a ocorrência de qualquer evento incomum. Tais alertas podem ser simples ou complexos, dependendo da capacidade do analisador que os cria. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.1.4. Conteúdo Dirigido.

O design do modelo de dados é de conteúdo dirigido. Isso significa que novos objetos são introduzidos para acomodar o conteúdo adicional, e não há diferenças semânticas entre os alertas. Este é um objetivo importante, como a tarefa de classificar e nomear as vulnerabilidades do computador é ao mesmo tempo extremamente difícil e muito subjetivo.

O modelo de dados deve ser inequívoco. Isto significa que enquanto for permitido que um analisador seja mais ou menos preciso do que o outro (ou seja, um analisador pode reportar mais informações sobre um evento do que o outro), não deve ser permitido produzir informações contraditórias em dois alertas que descrevem o mesmo evento, ou seja, os subconjuntos comuns da informação transmitida pelos dois analisadores devem ser idênticos, e precisam ser inseridas nos mesmos espaços reservados dentro da estrutura de dados do alerta. Claro, é sempre possível inserir todas as informações importantes sobre um evento em campos de extensão do alerta em vez de inserir nos campos onde ele pertence, no entanto, esta prática reduz a interoperabilidade e deve ser evitada sempre que possível. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.1.5. Relação entre processos.

Alertas de detecção de intrusão podem ser transmitidos a vários níveis. O documento RFC sobre o IDMEF aplica-se a todo o alcance, a partir de alertas muito simples (por exemplo, os alertas que são o resultado de uma única ação ou operação do sistema, como um relatório de *login* que falhou) até os muito complexos (por exemplo, a agregação de vários eventos causando um alerta a ser gerado). (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2. Arquitetura

Originalmente, duas formas de implementação foram propostas ao IDWG (*Intrusion Detection Working Group*), onde uma utilizava SMI (*Structure of Management Information*) para descrever um Protocolo de Gerenciamento de Redes Simples (SNMP - *Simple Management Network Protocol*) MIB, e outra utilizava DTD para descrever documentos em XML (*Extensible Markup Language*).

As propostas de implementação foram revisadas pelo IDWG nas reuniões realizadas em Setembro de 1999 e Fevereiro de 2000. Foi decidido na reunião de Fevereiro que a solução implementada em XML era a que melhor cumpria os requisitos do IDWG. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.1. A Linguagem XML.

A Linguagem XML é uma versão simplificada da Linguagem Padrão de Marcação Generalizada (SGML - *Standard Generalized Markup Language*), uma sintaxe utilizada para especificar a marcação de texto. A XML está ganhando atenção como uma linguagem para representação e troca de documentos e dados na Internet, e como a solução para a maioria dos problemas inerentes à *Hyper Text Markup Language* (HTML).

XML é uma metalinguagem que permite à um aplicativo definir a sua própria marcação, também permite a definição de linguagens de marcação personalizadas para diferentes tipos de documentos e aplicações distintos. Diferente do HTML, em que há um conjunto de identificadores predefinidos com os significados que necessitam ser adaptados para aplicações especializadas. Ambos XML e HTML usam elementos (*tags*) (identificadores delimitados por "<" e ">") e atributos (na forma de "*name = 'value'* "). Porém, onde "<p>" significa "parágrafo" em HTML, pode significar "parágrafo", "pessoa" ou "preço", como também pode não ter significado nenhum, dependendo de uma aplicação em particular.

A publicação do XML foi seguida por uma segunda recomendação feita pelo W3C (*Word Wide Web Consortium)*, onde definia o uso de *namespaces* em documentos XML. Um *namespace* em XML é uma coleção de nomes, identificados por um URI (*Uniform Resource Identifier*). Ao utilizar *namespaces*, cada *tag* é identificada pelo *namespace* que a segue, permitindo que *tags* de diferentes *namespaces* mas com o mesmo nome possam coexistir em um documento. Por exemplo, um único documento pode conter as *tags* "*usa*:*football*" e "*europe*:*football*", cada uma com significados diferentes. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.1.1. Justificativa para implementação do IDMEF em XML.

Aplicações baseadas em XML estão sendo utilizadas ou desenvolvidas para uma grande variedade de propósitos, incluindo transferência eletrônica de dados em uma variedade

de campos, a transferência de dados financeiros, cartões de visita eletrônicos, calendários e programações, distribuição de software empresarial, tecnologia *web* "*push*", e linguagens de marcação para química, matemática, música, dinâmica molecular, astronomia, publicações de livros e periódicos, publicações na *web*, observações meteorológicas, transações imobiliárias, e muitos outros.

A flexibilidade do XML torna uma boa escolha para estas aplicações; essa mesma flexibilidade torna uma boa escolha para a implementação do IDMEF também. Outras razões mais específicas para a escolha de XML para a implementação do IDMEF são:

- XML permite uma linguagem personalizada ser criada especificamente para o propósito de descrição de alertas de intrusão. Também define uma forma padrão para estender esta linguagem.
- Ferramentas de software para o processamento de documentos XML são amplamente disponíveis, em ambas as formas, comerciais e *open source*. Várias ferramentas e APIs para a análise e/ou validação de XML estão disponíveis em uma variedade de linguagens, incluindo Java, C, C++, Tcl, Perl, Python, e GNU Emacs Lisp. Um acesso mais difundido às ferramentas fará a adoção do IDMEF por desenvolvedores de produtos mais fácil, mais rápido.
- O XML atende aos requisitos do IDMEF, em que as mensagens apoiam completamente a internacionalização e localização. O padrão XML requer suporte para ambas codificações, UTF-8 e UTF-16 e Unicode, tornando todas as aplicações em XML (e, portanto, todas as aplicações IDMEF) compatíveis com estas codificações de caracteres.

XML também oferece suporte para a especificação, em uma base por elemento, na linguagem onde o elemento contido é escrito, tornando o IDMEF mais simples de adaptar para as versões de " suporte à linguagem natural" do produto.

- Os formatos de mensagem devem suportar filtragens e agregações. A integração do XML com XSL (*Extensible Stylesheet Language*), uma linguagem de estilo, permite que as mensagens sejam combinadas, descartadas e reorganizadas.
- Constantes projetos desenvolvidos em XML no W3C e em outros lugares, irão fornecer extensões orientadas a objeto, banco de dados entre outros recursos úteis. Se implementado em XML, o IDMEF automaticamente herda estes recursos também.
- XML é uma linguagem gratuita, sem licença, sem taxas de licença e sem *royalties*.

(CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.2. Declaração XML.

Documentos IDMEF sendo trocados entre os aplicativos compatíveis devem começar com uma declaração XML, e deve especificar a versão XML em uso.

Uma mensagem IDMEF deve, portanto, começar com:

```
1   <?xml version="1.0" encoding="UTF-8"?>
2
3     <idmef:IDMEF-Message version="1.0" xmlns:idmef="http://iana.org/idmef"/>
```

Figura 2: Exemplo de como deve começar uma mensagem IDMEF.

Aplicações IDMEF compatíveis podem optar por omitir a declaração XML internamente para conservar o espaço, acrescentando-a apenas quando a mensagem é enviada para outro destino (por exemplo, um navegador da *web*). Esta prática não é recomendada, a menos que ela possa ser realizada sem perder a versão de cada mensagem e informações de codificação.

Para ser válido, um documento XML deve conter um documento de definição de tipo. No entanto, isto representa uma sobrecarga significativa para uma aplicação IDMEF compatível, tanto em largura de banda que consome, bem como os requisitos que coloca no processador XML (e não apenas para analisar a declaração em si, mas também para analisar a DTD que referencia).

Os implementadores podem decidir, portanto, ter analisadores e gestores que concordam com a definição de tipo de documento fora-de-banda específico, que será utilizado para troca de mensagens (o padrão ou um com extensões), e depois omitir o tipo de documento de definição das mensagens IDMEF. Um grande cuidado deve ser tomado na negociação de tais acordos, pois como o gerente pode ter que aceitar mensagens de muitos analisadores diferentes, cada um usando um DTD com um conjunto diferente de extensões. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.3. Referência da Entidade dos Caracteres.

Recomenda-se que as aplicações IDMEF compatíveis usem o formulário de referência da entidade dos caracteres 'e', '<', '>', ' " ', e ' ' ' (aspas simples). Sempre escrever estes caracteres

em dados, para evitar qualquer possibilidade de má interpretação. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4. Tipos de Dados IDMEF.

Dentro de uma mensagem IDMEF XML, todos os dados serão expressos como "texto" (em oposição ao "binário"), uma vez que XML é uma linguagem de formatação de texto. Nós fornecemos a digitação de informações para os atributos das classes no modelo de dados, no entanto, para transmitir ao leitor o tipo de dados que o modelo prevê para cada atributo.

Cada tipo de dados no modelo tem requisitos específicos de formatação em uma mensagem IDMEF XML. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.1. Números Inteiros

Atributos inteiros são representados pelo tipo de dados INTEGER. Dados de número inteiro devem ser codificados em base 10 ou base 16 (Hexadecimal).

Codificações INTEGER em Base 10 utilizam os dígitos de 0 até 9 e um sinal opcional, podendo ser '-' ou '+', por exemplo, '123' e '-456'.

Codificações em Base 16 utilizam dígitos de 0 à 9 e letras de 'A' à 'F' (ou seus equivalentes em letra minúscula) e são precedidos pelos caracteres "0x", por exemplo, “0x1A2B". (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.2. Números Reais

Atributos reais são representados pelo tipo de dados REAL e devem ser codificados na Base 10. Codificação real é uma função POSIX 1003.1 da biblioteca "strtod": um sinal opcional ('+' ou '-') seguido por uma sequência não vazia de dígitos decimais, opcionalmente contendo um caractere base, em seguida, uma parte opcional expoente. Uma parte expoente consiste em um 'e' ou 'E' seguido de um sinal opcional, que será seguido de um ou mais dígitos decimais. Por exemplo, "123.45e02", "-567,89 e-03". Aplicações compatíveis com IDMEF

devem suportar tanto os caracteres '.' e ','. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.3. Strings e Caracteres.

Atributos de caracteres simples são representados pelo tipo de dados de caracteres. Atributos com vários caracteres de comprimento conhecido são representados pelo tipo de dados STRING. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.3.1. Entidade de Referência de Caracteres.

Dentro de documentos XML, alguns caracteres têm significados especiais em determinados contextos. Para incluir o próprio caractere real em um desses contextos, uma sequência especial de escape, chamado de entidade de referência, tem de ser usado. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

Os caracteres que às vezes precisam de escape, e suas referências de entidade, são:

Tabela 1: Tabela contendo os caracteres e suas respectivas entidades de referência segundo o padrão IDMEF

Caractere	Entidade de Referência
&	&amp;
<	&lt;
>	&gt;
"	&quot;
'	&apos;

2.2.4.3.2. Código de referência de caracteres.

Qualquer caractere definido pelas normas ISO/IEC 10646 e Unicode podem ser incluídos em um documento XML com o uso de uma referência de caractere. A referência de

caractere é iniciada com os símbolos 'e' e '#', e terminam com o caractere ';'. Entre esses caracteres, é inserido o código de caracteres para o caractere.

Se o código de caractere é precedido de um 'x', este então é interpretado em hexadecimal (base 16), caso contrário, é interpretada em decimal (base 10). Por exemplo, o símbolo comercial (&) é codificado como '&', ou '&' e o sinal de "menor" (<) é codificado como '<' ou '<'.

Qualquer caractere de um, dois ou quarto *bytes* pode ser incluso em um documento utilizando esta técnica. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.4. Bytes

Os dados binários são representados pelo tipo de dados BYTE (e byte []). Dados binários devem ser codificados em sua totalidade usando base 64. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.5. Tipos Enumerados

Os tipos enumerados são representados pelo tipo de dados ENUM, e consistem de uma lista ordenada de valores aceitáveis. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.6. Strings de Data e Hora

Strings de Data e Hora são representados pelo tipo de dados DATETIME. Cada sequência de data e hora identificam um instante específico no tempo.

Strings de data devem conter o formato AAAA-MM-DD (YYYY-MM-DD), enquanto o formato de horas deve ser hh:mm:ss. O tempo pode ser formatado para receber uma fração decimal de segundos, podendo ser hh:mm:ss.ss ou hh.mm.ss,ss. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.7. NTP Timestamps.

Timestamp NTP (*Network Time Protocol*) são representados pelo tipo de dados NTPSTAMP. Um *timestamp* NTP é um número de ponto fixo não assinado de 64 bits. A parte inteira é nos primeiros 32 bits, e a parte da fração é nos últimos 32 bits. Dentro mensagens IDMEF, *timestamps* NTP devem ser codificados como dois valores hexadecimais de 32 bits, separados por um ponto ('.'). Por exemplo, "0x12345678.0x87654321". (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.8. Port Lists

Port Lists são representados pelo tipo de dados PORTLIST e consistem em uma lista de números inteiros (individuais) e os intervalos separados por vírgulas (NM significa portas N através de M, inclusive). Qualquer combinação de números e intervalos podem ser utilizados em uma única lista. Por exemplo, "5-25,37,42,43,53,69-119,123-514". (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.2.4.9. Identificadores Únicos

Existem dois tipos de identificadores únicos usados na presente especificação. Ambos os tipos são representados por tipos de dados String. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3. O Modelo de Dados IDMEF e DTD

Nesta seção, os componentes individuais do modelo de dados IDMEF são explicados em detalhes. Os diagramas da Linguagem de Modelagem Unificados (UML - *Unified Modeling Language*) do modelo são fornecidos para mostrar como os componentes se relacionam entre si, e as secções relevantes do IDMEF DTD serão apresentadas para mostrar como o modelo é traduzido para o XML. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.1. Visão geral do modelo de dados.

O relacionamento entre os componentes principais do modelo de dados é mostrado na Figura 3 (indicadores de ocorrência, e os atributos são omitidos).

A classe de nível superior para todas as mensagens IDMEF é a IDMEF-Message; cada tipo de mensagem é uma subclasse desta. Existem atualmente dois tipos de mensagens definidas: alertas (*Alerts*) e batimentos cardíacos (*Heartbeats*). Dentro de cada mensagem, subclasses da classe de mensagem são utilizados para fornecer a informação detalhada realizada na mensagem.

É importante notar que o modelo de dados não especifica como um alerta deve ser classificado ou identificado. Por exemplo, uma varredura de portas pode ser identificada por um analisador como um único ataque contra alvos múltiplos, enquanto outro analisador pode identificá-lo como vários ataques de uma única fonte. Entretanto, uma vez que um analisador determinou o tipo de alerta que pretende enviar, o modelo de dados determina como esse alerta deve ser formatado. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

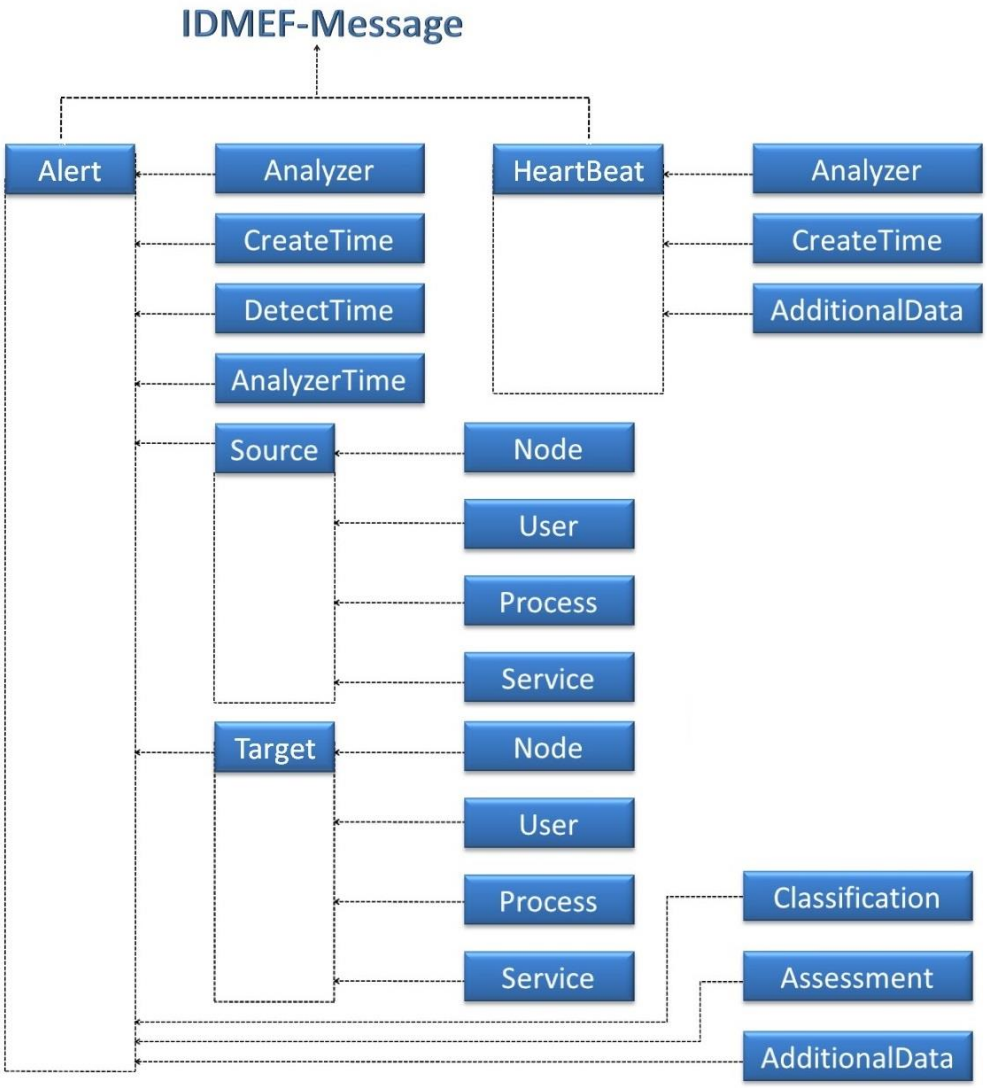

Figura 3: Visão Geral do Modelo de Dados (IDMEF).

2.3.2. Classes de Mensagem.

As classes serão individualmente descritas nos tópicos a seguir.

2.3.2.1. A Classe IDMEF-Message

Todas as mensagens IDMEF são instâncias da classe IDMEF-Message, esta é a classe de nível mais alto do modelo de dados IDMEF, bem como o IDMEF DTD. Existem atualmente dois tipos (subclasses) de IDMEF-Message: *Alert* e *Heartbeat*.

Esta classe possui um único atributo: Versão.

A versão da especificação IDMEF-Message está de acordo com esta mensagem. Aplicações especificando um valor para esse atributo devem especificar o valor "1.0”. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.2.1.1. Classe Alert

Geralmente, cada vez que um analisador detecta um evento que foi configurado para procurar, ele envia uma mensagem de alerta para o seu (s) gerente (s). Dependendo do analisador, uma mensagem de alerta pode corresponder a um único ou aos vários eventos detectados. Os alertas ocorrerem de forma assíncrona em resposta a eventos externos.

Uma mensagem de alerta é composta por várias classes de agregados, como mostrado na Figura 4.

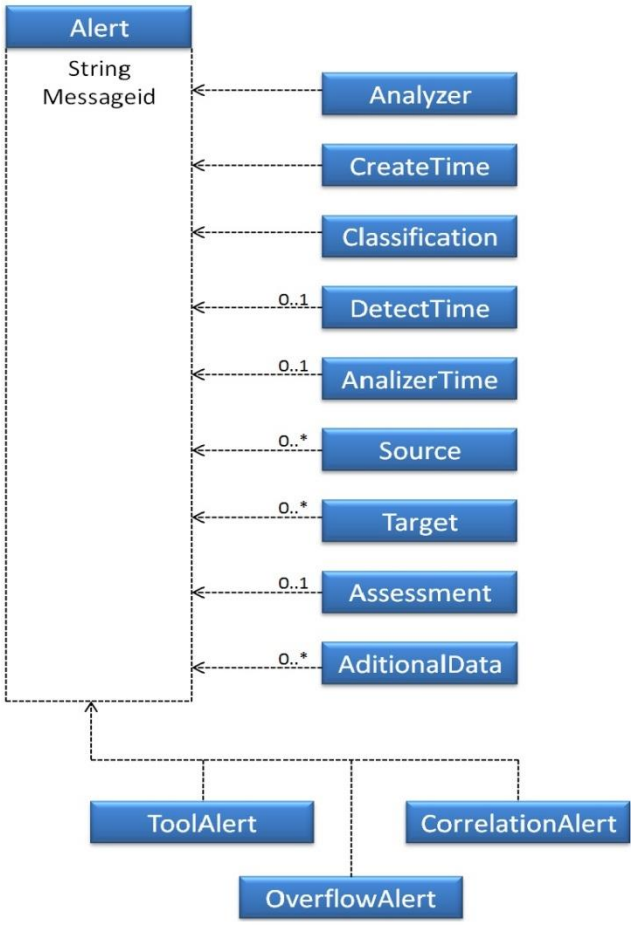

Figura 4: Classe Alert

As classes agregadas que compõem a classe Alerta são:

Analyzer (Analisador): Apenas um. Contém informações de identificação para o analisador que originou o alerta.

CreateTime (Hora de Criação): Apenas um. O tempo em que o alerta foi criado. Das três vezes que podem ser fornecidas com um alerta, este é o único que é necessário.

Classification (Classificação): Apenas um. O "nome" do alerta, ou outra informação que permita ao gerente determinar o que é.

DetectTime (Detector de Tempo): Zero ou um. Esta classe informa a data e hora da produção de um (ou mais) evento (s) detectado (s) pelo analisador. No caso de mais de um evento, ele irá informar os dados do primeiro acontecimento. Em algumas circunstâncias, isso pode não ser o mesmo valor que CreateTime.

AnalyzerTime (Tempo do Analisador): Zero ou um. Contém o tempo atual do analisador.

Source (Fonte): Zero ou mais. A origem do (s) evento (s) que antecederam o alerta.

Target (Alvo): Zero ou mais. O (s) alvo (s) do (s) evento (s) que conduz ao alerta.

Assessment (Avaliação): Zero ou um. Informações sobre o impacto do evento, as ações tomadas pelo analisador em resposta a ele, e a confiança do analisador na sua avaliação.

AdditionalData (Dados Adicionais): Zero ou mais. Informações incluídas pelo analisador que não se encaixam no modelo de dados. Pode ser uma parte atômica de dados, ou uma grande quantidade de dados fornecidos através de uma extensão para o IDMEF.

Alertas são representados no IDMEF DTD conforme apresentado na Figura 5:

```
1  <!ELEMENT Alert(
2        Analyzer, CreateTime, DetectTime?, AnalyzerTime?,
3        Source*, Target*, Classification, Assessment?, (ToolAlert |
4        OverflowAlert | CorrelationAlert)?, AdditionalData*
5      )>
6    <!ATTLIST Alert messageid CDATA '0' %attlist.global; >
```

Figura 5: Código de representação da classe Alert.

A classe Alert possui um único atributo: messageId. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.2.1.2. Classe ToolAlert

A classe ToolAlert traz informações adicionais relacionadas ao uso de ferramentas de ataque ou programas mal intencionados, como *Trojan Horses* (Cavalos de Tróia), e podem ser usados pelo analisador quando se é capaz de identificar estas ferramentas. Destina-se a um grupo ou mais de alertas previamente enviados juntos, com o objetivo de informar "esses alertas foram o resultado de alguém usando esta ferramenta."

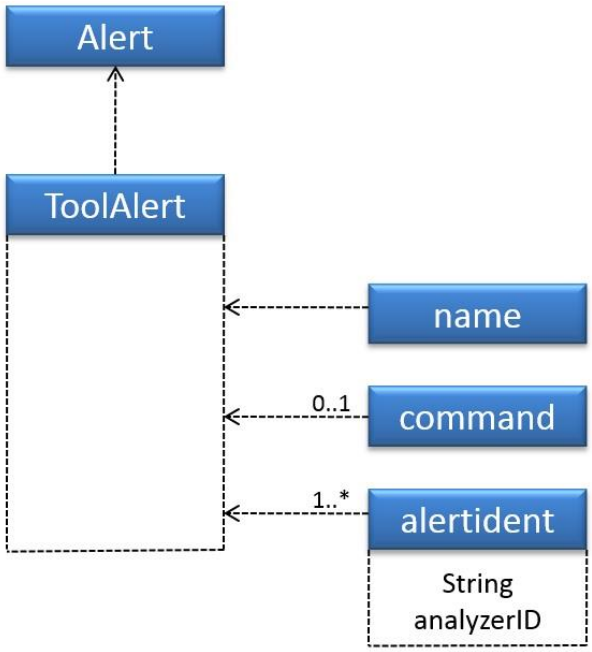

Figura 6: Classe ToolAlert.

As classes agregadas que compõem a ToolAlert são:

Name (Nome): Exatamente um. Tipo String. A razão para o agrupamento dos alertas em conjunto, por exemplo, o nome de uma ferramenta especial.

Command (Comando): Zero ou um. Tipo String. O comando ou operação que a ferramenta foi solicitada a realizar, por exemplo, o ping *BackOrifice*.

Alertident (Identificador): Um ou mais. Tipo String. A lista de identificadores de alerta que estão relacionados com este alerta. Identificadores de alerta são apenas exclusivos entre os alertas enviados por um único analisador, o "analyzerid" atributo opcional de "alertident" deve ser utilizado para identificar de qual analisador que uma determinada indicação veio. Se o "analyzerid" não é fornecido, assume-se que o alerta veio do mesmo

analisador que está enviando a ToolAlert. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

Isto é representado no IDMEF DTD da seguinte forma:

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

Figura 7: Código de representação da classe ToolAlert.

2.3.2.1.3. Classe CorrelationAlert

A classe CorrelationAlert carrega informações adicionais relacionadas com a correlação de informações de alertas. Destina-se a um grupo ou mais de alertas previamente enviados juntos, para dizer "estes alertas estão relacionados".

A classe é composta por outras duas classes agregadas, como pode ser vista na Figura 8, a seguir:

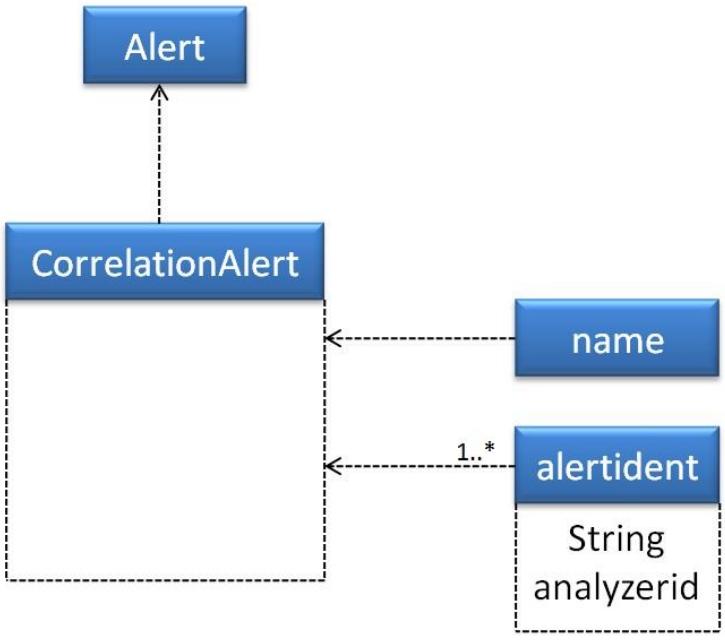

Figura 8: Classe CorrelationAlert.

As classes agregadas que compõem a CorrelationAlert são:

Name (Nome): Exatamente um. Tipo String. A razão para agrupar os alertas em conjunto, por exemplo, um método de correlação determinado.

Alertident: Um ou mais. Tipo String. A lista de identificadores de alerta que estão relacionados com este alerta. Identificadores de alerta são apenas exclusivos entre os alertas enviados por um único analisador, o "analyzerid" atributo opcional de "alertident" deve ser utilizado para identificar de qual analisador que uma determinada indicação veio. Se o "analyzerid" não é fornecido, assume-se que o alerta veio do mesmo analisador que está enviando a CorrelationAlert. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

É representado no modelo de dados da seguinte forma:

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

Figura 9: Código de representação da Classe CorrelationAlert.

2.3.2.1.4. Classe OverflowAlert

O OverflowAlert carrega informações adicionais relacionadas a ataques de *overflow* do *buffer*. Destina-se a permitir um analisador para fornecer os detalhes do ataque de *overflow*.

A classe OverflowAlert é composta por três classes agregadas, como mostrado na Figura 10.

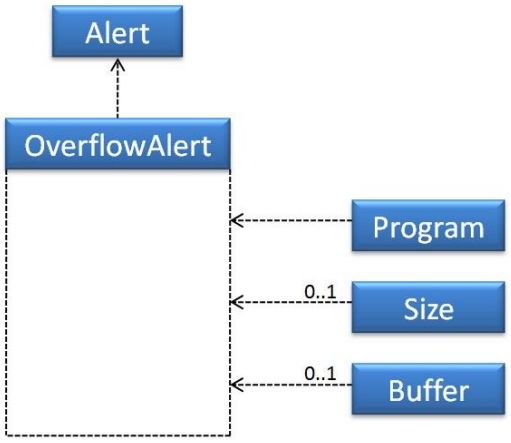

Figura 10: Classe OverflowAlert

As classes agregadas que compõem a OverflowAlert são:

Program (Programa): Exatamente um. Tipo String. O programa que o ataque de *overflow* tentou executar. (Este não é o programa que foi atacado).

Size (Tamanho): Zero ou um. Tipo Integer. O tamanho em *bytes* do *overflow*, ou seja, o número de *bytes* que o intruso enviou.

Buffer: Zero ou um. Tipo Byte[]. Alguns ou todos os dados de *overflow*, dependendo de quanto o analisador conseguir capturar. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

É representado pelo IDMEF DTD conforme apresentado pela Figura 11:

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

Figura 11: Código de representação da Classe OverflowAlert

2.3.2.1.5. Classe Heartbeat

Analisadores utilizam mensagem de pulsação (*Heartbeat Messages*) para indicar seu status atual para os gerentes. *Heartbeats* destinam-se a ser enviadas em um período regular, digamos, a cada dez minutos ou a cada hora. A recepção de uma mensagem de pulsação por um analisador, indica ao gerente que este está instalado e funcionando, a falta de uma mensagem de pulsação (ou, mais provavelmente, a falta de um certo número de mensagens de pulsação consecutivos) indica que o analisador ou sua conexão de rede falharam.

Todos os gerentes devem apoiar a recepção de mensagens de pulsação. No entanto, a utilização destas mensagens por analisadores é opcional. Os desenvolvedores de *softwares* gerenciais devem permitir que o mesmo seja configurado em uma base "por analisador" de usar/não usar mensagens de pulsação.

Uma mensagem de pulsação é composta de várias classes de agregados, tal como apresentado na Figura 12.

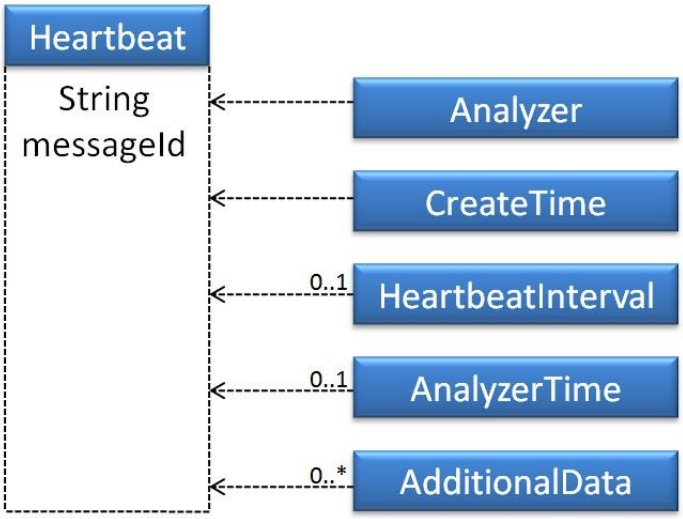

Figura 12: Classe Heartbeat.

As classes agregadas que compõem a Heartbeat são:

Analyzer (Analisador): Apenas um. Contém informações de identificação para o analisador que originou o *Heartbeat*.

CreateTime (Hora de Criação): Exatamente um. O momento em que o *Heartbeat* foi criado.

HeartbeatInterval (Intervalo de Pulsação): Zero ou um. O intervalo em segundos em que as pulsações são geradas.

AnalyzerTime (Tempo do Analisador): Zero ou um. O tempo atual no analisador.

AdditionalData (Dados Adicionais): Zero ou mais. São informações incluídas pelo analisador que não se encaixam no modelo de dados. Pode ser uma parte atômica ou uma grande quantidade de dados fornecidos através de uma extensão do IDMEF.

Esta classe é representada conforme a Figura 13:

```
1   <!ELEMENT Heartbeat(
2         Analyzer, CreateTime, HeartbeatInterval?, AnalyzerTime?,AdditionalData*)>
3     <!ATTLIST Heartbeat messageid CDATA '0' %attlist.global;>
```

Figura 13: Código de representação da Classe Heartbeat.

Esta classe possui apenas um atributo: messageId, este que é opcional, sendo o único identificador da classe Heartbeat. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.2.1.6. As Classes Core

São as classes principais: Analyzer, Source, Target, Classification e AdditionalData. São os principais componentes de Alert e Heartbeats, como apresentado na Figura 14. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

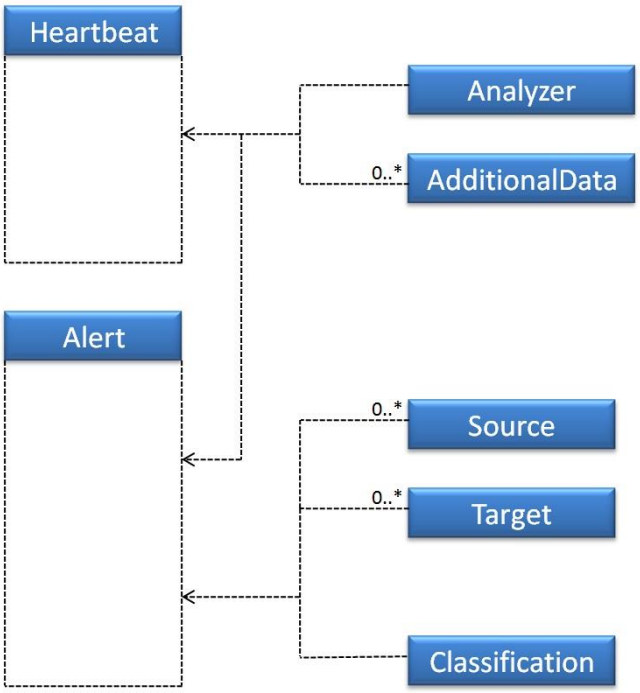

Figura 14: As Classes Core.

2.3.2.1.6.1. Classe Analyzer

A classe Analyzer identifica o analisador a partir de qual mensagem de *alert* ou *heartbeat* o originou. Apenas um analisador pode ser codificado para cada alerta ou pulsação, e deve ser o analisador em que o *alert* ou *heartbeat* foi originado. Embora o modelo de dados IDMEF não impede o uso de sistemas de detecção de intrusão hierárquicos (onde os alertas se retransmitidas para cima na árvore), e não fornece qualquer forma de registro da identidade dos analisadores "retransmissores" ao longo do trajeto do analisador de origem para o gerente que finalmente recebe o alerta.

A classe Analyzer é composto por três classes de agregados, tal como apresentado na Figura 15.

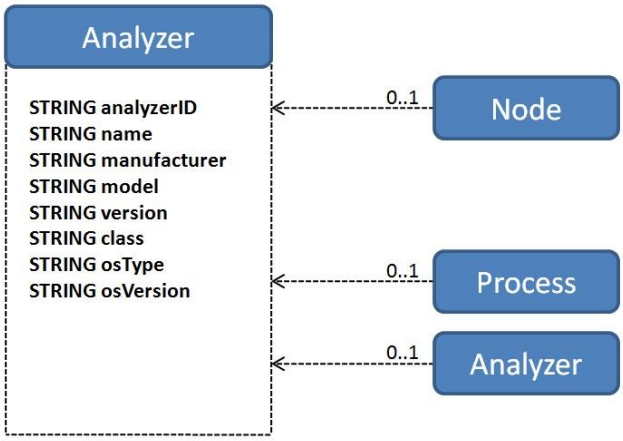

Figura 15: Classe Analyzer

As classes agregadas que compõem a classe Analyzer são:

Node (Nó): Zero ou um. Informações sobre o *host* ou dispositivo em que o analisador reside (Endereço da rede, nome da rede, etc.).

Process (Processo): Zero ou um. Informações sobre o processo em que o analisador está executando.

Analyzer (Analisador): Zero ou um. Informações sobre o analisador do qual a mensagem pode ter passado. A ideia por traz deste mecanismo é de que quando um gerente recebe um alerta e deseja enviá-lo para outro analisador, ele necessita substituir as informações do analisador original com suas próprias informações. Para preservar as informações do analisador, estas podem ser inclusas nas definições do novo analisador. Isto irá permitir rastrear o caminho do analisador.

É representado no IDMEF como mostra a Figura 16:

```
1   <!ELEMENT Analyzer(Node?, Process?, Analyzer?)>
2      <!ATTLIST Analyzer
3         analyzerid CDATA'0'
4         name CDATA #IMPLIED
5         manufacturer CDATA #IMPLIED
6         model CDATA #IMPLIED
7         version CDATA #IMPLIED
8         class CDATA #IMPLIED
9         ostype CDATA #IMPLIED
10        osversion CDATA #IMPLIED
11        %attlist.global;
12     >
```

Figura 16: Código de representação da Classe Analyzer.

Esta classe possui oito atributos:

AnalyzerId (ID do Analisador): Opcional (veja abaixo o porquê). Um único identificador para o analisador.

Este atributo é apenas "parcialmente" opcional. Se o analisador faz uso dos atributos de "ident" de outras classes para fornecer identificadores únicos para esses objetos, então também deve fornecer um atributo "analyzerid" válido. Esta exigência é ditada pelos requisitos de singularidade do atributo "ident" (que são exclusivos apenas dentro do contexto de uma "analyzerid" em particular). Se o analisador não faz uso do atributo "ident", no entanto, pode também omitir o atributo "analyzerid".

Name (Nome): Opcional. Um nome explícito para o analisador, que pode ser mais fácil de entender do que o analyzerid.

Manufacturer (Fabricante): Opcional. O fabricante do *software* e/ou *hardware* do analisador.

Model (Modelo): Opcional. O modelo do nome/número do *software* e/ou *hardware* do analisador.

Version (Versão): Opcional. A versão do *hardware* e/ou *software* do analisador.

Class (Classe): Opcional. A classe do *hardware* e/ou *software* do analisador.

OsType (Tipo do Sistema Operacional): Opcional. Consiste no nome do Sistema Operacional.

OsVersion (Versão do Sistema Operacional): Opcional. Contém a versão do Sistema Operacional.

Os conteúdos de "manufacturer", "model", "version" e "class" são específicos do fornecedor, mas podem ser utilizados em conjunto para identificar os diferentes tipos de analisadores. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.2.1.6.2. Classe Classification

A classe Classification fornece o "nome" de um alerta, ou outra informação que permita ao gerente determinar o que este é. O nome é escolhido pelo fornecedor de alerta.

Esta classe é constituída por uma classe agregada, tal como mostrado na Figura 17.

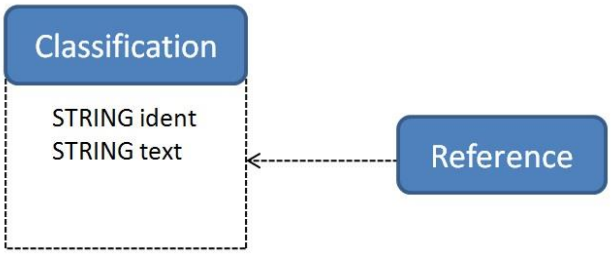

Figura 17: Classe Classification.

As classes agregadas compõem Classification são:

Reference (Referência): Zero ou um. Contém informações sobre a mensagem, apontando para sites de documentação externa, que irão fornecer informações básicas sobre o alerta.

Esta classe é representada pelo alerta como apresentado pela Figura 18:

```
1   <!ELEMENT Classification( Reference* )>
2      <!ATTLIST Classification
3         ident CDATA '0'
4         text CDATA #REQUIRED
5       >
```

Figura 18: Código de representação da Classe Classification.

Esta Classe possui dois atributos:

Ident (Identificador): Opcional. Um único identificador para esta classificação.

Text (Texto): Obrigatório. Uma String que identifica a mensagem de alerta, fornecida pelo fornecedor. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.2.1.6.3. Classe Source

A classe Source contém informações sobre a possível fonte (s) do (s) evento (s) que gerou um alerta. Um evento pode ter mais de uma fonte (por exemplo, em um ataque distribuído de negação de serviço). Esta classe é composta de outras quatro classes agregadas, conforme demonstrado na Figura 19.

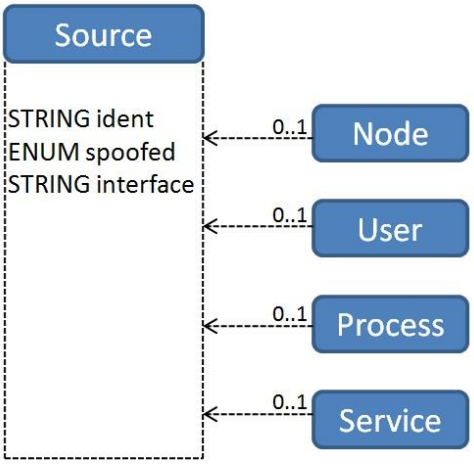

Figura 19: Classe Source

As classes agregadas que compõem esta classe são:

Node (Nó): Zero ou um. Informações sobre o *host* ou dispositivo que estão causando o (s) evento (s) (Endereço de Rede, Nome da Rede, etc.).

User (Usuário): Zero ou um. Informações sobre o usuário que parece causar o (s) evento (s).

Process (Processo): Zero ou um. Informações sobre o serviço de rede envolvido no evento.

É representado pelo IDMEF conforme apresentado na Figura 20:

```
1  <!ELEMENT Source(Node?, User?, Process?, Service?)>
2     <!ATTLIST Source
3        ident CDATA '0'
4        spoofed %attvals.yesno; 'unknown'
5        interface CDATA #IMPLIED
6        %attlist.global;
7     >
```

Figura 20: Código de representação da Classe Source.

A classe Source possui três atributos:

Ident (Identificador): Opcional. Um identificador exclusivo para esta fonte (*source*).

Spoofed (Falsificado): Opcional. Uma indicação se a fonte é, na medida em que o analisador pode determinar, um endereço falso usado para esconder a verdadeira origem do ataque. Os valores permitidos para este atributo são mostrados na Tabela 2, abaixo. O valor padrão é "*unknown*".

Tabela 2: Valores permitidos para o atributo Spoofed.

Rank	Palavra-chave	Descrição
0	Unknown	Precisão de informações de origem desconhecida.
1	Yes	Acredita-se que a fonte é um chamariz.
2	No	Acredita-se que a fonte é "real".

Interface (Interface): Opcional. Pode ser utilizado por um analisador baseado em redes com múltiplas interfaces para identificar em qual interface esta fonte foi visualizada. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.2.1.6.4. Classe Target

A classe Target contém informações sobre o possível alvo do evento que gerou um alerta. Um evento pode ter mais do que um alvo (por exemplo, no caso de uma remoção de porta). Como mostrado na imagem abaixo, a classe Target é composta por quatro classes agregadas, como apresentado na Figura 21.

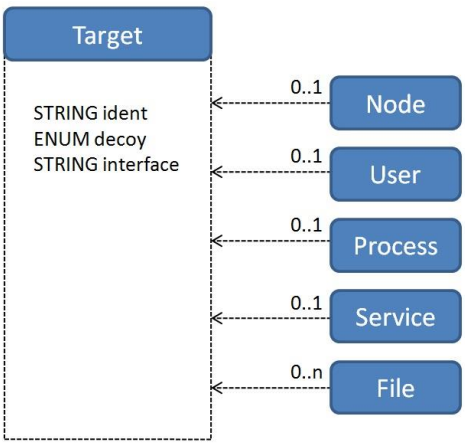

Figura 21: Classe Target

As classes agregadas que à compõem são:

Node (Nó): Zero ou um. Informações sobre o *host* ou dispositivo em que o evento está sendo dirigido.

User (Usuário): Zero ou um. Informações sobre o usuário no qual o evento está sendo dirigido.

Process (Processo): Zero ou um. Informações sobre o processo no qual o evento está sendo dirigido.

Service (Serviço): Zero ou um. Informações sobre os serviços de rede envolvidos no evento.

File (Arquivo): Opcional. Informações sobe arquivos envolvidos nos eventos.

É representada no IDMEF TDT conforme a Figura 22:

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

Figura 22: Código de representação da Classe Target

Esta classe possui três atributos:

Ident (Identificador): Opcional. Um único identificador para este *target*.

Decoy (Isca): Opcional. Uma indicação de que o alvo é, na medida em que o analisador pode determinar, ou seja, uma isca. Os valores permitidos para este atributo são apresentados na Tabela 3. O valor padrão é "*unknown*".

Tabela 3: Valores permitidos para o atributo decoy

Rank	Palavra-chave	Descrição
0	Unknown	Precisão de informações de origem desconhecida.
1	Yes	Acredita-se que o alvo é uma isca.
2	No	Acredita-se que o alvo é "real".

Interface (Interface): Opcional. Pode ser utilizado por um analisador baseado em redes com múltiplas interfaces para identificar em qual interface o alvo foi visualizado. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.3.2.1.7. Classe Assessment

A classe Assessment é usada para fornecer uma avaliação do analisador de um evento, tais como: o seu impacto, medidas tomadas em resposta, e confiança. Esta classe compõe-se de três classes agregadas, tal como mostrado na Figura 23.

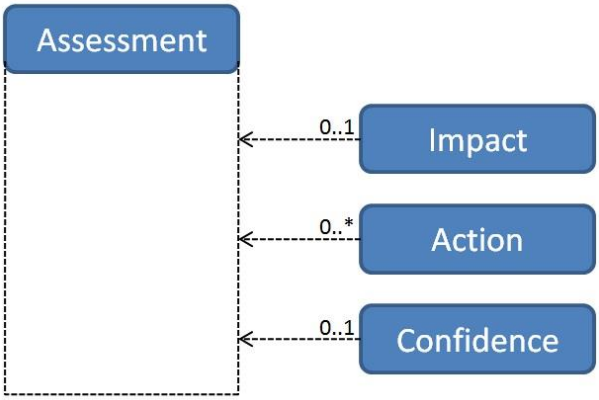

Figura 23: Classe Assessment

As classes que compões Assessment são:

Impact (Impacto): Zero ou um. A avaliação do analisador sobre o impacto do evento ao alvo.

Action (Ação): Zero ou mais. As ações tomadas pelo analisador em resposta ao evento.

Confidence (Confiança): Zero ou um. A medida da confiança que o analisador tem na sua avaliação do evento. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

Esta classe é representada pelo Modelo de Dados tal como apresentado na Figura 24:

```
1   <!ELEMENT Assessment(Impact?, Action*, Confidence?)>
2      <!ATTLIST Assessment
3         %attlist.global;
4       >
```

Figura 24: Código de representação da Classe Assessment

2.3.2.1.8. Classe AdditionalData

A classe AdditionalData é utilizada para prover informações que não podem ser representadas pelo modelo de dados. AdditionalData pode ser utilizada para fornecer dados atômicos (integers, Strings, etc.) nos casos em que pequenas quantidades de informações adicionais devem ser enviadas; pode ser utilizada também para estender o modelo de dados e o DTD para apoiar na transmissão de dados mais complexos. Na Tabela 4 pode-se visualizar os valores atômicos fornecidos por esta classe.

Tabela 4: Valores atômicos providos pela classe AdditionalData

Rank	Keyword	Descrição
0	Boolean	O elemento contém um valor booleano. (“*true*” ou “*false*”).
1	Byte	O conteúdo do elemento é um *byte* de 8-bits.
2	Character	O conteúdo do elemento é um caractere.
3	Date-time	O conteúdo do elemento é uma string *date-time*.
4	Integer	O conteúdo do elemento é um integer.
5	Ntpstamp	O conteúdo do elemento é uma NTP *timestamp*.
6	Portlist	O conteúdo do elemento é uma lista de portas.
7	Real	O conteúdo do elemento é um número real.
8	String	O conteúdo do elemento é uma String.
9	Byte-string	O conteúdo do elemento é um *byte*[].
10	Xmltext	O conteúdo do elemento são dados XML.

A classe AdditionalData é declarada conforme apresentado na Figura 25.

```
<!ENTITY % attvals.adtype                "
      ( boolean | byte | character | date-time | integer | ntpstamp |
        portlist | real | string | byte-string | xmltext )
    ">

  <!ELEMENT AdditionalData           (
    (boolean | byte        | character | date-time |
     integer | ntpstamp    | portlist  | real      |
     string  | byte-string | xmltext  )
   )>

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

Figura 25: Código de representação da Classe AdditionalData

Esta classe possui um único atributo:

Meaning (Significado): Opcional. Uma String que descreve o significado do conteúdo do elemento. Estes valores serão dependentes de acordo com vendedor e/ou implementação; o método para assegurar que os gestores possam compreender as Strings enviadas pelos analisadores está fora do escopo desta especificação. A lista de palavras-chave de significado aceitáveis não está dentro do escopo do documento. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

2.4. Demais classes do modelo de dados

Deve-se ressaltar que o projeto não se aplica apenas a estas classes descritas neste capítulo, todas as classes do modelo de dados foram implementadas, porém, não foram apresentadas neste tópico pois o mesmo se estenderia mais que o necessário. Nesta etapa foram explicadas apenas as classes mais importantes do projeto, no entanto, todas as outras classes restantes podem ser encontradas no Apêndice 1, Apêndice 2 e Apêndice 3, ao fim deste documento, onde são abordadas todas as classes que não se encontram aqui, contendo as mesmas especificações e detalhamento destas descritas.

3. Trabalhos Correlatos

Durante o desenvolvimento de todo o trabalho, foram utilizados alguns trabalhos correlatos, que deram suporte e também ideias em algumas etapas durante criação deste projeto. Neste tópico serão descritos alguns que mais se destacaram e que mais foram úteis para todo o desenvolvimento e implementação do projeto como um todo, também será relacionado cada um dos trabalhos com este projeto em questão.

3.1. Improvement and implementation of IDMEF Data Model

Este trabalho, que foi idealizado por Ming Han, Xu Dewu, Chen Wei e publicado pelo IEEE em 2010, propõe uma melhoria no modelo de dados IDMEF, buscando prover a interoperabilidade e maximizar a extensibilidade das mensagens que serão representados em XML, implementando um novo modelo de dados em XML.

3.1.1. Descrição do trabalho

De acordo com os autores deste artigo (HAN, Ming; DEWU, Xu; WEI, Chen. 2010), para que possa haver uma interoperabilidade na troca distribuída de mensagens de detecção de intrusão, o projeto proposto pelo IDWG (*Intrusion Detection Working Group*) em IDMEF foi estendido, neste novo modelo, foram adicionados três novos tipos de mensagens para os dois tipos fornecidos pelo IDMEF (*Alert* e *Heartbeat*), conforme a Figura 58. As cinco categorias de mensagens são descritas na sequência:

***Alerts*:** São respostas espontâneas pelo sistema de intrusão, como Snort. O agrupamento e supressão do *Alert* é um problema muito complexo em sistemas de detecção de intrusão atuais. O projeto do IDMEF define algumas classificações do alerta, como *ToolAlert*, *CorrelationAlert* e *OverflowAlert*, porém, apenas estas não são suficientes. Para este fim, foi adicionado a classe *CorrelationAlert Messages* em paralelo as classes de alerta.

Heartbeat: Analisadores utilizam mensagens de *Heartbeat* para indicar seu status atual aos gerentes. *Heartbeats* destinam-se a serem enviados em um período regular, podendo ser de dez em dez minutos ou a cada hora. Estas mensagens são usadas para indicar o status

atual para nós superiores da alavanca (IDS) e vice-versa. A falta de uma mensagem de *Heartbeat* indica que o analisador ou a sua conexão de rede falhou.

***CorrelationAlert*:** A função deste alerta é gerar um alerta global e sintético. Porém, esta classe é diferente da subclasse *CorrelationAlert*, uma das classes de *Alert*. Foi desenvolvido um novo modelo para alertas de correlação, este modelo adiciona um mecanismo de resposta para determinar se o alerta será realmente transmitido. O princípio desta correlação é um console que coleta primeiramente todas as mensagens de *correlationAlert* de sensores distribuídos, e em seguida, por meio de um algoritmo de correlação entre outros métodos, é determinado se o alerta deve ser enviado para a troca com outros alertas. Tais métodos possuem duas vantagens: Primeiro, diminuir a complexidade de comunicação e largura de banda, e em segundo lugar, diminuir o número de falsos alertas.

***Topology Messages*:** Os tipos básicos de topologia são: adotar (*adopt*), recordar (*recall*) e negar (*deny*). Em um sistema distribuído e dinâmico de intrusão, alguns IDS irão se juntar em sistemas de detecção de intrusão para defender aos ataques cooperativamente. IDS de usuários autorizados serão adotados. Porém, intrusos não podem ser permitidos para se juntar ao sistema, visando preservar dados sensíveis. Mensagens de *recall* irão verificar se IDSs autorizadas irão se juntar ao sistema em tempos periódicos.

***DBUpdates Messages*:** A detecção de intrusão distribuída possui um mecanismo automático para atualizar seu conjunto de regras. Pode-se utilizar o *Bloom Filtre* para filtrar algum alerta armazenado, para que este tenha uma grande vantagem espacial para representar conjuntos de alerta.

A Figura 26 abaixo apresenta um diagrama com as alterações propostas para o IDMEF-Message.

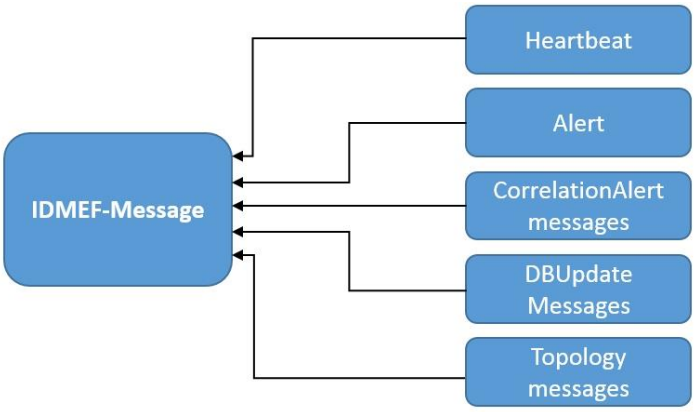

Figura 26: Modelos de Alerta propostos para o modelo de dados IDMEF. Fonte: (HAN, Ming; DEWU, Xu; WEI, Chen. Improvement and implementation of IDMEF Data Model, 2010)

3.1.2. Relação com este trabalho

O trabalho proposto acima se assemelha muito com o proposto neste projeto, porém, o foco é um pouco diferente, em sua proposta, o trabalho "*Improvement and implementation of IDMEF Data Model"* busca, como o próprio nome diz, melhorar o modelo de dados IDMEF, adicionando características e algumas funcionalidades novas, visando o aperfeiçoamento da interoperabilidade e já fornecendo um algoritmo de resposta, o que não é feito neste projeto em questão, porém, em um projeto futuro, poderá ser feito uma utilização destas alterações no modelo de dados para buscar uma melhora neste modelo que já está pronto.

3.2. Web Application Security: Communications in Computer and Information Science

Este trabalho avaliou a aplicabilidade de tecnologias de *web* semântica em sistemas de gestão de segurança da informação, proporcionando uma forma de compartilhar informações semanticamente entre diferentes domínios de segurança. Para isso, foi definida uma ontologia baseada em IDMEF, que pode conter todas as informações de qualquer mensagem IDMEF. (SERRÃO, Carlos; DÍAZ, Vicente Aguilera; CERULLO, Fabio. 2009)

3.2.1. Descrição do trabalho

Segundo seus autores, o modelo de dados IDMEF fornece uma linguagem comum para gerar alertas sobre eventos suspeitos, que permitem vários sistemas colaborarem na detecção dos ataques, ou no tratamento de alertas armazenados. Mesmo que o modelo IDMEF possua algumas vantagens (integração de diversas fontes, como também o uso de um formato bem suportado), este possui algumas desvantagens (fontes de dados heterogêneas pode levar o sistema a gerar vários alertas do mesmo ataque, mas que não contém as mesmas informações).

Para que os autores pudessem resolver os problemas identificados, os mesmos definiram uma ontologia de alerta baseada na estrutura do IDMEF. Neste processo, vale ressaltar que o modelo de dados IDMEF foi definido seguindo um modelo de classes e prioridades, tornando a definição das ontologias mais fácil, e aproveitando as vantagens da *web* semântica (distribuição, consulta, inferência, etc.). Várias restrições de classe foram definidas (cardinalidade, tipos de dados), analisando as definições do IDMEF.

De acordo com o documento feito pelos autores do projeto (SERRÃO, Carlos; DÍAZ, Vicente Aguilera; CERULLO, Fabio. 2009), as seguintes normas contidas no modelo IDMEF foram utilizadas:

- Nomes de classes iniciam com letras maiúsculas e são exatamente os mesmos nomes usados pelo IDMEF.
- Nomes de atributos iniciam com letra minúscula e possuem o formato *domain_propertyName*, onde *domain* é o nome da classe à qual o atributo pertence, e *propertyName* é o nome do atributo.
- Cada classe na IDMEF-Message mapeia a uma classe na ontologia IDMEF.
- Cada atributo em uma classe IDMEF é mapeado a um tipo de dados em sua correspondente classe da ontologia.
- Classes que estão contidas em outra classe são mapeados em propriedades gerais de tipo-objeto. Uma exceção a isso são as classes agregadas que contêm texto, a qual são mapeadas para propriedades de tipo de dados *data*.
- Uma subclasse IDMEF também é representada como subclasse na ontologia, herdando todas as propriedades de sua classe pai.
- Quando um atributo do IDMEF não pode conter vários valores, este é mapeado a uma classe funcional.

- Quando um atributo do IDMEF pode apenas conter alguns valores específicos, a ontologia define quais valores serão permitidos.
- Atributos numéricos são representados como tipo de dados numérico, datas são representadas como tipo de dados *datetime*, e o restante como tipo de dados String.

3.2.2. Relação com este trabalho

O projeto criado por (SERRÃO, Carlos; DÍAZ, Vicente Aguilera; CERULLO, Fabio. 2009) mostra uma utilização prática para o projeto desenvolvido nesta monografia, no caso deste projeto correlato, os autores utilizaram o modelo de dados IDMEF para criar uma ontologia específica para o que necessitavam, utilizando de *web* semântica, porém, essa ontologia se assemelha muito com o IDMEF.

Com este trabalho, foram utilizados certos aspectos do IDMEF para criar uma certa segurança em uma aplicação *web*, não se restringindo apenas ao uso em redes de computadores, como acontece normalmente. Com esta visão, pode-se imaginar uma ampliação deste projeto para o futuro, visando não apenas redes de computadores.

3.3. Considerações finais sobre o capítulo

Muitos outros trabalhos foram utilizados como suporte e também estudos para o desenvolvimento desta monografia e da implementação da classe IDMEF-Message, porém não seria possível citar todos nesta sessão, pois estenderia demais o trabalho, no entanto, todos foram úteis, e ajudaram amplamente no resultado final deste trabalho, tanto em partes conceituais quanto práticas durante toda a etapa de desenvolvimento.

4. Implementação do modelo IDMEF como API Java

O projeto foi concebido para ser implementado em Java, por ser uma linguagem amplamente utilizada atualmente, por funcionar em sistemas operacionais distintos, e possuir orientação a objetos, afinal, o modelo de dados IDMEF foi criado para ser executado com este paradigma.

4.1. Desenvolvimento da aplicação principal.

Nesta etapa, foi realizado o desenvolvimento de todo o corpo da aplicação, nesta fase que foram criadas todas as classes do IDMEF (descritas no Capítulo 2), juntamente com seus métodos e funções.

Uma tela visual foi implementada para que fosse possível realizar testes na aplicação enquanto seu desenvolvimento ainda estava em andamento, e também para uma visão mais ampla sobre o projeto em si, desta forma, facilitando o entendimento e utilização do mesmo.

Dentro da tela principal citada acima, encontra-se campos de texto referentes a todos os atributos de todas as classes do modelo de dados, aglomerados entre seis abas. Há também três botões que podem ser acionados, estes botões que comandam todas as funções da aplicação.

As figuras abaixo apresentam todas as telas do programa, contendo as classes com seus respectivos atributos e botões.

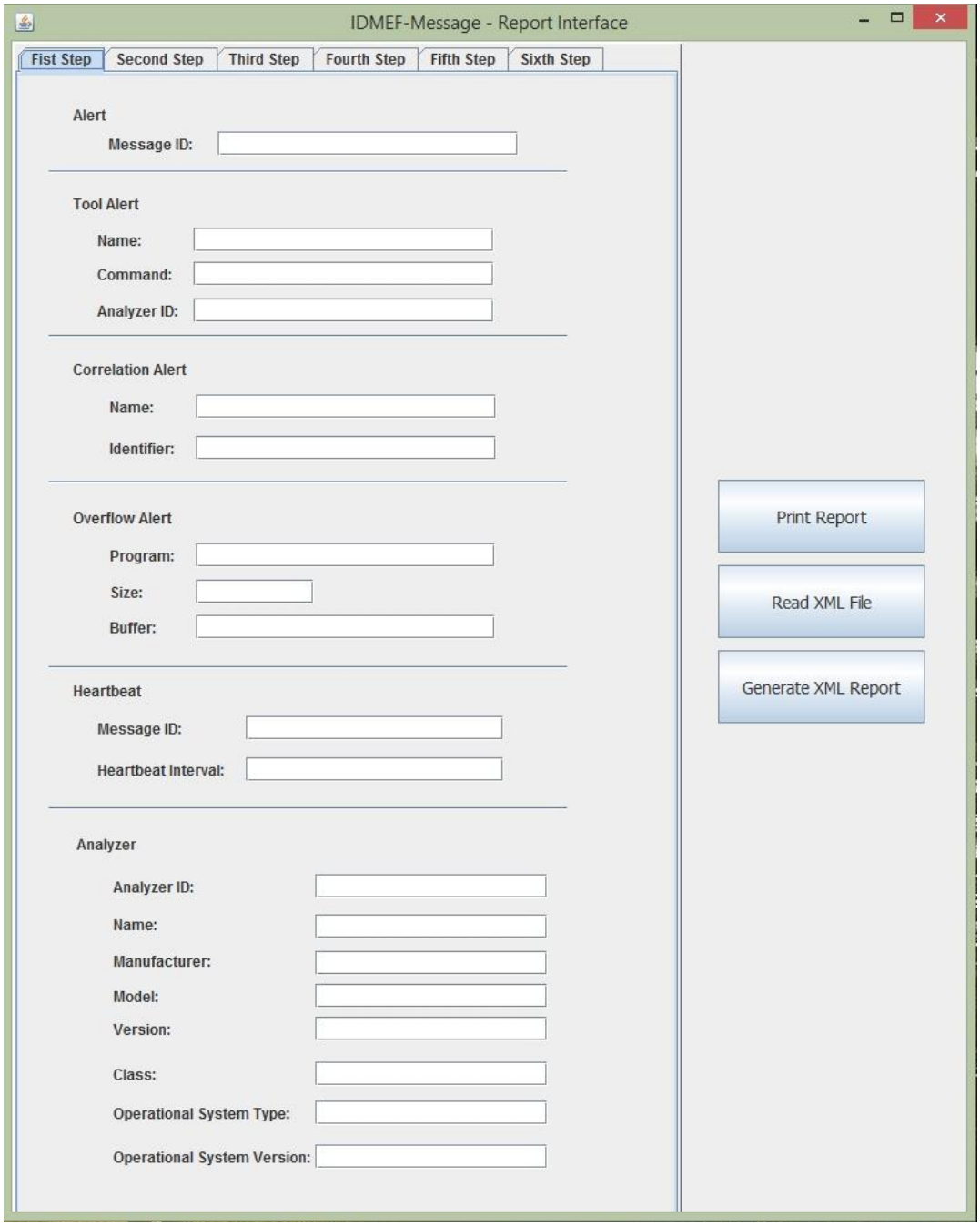

Figura 27: Aplicação IDMEF-Message, tela inicial para criação de relatório, apresentando conteúdo da primeira Aba

A Figura 27 acima apresenta a tela inicial da aplicação IDMEF-Message, contendo algumas classes e seus atributos, pode-se notar, no canto superior esquerdo, possuem seis abas enumeradas, cada uma destas é uma tela distinta, contendo suas respectivas classes e atributos, já ao lado direito, no centro, estão localizados os botões que ativam as funções do programa, porém, estes serão descritos a seguir. Os atributos alocados em caixas de seleção, garantem que apenas os valores permitidos sejam atribuídos, de acordo com o modelo de dados.

A Figura 28 abaixo apresenta as características e funções da segunda aba do programa, apresentando as classes Confidence, Reference, Node, Address, e User, juntamente com seus respectivos atributos.

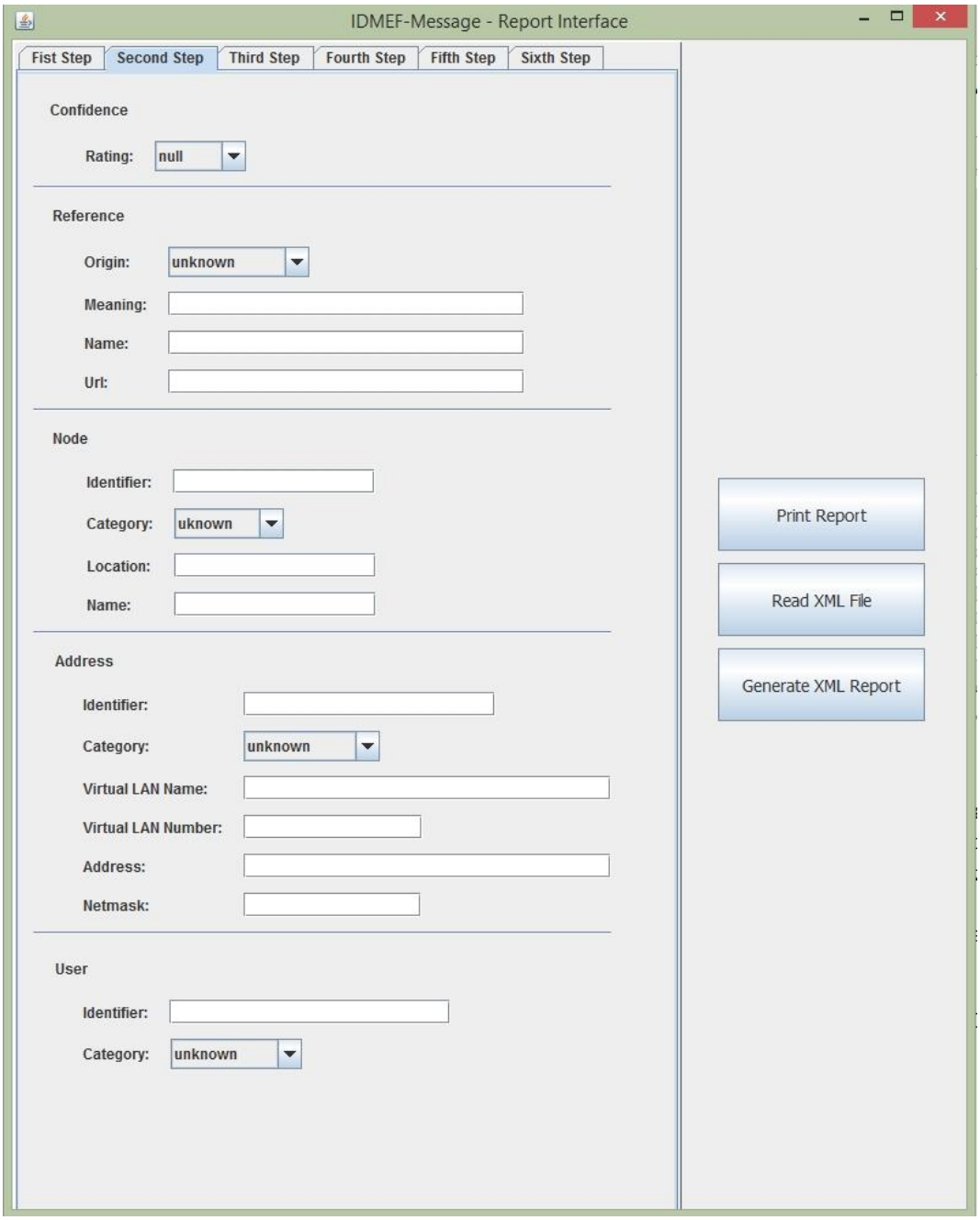

Figura 28: Aplicação IDMEF-Message, tela inicial para criação de relatório, apresentando conteúdo da segunda Aba

A Figura 29 apresenta as características da terceira aba da aplicação, contendo as classes UserId, Process e Service, contendo cada uma os campos referentes a seus atributos, para que possa ser realizado o preenchimento de acordo com as funções do programa.

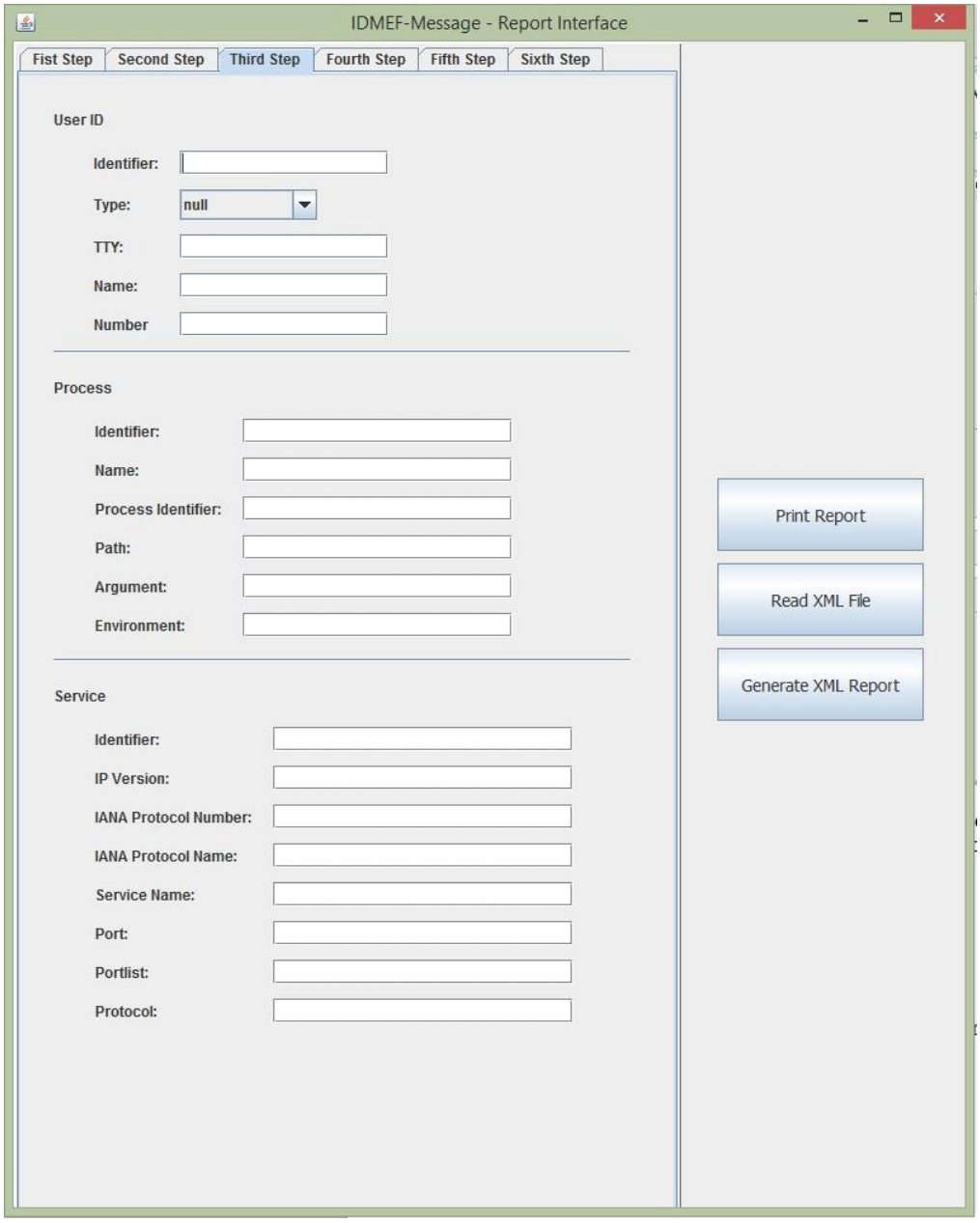

Figura 29: Aplicação IDMEF-Message, tela inicial para criação de relatório, apresentando conteúdo da terceira Aba

A Figura 30 apresenta as características da quarta aba do programa, contendo as classes WebService, SNMPService, e File, juntamente com os campos de seus respectivos atributos, permitindo seu preenchimento, conforme ocorre nas outras abas.

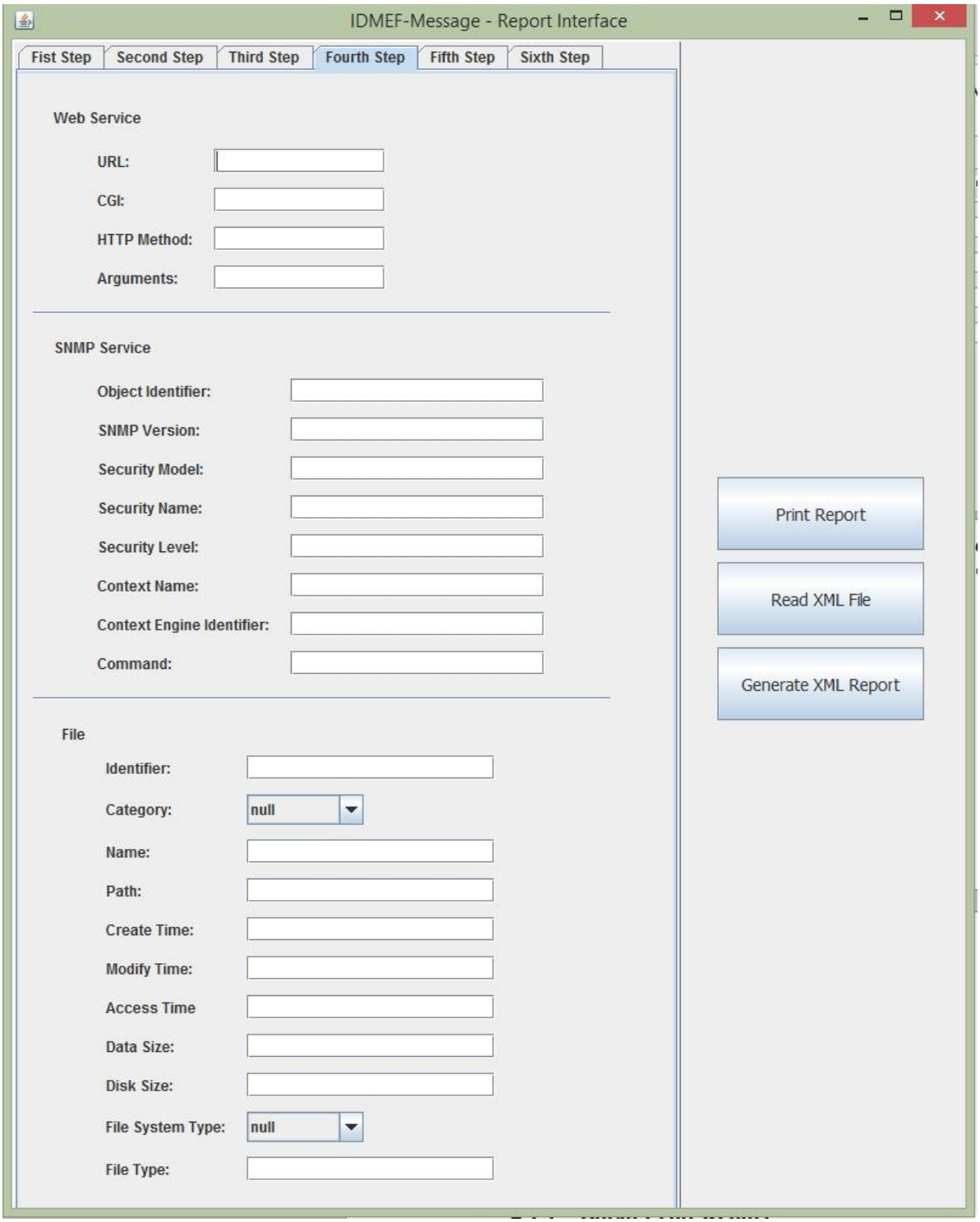

Figura 30: Aplicação IDMEF-Message, tela inicial para criação de relatório, apresentando conteúdo da quarta Aba

A Figura 30 apresenta as características da quinta aba do programa, contendo as classes Classification, Source, Target AdditionalData, Action e as TimeClasses CreateTime, DetectTime e AnalyzerTime, juntamente com os campos de seus respectivos atributos

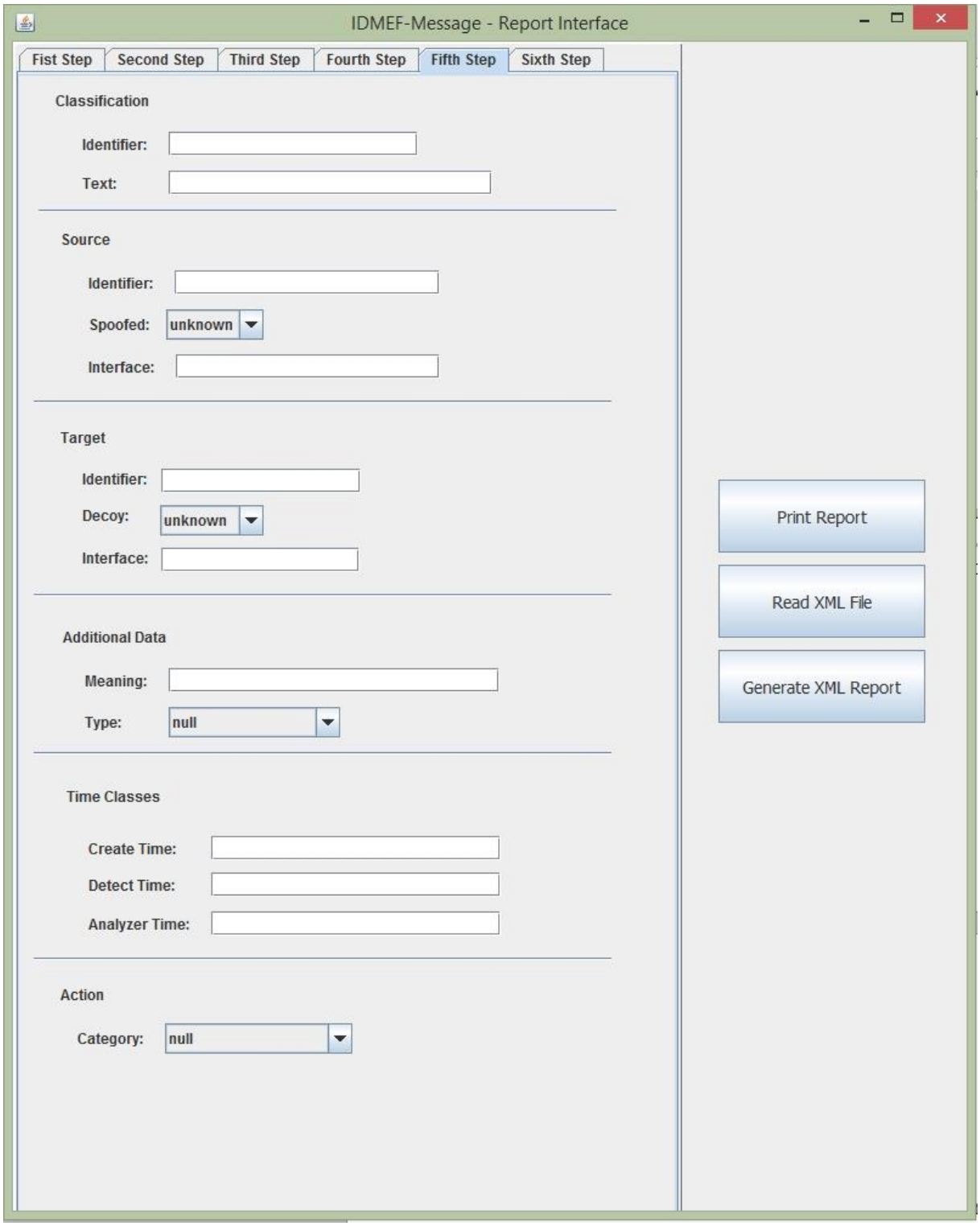

Figura 31: Aplicação IDMEF-Message, tela inicial para criação de relatório, apresentando conteúdo da quinta Aba

A sexta e última aba da aplicação é exibida na Figura 32, apresentando as classes FileAccess, Linkage, Inode, Checksum e Impact, juntamente com os atributos respectivos a cada classe.

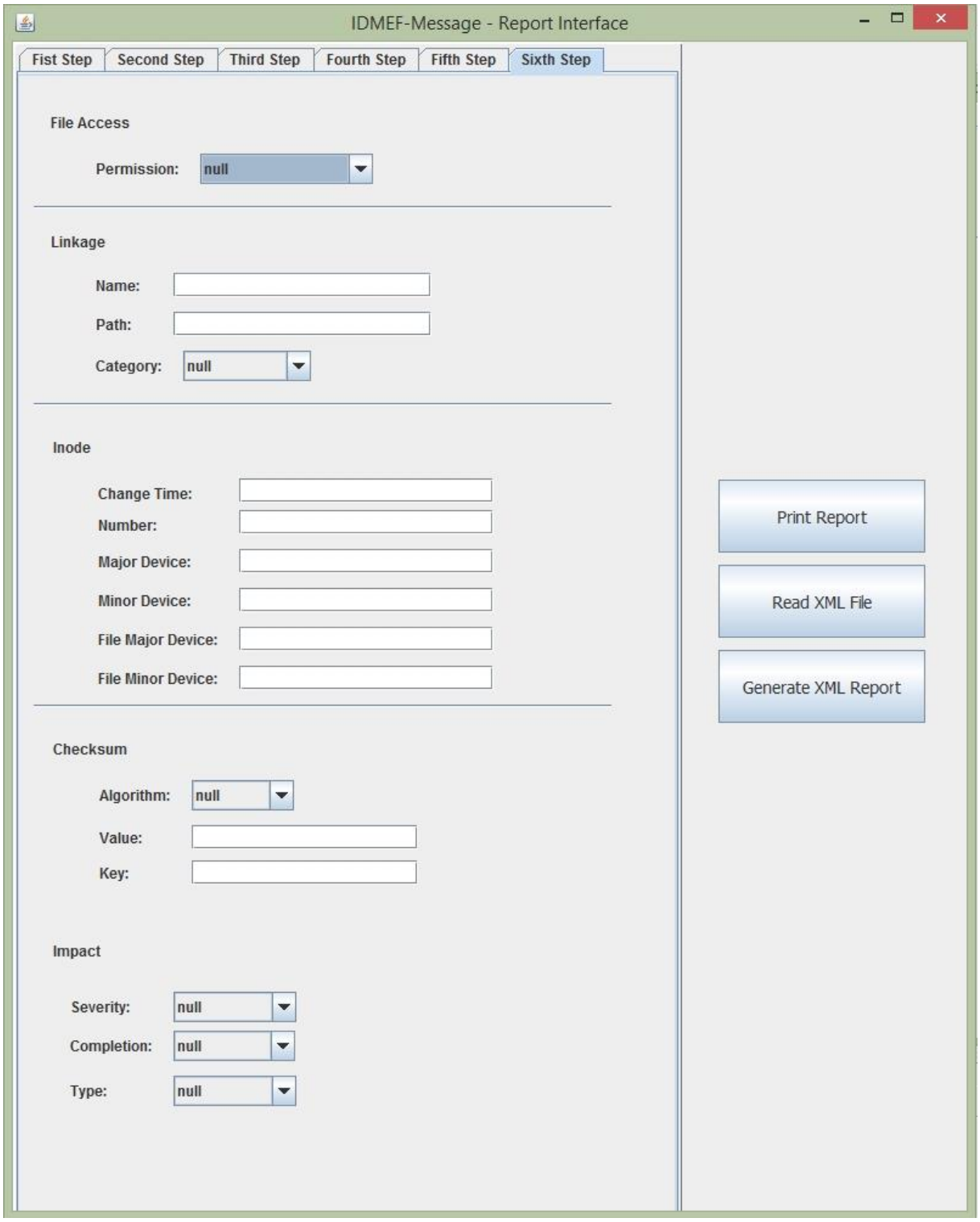

Figura 32: Aplicação IDMEF-Message, tela inicial para criação de relatório, apresentando conteúdo da sexta Aba

4.1.1. Botão Print Report

Este botão gera um relatório dentro da própria aplicação, a partir dos dados inseridos nos campos de texto.

4.1.2. Read XML File

Ao clicar neste botão, a aplicação abre uma tela de seleção de arquivo, onde deve ser escolhido um arquivo em XML, podendo ser um arquivo gerado por um IDS, para que o programa obtenha os dados deste e insira-os nos campos de texto da aplicação. O botão "*Read XML File*" nada mais é que um *parser* de um arquivo XML para um objeto Java específico, criado especialmente para converter os dados contidos no arquivo para as classes e atributos do IDMEF, ou seja, este conversor funciona inicialmente, apenas para o propósito do projeto, pois poderá não fornecer as conversões corretas caso seja utilizado como um *parser* XML genérico.

Quando sua função é ativada, a aplicação fornece ao utilizador a opção para selecionar um arquivo XML, quando este é escolhido, o programa abre o arquivo e faz uma leitura sobre ele, gerando vários *tokens*, que serão utilizados para identificar possíveis atributos ou classes. A partir do momento que uma classe é identificada, esta é adicionada a uma fila de classes, logo após, são localizados os atributos informados sobre esta classe, então são lidos e em seguida inseridos na tela principal da aplicação. Quando uma outra classe é encontrada, esta é novamente inserida na fila, porém, caso o programa encontre seu fechamento, por exemplo </idmef:ClasseExemplo> (conforme especificado pelo modelo de dados), esta é retirada da lista, desta forma, não haverá risco de um atributo que não seja pertencente à ela, seja atribuído erroneamente. Sempre que uma classe é inicializada, é inserida no fim da fila (caso a fila esteja vazia, é alocada na primeira posição), sendo assim, sempre que uma classe for removida da fila, ela estará alocada na última posição (exceto se for o único elemento da fila, pois assim obtém ao mesmo tempo a primeira e última posição), deste modo, quando a fila de classes estiver vazia, significa que todas as classes e atributos contidos no documento foram lidos e atribuídos a seus respectivos campos na aplicação.

Para que toda a função explicada anteriormente seja melhor entendida, será utilizado um pseudocódigo em XML, contendo duas classes com três atributos cada, conforme apresentado na Figura 33.

```
<idmef:ClasseExemplo1 atributo1=“abcd”>
        <idmef:atributo2>efgh</idmef:atributo2>
        <idmef:atributo3>bc524</idmef:atributo3>
</idmef:ClasseExemplo1>
<idmef:ClasseExemplo2 atributo4=“abc123”>
        <idmef:atributo5>def456</idmef:atributo5>
        <idmef:atributo6>ghi789</idmef:atributo6>
</idmef:ClasseExemplo2>
```

Figura 33: Exemplo de um pseudocódigo gerado a partir de um XML simples

Assim que o *parser* receber as informações contidas da Figura 33, o mesmo irá dividir o texto em vários *tokens*, como já foi descrito anteriormente, a Tabela 5 a seguir, mostra esta divisão dos *tokens* reconhecidos pela aplicação:

Tabela 5: Exemplo de separação de *tokens* realizada pelo *parser* em um arquivo XML

Token de Classe	Token de Atributo	Token de Valor
ClasseExemplo1	atributo1	abcd
ClasseExemplo1	atributo2	efgh
ClasseExemplo1	atributo3	bc524
ClasseExemplo2	atributo4	abc123
ClasseExemplo2	atributo5	def456
ClasseExemplo2	atributo6	ghi789

Tendo estas informações, o *parser* já é capaz de identificar exatamente em qual classe deve atribuir o valor do atributo em questão, utilizando os valores da Tabela 5, a aplicação iria atribuir a ClasseExemplo1, o valor “abcd” para o atributo “atributo1”, e desta forma, irá continuar a atribuir os valores para ClasseExemplo1 até que não haja mais nenhum atributo informado, ou que esta encontre seu fechamento, dado por “</idmef:ClasseExemplo1>”, então logo em seguida iniciaria as atribuições dos valores referentes a ClasseExemplo2, e assim por diante, até que todas as classes do documento XML fossem lidas.

4.1.3. Generate XML Report

Esta função, ao ser iniciada, recebe todos os dados dos campos de texto da aplicação, gerando um arquivo XML no padrão IDMEF, oferecendo a opção de escolher o nome do relatório gerado e também o local onde salvar o arquivo, para que este possa ser utilizado por um gerente, uma aplicação de tomada de decisão ou qualquer outro usuário ou programa que o deseje.

4.2. Funcionamento da aplicação

A aplicação como um todo, possui um modo de funcionamento muito simples para qualquer um que deseje utilizá-la. Esta mesma possui algumas funções úteis, porém bem simples de serem entendidas. Seu modo de operação pode ser entendido de acordo com a Figura 34 descreve seu modo de funcionamento através de um diagrama de atividade.

Com base neste diagrama, vê-se as formas de como a aplicação pode ser utilizada, onde ao ser inicializada, o programa oferece algumas opções ao utilizador, onde este poderá preencher manualmente os campos com os atributos respectivos a cada classe e atribuindo a cada um destes seu valor específico, como pode também, ao acionar o botão "*Read XML File*", escolher um arquivo XML para que a aplicação leia o documento e receba os valores estabelecidos pelo arquivo, poupando o tempo do usuário ou dispositivo que irá utilizar esta ferramenta.

Caso o utilizador opte por abrir um documento XML já com todas suas informações, ele poderá acrescentar informações adicionais fornecidas pela aplicação, que não foram estabelecidas no documento, ou até mesmo alterar informações fornecidas pelo arquivo.

Após todas as informações necessárias serem atribuídas, pode-se gerar um outro documento XML já com todas as novas especificações, e para que isso ocorra, basta o utilizador clicar no botão "*Generate XML Report*", pois logo em seguida, o programa irá atribuir todos os valores dos campos para suas respectivas classes e atributos, e em seguida, adicionando todos estes em um documento XML, e assim que esta tarefa estiver concluída, a aplicação apresenta ao usuário a opção de escolher o nome e local de destino para gerar o relatório criado.

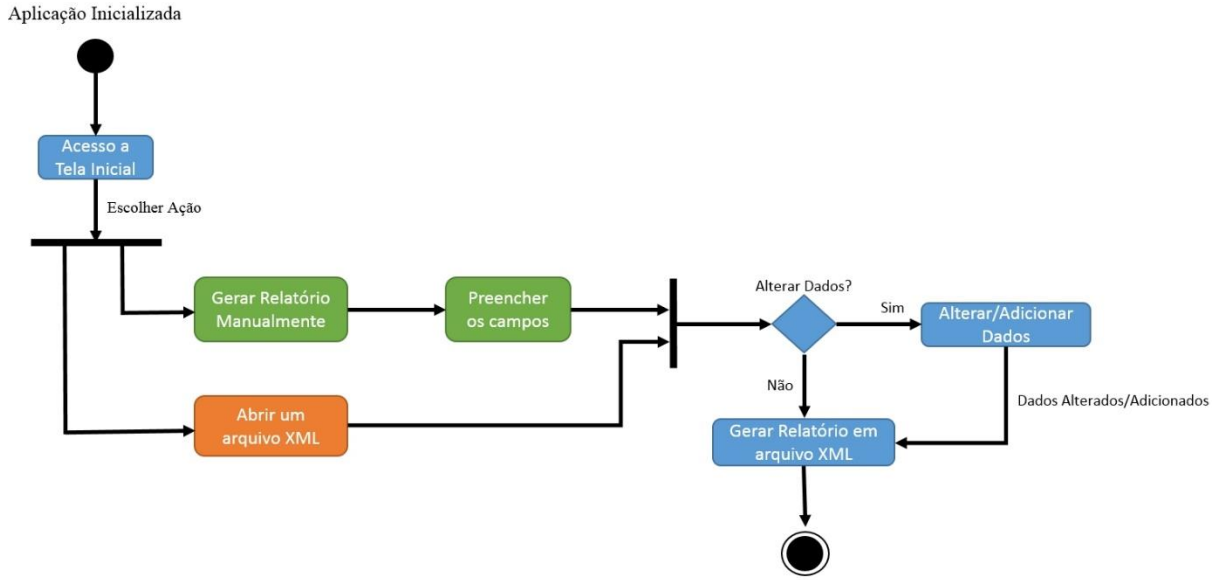

Figura 34: Diagrama de Atividade da aplicação IDMEF-Message

Utilizando o programa desta forma, o gerente, ou qualquer outro usuário que utilize esta ferramenta, poderá criar um relatório manual, digitando os valores dos atributos referentes às classes da forma que julgar necessário, ou até mesmo abrir um arquivo XML fornecido, obter seus dados e inseri-los na aplicação automaticamente, e caso queira, poderá alterar ou adicionar outros valores não informados manualmente antes de gerar um relatório final.

4.3. Ataques reconhecidos pela aplicação

Neste tópico serão descritos os possíveis tipos de ataque no qual a aplicação consegue gerar os relatórios a partir de dados fornecidos pelo IDS. Os ataques serão brevemente explicados, e um modelo de relatório para cada um destes será apresentado, porém, é necessário ressaltar, que esta forma a ser apresentada pode não representar a única, ou até mesmo a melhor forma de codificar estes possíveis ataques.

Tais modelos de ataque foram escolhidos pois os mesmos foram propostos na própria RFC e também demonstrados pela documentação original da proposta do modelo de dados do padrão IDMEF.

4.3.1. Ataque Teardrop

O *Teardrop Attack*, ou Ataque por Fragmentação, é um ataque de rede, por saturação, ou seja, negação de serviço, explorando o princípio de fragmentação do Protocolo IP. O protocolo IP fragmenta os pacotes em vários outros pacotes, tendo cada um destes seu número de sequência e identificação comuns. Quando são entregues ao destinatário, ele reagrupa os pacotes utilizando os valores de *offset* que cada um destes contém. O ataque de Teardrop busca inserir nos pacotes fragmentados, a informação de *offset* errada, para que assim, no momento em que o receptor for reagrupar estes pacotes, existam vazios ou sobreposições, provocando instabilidades no sistema. A Figura 35 demonstra uma detecção deste modelo de ataque feita pela aplicação:

```
<?xml version="1.0" encoding="UTF-8"?>
<idmef:IDMEF-Message version="1.0"xmlns:idmef="http://iana.org/idmef">
    <idmef:Alert messageid="abc123456789">
    </idmef:Alert>
    <idmef:Analyzer analyzerid="hq-dmz-analyzer01">
    </idmef:Analyzer>
    <idmef:Classification>
            <idmef:text>" Teardrop detected"</idmef:text>
    </idmef:Classification>
    <idmef:Source ident="a1b2c3d4">
    </idmef:Source>
    <idmef:Target ident="d1c2b3a4">
    </idmef:Target>
    <idmef:CreateTime createTime="2000-03-09T10:01:25.93464-05:00">
    </idmef:CreateTime>
    <idmef:Reference origin="bugtraqid">
            <idmef:name>"124"</idmef:name>
            <idmef:url>"http://www.securityfocus.com/bid/124"</idmef:url>
    </idmef:Reference>
    <idmef:Node ident="d1c2b3a4-001">
            <idmef:category>"dns"</idmef:category>
            <idmef:location>"Headquarters DMZ Network"</idmef:location>
            <idmef:name>"badguy.example.net"</idmef:name>
    </idmef:Node>
    <idmef:Address ident="a1b2c3d4-002">
            <idmef:category>"ipv4-addr-hex"</idmef:category>
            <idmef:address>"0xde796f70"</idmef:address>
            <idmef:netmask>"255.255.255.255"</idmef:netmask>
    </idmef:Address>
</idmef:IDMEF-Message>
```

Figura 35: Relatório gerado a partir de um ataque Teardrop

4.3.2. Ping of Death

Ping of Death, também conhecido como Ping da Morte, é um dos métodos mais antigos de ataques à rede conhecidos, seu funcionamento é bem simples, consiste basicamente em enviar um ping malicioso e malformado, ou seja, são solicitações de ping com pacotes muito elevados e em uma frequência muito alta, podendo ser milhares de vezes por segundo. Caso ocorra com sucesso, esta técnica pode acarretar em um mal funcionamento do sistema, podendo deixa-lo lento ou até mesmo travando o servidor em questão. A Figura 36 demonstra uma detecção deste modelo de ataque feita pela aplicação:

```
<?xml version="1.0" encoding="UTF-8"?>
<idmef:IDMEF-Message version="1.0"xmlns:idmef="http://iana.org/idmef">
    <idmef:Alert messageid="abc123456789">
    </idmef:Alert>
    <idmef:Analyzer analyzerid="bc-sensor01">
    </idmef:Analyzer>
    <idmef:Classification>
            <idmef:text>" Ping-of-death detected"</idmef:text>
    </idmef:Classification>
    <idmef:Source ident="a1a2">
            <idmef:spoofed>yes</idmef:spoofed>
    </idmef:Source>
    <idmef:Target ident="d7d8">
    </idmef:Target>
    <idmef:CreateTime createTime="000-03-09T10:01:25.93464Z">
    </idmef:CreateTime>
    <idmef:Reference origin="cve">
            <idmef:name>"CVE-1999-128"</idmef:name>
            <idmef:url>"http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-128"</idmef:url>
    <idmef:Node ident="d7d8-1">
            <idmef:category>"nisplus"</idmef:category>
            <idmef:location>"Cabinet B10"</idmef:location>
            <idmef:name>"Cisco.router.b10"</idmef:name>
    </idmef:Node>
    <idmef:Address ident="a1a2-2">
            <idmef:category>"ipv4-addr"</idmef:category>
            <idmef:address>"192.0.2.200"</idmef:address>
    </idmef:Address>
<idmef:IDMEF-Message>
```

Figura 36: Relatório gerado a partir de um ataque *Ping of Death*

4.3.3. Simple Port Scanning

Simple Port Scanning, consiste em um programa capaz de realizar uma varredura em um sistema a fim de encontrar portas abertas, podendo ser este sistema local ou remoto. Estas

portas abertas podem disponibilizar certos serviços, que caso o intruso consiga obter acesso, poderá atrapalhar ou até mesmo obstruir várias das funcionalidades do sistema. A Figura 37 demonstra uma detecção deste modelo de ataque feita pela aplicação:

```
<?xml version="1.0" encoding="UTF-8"?>
<idmef:IDMEF-Message version="1.0"xmlns:idmef="http://iana.org/idmef">
    <idmef:Alert messageid="abc123456789">
    </idmef:Alert>
    <idmef:Analyzer analyzerid="hq-dmz-analyzer62">
    </idmef:Analyzer>
    <idmef:Classification>
            <idmef:text>"Simple Portscan"</idmef:text>
    </idmef:Classification>
    <idmef:Source ident="abc01">
    </idmef:Source>
    <idmef:Target ident="def01">
    </idmef:Target>
    <idmef:CreateTime createTime="2000-03-09T15:31:00-08:00">
    </idmef:CreateTime>
    <idmef:Reference origin="vendor-specific">
            <idmef:name>"portscan"</idmef:name>
            <idmef:url>"http://www.vendor.com/portscan"</idmef:url>
    </idmef:Reference>
    <idmef:Node ident="def01-01">
            <idmef:category>"dns"</idmef:category>
            <idmef:location>"Headquarters Web Server"</idmef:location>
            <idmef:name>"www.example.com"</idmef:name>
    </idmef:Node>
    <idmef:Address ident="def01-02">
            <idmef:category>"ipv4-addr"</idmef:category>
            <idmef:address>"192.0.2.50"</idmef:address>
    </idmef:Address>
    <idmef:Service ident="def01-03">
            <idmef:portlist>"5-25,37,42,43,53,69-119,123-514"</idmef:portlist>
    </idmef:Service>
<idmef:IDMEF-Message>
```

Figura 37: Figura 62: Relatório gerado a partir de um ataque *Simple Port Scan*

5. Conclusões

Como destacado durante os capítulos iniciais deste projeto, o enorme crescimento do mundo computacional, e com o número crescente de usuários e dispositivos, está sendo cada vez mais difícil manter a segurança dos dados transmitidos por sistemas computacionais, pois mesmo que hajam estudos sobre segurança, afim de manter a integridade das informações, usuários mal intencionados também se esforçam para conseguir quebrar essas regras, ou seja, por mais seguro que seu sistema possa ser, ainda assim é impossível dizer que este é cem por cento seguro de invasões, ataques ou falhas. No entanto, existem muitas pesquisas que buscam melhorar cada vez mais a segurança, e este trabalho pode ser inserido entre elas. Como dito no primeiro capítulo, as IDSs não possuem um padrão para as mensagens de informação de intrusão (MII), impossibilitando que haja uma interoperabilidade entre elas, desta forma, o modelo de dados IDMEF busca criar este caminho para a comunicação entre sistemas de detecção distintos, desta forma, toda a proposta deste trabalho foca-se na implementação de uma API para este modelo de dados, buscando facilitar a distribuição desta interoperabilidade.

Durante os estágios iniciais para o desenvolvimento deste projeto, foram estipulados certos objetivos à serem alcançados, os quais contavam com um estudo aprofundado sobre o tema, pesquisa de trabalhos correlatos, a escolha de um modelo de padrão e MII, a implementação deste modelo, a geração de um ambiente de testes e a geração de relatórios, onde em sua grande maioria, foram compridos, com exceção do ambiente de testes, o qual não foi necessário ser criado, pois o modelo implementado para o projeto já poderia ser testado e validado a partir dos relatórios gerados pelos ataques.

A API criada em Java para a geração dos relatórios de intrusão funcionou corretamente, gerando os relatórios de ataques conforme especificado pelo modelo de dados, porém não foi possível testar todos os tipos de ataques no qual o IDMEF consegue relatar, pois não foi definido no escopo inicial do projeto, no entanto, como estipulado no trabalho inicial, eram pra ser realizados testes apenas com os modelos de ataque *Teardrop* e *Ping of Death*, mas foi possível também realizar testes com os ataques de *Simple Port Scan*, onde a aplicação gerou um relatório funcional sobre os três, validando assim suas detecções. Todos os relatórios gerados são em formato XML, contendo todas as especificações provenientes do modelo de dados IDMEF (Estes relatórios podem ser encontrados na seção 4.3 deste trabalho).

Mesmo o trabalho tendo sido finalizado durante o tempo hábil disponível e os objetivos iniciais foram cumpridos, o mesmo pode ser melhorado em trabalhos futuros, onde serão realizado mais testes e com outros modelos de ataque, poderá haver algumas alterações na estrutura do modelo de dados IDMEF, como visto em alguns exemplos na seção de trabalhos correlatos (seção 3.1), com a finalidade de melhorar o modo de detecção dos ataques, visando assim melhorar a eficiência do sistema. Assim que estes objetivos futuros para a API forem alcançados, este será o momento ideal para sua utilização em IDSs, pois assim os usuários poderão interligar este sistema juntamente com o sistema de detecção e proporcionar ao ambiente escolhido um IDS confiável, reconhecendo vários modelos de ataque e trabalhando com outros sistemas de segurança distintos.

O código fonte da aplicação IDMEF-Message será disponibilizado em repositórios de desenvolvedores para o desenvolvimento compartilhado, para que outros possam contribuir com a evolução deste projeto, tornando-o assim mais confiável sem desprover de sua integridade, visando cada vez mais sua integração e interoperabilidade com qualquer dispositivo de segurança utilizado.

Vale ressaltar, que mesmo a API tendo sida desenvolvida para *desktop* a mesma pode ser utilizada em aplicações web sem problema, pois para isso seria apenas acoplar a API na aplicação, utilizando suas funções da mesma forma como seria utilizado em uma aplicação *desktop*.

5.1. Dificuldades encontradas

Durante o desenvolvimento do projeto, a maior dificuldade encontrada foi a criação de um *parser* XML próprio, pois ao tentar utilizar modelos providos pelo Java, não consegui chegar nos resultados esperados por meio destas ferramentas, portanto foi decidido a implementação de um conversor de XML para objeto Java próprio para a API, onde foi necessário um estudo sobre os dados do arquivo XML, padrões de formatação e regras do documento XML.

Teve de ser realizado também um amplo estudo sobre IDSs e sobre o modelo de dados IDMEF, e por todos os textos, artigos e livros serem, em sua maioria, em inglês, tal fato gerou

certa dificuldade, não pela leitura em si, mas no tempo necessário para ser realizada a tradução dos documentos.

REFERÊNCIAS

ANDROULIDAKIS, G. Papavassiliou, S., **Improving network anomaly detection via selective flow-based sampling, Institution of Engineering and Technology (IET)**, Vol. 2, No. 3, 2008.

AXELSSON, Stefan. **Intrusion Detection Systems: A survey and Taxonomy**. Department of Computer Engineering, Chalmers University of Technology. Göteborg, Sweden. 2000.

BERNARDES, Mauro Cesar. **Avaliação do Uso de Agentes Móveis em Segurança Computacional**. 1991. 118 f. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional). Instituto de Ciências Matemáticas e de Computação (ICMC - USP). São Carlos, SP. 1999. Acessado em 26/04/2014. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-04022002-103542/pt-br.php >.

BURNETT, Steve; PAINE, Stephen. **Criptografia e Segurança: O guia oficial RSA**. Campus, 2002.

CROSBIE, Mark; SPAFFORD, Eugene H.; **Defending a computer system using autonomous agents**. Department of Computer Sciences, West Lafayetee, Purdue University, 1995.

CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. **The Intrusion Detection Message Exchange Format (IDMEF)**. Acessado em: 20/04/2014. Disponível em: <http://www.ietf.org/rfc/rfc4765.txt >.

FAGUNDES, Leonardo Lemes. **Metodologia para avaliação de Sistemas de Detecção de Intrusão**. São Leopoldo, Brasil, 2002.

FREDERICK, Karen Kent; NORTHCUTT, Stephen; RITCHEY, Ronald W.; WINTERS, Scott; ZELTSER, Lenny. **Desvendando segurança em redes**. Campus, 2002.

HAN, Ming; DEWU, Xu; WEI, Chen. **Improvement and implementation of IDMEF Data Model**. Zhejiang Normal University. Jinhua, China. 2010.

KEMMERER, Richard A.; VIGNA, Giovanni. **Intrusion Detection System: A Brief History and Overview**. Reliable Software Group, Computer Science Department, University of California Santa Barbara, 2002.

KUMAR, Sandep. **Classification and Detection of Computer Intrusions**. 1995. 180 f. Tese (Doutorado em Filosofia). Purdue University, 1995. Acessado em 15/03/2014. Disponível em: <https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/95-08.pdf>.

LIMA, Igor Vinícius Mussoi. **Uma abordagem simplificada de detecção de intrusão baseada em redes neurais artificiais**. Programa de pós graduação da Universidade Federal de Santa Catarina, Florianópolis, 2005. Acessado em 25/04/2014. Disponível em: <http://www.inf.ufsc.br/~bosco/grupo/MestradoIgor.pdf>.

MIIKA, T. **Intrusion detection systems**. Seminar on Distributed Security, Department of Computer Science: University of Helsinki, v. 01, n. 6, 2000.

MOURA, José Antão Beltrão; SUAVÉ, Jacques Philippe; TEIXEIRA JUNIOR, José Helvécio; TEIXEIRA, Suzana De Queiroz Ramos. **Redes de Computadores: Serviços, Administração e Segurança**. MakronBooks, 1999.

PEREIRA, Fábio Dacêncio. **Proposta e implementação de uma camada de integração de serviços de segurança (CISS) em SOC e multiplataforma**. 2009. 149 f. Escola Politécnica da Universidade de São Paulo, São Paulo, 2009. Acessado em 27/04/2014. Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3142/tde-18122009-124154/pt-br.php>.

PEREIRA, Hermano. **Sistema de detecção de intrusão para serviços web baseado em Anomalias**. Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica do Paraná, Curitiba, 2011. Acessado em 20/04/2014. Disponível em: <http://www.ppgia.pucpr.br/lib/exe/fetch.php?media=dissertacoes:2011:hermanopereira_oficial_v2.pdf>.

PINHEIRO, José Mauricio dos Santos. **Ameaças e ataques aos sistemas de informação: Prevenir e antecipar**. Caderno UniFOA, edição no. 5, 2007.

SALTZER, Jerome H.; SCHROEDER, Michael D.; **The Protection of Information in Computer Systems.** Fourth ACM Symposium on Operating System Principles. Department of Computer Science. University of Virginia, 1975.

SERRÃO, Carlos; DÍAZ, Vicente Aguilera; CERULLO, Fabio. **Web Application Security**. Madrid, Espanha. 2009.

STREBE, Matthew; PERKINS, Charles. **Firewalls 24seven**, 2 ed. Alameda: Sybex, 2002.

TANENBAUM, Andrew S. **Redes de computadores**. Rio de Janeiro: Campus, 2003.

APÊNDICE

APÊNDICE 1

1. Classes Time

O modelo de dados apresenta três classes para representação de tempo. Estas classes são elementos das classes Alert e Heartbeat, e são representadas conforme a Figura 38.

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

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

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

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

Figura 38: Código de representação das classes Time

(CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.1. Classe CreateTime

Esta classe é utilizada para indicar a data e hora no qual o *Alert* ou *Heartbeat* foi criado pelo analisador. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.2. Classe DetectTime

A classe DetectTime é utilizada para indicar a data e hora em que o(s) evento(s) produziu um alerta que foi detectado pelo analisador. No caso de mais de um evento, é a hora no qual o primeiro evento foi detectado. Este pode ou não ser a mesma data e hora que o

CreateTime, pois analisadores não precisam necessariamente emitir os alertas imediatamente após sua detecção. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.3. Classe AnalyzerTime

Esta classe é utilizada para indicar a data e hora atual do analisador. Seus valores podem ser preenchidos o mais tarde possível no processo de transmissão da mensagem. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

APÊNDICE 2

1. Classes Assessment

O modelo de dados provê três tipos de "*assessment*" (avaliação) em que o analisador pode criar sobre um evento. Estas classes são agregadas da classe Assessment. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.1. Classe Impact

A classe Impact é usada para fornecer uma avaliação do analisador do impacto do evento sobre o(s) alvo(s). É representado pelo modelo de dados conforme apresentado na Figura 39.

```
<!ENTITY % attvals.severity              "
      ( info | low | medium | high )
    ">
  <!ENTITY % attvals.completion          "
      ( failed | succeeded )
    ">
  <!ENTITY % attvals.impacttype          "
      ( admin | dos | file | recon | user | other )
    ">

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

Figura 39: Código de representação das classes Time

A classe Impact possui três atributos:

Severity (Gravidade): Uma estimativa relativa a gravidade do evento. Os valores permitidos para este atributo são descritos na Tabela 6.

Tabela 6: Valores permitidos para o atributo Severity, referente a classe Impact

Rank	Keyword	Descrição
0	Info	O alerta representa atividade informativa.
1	Low	Baixa gravidade.
2	Médium	Gravidade média.
3	High	Gravidade alta.

Completion (Conclusão): Uma indicação se o analisador acredita que a tentativa de que o evento descreve foi bem sucedida ou não. Os valores permitidos são apresentado na Tabela 7. Não há valor padrão.

Tabela 7: Valores permitidos para o atributo Completion da classe Impact

Rank	Keyword	Descrição
0	Failed	A tentativa não sucedida.
1	Succeeded	Tentativa sucedida.

Type (Tipo): O tipo de tentativa representada por este evento, em categorias relativamente amplas. Os valores permitidos são declarados na Tabela 8, juntamente com o valor padrão: "*other*".

Tabela 8: Valores permitidos para o atributo Type, referente a classe Impact

Rank	Keyword	Descrição
0	Admin	Realizada a tentativa ou conclusão de obter privilégios de administrador.
1	Dos	Realizada a tentativa ou conclusão de uma negação de serviço.
2	File	Realizada a tentativa ou conclusão de uma ação em um arquivo.
3	Recon	Realizada a tentativa ou conclusão de uma sonda de reconhecimento.
4	User	Realizada a tentativa ou conclusão de obter privilégios de usuário.
5	Other	Qualquer ação não descrita nas alternativas anteriores.

Todos os atributos descritos acima são opcionais. O elemento por si próprio pode ser vazio, ou conter uma descrição textual do impacto, caso o analisador for capaz de prover tais detalhes. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.2. Classe Action

A classe Action é utilizada para descrever quaisquer ações realizadas pelo analisador em resposta ao evento. É descrita pelo modelo de dados conforme a Figura 40 a seguir:

```
<!ENTITY % attvals.actioncat
    "( block-installed | notification-sent | taken-offline | other )">

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

Figura 40: Código de representação da classe Action

Esta classe possui um único atributo:

Category (Categoria): O tipo de ação tomada. Os valores permitidos serão descritos na Tabela 9. O valor padrão é "*other*".

Tabela 9: Valores permitidos para o atributo Category da classe Action

Rank	Keyword	Descrição
0	Block-installed	Um bloqueio de algum tipo foi instalado para evitar um ataque de atingir o seu destino. O bloqueio pode ser um bloqueio de porta, bloqueio de endereço, etc., ou desabilitar uma conta de usuário.
1	Notification-sent	Uma mensagem de notificação de algum tipo foi enviada fora da banda (via *pager*, *e-mail*, etc.). Não inclui a transmissão do alerta.
2	Taken-offline	Um sistema, computador ou usuário foi tirado do ar, como quando o computador é desligado ou um usuário está desconectado.
3	Other	Qualquer outra ação diferente das descritas acima.

O elemento por si próprio pode ser vazio, ou conter uma descrição textual da ação, caso o analisador for capaz de prover tais detalhes. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.3. Classe Confidence

A classe Confidence é utilizada para representar a melhor estimativa de validação das análises do analisador. É representada pelo modelo de dados como apresenta a Figura 41.

```
1       <!ENTITY % attvals.rating
2     "( low | medium | high | numeric )">
3
4       <!ELEMENT Confidence (#PCDATA) >
5       <!ATTLIST Confidence
6          rating %attvals.rating; 'numeric'
7          %attlist.global;
8        >
```

Figura 41: Código de representação da classe Confidence

A classe Confidence possui apenas um atributo:

Rating (Classificação): A classificação do analisador sobre sua validade analítica. O valor padrão para este atributo é "*numeric*". Os valores permitidos para este, serão descritos a seguir, na Tabela 10.

Tabela 10: Valores permitidos para o atributo Rating, referente a classe Confidence

Rank	Keyword	Descrição
0	Low	O analisador possui uma pequena confiança em validade.
1	Medium	O analisador possui uma confiança mediana em sua validade.
2	High	O analisador possui alta confiança em sua validade.
3	Numeric	O analisador apresentou um valor de probabilidade posterior indicando a sua confiança na sua validade.

Este atributo deve ser utilizado somente quando o analisador puder produzir informações significativas. Sistemas que podem ter como saída apenas uma heurística bruta deve usar "*low*", "*medium*" ou "*high*", como valor de classificação. Neste caso, o conteúdo do elemento deve ser omitido.

Sistemas capazes de produzir estimativas de probabilidade razoáveis devem usar "*numeric*" como o valor de classificação, e incluir um valor numérico de confiança no conteúdo do elemento. Este valor numérico deverá refletir uma probabilidade posterior (a probabilidade de que um ataque ocorreu com base nos dados observados pelo sistema de detecção e o modelo usado pelo sistema). É um número de ponto flutuante entre 0.0 e 1.0. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

APÊNDICE 3

1. Classes Support

As classes Support, ou classes de apoio, compõem as classes mais importantes entre as classes *Core*, e são compartilhadas entre eles. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.1. Classe Reference

Esta classe fornece o "*name*" de um alerta, ou outra informação relevante que permita ao gerente entender o que está acontecendo.

Podemos ver o funcionamento desta classe na Figura 42, a seguir:

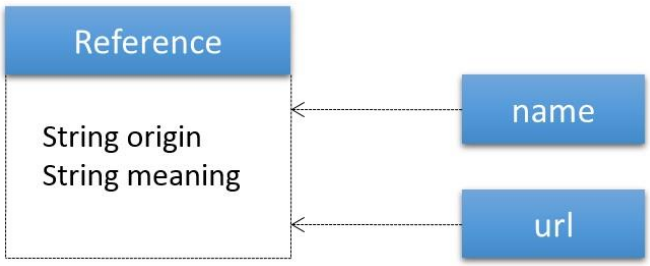

Figura 42: Classe Reference

As classes agregadas que compõem Reference são:

Name (Nome): Exatamente um. O nome do alerta, de uma das origens.

URL: Exatamente um, tipo String. A URL (*Uniform Resource Locator*) no qual o gerente (este podendo ser um operador humano ou uma máquina) poderá encontrar informações adicionais sobre o alerta. O documento indicado pela URL pode incluir uma descrição aprofundada sobre o ataque, contramedidas apropriadas, ou qualquer outra informação considerada relevante pelo vendedor.

Na Figura 43 pode-se ver o código de representação desta classe.

```
1  <!ENTITY % attvals.origin                "
2        ( unknown | vendor-specific | user-specific | bugtraqid | cve | osvdb )">
3
4     <!ELEMENT Reference(name, url)>
5     <!ATTLIST Reference
6        origin %attvals.origin; 'unknown'
7        meaning CDATA #IMPLIED>
```

Figura 43: Código de representação da Classe Reference.

A classe Reference possui dois atributos, que serão descritos a seguir:

Origin (Origem): Obrigatório. A fonte que originou o nome do alerta. Os valores permitidos para este atributo serão listados na Tabela 11 a seguir. O valor padrão deste é "*unknown*".

Tabela 11: Valores permitidos para o atributo origin, referente a classe Reference

Rank	Keyword	Descrição
0	Unknown	A origem do nome não é conhecida.
1	Vendor-specific	Um nome específico do fornecedor (e, portanto, a URL); este pode ser utilizado para fornecer informações específicas do produto.
2	User-specific	Um nome específico do usuário (e, portanto, a URL); este pode ser utilizado para fornecer informações específicas de instalação.
3	Bugtraqid	O identificador de vulnerabilidade SecurityFocus ("Bugtraq") do banco de dados (http://www.securityfocus.com/bid)
4	Cve	O nome de vulnerabilidades e exposições comuns (CVE) (http://www.cve.mitre.org/)
5	Osvdb	Banco de Dados de vulnerabilidade *Open Source* (http://www.osvdb.org)

Meaning (Significado): Opcional. O significado da referência, tal como entendido pelo provedor do alerta. Este campo só é válido se o valor do atributo <*Origin*> for definido como "*vendor-specific*" ou "*user-specific*". (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.2. Classe Node

A classe node é utilizada para identificar hosts e outros dispositivos de rede, tais como: roteadores, *switches* e etc.

Esta classe é composta por outras três classes agregadas, como mostrado na Figura 44, a seguir:

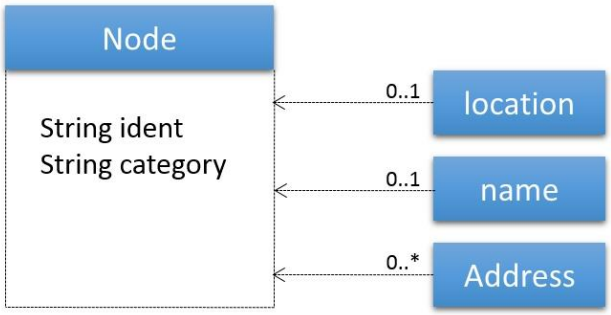

Figura 44: Classe Node

As classes que a compõem são:

Location (Localização): Zero ou um, atributo tipo String. A localização do equipamento.

Name (Nome): Zero ou um, atributo tipo String. O nome do equipamento. Esta informação deve ser informada caso não sejam dadas informações sobre o *address*.

Address (Endereço): Zero ou mais. O endereço de rede ou de *hardware* do equipamento. A menos que um "*name*" (acima) seja fornecido, pelo menos um endereço deve ser especificado.

Esta classe pode ser representada pelo modelo de dados como mostra a Figura 45.

```
1   <!ENTITY % attvals.nodecat
2       "( unknown | ads | afs | coda | dfs | dns | hosts | kerberos |
3           nds | nis | nisplus | nt | wfw )">
4
5   <!ELEMENT Node (location?, (name | Address), Address*)>
6   <!ATTLIST Node
7       ident CDATA '0'
8       category %attvals.nodecat; 'unknown'
9       %attlist.global;
10    >
```

Figura 45: Código de representação da Classe Node

A classe Node possui dois atributos:

Ident (Identificador): Opcional. Um único identificador para o nó (*node*).

Category (Categoria): Opcional. O "domínio" a partir do qual as informações do name foram obtidas, caso seja relevante. Os valores permitidos para este atributo serão descritos na Tabela 12. O valor padrão para este atributo é "*unknown*".

Tabela 12: Valores permitidos para o atributo Category, referente a classe Node

Rank	Keyword	Descrição
0	Unknown	Domínio desconhecido ou irrelevante.
1	Ads	Windows 2000 Advanced Directory Services.
2	Afs	Andrew File System (Transarc).
3	Coda	Coda Distributed File System.
4	Dfs	Distributed File System (IBM).
5	Dns	Domain Name System.
6	Hosts	Local hosts File.
7	Kerberos	Kerberos realm.
8	Nds	Novell Directory Services.
9	Nis	Network Information Services (Sun).
10	Nisplus	Network Information Services Plus (Sun).
11	Nt	Windows NT domain.
12	Wfw	Windows for Workgroups.

(CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.2.1. Classe Address.

Esta classe é utilizada para representar os endereços de rede, *hardware* e aplicações. Pode-se ver sua composição na Figura 46, a seguir:

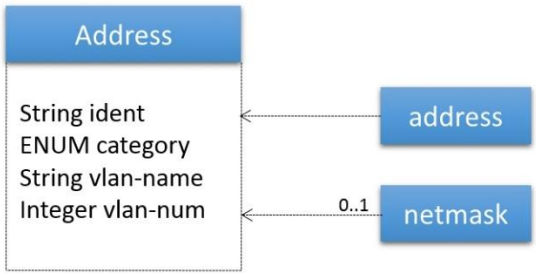

Figura 46: Classe Address

As classes que compõem Address são:

Address (Endereço): Exatamente um, tipo String. O endereço da informação.

Netmask (Máscara de Rede): Zero ou um, tipo String. A máscara de rede do endereço, se necessário.

Esta classe é representada como descrito na Figura 47:

```
<!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 )">

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

Figura 47: Código de representação da Classe Address

Address possui quatro atributos, que serão expostos a seguir:

Ident (Identificador): Opcional. Um identificador único para o endereço.

Category (Categoria): Opcional. O tipo do endereço apresentado, onde seus valores permitidos serão descritos na Tabela 13. O valor padrão para estre atributo é "*unknown*".

Tabela 13: Valores permitidos para o atributo Category, referente a classe Address

Rank	Keyword	Descrição
0	Unknown	Tipo de endereço desconhecido.
1	Atm	Asynchronous Transfer Mode network address
2	E-mail	Electronic mail address (RFC 2822 [12]).
3	Lotus-notes	Lotus Notes e-mail Address.
4	Mac	Media Access Control (MAC) address.
5	Sna	IBM Shared Network Architecture (SNA) address.
6	Vm	IBM VM ("PROFS") e-mail address.
7	Ipv4-addr	Endereço de host ipv4 em decimal pontuado (a.b.c.d).
8	Ipv4-addr-hex	Endereço de host ipv4 em notação hexadecimal.
9	Ipv4-net	Endereço de rede ipv4 em notação decimal pontuada, com bits mais significativos (a.b.c.d/nn).

10	Ipv4-net-mask	Endereço de rede ipv4 em notação decimal pontuada, com máscara de rede no mesmo padrão (a.b.c.d/w.x.y.z).
11	Ipv6-addr	Endereço de host ipv6.
12	Ipv6-addr-hex	Endereço de host ipv6 em notação hexadecimal.
13	Ipv6-net	Endereço de rede ipv6, com bits mais significativos.
14	Ipv6-net-mask	Endereço de rede ipv6 com máscara de rede.

VLan-Name: Opcional. O nome da *Virtual LAN* (LAN - *Local Area Network*) à qual o endereço pertence.

VLan-Num: Opcional. O número da *Virtual LAN* à qual o endereço pertence. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.3. Classe User.

Esta classe é utilizada para descrever os usuários. É utilizado principalmente como uma classe de "recipiente" para a classe agregada UserId, como mostrado na Figura 48.

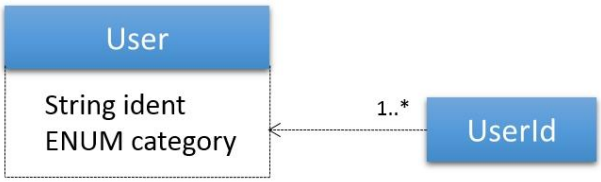

Figura 48: Classe User

As classes contidas em User são e sua representação são mostradas abaixo:

UserId: Um ou mais. Identificadores do usuário.

```
<!ENTITY % attvals.usercat
    "( unknown | application | os-device )" >

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

Figura 49: Código de Representação da classe User

Esta classe possui apenas dois atributos:

Ident (Identificador): Opcional. Um identificador único para o usuário.

Category (Categoria): Opcional. O tipo de usuário representado. O valor padrão para este atributo é "*unknown*", os outros valores permitidos para este serão descritos na Tabela 14 a seguir:

Tabela 14: Valores permitidos para o atributo Category, referente à classe User

Rank	Keyword	Descrição
0	Unknown	Tipo de usuário desconhecido.
1	Application	Um usuário da aplicação.
2	Os-device	Um Sistema Operacional ou usuário do dispositivo.

(CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.3.1. Classe UserId

A classe UserId é usada para especificar informações sobre os usuários. A classe User pode conter mais de um UserId para indicar tentativas de transições de um usuário para outro, ou para fornecer informações mais completas sobre privilégios de usuários (ou processos).

A composição desta classe pode ser vista na Figura 50, a seguir:

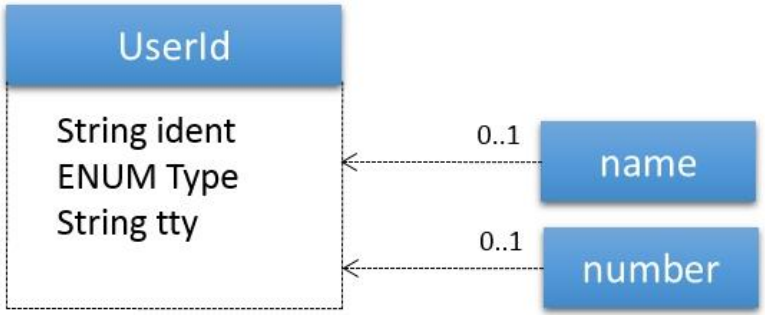

Figura 50: Classe UserId

As classes que compõem esta são:

Name (Nome): Zero ou um, tipo String. O nome de um usuário ou grupo.

Number (Número): Zero ou um, tipo Integer. O número de um usuário ou grupo.

A classe é representada pelo modelo de dados na Figura 51:

```
<!ENTITY % attvals.idtype
    "( current-user | original-user | target-user | user-privs |
       current-group | group-privs | other-privs )">

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

Figura 51: Código de representação da classe UserId

Esta classe possui três atributos, que serão descritos abaixo:

Ident (Identificador): Opcional. Um identificador único para o Id do Usuário.

Type (Tipo): Opcional. O tipo de informação do usuário representado. O valor padrão para este atributo é "*original-user*", os outros valores permitidos para este, são descritos na Tabela 15.

Tabela 15: Valores permitidos para o atributo Type, referente a classe UserId

Rank	Keyword	Descrição
0	Current-user	O Id atual sendo utilizado pelo usuário ou processo. Em sistemas Unix, este pode ser o id de usuário "real", em geral.
1	Original-user	A identidade real do usuário ou processo reportado. Em sistemas que (a) realizam algum tipo de auditoria e (b) apoiam a extração do Id do usuário pelo *token* "*audit* id", este valor deve ser utilizado. Em sistemas no qual esta tarefa não é suportada, e onde o usuário está logado no sistema, o "*login* id" deve ser usado.
2	Target-user	O Id em que o usuário ou processo está tentando se tornar. Isto se aplica em sistemas Unix, por exemplo, quando o usuário tenta utilizar "su", "rlogin", "telnet", etc.

3	User-privs	Outro Id de usuário no qual o usuário ou processo tenha a habilidade de utilizar, ou um Id de usuário associado à uma permissão de um arquivo.
4	Current-group	O Id de grupo atual sendo utilizado pelo processo ou pelo usuário.
5	Group-privs	Outro Id de grupo no qual o usuário ou processo tenha a habilidade de utilizar, ou um Id de grupo associado à uma permissão de um arquivo.
6	Other-privs	Não usado em um usuário, grupo ou contexto de processo, utilizado somente no contexto de arquivo. As permissões de arquivos atribuídos a usuários que não correspondem tanto às permissões do usuário ou grupo.

TTY (Tele Typewriter): Opcional, tipo String. O TTY que o usuário está utilizando. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.4. Classe Process

Esta classe é utilizada para descrever processos sendo executados nas fontes, alvos ou analisadores. Sua composição pode ser vista na Figura 52.

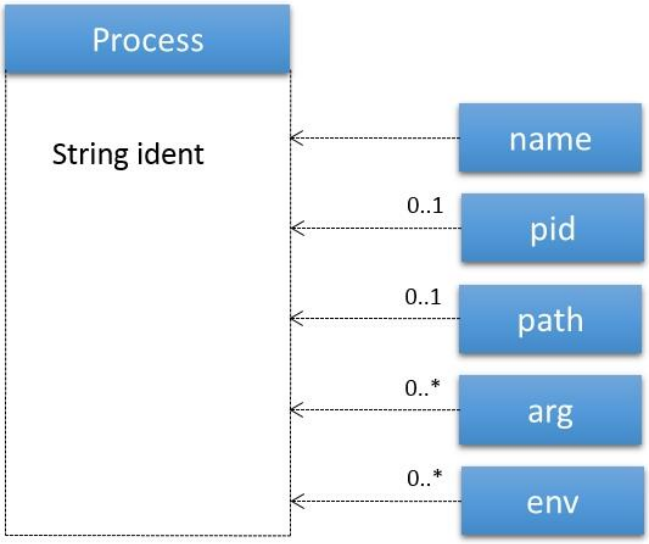

Figura 52: Classe Process

As classes que compõem Process são:

Name (Nome): Exatamente um, tipo String. O nome do programa sendo executado. Este deve ser um nome curto; *path* e informações de argumentos devem ser informadas em outro lugar.

Pid: Zero ou mais, tipo Integer. O identificador de processos do processo.

Path (Caminho): Zero ou mais, tipo String. O caminho completo do programa em execução.

Arg (Argumento): Zero ou mais, tipo String. Um argumento de linha de comando para o programa. Vários argumentos podem ser especificados (eles são assumidos como tendo ocorrido na mesma ordem em que são fornecidos), com múltiplos usos da *arg*.

Env (Ambiente): Zero ou mais, tipo String. Uma String de ambiente associada ao processo; geralmente no formato "Variável=valor". Múltiplas Strings podem ser especificadas com múltiplos usos do ambiente.

Esta classe é representada no modelo de dados como apresentado na Figura 53.

```
1   <!ELEMENT Process (
2           name, pid?, path?, arg*, env*
3       )>
4
5   <!ATTLIST Process
6           ident CDATA '0'
7           %attlist.global;
8       >
```

Figura 53: Código de representação da classe Process

A classe Process possui um único atributo:

Ident (Identificador): Opcional. Um único identificador para o processo. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.5. Classe Service

A classe Service descreve os serviços de rede de origens e destinos. Ele pode identificar serviços pelo nome, porta e protocolo. Quando o serviço ocorre como uma classe agregada de Source, entende-se que o serviço é aquele do qual atividade de interesse é originário; e que o serviço está "ligado" às informações do Node, Process User, também contida

em Source. Da mesma forma, quando o Service ocorre como uma classe de agregados de Target, entende-se que o serviço é aquele para o qual a atividade de interesse está a ser direcionada; e que o serviço está "ligado" às informações de Node, Process e User, também contida em Target. Se Service ocorre tanto em Source e Target, as informações em ambos os locais devem ser a mesma. Se a informação é a mesma em ambos os locais e os implementadores desejam deixa-la em apenas um local, devem especificá-la como um agregado da classe Target.

Pode-se ver a composição da classe Service na Figura 54, a seguir:

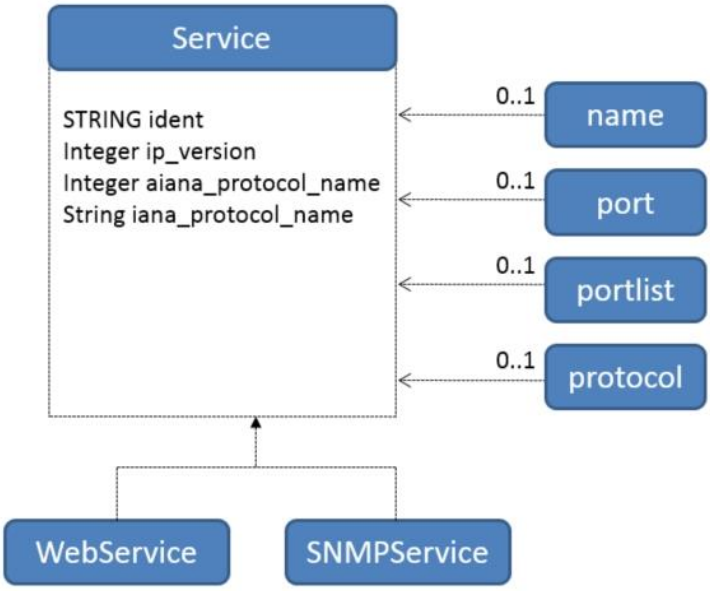

Figura 54: Classe Service

As classes agregadas que compõem Service são:

Name (Nome): Zero ou um, tipo String. O nome do serviço. Sempre que possível, o nome da lista de portas IANA conhecidas deve ser usado.

Port (Porta): Zero ou um, tipo Integer. O número da porta sendo utilizada.

Portlist (Lista de Portas): Zero ou um, tipo Portlist. Uma lista com as portas sendo utilizadas. Se uma *Portlist* é dada, o iana_protocol_number e iana_protocol_name deve se aplicar a todos os elementos da lista.

Protocol (Protocolo): Zero ou um, tipo String. Informações adicionais sobre o protocolo utilizado. A função deste campo é conter informações relacionadas ao protocolo sendo utilizado, quando forem preenchidos os atributos iana_protocol_number e/ou iana_protocol_name da classe Service.

Um serviço deve ser especificado como um nome ou uma porta, ou uma *portlist*. O protocolo é opcional em todos os casos, mas não são permitidas outras combinações.

A classe Service é representada pelo modelo de dados conforme a Figura 55.

```
<!ELEMENT Service(
        (((name, port?) | (port, name?)) | portlist), protocol?,
        SNMPService?, WebService?)>

  <!ATTLIST Service
        ident CDATA '0'
        ip_version CDATA #IMPLIED
        iana_protocol_number CDATA #IMPLIED
        iana_protocol_name CDATA #IMPLIED
        %attlist.global;
   >
```

Figura 55: Código de representação da classe Service

Esta classe possui quatro atributos, que serão descritos a seguir:

Ident (Identificador): Opcional. Um identificador único para o serviço.

Ip_Version (Versão do IP): Opcional, tipo Integer. O número da versão do IP.

IANA_Protocol_Number: Opcional, tipo Integer. O número do protocolo IANA (IANA - *Internet Assigned Numbers Authority*).

IANA_Protocol_Name: Opcional, tipo String. O nome do protocolo IANA. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.5.1. Classe WebService.

A classe WebService transmite informações adicionais relacionadas ao tráfego *web*. Esta é composta por quatro classes agregadas, como visto na Figura 56, a seguir:

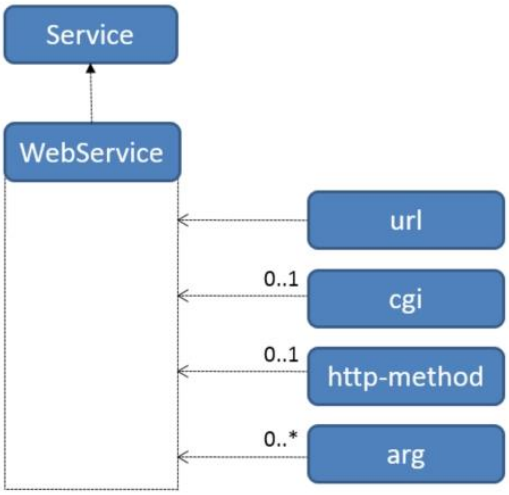

Figura 56: Classe WebService

As classes agregadas que compõem WebService são:

URL: Exatamente um, tipo String. A url da requisição.

CGI: Zero ou um, tipo String. O script CGI (*Common Gateway Interface*) da requisição, sem argumentos.

Http-Method: Zero ou um, tipo String. O método HTTP (*put*, *get*) usado na requisição.

Arg (Argumento): Zero ou mais, tipo String. Os argumentos do script CGI.

Esta classe é representada pelo modelo de dados na Figura 57:

(CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

```
1   <!ELEMENT WebService(
2           url, cgi?, http-method?, arg*)>
3       <!ATTLIST WebService
4           %attlist.global;
5       >
```

Figura 57: Código de representação da classe WebService

1.5.2. Classe SNMPService.

A classe SNMPService transmite informações adicionais referente ao tráfego SNMP (*Simple Network Management Protocol*). Esta classe é composta por oito classes agregadas, como mostrado na Figura 58:

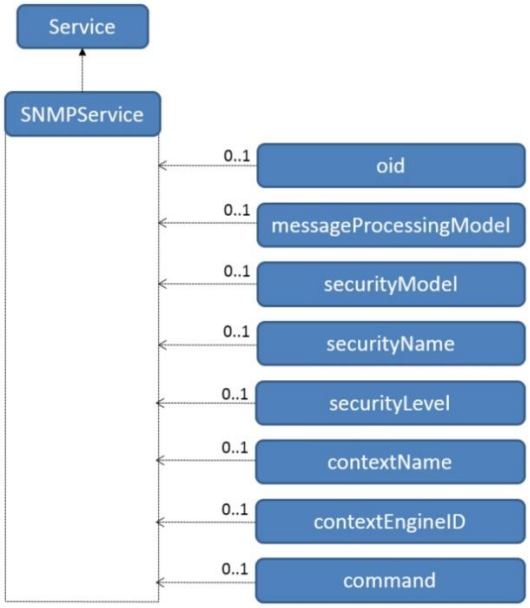

Figura 58: Classe SNMPService

As classes agregadas que compõem esta são:

Oid: Zero ou um, tipo String. O identificador do objeto na requisição.

MessageProcessingModel (Modelo de Processamento de Mensagem): Zero ou um, tipo Integer. A versão do SNMP, tipicamente 0 para SNMPv1, 1 para SNMPv2u e SNMPv2*, e 3 para SNMPv3.

SecurityModel (Modelo de Segurança): Zero ou um, tipo Integer. A identificação do modelo de segurança em uso, normalmente sendo 0 para qualquer, 1 para SNMPv1, 2 para SNMPv2c e 3 para USM.

SecurityName (Nome de Segurança): Zero ou um, tipo String. O nome do objeto de segurança.

SecurityLevel (Nível de Segurança): Zero ou um, tipo Integer. O nível de segurança da requisição SNMP.

ContextName (Nome do Contexto): Zero ou um, tipo String. O nome do contexto do objeto.

ContextEngineID: Zero ou um, tipo String. Identificador do mecanismo de contexto do objeto.

Command (Comando): Zero ou um, tipo String. O comando enviado ao servidor SNMP (GET, SET, etc.). (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

A classe é representada pelo modelo de dados como descrito na Figura 59, abaixo:

```
<!ELEMENT SNMPService(
        oid?, messageProcessingModel?, securityModel?, securityName?,
        securityLevel?, contextName?, contextEngineID?, command?
      )>
    <!ATTLIST SNMPService
        %attlist.global;
    >
```

Figura 59: Código de representação da classe SNMPService

1.6. Classe File

A classe File fornece informações específicas sobre um arquivo ou outro objeto parecido que foi criado, deletado, ou modificado no alvo. Esta descrição pode prover tanto as configurações do arquivo no momento do evento como antes do evento, conforme especificado utilizando o atributo "*category*".

Sua composição pode ser observada na Figura 60, a seguir:

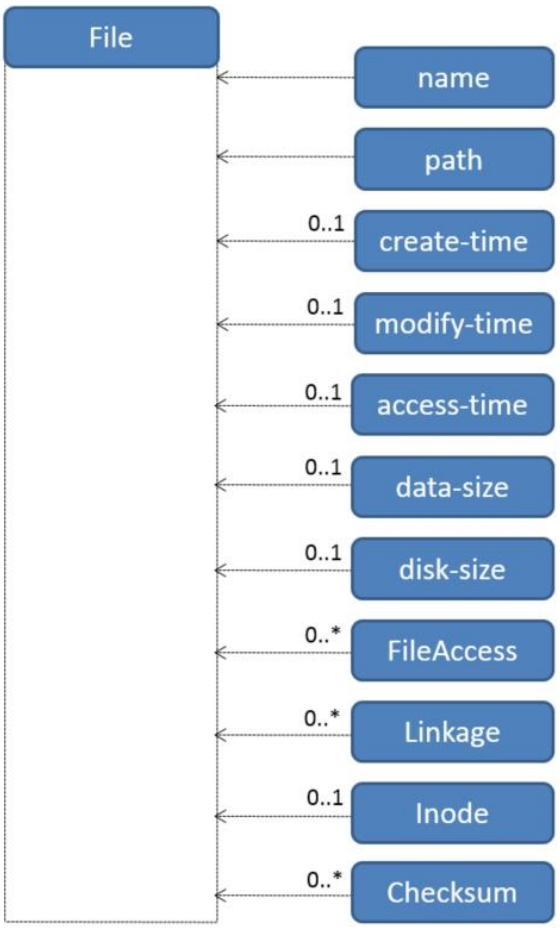

Figura 60: Classe File

As classes agregadas que compõem File são:

Name (Nome): Exatamente um, tipo String. O nome do arquivo no qual o alerta se aplica, não inclui o caminho para o arquivo.

Path (Caminho): Exatamente um, tipo String. O caminho completo do arquivo, incluindo seu nome. O nome do caminho deve ser representado da forma mais "universal" possível, para facilitar o processamento do alerta.

Create-Time: Zero ou um, tipo Datetime. A data em que o arquivo foi criado.

Modify-Time: Zero ou um, tipo Datetime. A data em que o arquivo foi modificado.

Acess-Time: Zero ou um, tipo Datetime. A data no qual o arquivo foi acessado pela última vez.

Data-Size: Zero ou um, tipo Integer. O tamanho dos dados, em *bytes*.

Disk-Size: Zero ou mais, tipo Integer. O espaço físico no disco consumido pelo arquivo, em *bytes*.

FileAccess: Zero ou mais. Permissões de acesso ao arquivo.

Linkage: Zero ou mais. Sistema de arquivos objetos para que este arquivo é vinculado (outras referências para o arquivo).

Inode: Zero ou mais. Informações de *inode* do arquivo (Relevante para sistemas Unix).

Checksum: Zero ou mais. Informações de verificação do arquivo.

Esta classe pode ser representada pelo modelo de dados como mostrado na Figura 61.

```
<!ENTITY % attvals.filecat "( current | original )">

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

Figura 61: Código de representação da classe File

Esta classe possui quatro atributos, dentre eles um obrigatório e três opcionais:

Ident (Identificador): Opcional. Um identificador único para este arquivo.

Category (Categoria): Obrigatório. O contexto da informação fornecida. Os valores permitidos são descritos na Tabela 16.

Tabela 16: Valores permitidos para o atributo Category, referente a classe File

Rank	Keyword	Descrição
0	Current	As informações relatadas do arquivo são após a mudança.
1	Original	As informações relatadas do arquivo são anteriores à mudança.

Fstype: Opcional. O tipo de arquivo de sistema no qual o arquivo está. Este atributo controla na forma em que o nome do caminho e outros atributos são interpretados. Os possíveis valores para este, são descritos na Tabela 17.

Tabela 17: Valores permitidos para o atributo Fstype, referente a classe File

Rank	Keyword	Descrição
0	Ufs	Arquivos de Sistema Rápido Berkeley Unix
1	Efs	Arquivo de Sistema “efs” Linux
2	Nfs	Arquivos de Sistema de Rede (Network File System)
3	Afs	Arquivos de Sistema Andrew (Andrew File System)
4	Ntfs	Arquivos de Sistema Windows NT
5	Fat16	Arquivos de Sistema Windows FAT 16-bits
6	Fat32	Arquivos de Sistema Windows FAT 32-bits
7	Pcfs	Arquivos de Sistema “PC” (MS-DOS) de CD-ROM
8	Joliet	Arquivos de Sistema Joliet de CD-ROM
9	Iso9660	Arquivos de Sistema ISSO 9660 de CD-ROM

File-Type: Opcional. O tipo de arquivo, como um *mime-type*. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.6.1. Classe FileAccess

A classe FileAccess representa as permissões de acesso ao arquivo. Esta representação visa ser útil em Sistemas Operacionais. Sua composição é feita conforme a Figura 62:

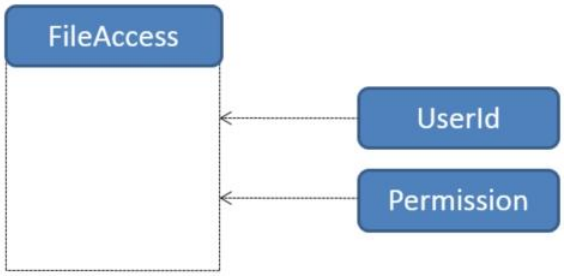

Figura 62: Classe FileAccess

As classes agregadas que compõem FileAccess são:

UserID (ID de Usuário): Exatamente um. O usuário ou grupo no qual as permissões são aplicadas. O valor do atributo "type" deve ser "*user-privs*", "*group-privs*", ou "*other-privs*". Outros valores para este atributo não devem ser utilizados neste contexto.

Permission (Permissão): Um ou mais, tipo ENUM. Nível de acesso permitido. Os valores permitidos para este serão descritos na Tabela 18.

Tabela 18: Valores permitidos para Permission, referente a classe FileAccess

Rank	Keyword	Descrição
0	noAccess	Nenhum acesso é permitido para este usuário.
1	Read	Este usuário possui acesso de leitura no arquivo.
2	Write	Este usuário possui acesso de escrita no arquivo.
3	Execute	Este usuário possui permissão para executar o arquivo.
4	Search	Este usuário possui a permissão para "procurar" este arquivo.
5	Delete	Este usuário possui a permissão para deletar este arquivo.

6	executeAs	Este usuário possui a permissão para executar este arquivo como outro usuário.
7	changePermissions	Este usuário pode alterar as permissões de acesso do arquivo.
8	takeOwnership	Este usuário tem a permissão para tomar a posse sobre o arquivo.

Os valores "*changePermissions*" e "*takeOwnership*" representam conceitos do Windows. Em Unix, o proprietário do arquivo sempre tem acesso à "*changePermissions*", mesmo que nenhuma outra permissão de acesso seja concedida a este usuário. (CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

Esta classe pode ser representada pelo modelo de dados conforme a Figura 63.

```
<!ELEMENT Permission EMPTY >
   <!ATTLIST Permission
      perms %attvals.fileperm; #REQUIRED
      %attlist.global;
   >

   <!ENTITY % attvals.fileperm "( noAccess | read | write | execute |
    search | delete | executeAs | changePermissions |
    takeOwnership)" >
```

Figura 63: Código de representação da classe FileAccess

1.6.2. Classe Linkage

A classe Linkage representa conexões do sistema de arquivos entre o arquivo descrito no elemento <File> e outros objetos no sistema de arquivos. Por exemplo, se o elemento <File> é um *link* simbólico ou um atalho, então o elemento <Linkage> deve conter o nome do objeto no qual o *link* aponta. Mais informações podem ser fornecidas sobre o objeto no elemento <Linkage> com outro elemento <File>, caso necessário.

A composição da classe Linkage pode ser vista na Figura 64, a seguir:

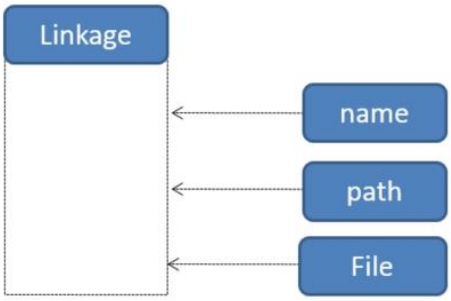

Figura 64: Classe Linkage

As classes que compõem Linkage são:

Name (Nome): Exatamente um, tipo String. O nome do objeto de sistema de arquivos, não incluso o caminho.

Path (Caminho): Exatamente um, tipo String. O caminho completo do objeto de sistema de arquivos, incluindo o nome. O nome do caminho deve ser representado da forma mais "universal" possível, para facilitar o processamento do alerta.

File (Arquivo): Exatamente um. O elemento <File> pode ser utilizado no lugar de <name> e <path> se informações adicionais sobre o arquivo são para serem incluídas.

Esta classe é representada pelo modelo de dados como visto na Figura 65.

```
1   <!ENTITY % attvals.linkcat
2           "( hard-link | mount-point | reparse-point | shortcut |
3             stream |symbolic-link )
4       ">
5
6     <!ELEMENT Linkage(
7         (name, path) | File)>
8     <!ATTLIST Linkage
9         category %attvals.linkcat; #REQUIRED
10        %attlist.global;
11      >
```

Figura 65: Código de representação da classe Linkage

A classe Linkage possui um único atributo:

Category (Categoria): O tipo de objeto no qual o *link* descreve. Os valores permitidos são descritos na Tabela 19.

Tabela 19: Valores permitidos para o atributo Category, referente a classe Linkage

Rank	Keyword	Descrição
0	Hard-link	O elemento <name> representa outro nome para o arquivo. Esta informação é mais facilmente obtida em sistemas de arquivo NTFS.
1	Mount-point	Um *alias* (apelido) para o diretório especificado pelos parentes dos elementos <name> e <path>.
2	Reparse-point	Aplicado apenas em Windows; exclui links simbólicos e pontos de montagem, que são tipos específicos de *reparse-points*.
3	Shortcut	O arquivo representado por um "atalho" do Windows. Um atalho é distinto de um *link* simbólico, por causa da diferença de seus conteúdos, que podem ser importantes para o gerente.
4	Stream	Uma ADS (Alternate Data Stream – Fluxo de dados alternado) em Windows, ou uma *fork* em MacOS. Entidades de sistemas de arquivos separadas que são consideradas uma extensão do <File> principal.
5	Symbolic-link	O elemento <name> representa o arquivo no qual o *link* aponta.

(CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).

1.6.3. Classe Inode

A classe Inode é usada para representar informações adicionais contidas em um sistema de arquivos Unix *i-node*. Sua composição pode ser vista na Figura 66, a seguir:

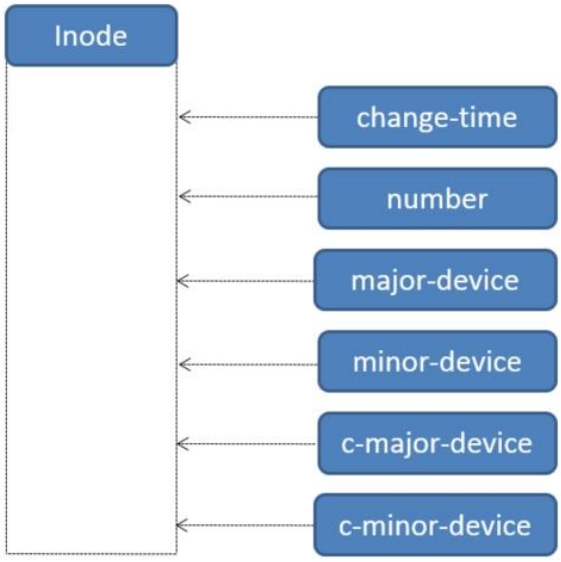

Figura 66: Classe Inode

As classes agregadas que compõem Inode são:

Change-Time: Zero ou um, tipo Datetime. A hora no qual o último *inode* mudou.

Number (Número): Zero ou um, tipo Integer. O número do *inode*.

Major-Device: Zero ou mais, tipo Integer. O maior número de dispositivos em relação ao dispositivo em que o arquivo reside.

Minor-Device: Zero ou mais, tipo Integer. O menor número de dispositivos em relação ao dispositivo em que o arquivo reside.

C-Major-Device: O maior dispositivo do próprio arquivo, se ele é um dispositivo especial de caracteres.

C-Minor-Device: O menor dispositivo do próprio arquivo, se ele é um dispositivo especial de caracteres.

Os valores de <Number>, <major-device> e <minor-device> devem ser dados juntos, como também devem ser os valores de <c-major-device> e <c-minor-device>.

Na Figura 67 abaixo, pode-se ver a representação da classe Inode de acordo com o modelo de dados.

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

Figura 67: Código de representação da classe Inode

1.6.4. Classe Checksum

A classe Checksum representa informações de verificação associadas ao arquivo. Esta informação de verificação pode ser feita por verificadores de integridade de arquivos, entre outros.

Sua composição será apresentada a seguir, na Figura 68:

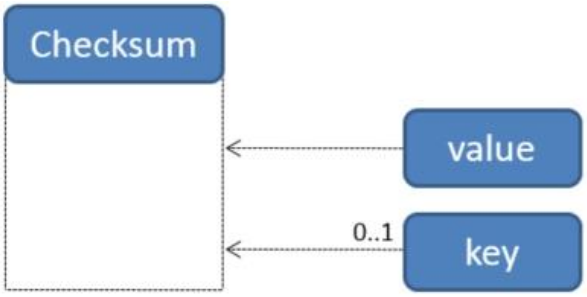

Figura 68: Classe Checksum

As classes agregadas que compõem Checksum são:

Value (Valor): Exatamente um, tipo String. O valor da verificação (*checksum*).

Key (Chave): Zero ou um, tipo String. A chave da verificação, se necessário.

Esta classe pode ser representada pelo modelo de dados conforme apresentado na Figura 69 abaixo:

```
<!ENTITY % attvals.checksumalgos
    "( MD4 | MD5 | SHA1 | SHA2-256 | SHA2-384 | SHA2-512 | CRC-32 |
    Haval | Tiger | Gost )"
>

<!ELEMENT Checksum (
    value, key?
  )>
<!ATTLIST Checksum
    algorithm %attvals.checksumalgos; #REQUIRED
    %attlist.global;
  >
```

Figura 69: Código de representação da classe Checksum

Checksum possui apenas um atributo:

Algorithm (Algoritmo): O algoritmo criptográfico utilizado na computação da verificação. Os valores permitidos para este atributo são descritos na Tabela 20.

Tabela 20: Valores permitidos para o atributo Algorithm, referente a classe Checksum

Rank	Keyword	Descrição
0	MD4	Algoritmo MD4
1	MD5	Algoritmo MD5
2	SHA1	Algoritmo SHA1
3	SHA2-256	Algoritmo SHA2 com tamanho de 256 bits.
4	SHA2-384	Algoritmo SHA2 com tamanho de 384 bits.
5	SHA2-512	Algoritmo SHA2 com tamanho de 512 bits.
6	CRC-32	Algoritmo CRC com tamanho de 32 bits.
7	Haval	Algoritmo Haval
8	Tiger	Algoritmo Tiger
9	Gost	Algoritmo Gost

(CURRY, David A.; FEINSTEIN, Benjamin S.; DEBAR, Herve. 2007).