Uma Ferramenta de Agregação e Extração de Alertas para Soluções Colaborativas

Bruno Lins[1], Eduardo Luzeiro Feitosa[2], Djamel Sadok[1]

[1]Centro de Informática
Universidade Federal de Pernambuco (UFPE)
Caixa Postal 7851 – Cidade Universitária - Recife - PE

[2]Departamento de Ciência da Computação
Universidade Federal do Amazonas (UFAM)
CEP 69077-000 Campus Universitário - Manaus - AM

{bfol, jamel}@cin.ufpe.br, efeitosa@dcc.ufam.edu.br

Resumo. *O uso de soluções colaborativas tem se mostrado cada vez mais comum e eficiente na detecção de ataques, intrusões e anomalias. Contudo, devido a sua aplicabilidade, tais soluções tem que processar uma grande quantidade de alertas produzidos pelos mais diferentes detectores. O objetivo deste artigo é implementar uma ferramenta capaz de agregar grandes volume de alertas e extrair somente aqueles mais significativos. Para validação, foram realizados testes usando a base DARPA 2000 e tráfego real.*

Abstract. *The use of collaborative solutions had proved increasingly common and effective in detecting attacks, intrusions and anomalies. However, due to its applicability, these solutions have to process a large volume of alerts produced by many different detectors. The goal of this paper is to implement a tool capable to aggregate large volume of alerts and extract only those alerts most significant. For validation, tests were performed in a controlled environment using DARPA 2000 dataset and real traffic.*

1. Introdução

Soluções colaborativas, tipicamente conhecidas como CIDS (*Collaborative Intrusion Detection Systems*) e CAIDS (*Collaborative Anomaly and Intrusion Detection Systems*), têm sido apontadas como a melhor saída para detecção de ataques massivos e aqueles realizados em múltiplas fases, tais como ataques distribuídos de negação de serviço (DDoS – do inglês *Distributed Denial-of-Service*) e a proliferação de worms (Storm e Blaster, por exemplo). Embora tais soluções sejam bastante efetivas, graças à habilidade que seus componentes e subsistemas têm de trocar informações relevantes, o principal problema é como manipular grandes quantidades de alertas.

Soleimani e Ghorbani [Soleimani e Ghorbani 2008] apontam três pontos que justificam a necessidade de mecanismos de agregação de alertas. Primeiro, nem sempre é fácil localizar a origem ou o destino dos ataques examinando os alertas um a um (em baixo nível). Segundo, soluções menos robustas consideram os alertas de forma "crua", sem levar em consideração possíveis relações e conexões entre eles. Por fim, respostas automáticas tendem a ser ineficientes quando baseadas na análise de alertas isolados, assim como sua contribuição para o processo de tomada de decisão.

Visando solucionar este problema, este artigo apresenta o projeto e implementação de uma ferramenta de agregação e extração de alertas. Baseada na técnica proposta por Xu *et al.* [Xu *et al.* 2005], voltada para detecção de anomalias de tráfego em *backbones*, este trabalho faz uso da agregação dos alertas e extração de

conjuntos (*clusters*) relevantes de informação através de três dimensões distintas: endereço IP de origem, endereço IP destino e classe de ataque.
O objetivo é, após a agregação, obter somente os mais relevantes alertas de acordo com os conjuntos de interesse. Isto permite que tais alertas possam ser usados na correlação e possível descoberta da estratégia de ataque, ajudando operadores de rede e gerentes de TI a enxergar as reais intenções do ataque e a tomar decisões mais adequadas. A solução proposta tem o potencial de reduzir o número de alertas (com baixa carga computacional), diminuindo a taxa de falsos positivos e priorizando os alertas mais relevantes.

O restante deste artigo está estruturado da seguinte forma. Primeiramente, alguns trabalhos relacionados (2) são discutidos para melhor contextualizar o problema. Em seguida, os conceitos fundamentais (3) para o entendimento da proposta são apresentados. Após isso, o processo de extração de conjuntos significantes (4) é explicado. A próxima seção apresenta uma visão geral da solução (5), incluindo detalhes de cada componente. Em seguida são apresentados os testes de desempenho e extração de alertas significantes (6), visando à validação da solução. Por último, são feitas as conclusões finais (7).

2. Trabalhos Relacionados

É fato que a agregação de alertas é uma atividade essencial para qualquer solução colaborativa. Normalmente, a agregação é realizada através da combinação de similaridades entre determinados atributos (campos) dos alertas recebidos em um determinado intervalo de tempo.

Na literatura, trabalhos como os de Valdes e Skiner [Valdes e Skiner 2001], Julish [Julicsh 2003] e Cuppens [Cuppens 2001] tem sido bastante referenciados tendo em vista que apresentam conceitos como similaridade e clusters que são a base para as técnicas e soluções utilizadas hoje.

Valdes e Skiner [Valdes e Skiner 2001] utilizam endereços IP (origem e destino), portas (origem e destino), tempo e classe de ataque para extrair as semelhanças (similaridades) entre os alertas. O trabalho de Julicsh [Julicsh 2003] agrega todos os alertas que compartilham as mesmas causas, o que é, intuitivamente, a razão da ocorrência desses alertas. Para tanto, estruturas hierárquicas, chamadas hierarquias de generalização (em tradução livre de *generalization hierarchy*), são usadas para separar os atributos dos alertas, dos valores mais gerais aos mais específicos. Desta forma, as diferenças entre dois alertas podem ser medidas comparando o caminho mais longo entre os valores desses atributos na estrutura correspondente. Já o trabalho de Cuppens [Cuppens 2001] emprega uma base de dados relacional para armazenar alertas e avaliá-los usando um conjunto de regras de similaridade para grupos, de acordo com a ocorrência de um mesmo ataque.

Mais recentemente, Zhihong *et al.* [Zhihong *et al* 2008] propuseram um sistema para agregação e correlação de alertas em tempo real, chamado Alertclu, que utiliza o conceito de clusters para analisar a similaridade dos alertas e assim melhorar o processo de agregação. Além disso, Alertclu permite que operadores humanos classifiquem os alertas como verdadeiro e falso positivos. Contudo, o sistema proposto gera apenas clusters baseados na classificação dos alertas (nome ou tipo do alerta) e emprega um algoritmo simples para medir a proximidade entre endereços IP.

Maggi *et al.* [19], explorando os conceitos de métricas e conjuntos fuzzy, propuseram algoritmos para descobrir se dois alertas são "próximos" ou não em relação ao tempo. Os autores definiram um critério para computar uma medida de "distância-tempo" entre os alertas de modo a considerar incertezas nas medições e incorporar erros como, por exemplo, atrasos na detecção.

Diferente dos trabalhos descritos anteriormente, a solução proposta neste artigo emprega o conceito de clusters significantes [Xu *et al.* 2005], onde são gerados clusters para diferentes dimensões de interesse, no caso, classe de ataque, endereço de origem e endereço de destino. Desta forma, ao invés de gerar apenas clusters relativos à classe de ataque, como no trabalho de Zhihong *et al.*, a solução proposta permite avaliar em tempo real alertas de múltiplas fontes de diferentes pontos de vista, reduzindo o número de alertas e a incerteza no processo de agregação. Além disso, nenhum tipo de conhecimento prévio dos alertas e/ou fase de treinamento são necessárias para o funcionamento da solução.

3. Conceitos Básicos

O trabalho de Xu *et al.* [Xu *et al.* 2005] propõe o uso de uma medida de teoria de informação, chamada **incerteza relativa** (do inglês *relative uncertainty* ou *RU*), para extrair dados significantes baseados no conceito matemático de entropia.

Entropia mede essencialmente a "quantidade de incerteza" contida em uma determinada informação. Dado X uma variável randômica que pode ter N discretos valores $\{x_1, \dots, x_n\}$ com vetor de probabilidade $\{p_1, \dots, p_n\}$, a entropia de uma variável randômica X é definida como:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \quad (1)$$

onde, por convenção, $0 \log 0 = 0$.

Uma vez que a entropia mede a "variabilidade observada" nos valores de X, é correto afirmar que $0 \leq H(X) \leq H_{max}(X) \coloneqq \log \min(N_x, m)$, onde N_x são os possiveis valores discretos para X, m é o numero de vezes em que a variável X é observada e $H_{max}(X)$ é definido com a entropia máxima de X quando $p(x_i) = 1/n$.

Então, assumindo que existe uma "variabilidade observada" $m \geq 2$ e $N_x \geq 2$, Xu *et al.* introduziu uma entropia padronizada, chamada incerteza relativa (RU), para prover um índice de variedade ou uniformidade independentemente do suporte ou tamanho da amostra definido como:

$$RU(X) = \frac{H(X)}{H_{max}(X)} = H(X) / \log \min\{N_x, m\} \quad (2)$$

Uma vez que a incerteza relativa fornece um índice de variedade ou uniformidade nos valores observados de X, se $RU(X) = 0$, então todas as observações de X são do mesmo tipo como, por exemplo, $p(x_i) = 1$ para qualquer $x_i \in X$, o que significa que variabilidade observada é completamente ausente. Por outro lado, quando $m \leq N_x$, $RU(X) = 1$ se e somente se $|A| = m$ e $p(x_i) = 1/m$ para cada $x_i \in A$, onde A denota um subconjunto de valores observados de X. Assim, todos os valores observados de X são diferentes ou únicos tendo um alto grau de variabilidade ou incerteza. Se $m > N_x$, $RU(X) = 1$ se e somente se $m_i = m/N_x$. Assim, $p(x_i) = 1/N_x$ para

$x_i \in A = X$, por exemplo, os valores observados são uma distribuição uniforme sobre X. Neste caso, $RU(X)$ mede o grau de uniformidade nos valores observados de X.

Xu *et al.* consideram a entropia condicional $H(X|A)$ e a incerteza relativa condicional $RU(X|A)$ pelo condicionamento de X em A, onde $H(X|A) = H(X)$, $H_{max}(X|A) = log|A|$ e $RU(X|A) = H(X)/log|A|$. Assim, $RU(X|A) = 1$ se e somente se $p(x_i) = 1/|A|$ para cada $x_i \in A$. De forma geral, $RU(X|A) \approx 1$ significa que os valores observados de X estão mais próximos de serem uma distribuição uniforme, portanto menos distinguíveis um dos outros, enquanto que $RU(X|A) \ll 1$ indica que a distribuição é mais enviesada, com alguns poucos valores mais frequentemente observados. É importante enfatizar que esta medida de uniformidade é usada para definir "conjuntos de interesses significantes".

4. Extração de Conjuntos Significantes

Antes de explicar o processo usado para extrair conjuntos significantes, é necessário esclarecer que o foco da proposta de [Xu *et al.* 2005] foi alterado. Ao invés de utilizar a extração de conjuntos significantes para detecção de ataques e anomalias, esse processo é utilizado para minimizar a carga de alertas sobre sistema de gerenciamento.

Além disso, ao contrário do trabalho original que usa uma tupla formada por endereço IP de origem, endereço IP de destino, porta de origem, porta de destino e protocolo para determinar os padrões de comunicação entre computadores e serviços, este trabalho adota uma tupla composto por apenas três elementos: endereço IP origem (*srcIP*), endereço IP destino (*dstIP*) e classe de ataque (*class*), onde a extração dos conjuntos de *srcIP* e *dstIP* é feita de forma idêntica ao trabalho original (representando os padrões de comunicação entre os computadores de interesse), enquanto o conjunto *class* representa o conjunto de informações sobre classe e impacto dos alertas.

Neste trabalho, optou-se por não usar as informações sobre as portas de origem e destino na identificação de conjuntos significantes, uma vez que a porta de origem pode ser facilmente alterada para esconder um ataque e a porta de destino é normalmente relacionada com o campo de classe de ataque.

No que se diz respeito ao processo de extração de conjuntos significantes, a ideia é que uma vez dado:

- X, uma variável randômica, que representa, por exemplo, as características de uma dimensão como *srcIP*;
- T, um intervalo de tempo;
- m, o número total de alertas observados durante o intervalo de tempo T;
- $\mathrm{A} = \{\mathrm{a}_1, \ldots, \mathrm{a}_\mathrm{n}\}, \mathrm{n} \geq 2$, o conjunto dos destintos valores de X,

a distribuição de probabilidade $\mathrm{P_A}$ em X é dada por $\mathrm{p_i} = \mathrm{P_A} = \mathrm{m_i}/\mathrm{m}$, onde $\mathrm{m_i}$ é o número de alertas que tem o valor $\mathrm{a_i}$ e a incerteza relativa (condicional), $\mathrm{RU(P_A)} = \mathrm{RU(X|A)}$ mede o grau de uniformidade nas características observadas de A. Se $\mathrm{RU(P_A)}$ se aproxima de 1, diz-se que, $> \beta = 0.9$, então os valores observados estão perto de serem uniformemente distribuídos, logo, indistinguíveis. Caso contrário, as características dos valores encontrados em A se destacam do restante.

Consequentemente é possível definir um subconjunto S em A que contém os mais significantes (interessantes) valores de A se S é um pequeno subconjunto de A tal que: (i) a probabilidade de qualquer valor em S é maior que qualquer outro valor encontrado em A; e (ii) a distribuição de probabilidade (condicional) sobre o conjunto

dos valores restantes, $R := A - S$, está perto de ser uniformemente distribuída. De forma intuitiva, S contém os mais significantes elementos de A, enquanto os valores restantes (R) são indistinguíveis entre si.

O Algoritmo 1 apresenta uma visão simplificada do algoritmo usado (pseudocódigo) para extração do conjunto significante S de A.

Algoritmo 1. Pseudocódigo do algoritmo de extração de conjuntos significantes

Entrada: $\alpha := \alpha_0$; $\beta := 0.9$; $S := \emptyset$;

```
01: S := ∅; R := A; k := 1
02: calcule a distribuição de probabilidade P_R e RU θ = RU(P_R);
03: while θ ≤ β do
04:     α = α x 2^-k;
05:     for each α_i ∈ R do
06:         if P_A(α_i) ≥ α then
07:             S := S ∪ {α_i}; R := R − {α_i};
08:         endif
09:     end for
10:     calcule a distribuição de probabilidade (condicional) P_R e θ = RU(P_R);
11: end while
```

Inicialmente, com $\alpha_0 = 2\%$, o algoritmo busca o melhor valor limitante de corte α^* através de uma "aproximação exponencial" (reduzindo o limitante α através de um fator de decréscimo $1/2^k$ com k constante). Enquanto a incerteza relativa da distribuição de probabilidade P_R no conjunto R é inferior ao valor de β (linha 03), o algoritmo examina cada valor em R e inclui em S os elementos cuja probabilidade excede o limitante α (linhas 06 e 07). O algoritmo só para quando a distribuição de probabilidade do conjunto dos elementos remanescentes R se aproxima de uma distribuição uniforme ($> \beta = 0.9$), ou seja, elementos indistinguíveis (linha 03).

Os resultados do algoritmo são vetores, um para cada conjunto de interesse, onde cada conjunto contém a chave de interesse (*srcIP*, por exemplo), a frequência de cada chave e o ponteiro para todos os elementos dessas chaves.

5. Projeto e Implementação

Uma vez que o objetivo é implementar um mecanismo eficiente de agregação de alertas gerados por múltiplas origens, sem a necessidade do aumento da carga computacional, esta seção descreve o projeto e desenvolvimento de uma ferramenta modular de agregação de alertas, capaz de receber alertas de diferentes detectores, convertê-los para o formato padrão IDMEF [Debar *et al.* 2007] e então agregá-los baseado em clusters similares. A Figura 1 mostra o diagrama funcional da ferramenta.

Os principais componentes que compõem a ferramenta são:

- **Módulo de Manipulação de Alertas** – tem a função de receber a informação (alertas) de diferentes detectores e prepará-los para análise. Este componente serve como porta de entrada para as outras funcionalidades da ferramenta. Especificamente, é responsável por duas atividades:
    - *Tradução e Validação* – converter as mensagens provenientes de diferentes fontes e em diferentes formatos para o padrão IDMEF, validando-as de forma a garantir sua conformidade.
    - *Ordenação* – ordena e sincroniza os alertas, já no formato IDMEF, de acordo com as marcas de tempo (*timestamps*). É fundamental

salientar a necessidade de que todos os detectores estejam com os relógios sincronizados para assegurar uma correta ordenação dos alertas. Uma maneira simples de garantia a sincronia desses detectores é por meio do protocolo NTP [Mills, 1992]

Em relação à implementação, este o módulo é dividido em: Agente de Manipulação de Alertas (AMA) e Servidor de Manipulação de Alertas (SMA). O primeiro é executado continuamente, coletando os alertas gerados, traduzindo-os, validando-os quando necessário e finalmente enviando-os para o SMA. O segundo, que também é executado continuamente, recebe os alertas dos AMAs, os ordena e os encaminha para o módulo de agregação.

- **Módulo de Agregação –** considerado o coração da ferramenta, este módulo tem como principal tarefa agregar os alertas que têm valores de interesse comuns. Basicamente, recebe os alertas, os ordena e executa o algoritmo de agregação para extrair apenas os conjuntos de alertas significantes. O resultado obtido é um sumário (lista) de alertas significantes que são encaminhados para uma possível correlação para checar ou confirmar a existência de ataques ou anomalias.

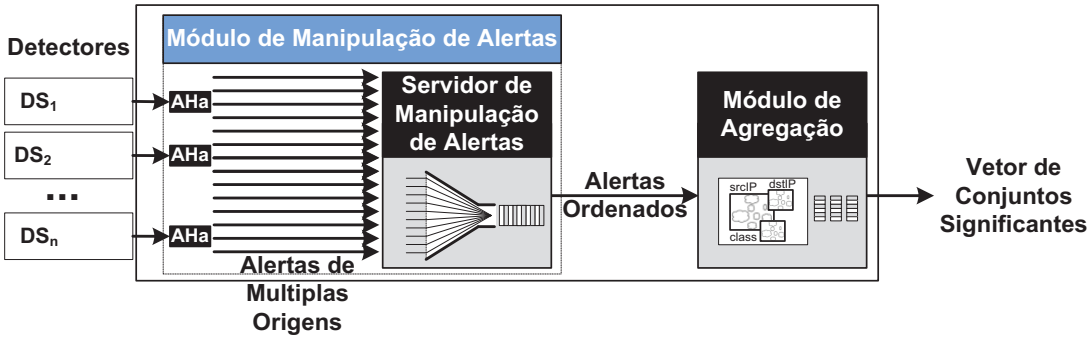

Figura 1. Diagrama funcional da ferramenta.

5.1 Implementação

Esta seção descreve o processo de implementação da ferramenta de agregação de alertas focando principalmente aspectos de sua estrutura de dados.

5.1.1 Módulo de Manipulação de Alertas

Uma vez que deve receber alertas de múltiplas fontes, este componente é implementado baseado no esquema clientes/servidor, Agentes de Manipulação de Alerta (AMA) e Servidor de Manipulação de Alertas (SMA), respectivamente.

Agente de Manipulação de Alertas (AMA)

Desenvolvido usando Java 1.6, o AMA é implementado para trabalhar na forma de *deamon* em conjunto com os detectores (Snort, por exemplo). Sua operação é dividida em três tarefas.

1. Verificação periódica da existência de arquivos de alertas gerados pelos detectores. Quando um novo arquivo de alertas é encontrado, uma cópia é feita para que se inicie o processo de tradução e validação.
2. Tradução e validação dos alertas, de sua forma original, para o formato IDMEF. Essa tarefa é implementada usando o DocumentBuilderFactory,

uma classe do pacote javax.xml. Atualmente, os AMAs operam somente com detectores Snort [Snort 2010].

3. A última tarefa consiste no envio dos alertas já traduzidos e validados, através de comunicação via sockets, para o Servidor de Manipulação de Alertas (SMA).

Servidor de Manipulação de Alertas (SMA)

Também desenvolvido com Java 1.6 e projetado para atuar como um serviço, o SMA tem como finalidade receber, via *socket*, e armazenar alertas. Depois, eles são ordenados e então repassados para o modulo de agregação. É importante enfatizar que este trabalho considera que todos os detectores estão trabalhando com seu tempo sincronizado de alguma forma.

5.1.2 Módulo de Agregação

O módulo de agregação tem a função de extrair os alertas mais significantes. Para alcançar este objetivo, foram implementada duas estruturas de dados, chamadas de *ATable* e *CTable*. A Figura 3 ilustra as estruturas *ATable* e *CTable*.

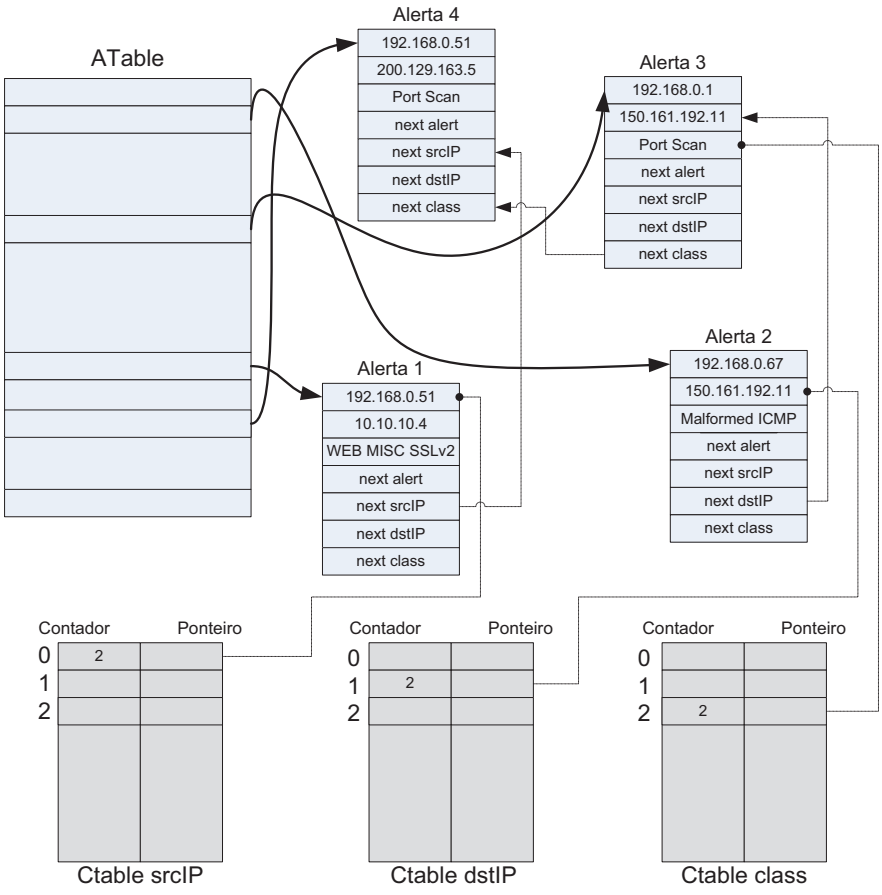

Figura 2. Estrutura das ATable e CTable

A *ATable* armazena todos os alertas recebidos de todos os detectores em um intervalo de tempo X enquanto as *CTables* agrupam os alertas levando em consideração a valoração de um determinado atributo (dimensão). Neste trabalho são utilizadas três dimensões, endereços IP origem e destino e classe do ataque. Detalhes sobre cada um destes elementos serão dados a seguir.

Em relação à implementação, *ATable* é uma *hash table* para o armazenamento de todos os alertas e suas informações relevantes, no caso o endereço IP de origem, endereço IP de destino, portas de origem e destino, classe de ataque, *timestamp* e severidade do alerta. Além destes atributos, cada alerta inserido na *ATable* também armazena três ponteiros (*next srcIP*, *next dstIP* e *next Class*) para ligar alertas que compartilham os mesmos valores em uma determinada dimensão, ou seja, todos os elementos que possuam mesma valoração para um determinado atributo serão interligados. Essa simples ideia remove a necessidade de duplicação de alertas no momento da geração das *CTables*, sem mencionar que diminui o custo com espaço de memória utilizado e aumenta a escalabilidade e eficiência da agregação, uma vez que arquivos de alertas podem conter centenas de milhares de entradas.

Após todos alertas estarem armazenados na *ATable*, as *CTables* colocam em operação o processo de referenciar a primeira ocorrência de cada um dos alertas, fornecendo uma maneira fácil e simples de localizar cada um dos alertas para cada determinada dimensão. Já que são usadas três dimensões, foram criadas três instâncias de *CTable* para gerenciar cada uma delas.

Apesar de simples, as *CTables* são essenciais para o processo de extração de conjuntos significantes. Cada *Ctable* armazena um contador que contém o número de ocorrências para cada uma das chaves e um ponteiro para a primeira ocorrência dessa chave na *ATable*. Por exemplo, ao avaliar o alerta 1 na Figura 2, tem-se o campo endereço IP de origem (192.168.0.51) como a primeira ocorrência na *scrIP CTable*. Desta forma é criada uma referência para esse elemento na *ATable* e o contador de alertas é incrementado em um. O mesmo procedimento é tomado para as dimensões *dstIP* e *Class*, representadas na Figura 2 por *dstIP CTable* e *class CTable* respectivamente. Contudo, ao avaliar o alerta 3, nota-se uma repetição no campo *dstIP* (150.161.192.11). Então ele é endereçado como o *next dstIP* do alerta 2 e o contador de ocorrências é, finalmente, incrementado em um.

Por exemplo, quando o alerta 1 é analisado, verificasse que seu endereço IP de origem não contem ocorrência para na *srcIP CTable*, logo este elemento é referenciado e seu contador é acrescido de um. No entanto quando o endereço IP de origem do alerta 4 é procurando na *srcIP CTable* encontramos a ocorrência do Alerta 1, então o alerta 4 é referenciado como o *next srcIP* do alerta 1 é o contador desta chave é incrementado. Este procedimento é feito para todos os alertas da *Atable* e procede de forma análoga para os atributos endereço IP de destino e classe de ataque, que serão referenciados nas *dstIP CTable* e *class CTable.*

Quando as *CTables* são finalizadas (todas os elementos da *ATable* já foram devidamente referenciados e reagrupados de acordo com as dimensões), o processo de extração é iniciado. De acordo com o algoritmo 1, ele resulta em três listas (uma para cada dimensão) compostas pelas chaves mais significantes(*srcIP*, *dstIP* ou *classe do ataque*), frequência de ocorrência e um ponteiro para a primeira ocorrência de cada uma das chaves em sua respectiva *CTable*. Como produto final do módulo de agregação, cada lista é utilizada para a geração de vetores que contêm, para cada um das dimensões, os mais significantes elementos (alertas).

6. Avaliação e Resultados Iniciais

6.1 Ambiente de teste

Para a realização das avaliações da ferramenta foram utilizadas as instalações do GPRT (Grupo de Pesquisa em Redes e Telecomunicações) da UFPE (Universidade Federal de

Pernambuco). O ambiente de teste foi composto por um computador Intel Core2Duo T5300, 2 GB de RAM, e 250 GB de HDD, executando a distribuição do Linux Ubuntu 8.04.

6.2 Desempenho

Apesar de a solução proposta empregar o modelo cliente/servidor, optou-se por avaliar o consumo de CPU e memória apenas do módulo de agregação, uma vez que as outras atividades, principalmente as realizadas pelos AMAs pode variar consideravelmente.

Para melhor apresentar os resultados do processo de medição, o módulo de agregação foi avaliado de duas formas: construção da *ATable* e extração de conjuntos significante (que também inclui a criação das *CTables*). Esta mesma divisão é feita na avaliação do consumo de memória. Para tanto, foram inseridos pontos de verificação em cada uma das partes no código do módulo de agregação. O valor medido nesses pontos indica o consumo de memória e CPU de cada uma das partes e a soma, o consumo total do módulo de agregação.

Para testar o módulo de agregação foram usados dois conjuntos de dados (*AF1* e *AF2*), coletados no GPRT e contendo, respectivamente 5.224 e 24.824 alertas no formato IDMEF. A Tabela 1 mostra a carga da CPU e o uso de memória resultante dos dois arquivos de alerta (*AF1* e *AF2*). As Figuras 3 e 4 mostram, respectivamente, o consumo de CPU e memória do módulo de agregação para ambos arquivos, onde nota-se que quanto maior o número de alertas a serem processados, menor é o consumo de memória e CPU, devido ao processo de construção da *ATable*.

Tabela 1. Consumo de CPU e memória no modulo de Agregação

Arquivo	CPU (%)			Memória (MB)		
	mínimo	média	máximo	mínimo	média	máximo
AF1	22,47	34,51	64,43	1,4	29,13	84,72
AF2	24,75	28,33	61,81	0,83	26,62	74,37

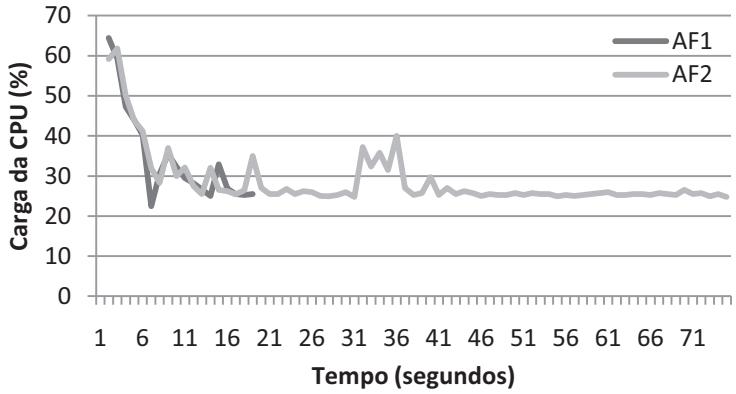

Figura 3. Consumo de CPU do módulo de agregação para AF1 e AF2

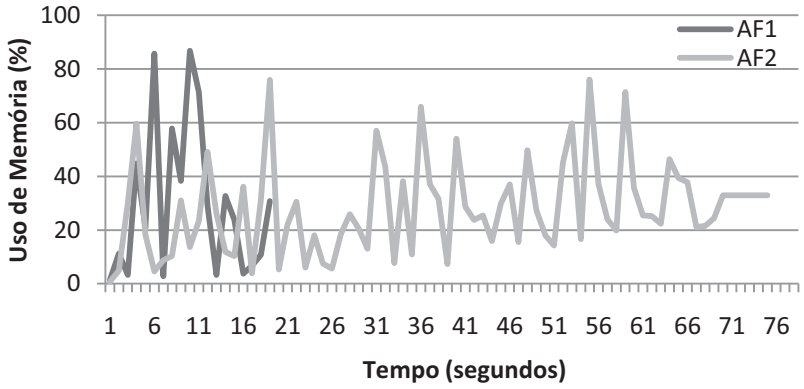

Figura 4. Uso de memória do módulo de agregação para AF1 e AF2

Em resumo, o consumo de CPU tende a ser constante e está relacionado com o número de alertas a serem armazenados na *ATable* e consequentemente na *CTable*, assim como o tamanho dos conjunto de elementos significativos. Por outro lado, o uso de memória é determinado pelo número de alertas a serem avaliados e armazenados na *ATable* e está representado pelo intervalo de 1 a 69 segundos na Figura 4 para o alerta AF2. Essa oscilação é consequência dos processos de leitura, avaliação e inserção de alertas na *ATable*. O intervalo de 69 a 75 segundos, também na Figura 4, corresponde a construção da *CTable*, extração de conjuntos significantes e geração dos vetores resultantes para o alerta AF2.

6.3 Extrações de Significantes

Os testes de desempenho do uso de CPU e memória demonstram a viabilidade operacional da solução proposta. Entretanto, faz-se necessário provar que o processo de extração de conjuntos significantes também é eficiente. A fim de atender a este requisito, três experimentos foram conduzidos.

6.3.1 AF1 e AF2

A Tabela 2 resume o resultado da extração de elementos significantes nos arquivos de alerta *AF1* e *AF2* realizada pelo módulo de agregação.

Tabela 2. Características dos arquivos de alerta AF1 e AF2

Arquivo	Dimensão *class*		Dimensão *dstIP*		Dimensão *srcIP*	
	Únicos	Extraídos	Únicos	Extraídos	Únicos	Extraídos
AF1	8	4	467	13	2	1
AF2	16	7	22	7	35	20

Do arquivo AF1, formado por 1.399 alertas, foram extraídas 4 classes de ataque significantes de um total de 8 classes encontradas nos alertas. Para a dimensão *dstIP*, 13 conjuntos foram considerados significantes de um total de 467. Para a dimensão *srcIP*, apenas 1 conjunto significante de endereço IP de origem foi extraído de um total de 2. Já para o arquivo AF2, formado por 24.824 alertas, foram extraídos 7 conjuntos significantes de classes de ataque de um total de 16, 7 conjuntos significantes de endereço IP destino de um total de 22 e 20 conjuntos de endereço IP origem de um total de 35 conjuntos distintos. A Tabela 3 exemplifica as classes de ataque significantes extraídos do alerta AF2.

Tabela 3. Valores de RU para classe de ataque no AF2

Classe de Ataque	RU	Probabilidade [$P_A(\alpha_i)$]	α
Bad Traffic same src/dst IP	0.387171712517231	0.450451176281018	0.02
Bad Traffic Loopback IP	0.396398827192577	0.449685787947147	0.02
Misc UPnP Malformed Advertise	0.406761881088819	0.056235900741218	0.02
SQL Probe Response Overflow	0.418514276148485	0.018329036416371	0.01
ICMP destination unreachable port	0.519671597203675	0.017603931679020	0.01
Storm Worm Phone Home Address	0.602436402138431	0.004028359651950	0.0025
Policy Outbound Teredo Traffic	0.627372862460001	0.002054463422494	0.00125
RU final	**0.925323538573275**		

A Tabela 3 representa a extração dos conjuntos significantes baseada na incerteza relativa (RU), descrita no algoritmo 1. Antes de iniciar o processo de extração, o RU inicial é calculado enquanto todos os elementos ainda estão no conjunto, $RU(A) = 0.387171712517231$. Quando o primeiro conjunto (*bad traffic loopback ip*) começa a ser avaliado, a probabilidade é calculada através da divisão da frequência de repetição desse elemento pelo número total de alertas do conjunto (522/1109), resultando em $P_A(a_1) = 0.450451176281018$. Depois a probabilidade $P_A(a_1)$ é comparada com o parâmetro α (linha 6 do algoritmo). Se a probabilidade é maior ou igual a α, então o conjunto é considerado significante e é adicionado ao conjunto S e removido de R. Este processo é repetido até que todos os elementos de R tenham sido comparados com α. Na Tabela 3, os três primeiros conjuntos são extraídos na primeira interação (α = 0.02), enquanto que o quarto, quinto, sexto e sétimo só serão extraídos na segunda (α = 0.01), terceira (α = 0.0025) e quarta (α = 0.00125) interações, respectivamente. Note que a cada interação o parâmetro α é reduzido à metade para prover uma aproximação. A extração de conjuntos significante é finalizada quando o RU do conjunto R é maior que o parâmetro β, neste caso $0.92323538573275 > 0.9$.

6.3.2 DARPA 2000 Dataset

O DARPA 2000 dataset [MIT 2010] é um conhecido arquivo de log (*trace*), bastante empregado na avaliação de IDS, contendo dois cenários (LLDOS 1.0 e LLDOS 2.0.2), onde o tráfego foi capturado em uma rede interna (*inside*) e externa (*outside*).

No cenário LLDOS 1.0, existem 1.109 alertas inside e 2.465 alertas outside. Para o cenário inside, dos 29 conjuntos de classe de ataque, 37 conjuntos de endereços IP de destino e 294 conjuntos de endereço IP de origem, foram extraídos 9 para classe de ataque, 12 para IP de destino e nenhum para IP de origem. A Tabela 4 sumariza os valores de RU para a dimensão dstIP.

Tabela 4 Valores de RU para dimensão IP de destino no cenário LLDOS 1.0 inside

IP de Destino	RU (*dstIP*)	Probabilidade [$P_A(\alpha_i)$]	α
131.84.1.31	0.349490430592109	0.7556357078449053	0.02
172.16.112.100	0.352162574775310	0.05049594229035167	0.02
172.16.112.105	0.354952942524035	0.026149684400360685	0.02
172.16.115.20	0.357870741608349	0.018034265103697024	0.01
194.7.248.153	0.860696355068559	0.015329125338142471	0.01
172.16.113.148	0.868338326412219	0.013525698827772768	0.01

172.16.112.194	0.876366490152678	0.013525698827772768	0.01
172.16.112.10	0.884815250608347	0.012623985572587917	0.01
172.16.116.20	0.893723469979750	0.012623985572587917	0.01
172.16.112.50	0.903135238078656	0.012623985572587917	0.01
172.16.115.87	0.913100810499855	0.010820559062218215	0.01
172.16.113.105	0.923677760466658	0.010820559062218215	0.01
RU final	**0.934932404156025**		

Para o cenário *outside* do LLDOS 1.0, foram extraídos 9 conjuntos de classe de ataque, 9 de IP de destino e 10 de IP de origem de um total de 24, 42 e 29, respectivamente.

Em relação à dimensão IP de origem (*srcIP*), para o cenário *inside*, nenhum conjunto significante pode ser encontrado porque o RU inicial do conjunto é 0.922710283397728, ou seja, $RU(A) > 0.9$. Isso acontece porque apesar de existirem 294 conjuntos, cada um contém poucos (entre 1 e 2) elementos.

No cenário LLDOS 2.0.2 do DARPA 2000, existem 935 alertas *inside* e 1.108 alertas *outside*. Usando os alertas *inside*, foram extraídos 6 conjuntos para classe de ataque, 2 para IP de destino e nenhum para IP de origem, de um total de 29, 26 e 434, respectivamente. O motivo da não extração na dimensão de IP de origem é igual no cenário LLDOS 1.0 *inside*, RU(A) = $0.9803433665539428 > \beta = 0.9$. Usando os alertas *outside*, foram extraídos 5 conjuntos para classe de ataque, 8 para IP de destino e 1 para IP de origem, de um total de 23, 28 e 20, respectivamente.

Para sumarizar os resultados do DARPA 2000, a Tabela 5 mostra a relevância do processo de extração de clusters significantes no DARPA 2000 dataset.

Tabela 5. Sumário de Extração do DARPA 2000 Dataset

Scenarios	Alerts per Clusters			
	Class	srcIP	dstIP	Class ∪ srcIP ∪ dstIP
LLDOS1.0 Inside	740	0	756	670
LLDOS1.0 Outside	1433	1413	1320	1303
LLDOS2.0.2 Inside	900	0	906	888
LLDOS2.0.2 Outside	1077	1071	1078	961

No cenário LLDOS 1.0 *inside*, dos 1109 alertas iniciais, a união de todas as dimensões de interesse (740 elementos para *class*, 0 para *srcIP* e 756 para *dstIP*) mostra que apenas 670 alertas foram considerados relevantes. No cenário LLDOS 1.0, a união das dimensões de interesse (1433 elementos para *class*, 1413 para *srcIP* e 1320 para *dstIP*) mostram que somente 1303 alertas foram considerados relevantes. Já para o cenário LLDOS 2.0.2 *inside*, a combinação das dimensões (900 elementos para *class*, 0 para *srcIP* e 906 para *dstIP*) revela que somente 888 alertas foram considerados relevantes e no cenário LLDOS 2.0.2 *outside*, dos 1077 elementos para dimensão *class*, 1071 para *srcIP* e 1078 para *dstIP*, somente 961 foram considerados relevantes.

É importante esclarecer que o número de alertas extraídos é próximo ao número inicial em alguns casos devido ao fato do Snort não ser capaz de detectar a fases 1 e parte da fase 4 dos ataques descritos em [MIT 2010], em ambos os cenários. A

explicação para tal fato se deve a não existência de regras Snort para tratar *ICMP requests* e conexões telnet como atividades maliciosas.

6.3.3 MAWI Dataset

O repositório de tráfego MAWI [MAWI 2010], parte integrante do projeto WIDE vem coletando e arquivando pacotes, nos últimos 5 anos (2006 a 2010) do enlace trans-pacífico (samplepoint-F, 150Mbps) entre o Japão e os Estados Unidos. Para avaliação, foram utilizados 4 arquivos (traces), consistindo de 15 minutos cada, coletados no dia 13 de Abril de 2010, entre 00:00 e 01:00, como parte do projeto um dia na vida da Internet (*a Day in the Life of the Internet project*[1]). As principais características destes arquivos (*traces*) são apresentadas na Tabela 6.

Tabela 6. Características dos traces MAWI

Trace	Size (MB)	#Packets	#Flows	Time (s)
1	2489.72	35.618.581	1.189.259	899.81
2	2447.38	35.137.496	1.260.941	899.66
3	2343.92	33.587.973	1.176.762	899.57
4	2442.66	34.949.704	1.201.515	900.70

Usando o IDS Snort para processar os traces, foram gerados 1355, 1377, 2455 e 1411 alertas, respectivamente. A Tabela 7 apresenta os alertas e as dimensões observadas e extraídas para todos os traces.

Tabela 7. Informações extraídas de todos os traces MAWI

Trace	#Alerts	Class	SrcIP	DstIP	Time (s)
1	1355	7/6	399/46	332/30	6
2	1377	7/6	367/42	370/28	6
3	2455	9/6	402/38	420/24	9
4	1411	7/6	383/35	384/19	5

Para melhorar o entendimento, a Figura 5 ilustra a distribuição de frequência do *cluster* classe de ataque para dos os arquivos. É possível notar que todas as classes de ataque extraídas representam atividades de varredura (*scan*). A mais alarmante classe de ataque é representada pelo *nkiller2 DDoS attack*, que é caracterizado pela grande quantidade de pacotes TCP com janela igual a zero (*TCP zerowindow packets*).

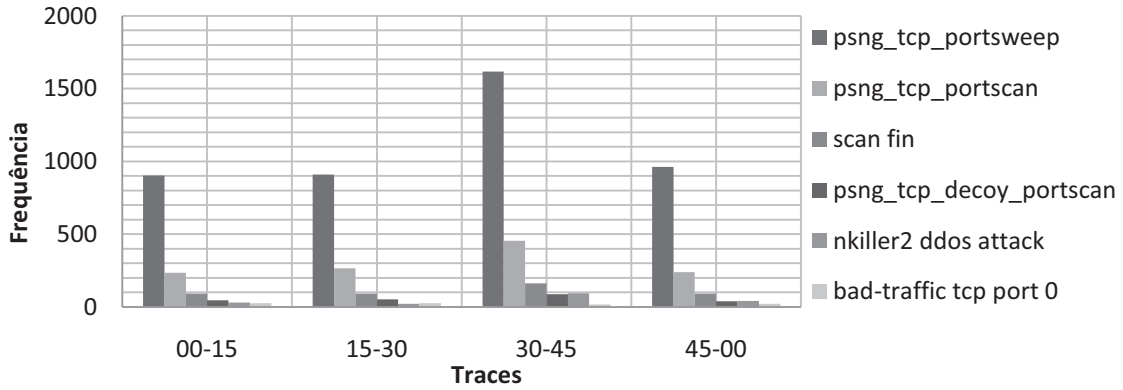

Figura 5. Distribuição de Frequência das Classes de Ataque de todos os traces MAWI

[1] http://www.caida.org/projects/ditl/

Para finalizar, a união de todas as dimensões de interesse (*class*, *srcIP* e *dstIP*) de todos os arquivos mostram que 2670 alertas foram considerados relevantes de um total inicial de 6598 alertas, uma redução de aproximadamente 60% no número de alertas.

7. Conclusão

Como discutido anteriormente, soluções colaborativas são caracterizadas pela necessidade de manipular grandes quantidades de alertas gerados por seus componentes. Para lidar com este problema, se faz necessário um esquema ou mecanismo para agregação desses alertas.

Este artigo apresentou o projeto e a implementação de uma solução capaz de converter alertas, em seus formatos originais, em formato IDMEF, agregando-os e, por fim, extraindo aqueles considerados significantes. De modo geral, a solução reduz o volume de alertas que precisam ser avaliados, de forma a permitir que um processo de correlação entre os alertas possa ser feito, aumentando o potencial de colaboração.

Para avaliar a solução proposta, foram realizados experimentos com tráfego real coletado no GPRT, tráfego malicioso do DARPA 2000 e tráfego real coletado no backbone MAWI. Como demonstrado, a solução tem a real capacidade de diminuir os alertas significativamente, mantendo um alto nível de precisão.

No que diz respeito a trabalhos futuros, pretende-se:

- Testar a ferramenta em novos cenários para melhor validar o protótipo;
- Implementar algum mecanismo de avaliação dos alertas considerados não relevantes, uma vez que pouca representatividade não significa falta de periculosidade;
- Projetar e implementar solução(ões) para avaliação de falso positivos e falsos negativos, o que melhorará a precisão da solução proposta.

Referências

Cuppens, F. (2001) "Managing alerts in a multi-intrusion detection environment", 17th Annual Conference on Computer Security Applications (ACSAC), páginas 22-31.

Debar, H., Curry, D., and Feinstein, B. (2007) "The Intrusion Detection Message Exchange Format (IDMEF)", RFC 4765, Março.

F. Maggi, M. Matteucci and S. Zanero Reducing false positives in anomaly detectors through fuzzy alert aggregation Information Fusion, 10(4):300-311. October 2009. DOI: 10.1016/j.inffus.2009.01.004.

Julicsh, K. (2003) "Mining Alarm Clusters to Improve Alarm Handling Efficiency", 17th Annual Conference on Computer Security Applications.

MAWI. MAWI Working Group Traffic Archive. http://mawi.wide.ad.jp/mawi. 2010.

MIT Lincoln Laboratory. (2000) “2000 DARPA Intrusion Detection Scenario Specific Data Sets”, http://www.ll.mit.edu/IST/ideval/data/2000/2000_data_index.html.

Snort. (2010) “Snort”, http://www.snort.org.

Soleimani, M. e Ghorbani, A. A. (2008) "Critical Episode Mining in Intrusion Detection Alerts", In *Proceedings of the Communication Networks and Services Research Conference (CNSR)*, páginas 157-164.

T. Zhihong, Q. Baoshan, Y. Jianwei and Z. Hongli. Alertclu: A Realtime Alert Aggregation and Correlation System. Proceedings of the International Conference on Cyberwolds, páginas 778-781. 2008.

Valdes A. and Skinner, K. (2001) "Probabilistc Alert Correlation", Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection, páginas 54-68, October 10-12, 2001

Xu, K., Zhang, Z. and Bhattacharyya, S. (2005) "Profiling internet backbone traffic: behavior models and applications", 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '05), Philadelphia, Pennsylvania, USA, 2005, páginas 169-180.

David L. Mills. Network Time Protocol (Version 3) Specification, Implementation and Analysis. IETF, Internet Standard RFC 1305, 1992