UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

SNIFFSTAT2DB - UMA FERRAMENTA PARA COLETA E ARMAZENAMENTO DOS CONTADORES DO TRÁFEGO DE UMA REDE

TRABALHO DE GRADUAÇÃO

Adler Hoff Schmidt

Santa Maria, RS, Brasil

2009

SNIFFSTAT2DB - UMA FERRAMENTA PARA COLETA E ARMAZENAMENTO DOS CONTADORES DO TRÁFEGO DE UMA REDE

por

Adler Hoff Schmidt

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientador: Prof. Raul Ceretta Nunes (UFSM)

Co-orientador: Msc. Tiago Perlin

Trabalho de Graduação N. 279

Santa Maria, RS, Brasil

2009

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

SNIFFSTAT2DB - UMA FERRAMENTA PARA COLETA E ARMAZENAMENTO DOS CONTADORES DO TRÁFEGO DE UMA REDE

elaborado por
Adler Hoff Schmidt

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof. Raul Ceretta Nunes (UFSM)
(Presidente/Orientador)

Profª Roseclea Duarte Medina (UFSM)

Prof. Antonio Marcos de Oliveira Candia (UFSM)

Santa Maria, 17 de Julho de 2009.

AGRADECIMENTOS

Inicialmente gostaria de agradecer aqueles que me ajudaram diretamente na construção desse trabalho. Ao professor Raul, por ter sempre me ajudado e orientado ao longo de todo o curso. A Fapergs pelo apoio financeiro que me deu durante esses seis meses que estive trabalhando. Ao amigo Tiago que também me ajudou muito no projeto desse trabalho. Também gostaria de agradecer à professora Oni e professora Andrea, por toda ajuda que sempre me deram ao longo dos anos em que fiz parte do grupo PET. Saibam que todos exerceram um grande papel na minha vida, obrigado por tudo. Também gostaria de agradecer à Fapergs pelo apoio financeiro que recebi durante a realização desse trabalho.

Dedico esse trabalho à minha família. Minha mãe Liane para com quem nem posso enumerar as coisas pelas quais sou grato, senão esse trabalho viraria um livro. Ao meu pai Canisio por todos os ensinamentos e sua teimosia que no fundo todos sabem é só preocupação conosco. E ao meu irmão que há tantos anos foi e será meu melhor amigo. A vó Edite que amo e sempre me dava conselhos que leu sobre tantos psicólogos.

Dedico também a todos os meus amigos que fiz ao longo desses anos de faculdade. Ao Rosca com quem me identifiquei logo no começo e virou um dos meus melhores amigos. Ao Magoo com quem tanto aprendi sobre multiprocessamento e ofuscação. Ao Gottin que há tantos anos vem sendo meu fã número um. Ao Cristiano e Eduardo, meus grandes amigos que fiz no intercâmbio e levarei para o resto da vida. Ao PL e as jogatinas épicas que um dia serão escritas liras sobre. E também a tantos outros amigos que fiz, e não foram poucos. Menos o Aaron Algarve, ainda o odeio.

O chef ruim é aquele que esconde a falta de talento na receita complicada.

AUTOR DESCONHECIDO

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

SNIFFSTAT2DB - UMA FERRAMENTA PARA COLETA E ARMAZENAMENTO DOS CONTADORES DO TRÁFEGO DE UMA REDE

Autor: Adler Hoff Schmidt
Orientador: Prof. Raul Ceretta Nunes (UFSM)
Co-orientador: Msc. Tiago Perlin

Local e data da defesa: Santa Maria, 17 de Julho de 2009.

Sistemas de detecção de intrusão são de grande importância para prevenção de ataques em redes de computadores. Eles realizam uma análise do tráfego de dados para constatar quando esses ataques estão ocorrendo. Porém necessitam de uma ferramenta que faça a coleta dos dados, os conhecidos *sniffers*, para que funcionem devidamente.

Esse trabalho apresenta o projeto de um *sniffer* que trate os dados capturados de maneira específica para uso de sistemas de detecção de intrusão por anomalias que utilizam séries temporais ou meros contadores do tráfego. Garantindo que o acesso às informações capturadas seja de uma maneira rápida e de baixo custo.

Palavras-chave: Coletores de tráfego de rede; sistemas de detecção de intrusão; gerência de redes.

ABSTRACT

Trabalho de Graduação
Undergraduate Program in Computer Science
Universidade Federal de Santa Maria

SNIFFSTAT2DB - A TOOL FOR COLECTION AND STORAGE OF THE NETWORK TRAFFIC COUNTERS

Author: Adler Hoff Schmidt
Advisor: Prof. Raul Ceretta Nunes (UFSM)
Coadvisor: Msc. Tiago Perlin

Intrusion detection systems are of great value for the prevention of attacks on computer networks. They make an analysis of the data traffic to take conclusions on when these attacks are occurring. But they need a tool that can do such data collection, the well known sniffers, so that they can properly work.

This work presents the project of a sniffer that deals with data capture on a specific way focused on the use for anomaly based intrusion detection systems that use time series or mere traffic counters. Making sure that the access to this captured information is in a fast and low cost way.

Keywords: network traffic colectors, intrusion detection systems, network managment.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

ACK	Acknowledgment
DoS	Denial Of Service
FIN	Finalize
Gmicro	Grupo de Microeletrônica
GTSeg	Gestão e Tecnologia em Segurança da Informação
ICMP	Internet Control Message Protocol
IP	Internet Protocol
IPFIX	Internet Protocol Flow Information Export
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
RFC	Request For Comments
SDI	Sistema de Detecção de Intrusão
SYN	Synchronization
TCP	Transmition Control Protocol
UDP	User Datagram Protocol
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Com a proliferação do uso de redes de computadores os mais diversos tipos de serviços foram incorporados a essa forma de acesso, devido à praticidade que propicia. Porém, usuários mal-intencionados sempre surgem com o intuito de causar um comportamento irregular a esse meio de comunicação. Na tentativa de evitar que ataques aconteçam, várias medidas foram e vem sendo tomadas. Uma delas é o uso de sistemas para a detecção de intrusão baseados em anomalias. Esses sistemas têm como objetivo prever um ataque através de uma comparação entre o comportamento atual de uma rede de computadores e o comportamento passado, em momentos que se sabia não estarem ocorrendo ataques (BARFORD et al., 2002).

Essa comparação se dá através de dados quantificados sobre todo tráfego que está ocorrendo. Porém, essa quantificação em si foge do escopo de um sistema de detecção de intrusão. Ela é realizada por um coletor de tráfego auxiliar, ferramenta que se responsabiliza pela escuta do que está acontecendo na rede e pelo armazenamento desses dados. Coletores de tráfego também são referenciados como *sniffers* (MCCLURE; SCAMBRAY; KURTZ, 1999).

A dependência existente entre esses dois sistemas frequentemente implica em problemas de compatibilidade. Nem sempre é possível usar o mesmo *sniffer* para sistemas que realizam a detecção de intrusão de forma própria. Cada uma das partes precisa estar em harmonia.

Um exemplo disse observou-se no desenvolvimento dos sistemas de detecção de intrusão por anomalias DIBSeT (LUNARDI et al., 2008) e DIBSeT-W (DALMAZO; NUNES; KOZAKEVICIUS, 2009), desenvolvidos dentro do grupo GTSeg/GMICRO, com o Internet Analysis System - IAS (POHLMANN; PROEST, 2006), que era utilizado como *sniffer*. Enquanto o IAS armazenava o máximo de informações possíveis sobre o tráfego

de uma rede visando uma análise *offline*, os detectores necessitavam de contadores de pacotes em certos intervalos de tempo. Isso resultava em um desempenho totalmente aquém do esperado.

Esse trabalho apresenta um projeto de *sniffer* que atende especificamente as necessidades de detectores de intrusão por anomalias que utilizam séries temporais , o caso do DIBSeT e DIBSeT-W, ou que necessitam só de contadores do tráfego da rede para realizar seu trabalho. Esse *sniffer* realiza toda a coleta e processamento dos dados de forma rápida para que o sistema de detecção possa operar em tempo real e é suportado nas mais diversas plataformas.

No capítulo 2 é apresentada uma série de conceitos que servem de fundamentação teórica para o que é apresentado nesse trabalho. Nele são abordados os Sistemas Detectores de Intrusão (SDI), os tipos dos mesmos, os ataques mais comuns de *Denial of Service* (DoS) e os coletores de tráfego. No capítulo 3 são apresentados os trabalhos anteriores já realizados, juntamente com um relato do impasse que eles estavam tendo e, finalizando, a apresentação de trabalhos relacionados que visam solucionar esse impasse. O capítulo 4 apresenta a solução proposta com todas suas peculiaridades. No capítulo 5 são apresentadas as especificações técnicas seguidas para que fosse possível tornar funcional a solução. O capítulo 6 apresenta resultados com uma discussão sobre eles e também possíveis trabalhos futuros.

1.1 Recursos Utilizados

Para realizar esse trabalho foi utilizada a rede do grupo de pesquisas GTSeg e GMICRO como testes para a captura dos pacotes de dados e contabilização deles. Computadores do mesmo grupo de pesquisa também foram utilizados para realizar a implementação do trabalho.

2 FUNDAMENTAÇÃO

Os Sistemas de Detecção de Intrusão (SDI) chamam bastante atenção de empresas e usuários de redes de computadores, pois provê uma melhora na segurança bastante procurada: de que algo maléfico poderá ser evitado previamente à sua ocorrência. Porém, para que essa garantia possa ser provida é necessário valer-se de ferramentas e métodos sempre atualizados.

Sendo que se trata de uma área com um bom grau de complexidade, podemos quebrar os conceitos aqui em partes: inicialmente apresentar um pouco mais sobre os SDI na seção 2.1, seguido de uma especificação um pouco maior no trato da diferença entre anomalias e assinaturas de redes na seção 2.2, para então apresentar as escolhas de filtros que esse trabalho realizou na seção 2.3 e, finalizando, tratar dos coletores de tráfego, que são de vital importância para um SDI, na seção 2.4.

2.1 O que é um SDI?

Um SDI nada mais é do que uma ferramenta que visa manter o funcionamento contínuo e correto de um sistema através da geração de avisos quando há a possibilidade de que falhas de segurança estejam sendo utilizadas. Com o grau de complexidade dos ataques que ocorrem hoje em dia e a quantidade dos mesmos, simplesmente não é mais viável valer-se de ferramentas que diminuam a probabilidade de sofrermos um ataque através da correção de falhas. Pois elas, em qualquer sistema, estão fadadas a aparecerem. Sendo assim, a idéia do uso de um SDI para detecção dessas falhas torna-se de vital importância para redes que prezem o seu funcionamento (DENNING, 1987).

Logo o objetivo inicial de uma ferramenta dessas é alertar a todos quando comportamentos anômalos estão ocorrendo. Para que a partir desse alerta os administradores analisem os acontecimentos e constatem uma possível nova brecha de segurança. Logi-

camente alguns SDI se encarregam de lidar com certos tipos de falhas automaticamente.

Existem várias maneiras de se operar para detectar algum comportamento anormal em uma rede. Uma forma é através da análise do comportamento durante um tempo presente e a comparação com o comportamento que existia previamente (que era considerado normal), na tentativa de identificar possíveis desvios (BARFORD et al., 2002). Outra seria através da busca de sequências de ações nitidamente caracterizadas como inválidas, registradas em uma base de dados que contém assinaturas dos ataques conhecidos (MIRKOVIC; REIHER, 2004).

Estes métodos são referenciados como detecção por anomalia e por assinatura, respectivamente. Algumas das diferenças características entre eles serão apresentadas na próxima seção.

2.2 Detecção de Intrusão por Anomalias ou por Assinaturas

Esse trabalho visa construir uma ferramenta de auxílio para SDIs que implementam o método de detecção de anomalias, porém, para um bom entendimento dessa proposta, é interessante esclarecer melhor a diferença entre esses dois métodos.

O significado da palavra anomalia é um evento irregular ou incomum que foge de uma lei ou regra padrão. Porém, cada rede possui as suas devidas peculiaridades, dessa maneira, o segredo para que esse tipo de método de detecção de intrusão funcione satisfatoriamente é conseguir estabelecer o que seria uma lei, ou uma regra padrão. Essa regra não é de tão fácil confecção, pois, afinal, a quantidade de tráfego de informações está em constante variância de acordo com o número de terminais conectados e programas utilizados. Um recurso que se pode utilizar é o comportamento prévio da rede, em um instante que se sabe não estar sofrendo um ataque. Assim é estabelecido o que seria um comportamento aceitável em um momento futuro e, quando esse momento chegar, constatarmos se tudo ocorreu dentro da previsão ou algum imprevisto está ocorrendo (BARFORD et al., 2002). Esse limite de comportamento aceitável trata-se de um limite dinâmico que está em constante variância.

Por outro lado temos os SDIs que trabalham com a detecção de assinaturas. Cada tipo de ataque a uma rede possui certa peculiaridade, uma característica que extrapola um comportamento normal e identifica o ocorrido como um ataque. Com uma base de dados construída, que possua uma lista das assinaturas de ataques conhecidos, é sempre

possível procurar nas estatísticas de tráfego da rede por essas peculiaridades. Caso seja encontrado, o propósito do SDI foi obtido e o sistema que ele defende sempre estará apto a lidar com ataques daquele comportamento (MIRKOVIC; REIHER, 2004). Porém será necessário sempre atualizar a base de dados das assinaturas, caso contrário esse método torna-se facilmente obsoleto.

2.3 Ataques de Redes de Corriqueiro Acontecimento

Com essa distinção entre dois tipos de SDI podemos dar um passo adiante e relatar sobre quais os dados da rede que desejaremos capturar e fornecer ao SDI para que ele faça sua análise. Afinal de contas existe um número muito grande de possibilidades de tipos de protocolos utilizados e, não obstante, ainda existem peculiaridades de cada protocolo que também podemos filtrar. O que torna necessário certas restrições para os filtros de captura.

Essa seção apresentará alguns dos protocolos mais observados pela maioria dos SDIs e apresentará uma breve justificativa sobre o porquê desse comportamento. Isso será de vital importância, pois o que for aqui apresentado será aplicado na solução proposta para diminuir a complexidade ao mesmo tempo que se captura os dados referentes aos ataques mais frequentes.

Três protocolos e três *flags* de um desses protocolos serão apresentados aqui nessa seção. Eles são organizados em subitens munidos da justificativa pela escolha de cada um e no mínimo um exemplo de ataque.

2.3.1 Pacotes TCP

Como o protocolo TCP/IP é vastamente utilizado, a probabilidade de um ataque ocorrer com o seu uso é grande. Um exemplo de ataque que pode ser citado são os do tipo *spoofing*, no qual um usuário imita um pacote TCP provindo de outro *host*, geralmente de confiança do destinatário, para realizar a comunicação com o seu alvo.

2.3.2 Pacotes UDP

Descrito na RFC 768 (POSTEL, 1980) esse protocolo permite que seja criado um datagrama dentro de um pacote IPv4 ou IPv6 e então seja enviado. Ele não possui nenhum dos mecanismos de segurança que o protocolo TCP implementa, sendo assim é necessário que as aplicações implementem esses mecanismos quando usarem o UDP. Porém, ele em

si é menos seguro fazendo com que vários ataques o utilizem. Nas próximas subseções teremos a exemplificação dos dois.

2.3.2.1 Ataques de Inundação UDP

Inunda um *host* remoto com pacotes UDP para as mais variadas portas na tentativa de causar um *Denial Of Service* (DoS) (HANDLEY; RESCORLA, 2006), momento no qual o alvo não está mais apto a responder pedidos pois ainda está processando os outros que recebeu. Assim o serviço que o alvo prestava não está mais disponível.

2.3.2.2 Ataque UDP de Diagnóstico de Portas

Similar ao anterior, porém nesse ataque são encaminhados pedidos de diagnóstico de uma certa porta. Mas o intuito ainda é o mesmo de causar um *Denial of Service* (DoS).

2.3.3 Pacotes ICMP

É parte do *Internet Protocol Suite*, como definido na RFC 792 (POSTEL, 1981). Tipicamente as mensagens ICMP são geradas em resposta a erros em datagramas IP ou para diagnósticos e propósitos de roteamento. Também podemos citar dois tipos de ataques que utilizam esse protocolo que serão apresentados nas próximas subseções.

2.3.3.1 Ataque de DoS ICMP

Pacotes com mensagens referentes a um destinatário inalcançável ou tempo de comunicação excedido são enviados pela rede na tentativa de desativar as comunicações existentes.

2.3.3.2 ICMP smurf

Enviam-se pacotes pedindo uma resposta de que tudo está funcionando para listas de *broadcast* de redes vulneráveis fazendo com que cada *host* dessas listas responda para a vítima. Assim tentando não só desabilitar o serviço que cada um poderia estar prestando (pois estarão ocupados respondendo os pedidos) como consumindo a banda da rede. Caracterizando dessa maneira um *Denial Of Service* (DoS) (HANDLEY; RESCORLA, 2006).

2.3.4 Pacotes TCP com a *Flag* ACK ativada

Esse tipo de mensagem simboliza um reconhecimento de que certo envio de pacotes foi recebido com sucesso. Um índice alto desses pacotes na rede pode indicar um DoS. Esses contadores também são utilizados para constatar um ataque SYN-ACK, que será explicado na seção 2.3.6.

2.3.5 Pacotes TCP com a *Flag* FIN ativada

Esses pacotes representam um pedido, de uma das pontas da conexão, para terminar a conexão existente. Um dos hosts envia uma mensagem FIN para o outro, que responde com uma ACK, e esse outro também envia uma FIN, também respondida, sinalizando a finalização da conexão. O ataque mais comum com essas mensagens é aquele nas quais várias requisições de conexão são feitas e logo são enviados pacotes FIN, sem transferência de dados. Dessa maneira tenta-se causar um DoS no intuito de sobrecarregar um alvo com várias requisições de conexão abertas e fechadas (HARRIS; HUNT, 1999).

2.3.6 Pacotes TCP com a *Flag* SYN ativada

Esses pacotes são caracterizados por realizar uma sincronização entre os *hosts* que irão estabelecer uma comunicação pelo protocolo TCP. Normalmente um pacote SYN é enviado, seguido de uma resposta SYN-ACK que reconhece o envio do SYN, e, finalizando, uma mensagem ACK é enviada. A partir desse momento a sincronização foi concluída. O ataque comumente visto é de inundação de pacotes SYN, tentando causar DoS: vários pacotes SYN são enviados à um alvo indicando que o SYN-ACK deverá ser enviado para outro *host* que não estava esperando. Quando os alvos alocam recursos antes do pacote ACK final, um DoS é causado, pois vários recursos são utilizados e jamais liberados pois a mensagem ACK final nunca chega. Existe uma vasta literatura na comunidade científica sobre esse ataque que pode ser vista nas publicações (MASELLI; DERI; SUIN, 2003), (PENG; LECKIE; RAMAMOHANARAO, 2007) e (LEVCHENKO; PATURI; VARGHESE, 2004). Para melhor visualização desse ataque temos a figura 2.1.

Com essa apresentação feita pode-se entender o porquê da escolha dos pacotes a serem contabilizados por esse *sniffer*. Essa escolha foi feita em conjunto com o trabalho do mestrando Tiago Perlin em cima do DIBSeT (LUNARDI et al., 2008), SDI que se concentra na tentativa de detectar ataques do tipo *Denial Of Service*, que transparece nas exempli-

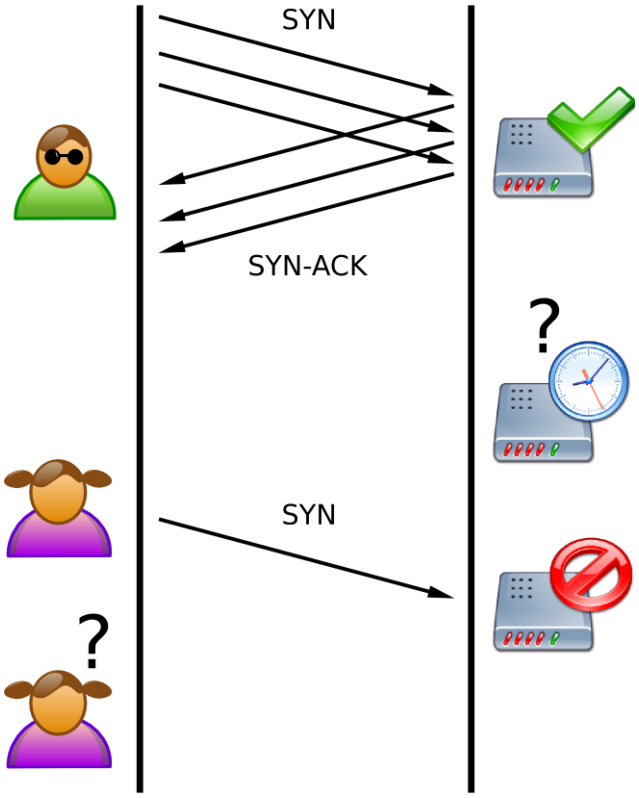

Figura 2.1: Ilustração do funcionamento de um ataque SYN-ACK (DAKE, 2006)

ficações das escolhas dos pacotes. Porém os trabalhos anteriores serão apresentados de maneira mais detalhada no próximo capítulo.

2.4 Coletores de Tráfego

Até o momento já se tem uma construção do que é um SDI, como ele pode funcionar e alguns tipos de pacotes frequentemente usados para ataques. Essa subseção, por sua vez, irá falar um pouco mais sobre os coletores de tráfego, responsáveis por fornecer os dados de entrada para a operação de um SDI.

Toda e qualquer rede possui um portão de entrada, chamado *Gateway*, demonstrado na figura 2.2, que é responsável por transmitir todas as informações de fora da rede para dentro e vice-versa. Esse *Gateway* não se faz ciente de todo o conteúdo que está sendo transportado, mas sim da forma como está. Essa forma se dá através de tipos de protocolos, *flags* que sinalizam algum estado alterado, portas que devem ser utilizadas e assim por diante.

Dessa maneira, torna-se possível utilizar um dispositivo que guarde informações sobre a forma que o tráfego está ocorrendo, realizando um catálogo dos protocolos que por ali

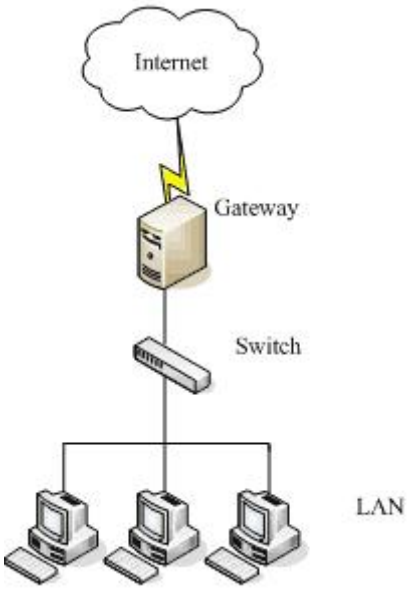

Figura 2.2: Estruturamento de uma rede com um *gateway* (LANCTRL, 2009)

transitam com as informações adicionais desejadas. Esse dispositivo se chama *Sniffer* (MCCLURE; SCAMBRAY; KURTZ, 1999), e nada mais é do que um coletor de tráfego. Esse mesmo dispositivo será o responsável por fornecer uma base de entrada para que o SDI possa realizar o que se propõe.

Ele será o responsável pela captura das informações de tráfego, porém o modo com que as armazena é uma peculiaridade que vai da implementação de cada *sniffer*. Enquanto alguns podem simplesmente guardar toda informação que capturam, outros podem realizar um trabalho mais elaborado em cima dessas informações, já guardando um dado de forma própria. E é essa maneira de armazenamento que se torna vital para a ferramenta que irá valer-se das suas informações. Sendo assim, sempre é desejável que o *sniffer* não guarde informações demasiadas e desnecessárias, que só causariam um desperdício de recursos, mas sim que armazene exatamente o que a aplicação precisa para atingir o seu objetivo.

Além disso, também podemos falar dos filtros que esses coletores de tráfego podem aplicar: filtros de captura e filtros de display. A diferença entre ambos é um tanto auto-explicativa devida sua nomenclatura: os filtros de captura são aqueles que se responsabilizam por determinar o que, de todo o tráfego que passa por definido ponto, deve ser capturado e os filtros de display realizam um trabalho de refinamento dessa informação para apresentar ao usuário de uma maneira que melhor convir. Existem várias ferramentas que realizam o trabalho de captura dos dados, dentre elas podemos citar o *tcpdump*

(TCPDUMP, 2009), que utiliza somente filtros de captura, e o *wireshark* (WIRESHARK, 2009), que fornece suporte à ambos os filtros. Ambas as ferramentas utilizam uma biblioteca em comum para realizar essa captura: a LIBPCAP (LIBPCAP, 2009). Essa é uma biblioteca utilizada por quaisquer programas interessados em realizar captura de tráfego de uma placa de rede, já que fornece os devidos meios para isso.

3 TRABALHOS ANTERIORES E SOLUÇÕES RELACIONADAS

Neste capítulo será visto no item 3.1 os trabalhos anteriores do SDI em desenvolvimento do GTSeg/Gmicro, através das versões de desenvolvimento DIBSeT e DIBSeT-W, com isso apresentado serão discutidos no item 3.2 todos os problemas que os SDIs previamente desenvolvidos tinham com a coleta de dados de tráfego em tempo real e, para finalizar, na seção 3.3 serão relatados trabalhos relacionados que já abordaram o problema que foi apresentado na seção 3.2.

3.1 Trabalhos anteriores do GMICRO

Para facilitar o entendimento essa seção será subdividida novamente em outras duas que irão fazer uma breve descrição do trabalho feito previamente pelos alunos Roben Castagna Lunardi com o DIBSeT na seção 3.1.1 e as modificações realizadas pelo aluno Bruno Lopes Dalmazo no DIBSeT-W na seção 3.1.2.

3.1.1 DIBSeT

O DIBSeT (LUNARDI et al., 2008) é um sistema de detecção de intrusão que visa encontrar anomalias nos contadores de tráfego da rede. Ele utiliza séries temporais do modelo ARIMA e o preditor de passos em séries temporais desenvolvido pelo mesmo orientador desse trabalho em sua tese de doutourado (NUNES, 2003). Com isso limites dinâmicos são estabelecidos para estipular-se a probabilidade de que uma anomalia se trata de um ataque quando os contadores ultrapassarem os mesmos.

Porém, a parte que é importante para entender o problema foi de que o uso do *Internet Analysis System* (IAS) (POHLMANN; PROEST, 2006), sistema de coleta de contadores do tráfego produzido pela Fachochschule Gelsenkirchen, não foi possível ser utilizado

para análise do tráfego em tempo real. Foi necessário usar o NTOP, ferramenta de análise do tráfego da rede que gera estatísticas, aliadas de métodos auxiliares que lidavam com os dados obtidos para realizar a análise.

3.1.2 DIBSeT-W

O DIBSeT-W (DALMAZO; NUNES; KOZAKEVICIUS, 2009) por sua vez tentou realizar um trabalho de filtragem dos alarmes gerados através do uso de *wavelets* para tentar obter um resultado de saída mais satisfatório. *Wavelet* é uma função matemática capaz de decompor uma função no domínio do tempo em diferentes escalas, de modo que seja possível uma análise da função nos domínios da frequência e tempo (STRANG, 1993).

Já o DIBSet-W por sua vez inicialmente utilizou a base de dados DARPA (DARPA, 2009) para validar a sua proposta. E, quando atingiu esse objetivo, utilizou os dados capturados pelo IAS (POHLMANN; PROEST, 2006). Porém o custo para acessar esse dado foi demasiadamente grande já que as tabelas geradas pelo IAS são de grande porte e armazenam muitas informações que frequentemente não eram utilizadas na construção das séries temporais. Sem mencionar no custo do processamento para criação dos contadores a partir dos dados brutos que eram encontrados nas tabelas.

Foram dois trabalhos complementares que atingiram cada um seu objetivo, porém um empecilho manteve-se entre ambos: a forma de obter-se osos contadores do tráfego da rede. Independentemente do caso, quando fosse feita uma análise em tempo real vários custos adicionais eram agregados ao utilizar as ferramentas que eram dispostas. Isso acontecia pois essas ferramentas não possuíam o objetivo original de provir sistemas como o DIBSeT e DIBSeT-W. Nasceu aqui um problema a ser resolvido neste trabalho.

3.2 O Problema do Coletor de Dados

Era necessária uma boa ferramenta para realizar *sniffing* que guardasse os contadores desejados e fornecesse uma via de rápido acesso. Esse *sniffer* deveria funcionar de tal maneira que os dados do tráfego fossem contabilizados e guardados nas tabelas do banco de dados. O que nos leva a duas possibilidades de como armazenar, a primeira podemos guardar toda a informação que conseguirmos obter (como destino e origem do pacote, informações do cabeçalho e assim por diante) e a segunda temos a possibilidade de sim-

plesmente quantizar todo o tráfego de acordo com os filtros que desejamos guardando só contadores.

Como o que ambos os SDIs realmente necessitavam como entrada eram os contadores, a decisão foi feita. As possibilidades de informações perdidas não seriam de relevância para os SDIs, pois eles somente necessitam como entrada contadores, logo o escopo em que fossem usados indicariam essas outras informações (como por exemplo calculando só os pacotes com destino a determinado IP ou então da sub-rede e assim por diante). E também com o uso de contadores o tempo de acesso das informações para processamento do SDI seria reduzido drasticamente. Terminando assim o esboço do que se procurava para solucionar esse problema apresentado: um *sniffer* que contabiliza certos pacotes em determinados intervalos de tempos e fornece esses dados para o SDI utilizá-los.

3.3 Soluções Relacionadas

Várias soluções existem para resolver esse problema. Algumas dessas realizam a coleta com coletores próprios, utilizando cada um o seu protocolo, porém a grande maioria utiliza o *NetFlow* (CLAISE, 2004), pois é um protocolo feito para diminuir o custo na captura dos dados sobre os pacotes transitando.

Dentre as soluções existem várias aplicações que utilizam o *NetFlow*, dentre os quais as ferramentas proprietárias (ADVENTNET, 2009) e (NETWORKS, 2009). Porém o foco para a resolução do problema apresentado na subseção anterior encontra-se no armazenamento dos mesmos em uma base de dados para que terceiros utilizem-nos, realizando algum processamento adicional. Um trabalho bem similar que merece ser referenciado constrói um modelo de armazenamento de fluxos de rede para análise de trafego e segurança (CORREA; PROTO; CANSIAN, 2008).

Nele foi usado o padrão IPFIX (IP Flow Information Export) (QUITTEK et al., 2004), cuja finalidade era estabelecer uma arquitetura para análise de tráfego, e o *NetFlow* RFC 3954 (CLAISE, 2004), criado pela *Cisco Systems*, para realizar a coleta dos dados de uma rede. O *NetFlow* funciona da seguinte maneira: ele está presente em roteadores ou *switches* da *Cisco Systems* e toda vez que ele recebe um pacote, analisa uma série de atributos do pacote e determina se ele é único ou similar a outros pacotes. Caso seja similar, ele o agrupa com os outros, para então armazenar essas informações em um *cache* que futuramente será exportado para algum coletor. Uma visualização mais fácil pode ser

vista na figura 3.1.

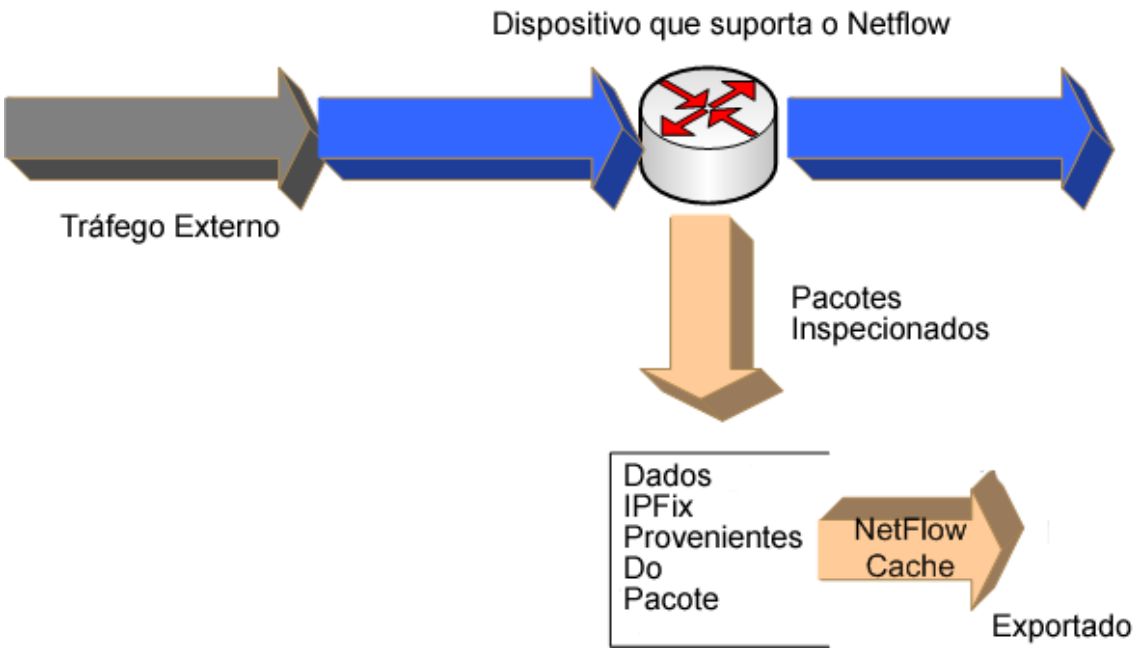

Figura 3.1: Ilustração do funcionamento do protcolo *NetFlow* (NETFLOWANALYSIS, 2009)

No caso, o *cache* era exportado para uma aplicação Java que executava em paralelo três threads com as respectivas responsabilidades:

- A primeira *thread* é responsável por receber esses dados provindos da cache do roteador;
- A segunda é responsável pelo tratamento dos dados recebidos e a inserção em uma tabela que possui os dados do tráfego dos últimos trinta minutos;
- A terceira lida com movimentação dos dados da tabela dos últimos trinta minutos para uma tabela que arquiva todos os dados de cada dia.

Dentro do trabalho realizado pela segunda thread vale destacar o trabalho de processamento de dados adicional que nela ocorre. Uma modificação da representação do número IP é feita para ocupar menos espaço dentro da base de dados. Na tentativa de utilizar a menor quantidade de espaço para armazenar uma linha da tabela.

4 PROPOSTA PARA UM NOVO COLETOR

Nessa seção é feita uma apresentação da proposta para coleta de dados do tráfego de uma rede, com o intuito de melhor servir um SDI que utiliza séries temporais (ou quantificadores do tráfego), na subseção 4.1. E em seguida, na subseção 4.2, as principais diferenças entre a solução aqui proposta com a do trabalho relacionado apresentado. A partir desse momento essa solução que será proposta será referenciada como SniffStat2DB.

4.1 Solução Proposta

São cinco pontos base, que são apresentados em seguida, para o entendimento da proposta. Novas seções foram criadas para esclarecer de forma mais gradativa o que está sendo proposto. Ao final, na seção 4.1.6 será demonstrado um fluxograma de funcionamento.

4.1.1 Captura do Tráfego

Utilizar uma ferramenta de escuta do tráfego da rede para realizar a captura dos dados ao invés de delegar essa tarefa a um *switch* ou roteador que forneça suporte ao protocolo *NetFlow*. Essa ferramenta pode tanto ser implementada diretamente com os recursos da biblioteca Libpcap (LIBPCAP, 2009) como ser uma ferramenta já pronta: Tcpdump (TCPDUMP, 2009) ou Wireshark (WIRESHARK, 2009).

Logicamente, dessa maneira a estrutura da rede ficará diferente do apresentado na figura 3.1, pois ou um computador realizando essa captura deverá estar situado entre a rede externa e interna ou o *switch* deverá mandar cópias do tráfego para uma máquina que irá capturá-los e descartá-los. Visualmente a representação resultaria na apresentada na figura 2.2 com a ferramenta funcionando dentro do *gateway*. O principal ponto positivo atingido por essa abordagem é que a utilização do SniffStat2DB não dependeria da rede

possuir um dispositivo que suportasse o *NetFlow*, mas sim só de um modelo de arquitetura de rede (que já é comumente utilizado). Optou-se pelo uso direto da biblioteca Libpcap (LIBPCAP, 2009) para captura dos dados da placa de rede, os motivos para essa escolha são esclarecidos no capítulo 5.

4.1.2 Processamento dos Dados Capturados

Nas seções 3.1.1 e 3.1.2 tivemos uma breve idéia de como se dá o funcionamento das duas versões do detector de intrusão DIBSeT. O que podemos perceber em comum é que ambos trabalham com dados quantizados do tráfego da rede em certo intervalo de tempo. Então os analisando e tirando as conclusões cabíveis.

Sendo assim, sabemos que a entrada realmente necessária são contadores dos tipos de tráfego que deseja analisar. Logo, não é necessário armazenar todos os dados do tráfego da rede, como é feito no contador de tráfego do IAS (POHLMANN; PROEST, 2006). Ao invés disso, pode ser feito um trabalho adicional de processamento dos dados antes de realizar o armazenamento. Nessa seção são utilizados os filtros de display, que foram citados na seção 2.4, para realizar esse processamento.

4.1.3 Repasse dos Dados Capturados

O repasse dos dados pode se dar de duas maneiras: envio direto através de *sockets* TCP/IP (HALL, 2009) e armazenamento em bancos de dados. O envio direto tem o propósito de ser a maneira mais rápida e de baixo custo para o repasse das informações capturadas. A sua principal vantagem é a conexão direta com o SDI, sem qualquer outro meio que possa agregar algum custo, porém a função de armazenar os dados passa a ser do SDI e não mais do coletor.

Para o armazenamento será usado um modelo de dados relacional que é descrito como um conjunto de relações (ELMASRI; NAVATHE, 2005). Esse conjunto são tabelas que possuem uma série de tuplas, linhas que representam um conjunto de valores, e atributos, que são as colunas com um significado para cada um dos valores da tupla.

Usando esse modelo é necessária a criação de uma sistemática para o armazenamento, constituída de duas tabelas. Da mesma forma que o trabalho de (CORREA; PROTO; CANSIAN, 2008) a primeira tabela é responsável pelo armazenamento dos contadores dos últimos trinta minutos e a outra guarda um arquivo com todas as informações sobre o dia. Essas duas tabelas respeitam a mesma sintaxe: as suas linhas representam o intervalo

de tempo que o usuário deseja analisar, armazenando o horário inicial, e as colunas os contadores de cada tipo de protocolo contabilizado. Dessa maneira em cada execução são criadas duas tabelas dentro da base de dados. Porém, tabelas adicionais serão criadas devido às portas que o usuário deseja contabilizar, o que será mais bem explicado na seção 4.1.5. Na figura 4.1 podemos ver como é a estrutura da tabela que conta todos os pacotes possíveis.

Name	Type	Primary	Not Null
TSMP	TIME	☑	☑
TCP	INTEGER	☐	☑
UDP	INTEGER	☐	☑
ICMP	INTEGER	☐	☑
TCPACK	INTEGER	☐	☑
TCPFIN	INTEGER	☐	☑
TCPSYN	INTEGER	☐	☑

Figura 4.1: Campos de uma tabela padrão

4.1.4 Sistemática de Configurações do SniffStat2DB

Cada SDI que vier a utilizar o cadastro dos contadores pode ter diferentes necessidades quanto à quais portas, protocolos, banco de dados, porta para comunicação com *sockets* e assim por diante desejam utilizar. Até mesmo um SDI idêntico pode ter seus objetivos alterados de acordo com a vontade do usuário. Dado esse conjunto de fatos, é importante uma maior flexibilidade do SniffStat2DB.

Essa flexibilidade é atingida com o uso de um arquivo de configuração que irá guardar as propriedades que o usuário deseja utilizar. Essas propriedades podem ser divididas em quatro grupos:

- Propriedades da aplicação auxiliar que realizará o processo de *sniffing*;
- Propriedades do banco de dados que irá guardar as informações adquiridas;
- Propriedades dos filtros a serem aplicados: porta da rede que será escutada, quais dos filtros que foram citados na seção 2.3 serão ativados, qual o intervalo de análise, quantidade de intervalos e se o modo promíscuo (no qual se escuta inclusive as comunicações internas da rede) será ativado ou não;

- Propriedades para configuração do serviço de *sockets* TCP/IP.

A partir disso é utilizada uma formatação genérica para armazenamento dos dados dessas propriedades, visando a maior compatibilidade possível para que terceiros alterem esses dados. A escolha sobre qual formatação para armazenamento dos dados foi pelo uso de Extensible Markup Language (XML) (XML, 2009) e a razão para o mesmo será melhor esclarecido no capítulo 5.

4.1.5 Execução de Múltiplas Instâncias

Utilizando-se o modelo para armazenamento proposto podemos perceber que em lugar algum está armazenado em qual porta ocorreu a coleta dos dados. Isso ocorre, pois se buscou o máximo em rapidez no momento de acesso para a informação armazenada, assim o SniffStat2DB deverá ser instanciado para cada porta que se deseja filtrar, ou uma só vez se é desejado contabilizar todas. Isso acarretará em uma quantidade maior de tabelas, mas será compensado por dois fatos:

- Tamanho das tabelas: o tamanho será consideravelmente pequeno, pois elas irão armazenar somente os contadores, que são só dados na forma de inteiros com a possibilidade máxima de o usuário querer escutar os seis tipos de filtros estipulados. Sendo assim no pior caso cada linha guardaria 24 bytes de informação, relativa à contagem, mais a representação de um horário;

- Dimensão das tabelas: com a inserção de um atributo relativo à qual porta cada um dos contadores é relacionado é adicionada uma nova dimensão a todas as tabelas. Isso acarreta em um número muito maior de linhas em cada tabela, o que, por sua vez, acarreta em um custo maior no momento em que se deseja encontrar certa informação dentro da tabela desejada.

Na figura 4.2 podemos ver a quantidade de ataques *scans* (LEE; ROEDEL; SILENOK, 2003) separados por portas entre Janeiro e Março de 2009. Como podemos ver, existe um amplo uso de certas portas específicas para os ataques. Dessa maneira, com a separação, cada tabela irá guardar especificamente o que o usuário quer acessar. Sem necessidade de trabalho adicional no momento de acesso às informações.

Figura 4.2: Ataques scan por portas entre Janeiro e Marco de 2009 (CERT.BR, 2009)

4.1.6 Fluxograma de Funcionamento

Para facilitar o entendimento de como é o funcionamento do SniffStat2DB pode-se observar o fluxograma apresentado na figura 4.3. Esse fluxograma lida de forma genérica com a sistemática de funcionamento e as especificações serão apresentadas no capítulo 5. Vale a ressalva de que haverão duas *threads* principais: a de coleta e a de armazenamento dos dados. Dentro da de armazenamento dos dados, as comparações e chamadas a cada método de armazenamento também ocorrerão em paralelo pois são independentes.

4.2 Comparações com o Trabalho Relacionado

Como pode ser visto na solução proposta existem algumas similaridades com o trabalho (CORREA; PROTO; CANSIAN, 2008), porém também são acrescentadas várias sistemáticas de funcionamento diferenciadas. Essa seção serve para ressaltar essas diferenças, assim demonstrando uma relevância no que aqui foi feito em relação ao que já está pronto e divulgado.

A primeira diferença está no fato de que não é usado o protocolo *NetFlow* (CLAISE, 2004). Como já foi explicado anteriormente esse protocolo necessita arbitrariamente de um *switch* ou roteador que dê o devido suporte a esse protocolo. Porém nem todas as redes possuem tal artefato, o que acaba criando uma restrição física para o uso. Com a proposta do SniffStat2DB não é necessário possuir-se um artefato que suporte o *NetFlow* e basta utilizar um computador de *gateway*. O que normalmente já é utilizado em redes de porte médio ou grande como *firewall*, bastando assim executar o SniffStat2DB nessa máquina.

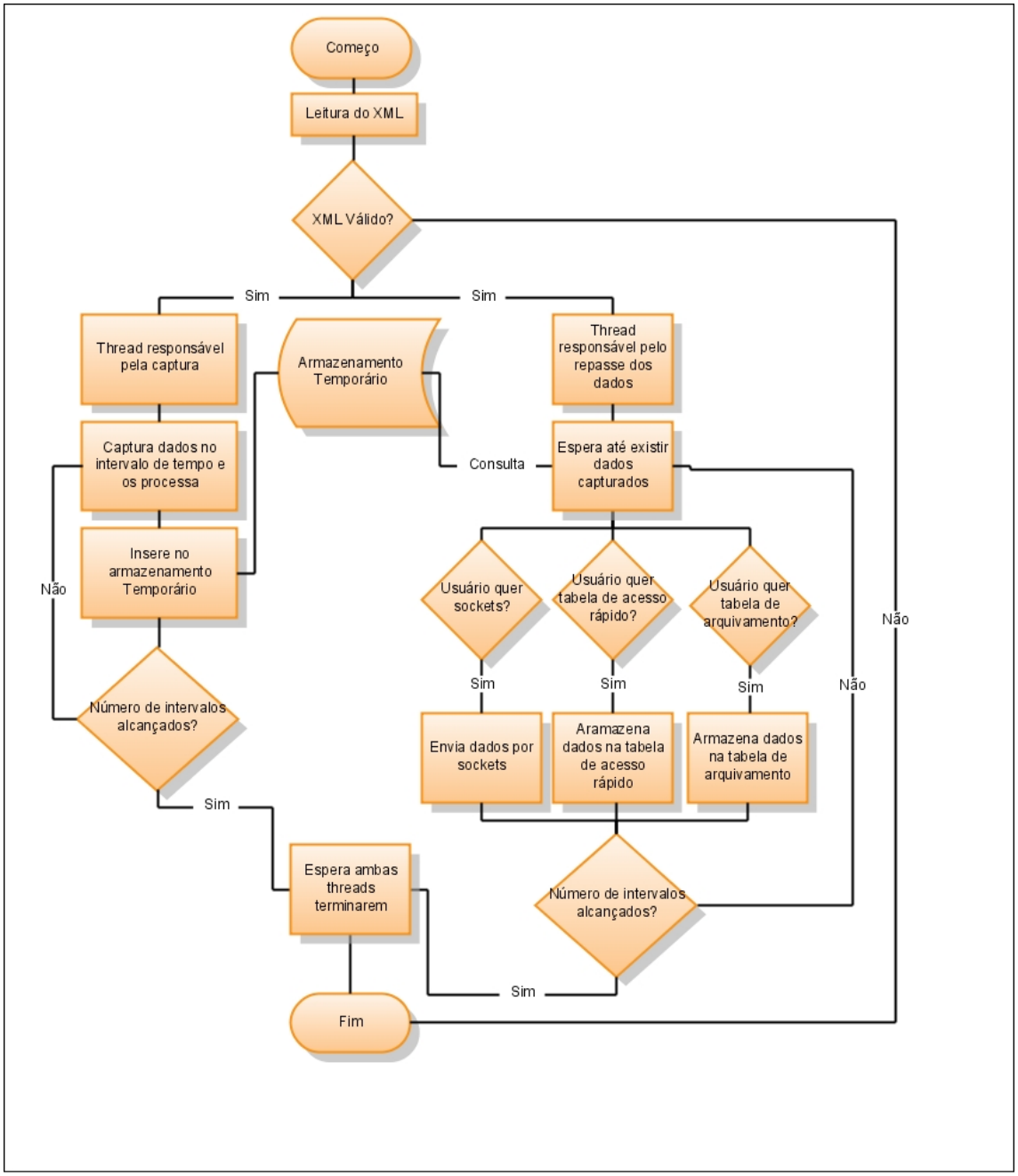

Figura 4.3: Fluxograma com o funcionamento do SniffStat2DB

A segunda diferença é a possibilidade do uso de *sockets* TCP/IP. Ponto de grande importância para o acesso rápido das informações, pois todo o custo relacionado ao acesso ao banco de dados é cortado fora. Deixando somente o custo de transferência de dados pela rede, caso o SDI e o SniffStat2DB estejam em máquinas diferentes, ou nem isso, caso estejam na mesma máquina.

A terceira diferença se dá na forma em que os dados são armazenados. Enquanto no SniffStat2DB filtros de display são utilizados para processar a informação da coleta

antes do armazenamento, no trabalho (CORREA; PROTO; CANSIAN, 2008) os dados são armazenados diretamente. Assim quando SDIs desejam trabalhar meramente com contadores o sistema gerenciador do banco de dados necessita trabalhar toda a informação para então retorná-la, ao contrário do SniffStat2DB que simplesmente retorna os dados guardados na tabela, economizando uma quantidade de recursos. Também vale lembrar o fato de que no modelo de armazenamento aqui proposto a dimensão de cada tabela foi drasticamente diminuída com a proposta da criação de várias tabelas.

A quarta diferença é o arquivo de configuração da execução da aplicação, inexistente no trabalho relacionado, que permite uma diminuição na quantidade de dados adquiridos visando excluir um eventual desperdício de recursos. Dessa forma somente será armazenado nas bases de dados o que o cliente realmente irá utilizar em um momento posterior.

5 IMPLEMENTAÇÃO DO SNIFFSTAT2DB

Nesse capítulo são apresentados todos os detalhes da implementação realizada. Ele está divido em três subseções: entrada de dados no SniffStat2DB, particularidades do processamento e saída de dados. Como um dos objetivos também era realizar uma ferramenta que desse o devido suporte entre plataformas heterogêneas, a construção do código se deu com a linguagem Java.

5.1 Entrada de Dados

A forma de interação de entrada com o SniffStat2DB se dá de duas maneiras: a alteração do arquivo de configuração, descrito na subseção 4.1.4, e a entrada direta, prévia à execução do programa. Na entrada direta são repassados três dados: a localização do arquivo de configuração, quais dos dispositivos disponíveis deseja-se filtrar e a senha para o banco de dados (caso deseja-se utilizá-lo). Para representação dos dados de configuração utilizou-se a *Extensible Markup Language* (XML) (XML, 2009) que provê uma forma de guardar os dados que se deseja através de *tags*, indicativos do início e final de certo tipo de dado. Dessa forma uma regra para a sintaxe do XML é criada e, obedecendo ela, qualquer terceiro pode alterar o comportamento do SniffStat2DB ao alterar o conteúdo do arquivo de configuração XML.

Um exemplo dos dados que o usuário terá que fornecer podem ser vistos na figura 5.1. Eles são quatro grupos de informação necessários: o do *sniffer*, da base de dados, dos filtros e dos *sockets*. O *sniffer* precisa saber qual porta ele deve filtrar, o intervalo de análise (deve contar os pacotes por quantos segundos) e a quantidade desses intervalos (ou um valor igual a -1 para ser interminável). A base de dados precisa das seguintes informações: o nome do banco que está sendo usado (será mais bem explicado na subseção 5.2.2), o nome do *host* em que a base de dados está, a porta para acesso, a localização da

base dentro da máquina hospedeira, o usuário para acesso, o desejo de utilizar as tabelas de acesso rápido, um nome especial para essa tabela, o desejo de utilizar as tabelas de arquivamento e um nome peculiar para a mesma.

Os filtros são campos binários que representam o desejo do usuário em utilizar, ou não, cada um dos filtros possíveis. E, por fim, os dados referentes aos *sockets*, ou seja, uma representação binária caso deseja-se criar um servidor de *sockets* que envie os dados capturados e, caso positivo, qual a porta que será utilizada para fazer a comunicação.

```
- <sniff2DBData xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ns0="http://xml.netbeans.org/schema/XMLProperties"
    xsi:schemaLocation="http://xml.netbeans.org/schema/XMLProperties XMLProperties.xsd">
  - <Sniffer>
      <port>8080</port>
      <intervalOfAnalysis>2</intervalOfAnalysis>
      <ammountOfIntervals>3</ammountOfIntervals>
    </Sniffer>
  - <Database>
      <dbName>firebird</dbName>
      <hostName>localhost</hostName>
      <port>3050</port>
      <location>C:/Documents and Settings/Tulkas/My Documents/SNIFFER.FDB</location>
      <username>SYSDBA</username>
      <statusFat>1</statusFat>
      <fatTableName>FAT</fatTableName>
      <statusArchive>1</statusArchive>
      <archiveTableName>hoff</archiveTableName>
    </Database>
  - <Filters>
      <tcp>1</tcp>
      <udp>1</udp>
      <icmp>1</icmp>
      <tcpAck>1</tcpAck>
      <tcpFin>1</tcpFin>
      <tcpSyn>1</tcpSyn>
    </Filters>
  - <Sockets>
      <status>0</status>
      <port>7000</port>
    </Sockets>
</sniff2DBData>
```

Figura 5.1: Arquivo XML válido para a configuração do SniffStat2DB

5.2 Particularidades do Processamento

Essa seção é dividida em duas subseções: a primeira fala da captura dos dados do tráfego e a segunda em como é o armazenamento dos mesmos para retorno ao cliente. Não só a apresentação está assim dividida como também a execução. Duas *threads* são criadas, uma com a função de captura e outra com o repasse, para que operem em paralelo em certos momentos. Isso serve para evitar que a *thread* de captura tenha que esperar pela de repasse realizar todo o seu trabalho. Dessa maneira o atraso que existe de 0,015 segundos só ocorre no primeiro ciclo de captura. Dentro da *thread* de armazenamento existe uma nova divisão para execução em paralelo de cada um dos métodos de armazenamento, já que são independentes.

5.2.1 Captura dos Dados

Inicialmente foi construído o SniffStat2DB com o uso da ferramenta (WIRESHARK, 2009). Porém, o desempenho obtido não foi satisfatório já que o tempo perdido aplicando os filtros de *display* era em média de 0.5 segundos (período em que o tráfego da rede não era contabilizado). Então essa iniciativa anterior foi descarta e realizou-se um trabalho de implementação própria com o uso da biblioteca auxiliar jNetPcap (JNETPCAP, 2009), um invólucro da Libpcap (LIBPCAP, 2009) que dá suporte à linguagem Java além de funcionalidades que a Libpcap não provê.

Dessa nova forma os filtros de *display* são aplicados ao mesmo tempo em que aplicam-se os de captura, fazendo desaparecer o custo de 0.5 segundos que o (WIRESHARK, 2009) tinha. O produto final da captura dos dados são os contadores de cada um dos pacotes que o cliente optou por filtrar. Esse produto é armazenado em uma estrutura de dados temporária, que será utilizada pela *thread* responsável pelo repasse dessas informações.

5.2.2 Armazenamento dos Dados

O armazenamento das informações é apresentado em três seções: transferência de dados por *sockets*, consulta à tabela de acesso rápido e consulta às tabelas de armazenamento. A segunda e terceira seções relatam trabalhos relacionados com um banco de dados e, visando uma maior flexibilidade, o SniffStat2DB optou por não tornar a escolha pelo Sistema Gerenciador de Banco de Dados (SGBD) (ELMASRI; NAVATHE, 2005) arbitrária. Assim, ele fornece uma escolha entre três que são vastamente usados e grátis: Firebird (FIREBIRD, 2009), MySQL (MYSQL, 2009) e PostgreSQL (POSTGRESQL, 2009), sendo que a escolha é feita dentro do arquivo de configuração XML.

5.2.2.1 Transferência de dados por sockets

A transferência em si não é uma forma de armazenamento dos dados, mas sim uma delegação da responsabilidade de realizar o mesmo para quem receber esses dados. O uso de um servidor *socket*, para envio direto dos dados a partir da captura, tem como principal objetivo cortar custos. Esse servidor aguarda a conexão de um cliente e, a partir dae, inicia a execução do SniffStat2DB. A partir desse momento inicia-se um ciclo, de tempo e quantidade de ocorrências estipulados pelo arquivo de configuração, em que o

registro do tempo em que iniciou a captura e os devidos contadores são enviados um por um. Os valores especiais estipulados são os seguintes: o número -1 para sinalizar que um conjunto de captura foi totalmente enviado e o número -2 que simboliza o final da execução do SniffStat2DB. Esses valores reservados servem para controle por parte do cliente.

5.2.2.2 Armazenamento na tabela de acesso rápido

A tabela de acesso rápido é outra opção para clientes, o seu princípio e organização já foram explicados e a sua implementação implicou na mera tarefa de manutenção dos dados já presentes antes da inserção. Ou seja, remover as linhas com marcador de tempo maior do que 30 minutos passados do momento da captura. Assim mantendo o princípio de poucas e recentes linhas com informação. Como cada instância do SniffStat2DB será responsável por uma porta específica, o número da porta que cada tabela representa está escrito juntamente com o nome escolhido pelo usuário para essa tabela. Quanto aos dados que ela guardará pode-se ter uma visualização na figura /reffig:3seg, que é uma tabela gerada em intervalos de análise de 3 segundos e que filtrava todos os pacotes possíveis.

TSMP	TCP	UDP	ICMP	TCPACK	TCPFIN	TCPSYN
Click here to define a filter						
12:09:16	13	0	0	10	4	3
12:09:19	24	0	0	24	6	3
12:09:22	8	0	0	6	0	4
12:09:25	10	0	0	8	4	0
12:09:28	11	0	0	8	2	4
12:09:31	10	0	0	6	0	7
12:09:34	27	0	0	26	8	2
12:09:37	1	0	0	0	0	1
12:09:40	8	0	0	8	2	1
12:09:43	36	0	0	32	8	8

Figura 5.2: Exemplificação dos dados armazenados em intervalos de 3 segundos

5.2.2.3 Armazenamento na tabela de arquivamento

O arquivamento serve principalmente para futuro uso *offline*. A implementação dessa tabela se deu de tal forma que uma tabela nova é criada para representar cada dia. No

nome da tabela constam as informações referentes à data e a porta que ela representa. Dentro dela estão todos os intervalos do dia que foi analisado. Dessa maneira cada ciclo que transpor um dia de análise, criará uma nova tabela.

5.3 Saída de Dados

A saída de dados está intimamente relacionada com a forma com que eles são armazenados. A primeira opção se dá através da construção de um cliente que faça uma conexão através de um *socket* com o SniffStat2DB. As vantagens dessa abordagem já foram esclarecidas e na figura 5.3 pode-se ver um exemplo de um cliente criado na linguagem Java.

```
public static void main(String[] args) throws IOException {
    Socket client = null;
    DataInputStream in = null;
    try {
        client = new Socket("localhost", 7000);
        in = new DataInputStream(client.getInputStream());
    } catch (UnknownHostException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    } catch (IOException ex) {
        Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
    }
    int c;
    do {
        System.out.println("Timestamp: " + in.readUTF());
        System.out.println("tcp: " + in.readInt());
        System.out.println("udp: " + in.readInt());
        System.out.println("icmp: " + in.readInt());
        System.out.println("tcp ack: " + in.readInt());
        System.out.println("tcp fin: " + in.readInt());
        System.out.println("tcp syn: " + in.readInt());
        c = in.readInt();
        System.out.println("c: " + c);
    } while (c != -2);
    in.close();
    client.close();
}
```

Figura 5.3: Código em linguagem Java para criação de um cliente *Socket*

Outra opção se dá através de consultas SQL às tabelas de armazenamento, tanto de acesso rápido como a de arquivamento. Na figura 5.4 tem-se o exemplo de uma consulta que irá retornar todas as linhas da tabela de acesso rápido.

```
SELECT * FROM FAST_ACCESS_P23;
```

Figura 5.4: Exemplo de consulta SQL para acessar os dados de uma tabela de acesso rápido que filtrou a porta 23

6 RESULTADOS OBTIDOS

Os resultados obtidos com o SniffStat2DB estão diretamente relacionados com dois fatores: o custo para repasse dos dados capturados e o custo para armazenamento nas tabelas de dados. Quanto ao custo de repasse, temos o acesso por comunicação de *sockets* e um acesso à base de dados.

Uma troca de mensagens por *sockets* entre a mesma máquina ocorreu um atraso médio de 0,015 segundos no envio do primeiro intervalo de captura. Porém, do segundo em diante esse atraso virou insignificante (menor do que 1 milésimo de segundo), com o SDI recebendo os dados quase que instantaneamente. Isso se deu graças à arquitetura com execução em paralelo.

Quanto à escalabilidade do SniffStat2DB não foram noticiadas eventuais sobrecargas ou perca de dados em redes de computadores de maior porte, como é o caso do tráfego da rede da UFSM. Grande parte do mérito desse fato é o uso direto da Libpcap (LIBPCAP, 2009). Um problema de escalabilidade evidente que pode ocorrer é relacionado com o uso de muitos servidores *sockets* para cada porta que estava sendo filtrada. Isso causaria a utilização de várias portas que também podem ser utilizadas por outros serviços. Essa possibilidade não foi testada, pois a utilização de uma quantidade grande de portas foge da proposta do servidor *socket*, que é acessar as informações principais de forma menos custosa possível. Como a relevância está diretamente relacionada com o interesse do usuário e a porta utilizada para comunicação, raramente todas as portas de uma rede terão o mesmo grau de importância e uma execução híbrida pode ser atingida, em que contadores de algumas portas são obtidos por *sockets* e outros por acesso ao banco de dados. Dessa maneira, com o conhecimento do administrador de quais portas são menos utilizadas o desempenho não será comprometido.

O acesso ao banco de dados foi realizada uma comparação com o IAS (POHLMANN;

PROEST, 2006) utilizando-se o MySQL (MYSQL, 2009), que é o único suportado pelo IAS. O IAS realiza o armazenamento dos dados do tráfego em três tabelas e em intervalos de cinco minutos, da mesma maneiras foi construída uma tabela equivalente com o SniffStat2DB em intervalos de 5 segundos. A partir disso foram realizadas dez consultas para medição de tempo. Enquanto o IAS retornava somente um tipo de pacote (contadores TCP ou UDP ou ICMP e assim por diante) por consulta, fazendo necessário seis consultas para obter-se todos contadores, o SniffStat2DB retornava todos em uma só consulta. O tempo levado para conexão ao banco de dados foi ignorado, somente contabilizou-se o tempo para realizar a consulta e obter os contadores. Na tabela 6.1 estão os dados do IAS e na tabela 6.2 estão os do SniffStat2DB.

8.882
8.902
8.932
8.910
8.929
8.896
8.906
8.900
8.929
8.920
Média: 8.9106

Tabela 6.1: Medidas de tempo de acesso do IAS em segundos

0.125
0.109
0.125
0.125
0.109
0.125
0.110
0.125
0.110
0.109
Média: 0.1172

Tabela 6.2: Medidas de tempo de acesso do SniffStat2DB em segundos

Pode-se ver uma diferença significativa entre os dois e vale ressaltar novamente o fato de que com o IAS era necessário um intervalo de tempo daqueles para cada pacote que deseja-se obter. Os outros sistemas gerenciadores de banco de dados não foram compara-

dos já que somente o SniffStat2DB os suporta.

7 CONCLUSÃO

A problemática existente na compatibilidade entre os sistemas detectores de intrusão e os respectivos coletores de tráfego já existe faz muito tempo. Com iniciativas de utilização de diferentes métodos para a realização da detecção, a demanda por novos coletores surge. Esse trabalho apresentou uma proposta para um novo coletor que teve a necessidade por ela vinda do desempenho precário obtido pelo SDI desenvolvido no GTSeg/Gmicro utilizando o coletor de tráfego IAS (POHLMANN; PROEST, 2006). A solução apresentada atendeu a demanda e forneceu um coletor de dados com baixíssimo custo para consulta a seus dados e sem necessidade de processamento adicional.

Três formas de acesso foram criadas: *sockets*, tabelas de acesso rápido e tabelas de arquivamento. A primeira delas com grande destaque já que o seu custo de resposta é muito menor que os outros. E as duas últimas soluções um pouco mais usuais, pois residem em lugar comum, mas seguro.

A utilização de um arquivo para configuração foi outro ponto interessante, pois deu uma ótima flexibilidade para o coletor. Os filtros de captura podem ser alterados e a maneira com que a conexão TCP/IP por *sockets* funciona também. Além disso, não só as propriedades de uma base de dados, como o SGBD em si podem variar. Isso sem contar a possibilidade dos intervalos de análise e quantidade serem configuráveis. Dessa maneira tentando abranger a maior quantidade possível de usuários.

7.1 Trabalhos Futuros

Como o objetivo desse trabalho de prover um serviço para sistemas que necessitam dos contadores do tráfego da rede e que tivesse acesso rápido aos dados foi atingido, o trabalho futuro visado é a ampliação das possibilidades de filtragem. Abrangendo uma quantidade maior de protocolos a serem filtrados e propriedades dos mesmos poderá fazer

com que mais clientes possam utilizar o SniffStat2DB. Provendo um melhor serviço aos sistemas detectores de intrusão.

REFERÊNCIAS

ADVENTNET. **ManageEngine NetFlow Analyzer**. Disponível em: http://www.manageengine.com/products/netflow/index.html Acesso em: Junho de 2009.

BARFORD, P.; KLINE, J.; PLONKA, D.; RON, A. A Signal Analysis of Network Traffic Anomalies. **Proceedings of ACM SIGCOMM Internet Measurement Workshop 2002**, [S.l.], 2002.

CERT.BR. Disponível em: http://www.cert.br/stats/incidentes/2009-jan-mar/scan-portas.html Acesso em: Julho de 2009.

CLAISE, B. **RFC3954 - Cisco Systems NetFlow Services Export Version 9**. Disponível em: http://www.ietf.org/rfc/rfc3954.txt Acesso em: Junho de 2009.

CORREA, J. L.; PROTO, A.; CANSIAN, A. M. Modelo de Armazenamento de Fluxos de Rede para Análises de Tráfego e de Segurança. **VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**, [S.l.], 2008.

DAKE. Disponível em: http://en.wikipedia.org/wiki/File:Tcp_synflood.png Acesso em: Junho de 2009.

DALMAZO, B. L.; NUNES, R. C.; KOZAKEVICIUS, A. J. Analisador de Alarmes de Tráfego de Redes Através de Wavelets. **XXXII Congresso Nacional de Matemática Aplicada e Computacional**, [S.l.], 2009.

DARPA. Disponível em: http://www.ll.mit.edu/mission/communications/ist/ Acesso em: Junho de 2009.

DENNING, D. E. An Intrusion-Detection Model. **IEEE Transactions on Software Engineering**, [S.l.], 1987.

ELMASRI, R. E.; NAVATHE, S. **Sistemas de Banco de Dados**. [S.l.: s.n.], 2005.

FIREBIRD. Disponível em: http://www.firebirdsql.org/ Acesso em: Julho de 2009.

HALL, B. **Beej's Guide to Network Programming**. [S.l.: s.n.], 2009.

HANDLEY, M.; RESCORLA, E. **Internet Denial-of-Service Considerations**. Disponível em: http://tools.ietf.org/html/rfc4732 Acesso em: Junho de 2009.

HARRIS, B.; HUNT, R. TCP/IP Security Threats and Attack Methods. **Computer Communications**, [S.l.], v.22, n.10, p.885–897, Junho 1999.

JNETPCAP. Disponível em: http://jnetpcap.com/ Acesso em: Julho de 2009.

LANCTRL. Disponível em: http://en.lanctrl.com/awall/manual/chapter3.htm Acesso em: Julho de 2009.

LEE, C. B.; ROEDEL, C.; SILENOK, E. Detection and Characterization of Port Scan Attacks. **CSE 222A: Computer Communication Networks**, [S.l.], 2003.

LEVCHENKO, K.; PATURI, R.; VARGHESE, G. On the Difficulty of Scalably Detecting Network Attacks. **CCS-ACM**, [S.l.], 2004.

LIBPCAP. Disponível em: http://www.tcpdump.org/ Acesso em: Junho de 2009.

LUNARDI, R.; DALMAZO, B. L.; AMARAL Érico; NUNES, R. C. DIBSeT: um detector de intrusão por anomalias baseado em séries temporais. **Anais do VIII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais**, [S.l.], 2008.

MASELLI, G.; DERI, L.; SUIN, S. Design and Implementation of an Anomaly Detection System: an empirical approach. **Proceedings of Terena Networking Conference (TNC 03)**, [S.l.], Maio 2003.

MCCLURE, S.; SCAMBRAY, J.; KURTZ, G. **Hacking Exposed**: network security secrets and solutions. 5th.ed. [S.l.: s.n.], 1999. 280-281, 369, 383-386p.

MIRKOVIC, J.; REIHER, P. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. **ACM SIGCOMM Computer Communications Review**, [S.l.], 2004.

MYSQL. Disponível em: http://www.mysql.com/ Acesso em: Julho de 2009.

NETFLOWANALYSIS. Disponível em: http://www.netflowanalysis.net/NetFlow.png Acesso em: Julho de 2009.

NETWORKS, F. **NetFlow Tracker**. Disponível em: http://www.flukenetworks.com/fnet/en-us/products/NetFlow+Tracker/ Acesso em: Junho de 2009.

NUNES, R. C. Adaptação dinâmica do timeout de detectores de defeitos através do uso de séries temporais. **Programa de Pós-Graduação em Computação: Tese de Doutourado Universidade Federal do Rio Grande do Sul, Porto Alegre**, [S.l.], 2003.

PENG, T.; LECKIE, C.; RAMAMOHANARAO, K. Survey of Network-Based Defense Mechanisms Countering the DoS and DDoS Problems. **ACM Computing Surveys**, [S.l.], v.39, n.1, 2007.

POHLMANN, N.; PROEST, M. **Internet Early Warning System**: the global view. [S.l.: s.n.], 2006. 377-286p.

POSTEL, J. **RFC768 - User Datagram Protocol**. Disponível em: http://tools.ietf.org/html/rfc768 Acesso em: Junho de 2009.

POSTEL, J. **RFC792 - Internet Control Message Protocol**. Disponível em: http://tools.ietf.org/html/rfc792 Acesso em: Junho de 2009.

POSTGRESQL. Disponível em: http://www.postgresql.org/ Acesso em: Julho de 2009.

QUITTEK, J.; ZSEBY, T.; CLAISE, B.; ZANDER, S. **RFC3917 - Requirements for IP Flow Information Export**: ipfix. Disponível em: http://www.ietf.org/rfc/rfc3917.txt Acesso em: Junho de 2009.

STRANG, G. Wavelet Transforms versus Fourier Transforms. **Bulletin of the American Mathematical Society**, [S.l.], 1993.

TCPDUMP. Disponível em: http://www.tcpdump.org/ Acesso em: Junho de 2009.

WIRESHARK. Disponível em: http://www.wireshark.org/ Acesso em: Junho de 2009.

XML. Disponível em: http://www.w3.org/XML/ Acesso em: Junho de 2009.