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

FÁBIO MASSALINO

APERFEIÇOAMENTO DO LIDS ATRAVÉS DO USO DE THREADS E COM CONTROLE DE BLOQUEIO

MARÍLIA
2006

FÁBIO MASSALINO

APERFEIÇOAMENTO DO LIDS ATRAVÉS DO USO DE THREADS E COM CONTROLE DE BLOQUEIO

Monografia apresentada ao Curso de Ciência da Computação da Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília - UNIVEM, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação. (Área de Concentração: Segurança em Redes).

Orientador:

Prof. Dr. Antonio Carlos Sementille

Marília
2006

MASSALINO, Fábio.
Aperfeiçoamento do LIDS através do uso de Threads e com Controle de Bloqueio / Fábio Massalino; orientador: Antonio Carlos Sementille. Marília, SP: [s.n.], 2006.
54 f.

Trabalho (Graduação) – Centro Universitário Eurípides de Marília – Fundação de Ensino Eurípides Soares da Rocha.

CDD: 004.6

FÁBIO MASSALINO

APERFEIÇOAMENTO DO LIDS ATRAVÉS DO USO DE THREADS E COM CONTROLE DE BLOQUEIO

Banca examinadora do Trabalho de Conclusão de Curso apresentado ao Programa de Graduação da UNIVEM/F.E.E.S.R, para obtenção do título de Bacharel em Ciência da Computação. Área de Concentração: Segurança em redes.

Resultado: 10,0

ORIENTADOR: Prof. Dr. Antonio Carlos Sementille

1º EXAMINADOR: Prof. Dr. José Remo Ferreira Brega

2º EXAMINADOR: Prof. Dr. Ildeberto Aparecido Rodello

Marília, 07 de Dezembro de 2006.

AGRADECIMENTOS

Aos meus pais e minha avó, pelo incentivo que me deram deste o começo de minha faculdade.

Ao professor orientador Antonio Carlos Sementille, que me orientou neste trabalho, e aos demais professores, que me deram base teórica, essencial para meu futuro profissional.

Aos meus amigos e colegas de trabalho pela paciência que tiveram comigo, nos momentos que eu já não tinha mais.

À minha namorada Fátima, companheira, que com muito carinho e dedicação, me deu ânimo nos momentos que mais precisei.

À Deus, por tudo que colocou em meu caminho...

MASSALINO, Fábio. Aperfeiçoamento do LIDS através do uso de threads e com controle de bloqueio. 2006. 54 f. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação. Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

RESUMO

A interconectividade entre as redes é fundamental. Uma organização, independente de sua missão ou tamanho, utiliza-se das redes de computadores para trafegar seus dados. Isto desperta um grande interesse de pessoas que queiram usufruir destas informações, ou mesmo, paralisar os recursos computacionais. Garantir que os dados trafegados não sejam interceptados e haja disponibilidade dos recursos das redes computacionais são elementos pertinentes da área de segurança de redes. Visando detectar um ataque e conseguir reagir, surgiram os Sistemas de Detecção de Intrusos (IDS). O presente trabalho implementa melhorias no IDS *Linux Intrusion Detection System*, tornando-o mais eficaz, e implementando um módulo de bloqueio, capaz de impedir o invasor da rede cause danos maiores.

Palavras-chave: redes de computadores; segurança, IDS.

MASSALINO, Fábio. Improvement of LIDS through the use of threads and controlling the blockade. 2006. 54 f. Monografia (Bacharelado em Ciência da Computação) – Curso de Ciência da Computação. Centro Universitário Eurípides de Marília, Fundação de Ensino Eurípides Soares da Rocha, Marília, 2006.

ABSTRACT

The interconnection among the nets is very important. An organization, despite of its mission or size, uses the computer networks to pass through its data. It start to grow a great interest of people who want to usufruct of these informations, or even, paralyze the computational resources. To guarantee that the passed through data are not intercepted and have availability of the resources of the computational nets are pertinent elements of the area of security of nets. Trying to detect an attack and obtain reaction, the Intrusion Detection Systems(IDS) had appeared. The present work implements improvements in the IDS Linux Intrusion Detection Systems, becoming it more efficient and implementing a module of blockade, that is capable to block the invader of the net who may causes big damages.

Keywords: computers network, security, IDS.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

BPF: *Berkeley Package Filter*

CIDF: *Common Intrusion Detection Framework*

DMZ: *Demilitarized Zone*

DoS: *Deny of Service*

FTP: *File Transfer Protocol*

GPL: *General Public License*

HIDS: *Host Intrusion Detection System*

HTTP: *Hipertext Transfer Protocol*

ICMP: *Internet Control Message Protocol*

IDMEF: *Intrusion Detection Message Exchange Format*

IDS: *Intrusion Detection System*

IEEE: Instituto de Engenharia Elétrica e Eletrônica

IETF: *Internet Engineering Task Force*

IDS: Intrusion Detection System

IP: *Internet Protocol*

LIDS: *Linux Intrusion Detection System*

NAT: *Network Addresss Translation*

NIDS: *Network Intrusion Detection System*

OSI: *Open Systems Interconnection*

PNF: *Post Office Protocol*

POP3: *Post Office Protocol*

POSIX: *Portable Operating System*

SGBD: Sistema Gerenciador de Banco de Dados

SMTP: *Simple Mail Transfer Protocol*

SOHO: *Small Office - Home Office*

TCP: *Transmission Control Protocol*

UDP: *User Datagram Protocol*

VPN: *Virtual Private Network*

SUMÁRIO

1. INTRODUÇÃO

A interconectividade entre as redes de computadores é fundamental. Não tem como ficar isolado do mundo. As redes computacionais têm sido prioridade nas mais diferentes organizações, independente de seu tamanho ou missão, pois a troca de informações entre departamentos e filiais, tornou-se uma necessidade para criar condições de competitividade e até mesmo, de sobrevivência.

Visando garantir a funcionalidade destas redes, uma área que têm ganhado destaque é a relacionada a segurança de redes. Nesta área, a grande preocupação é garantir o funcionamento desta, garantindo proteção dos dados trafegados por ela.

Ao falar sobre a segurança em redes, duas questões são vitais: a primeira, garantir a não violação do conteúdo que trafega por ela; e garantir a disponibilidade da mesma.

Visando a garantia destas duas questões vitais, existem ferramentas que auxiliam o administrador de rede. Estas ferramentas atuam basicamente de duas formas: a primeira, de maneira preventiva: criando uma barreira, impedindo o acesso não autorizado. A segunda, detectada a invasão, ser um mecanismo capa de reagir ao ataque, evitando assim, danos maiores.

Um dos mecanismos de segurança das redes são os Sistema de Detecção de Intrusão (IDS - *Intrusion Detection System* -), que monitora constantemente a rede, impedindo que esta possa ser atacada por agentes mal intencionados, com o objetivo de obter informações confidenciais, ou mesmo, a paralisação de todo o sistema. Detectada a tentativa de intrusão, o IDS pode reagir, evitando a invasão.

Analisando este panorama, desenvolveu-se um protótipo de Sistema de Detecção de Intrusão (IDS), chamado de LIDS – *Linux Intrusion Detection System* (FERREIRA, 2004). Desenvolvido em módulos e implementado com base no sistema operacional *Linux*, o LIDS,

através do Módulo de Captura, monitora o tráfego da rede. O Módulo de Análise compara o tráfego capturado com as assinaturas de ataque armazenadas em Banco de Dados, registrando a ocorrência de tentativas de intrusão, através do Módulo de Gravação.

Embora eficiente, este IDS apenas detecta o ataque, não dispondo de nenhum recurso de reação.

O presente trabalho aprimora os Módulos de Captura e Análise do LIDS e a criação de um Módulo de Bloqueio anexado ao LIDS, proporcionando com isso, condições deste protótipo reagir às tentativas de ataque, criando um componente ativo de segurança.

O trabalho foi organizado da forma que segue: o Capítulo 2 reúne fundamentação teórica para o desenvolvimento deste trabalho. Trata dos fundamentos da segurança, seus conceitos, o que é intrusão e as principais ferramentas, tais como *firewall*, sistemas de detecção de intrusão.

O estudo do LIDS é abordado no Capítulo 3. Tratará sua implementação original, apresentando seus principais módulos, e o uso da biblioteca *libpcap*.

A implementação do presente trabalho é exposta no Capítulo 4. Neste, será tratada a metodologia utilizada, o ambiente experimental e os testes realizados.

No Capítulo 5 é feita uma análise dos resultados obtidos, trazendo a conclusão da pesquisa e implementação.

Finalmente, o Capítulo 6 traz idéias para trabalhos futuros, utilizando este trabalho como base.

2. SEGURANÇA

Segurança é a “tentativa de minimizar a vulnerabilidade de valores e recursos” (ISO7498-2). Para que uma rede de comunicação seja segura, observam-se as seguintes características:

- Confidenciabilidade: Somente o remetente e o destinatário da mensagem devem entender o conteúdo da mesma. Caso terceiros tenham acesso à mesma, não consigam entendê-la, pois a mensagem deve ser cifrada, sendo que somente o destinatário consiga decifrá-la.

- Autenticação: O usuário precisa comprovar que ele é realmente quem diz ser. Precisam existir mecanismos de autenticação para que o usuário possa identificar-se.

- Autorização: Característica que permite ao usuário acessar somente os recursos que tem direito.

- Integridade e não-repudiação de mensagem: Garante que a mensagem que chega ao destinatário seja a mesma enviada pelo remetente, isto é, não seja alterada por acidente ou má-intenção, durante a transmissão ou armazenamento.

- Disponibilidade: O sistema precisa ser protegido para garantir que esteja disponível a maior parte do tempo possível, por exemplo, contra ataques que tenham como objetivo, a paralisação destes (LAFETÁ, ARRUDA, 2005).

Proteger a comunicação e os recursos da rede é a definição de comunicação segura. A segurança não envolve somente proteção, mas também a detecção de falhas e de ataques a rede e a reação a estes ataques. A segurança de rede é conseguida por meio de um ciclo contínuo de proteção, detecção e reação.

2.1. Intrusão

De acordo com Silva (2002), intrusão é a tentativa de invasão de um sistema ou de fazer mau uso do mesmo, obtendo ou não, sucesso na tentativa. Para que se possa caracterizar uma invasão, é necessário diferenciar as ações legítimas das nocivas, com isso, é necessária a definição de uma política de segurança. Enquanto não definir o que é permitido e o que não é, torna-se inútil tentar entender uma intrusão.

Enquanto muitos administradores de sistemas acreditam que os ataques, em sua maioria, provém de fontes externas, ou seja, originados de fora da instituição, estudos revelam que a maior parte dos ataques tem origem dentro da própria instituição (intrusos internos).

2.1.1. Classificação das intrusões

As intrusões podem ser classificadas em seis tipos principais:

- Tentativas de invasão: Invasores tentarem violar os mecanismos de validação ou usuários apresentam comportamentos atípicos;

- Ataques mascarados: Semelhante à tentativa de invasão, mas de determinação mais sensível;

- Penetração do sistema de controle de segurança: Detectado pela monitoração de padrões específicos de atividade;

- Vazamento: Detectável pelo uso atípico dos recursos de Entrada/Saída;

- Negação de serviço: Caracterizado pela tentativa de causar a indisponibilidade dos sistemas;

- Uso malicioso: Usuários legítimos cometem abusos de seus privilégios.

Estes tipos de intrusão podem ser monitorados pela análise dos perfis de

comportamentos atípicos, violações de regras de segurança, padrões específicos de atividades (SILVA, 2002).

Os ataques também podem ser classificados em:

- Passivos: Ataques cujo objetivo é a obtenção de informações;

- Ativos: Alteram as informações ou afetam a operação do sistema;

- Internos: Iniciados por um agente do perímetro da área de segurança, ou seja, uma pessoa autorizada a utilizar os recursos do sistema, mas utilizou seus privilégios de maneira errônea.

- Externos: O atacante está fora do perímetro de segurança. Explora as vulnerabilidades do sistema para iniciar seus ataques.

2.2. Sistema de detecção de intrusão

A finalidade de um IDS (Sistema de Detecção de Intrusão) é detectar o acesso indevido ou uso inadequado de um sistema computadorizado (CERIAS, 2006). A motivação do uso de um IDS reside em, por mais que um sistema de prevenção seja eficiente, existe sempre a possibilidade de falhas ou de desvios por onde um atacante possa invadir o sistema. O IDS entra em ação, impedindo intrusos de obterem informações sigilosas, ou de comprometerem o sistema. Mesmo que não consiga detecta-lo a tempo, quanto mais cedo isto ocorrer, menores serão os danos causados. O IDS pode também, coletar informações referentes a comportamentos maliciosos e técnicas de invasão, contribuindo para uma melhor ação dos sistemas de prevenção.

O IDS realiza suas operações a partir da análise dos perfis de comportamentos atípicos: violações de regras de segurança; padrões específicos de atividades; uso de recursos dos sistemas ou uso de privilégios especiais. A Figura 2.1 mostra a classificação de um IDS.

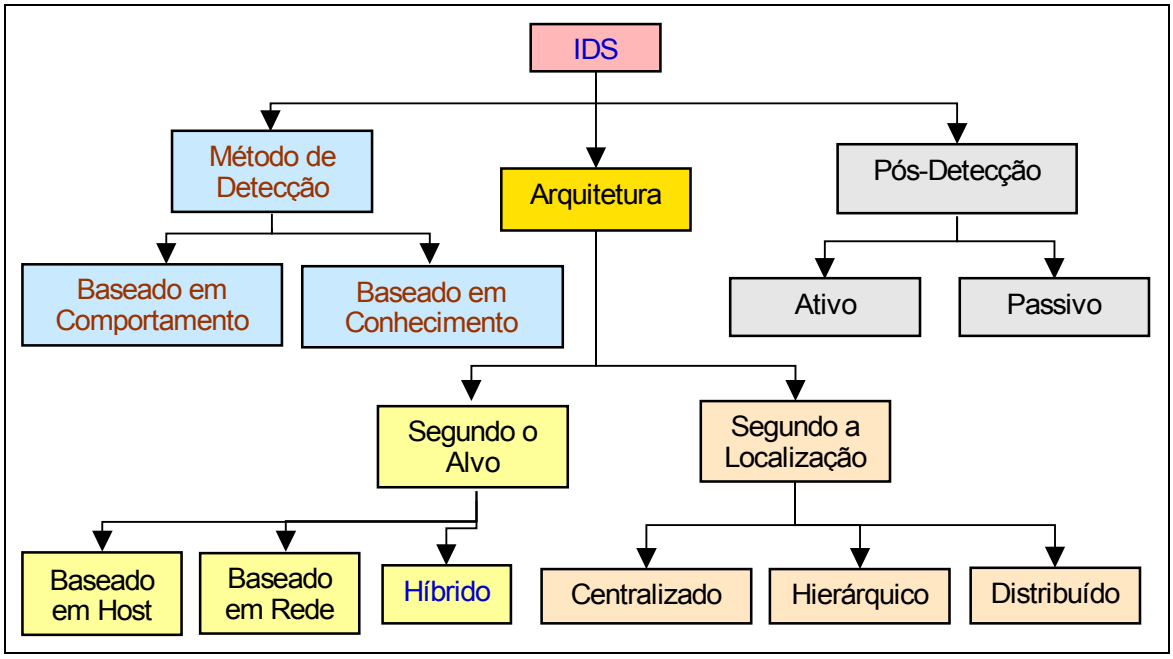

Figura 2.1 - Classificação de um IDS (SILVA, 2002)

2.2.1. Características de um IDS

As principais características de um IDS são:

- Executar continuamente sem interação humana (*background*);

- Tolerante a falhas;

- Resistir a tentativas de mudanças de sua base (subversão);

- Não degradar o desempenho do sistema;

- Detectar mudanças no seu funcionamento normal;

- Ser de fácil configuração e

- Adaptar-se as mudanças do sistema.

2.2.2. Modelo *Common Intrusion Detection Framework*

Com o surgimento de grande variedade de ferramentas de IDS, foi proposto um

modelo chamado CIDF (*Common Intrusion Detection Framework*) que reúne uma coleção de componentes que define uma ferramenta de IDS (OLIVEIRA, 2006):

- Gerador de eventos (*E-boxes*): A função deste componente é capturar os eventos, definidos através da política de segurança, como tentativas de invasão.

- Analisador de eventos (*A-boxes*): Recebe as informações do gerador de eventos e analisa as informações e envia para a base de dados de eventos.

- Base de eventos (*D-box*): Armazena os eventos e/ou resultados para uma necessidade futura.

- Unidade de resposta (*R-box*): Responsável pelo contra-ataque, tais como: matar o processo, reinicializar a conexão, alterar a permissão de arquivos, notificar o administrador, etc.

2.2.3. Tipos de IDS

De acordo com Campello (2001), as ferramentas de IDS podem ser classificadas em três tipos:

1. HIDS - IDS baseado em *Hosts*: São instalados em um servidor, observando as ações realizadas no sistema operacional, as ações de serviços e o comportamento da pilha TCP/IP. Um subtipo deste IDS é o baseado em aplicação, onde é analisada uma aplicação especifica.

2. NIDS – IDS baseado em redes: Localizam-se em pontos estratégicos, verificando o tráfego da rede, o formato dos pacotes, entre outros.

3. IDS Híbridos: Mescla de IDS baseado em *Host* com o baseado em Redes.

2.2.4. Assinaturas de ataque

Assinatura de ataque é uma base de dados com seqüências de ações consideradas indícios de intrusão, indicando a maioria dos ataques conhecidos e servindo como base para a busca de intrusos no sistema (SILVA, 2002).

Em um IDS usando assinaturas de ataques para identificar uma intrusão, podem ocorrer dois erros:

- Falso positivo: O IDS consulta a base de dados, e sinaliza como tentativa de invasão, uma atividade normal.

- Falso negativo: Uma tentativa de invasão passa pelo IDS, e este a considera como uma atividade normal.

Visando diminuir as ocorrências de falsos positivos e ou negativos, adota-se o conceito de sistemas especialistas, onde o conhecimento sobre ataques é codificado na forma de regras de condição do tipo *if-then*, isto é, essas regras especificam em seu lado esquerdo, as condições necessárias para que seja configurado como um ataque, e dispara ações ou novas avaliações colocadas no seu lado direito, com isso, os dados coletados são traduzidos em fatos que carregam seu significado semântico ao sistema especialista, e uma máquina de inferência traça conclusões usando as regras já definidas.

Cria-se uma base de ações ou de seqüência de ações não-aceitáveis, chamada de base de assinaturas, e esta é normalmente distribuída e atualizada pelo fabricante do IDS. Outras ferramentas atualizam suas base de assinaturas com pouca freqüência, deixando esta tarefa para seus usuários. No primeiro caso, a justificativa é a dificuldade em encontrar usuários dispostos a aprender o formato de descrição de ataques, e atualizar com a freqüência aceitável. No segundo, a justificativa é que as assinaturas existentes precisam ser adaptadas a realidade de cada organização.

O IDS Snort, por exemplo, usa seu próprio modelo assinaturas de ataque, em uma simples, leve e flexível linguagem, conhecida como "snort rules". Estas são divididas em duas seções lógicas: o cabeçalho e as opções. No cabeçalho, contém informações sobre protocolos, endereços de origem e destino, máscara de rede, e portas de origem e destino.

Nas opções, contém mensagens de alerta e informações de quais partes dos pacotes devem ser inspecionadas.

2.2.5. Exemplos de IDS

- OSSEC HIDS: Open Source HIDS – IDS baseado em host, open-source. Executa a análise e a correlação do registro, verificar da integridade, a detecção do *rootkit*, alertas *time-based* e *active response*. Funciona na maioria de sistemas operacionais, incluindo Linux, OpenBSD, FreeBSD, Solaris e Windows (OSSEC,2006).

- **SNORT:** IDS baseado em redes, *open-source*, isto é, código-fonte é aberto. Utiliza-se da análise de assinaturas verificando anomalias. Bastante popular por usuários do sistema operacional Linux, por ser bastante flexível nas configurações de regras e constantes atualizações, adequando o software para os novos tipos de ataque e ferramentas utilizadas para este fim. Possui um vasto número de assinaturas de ataque (SNORT, 2006).

- **PRELUDE:** É um *Framework* de IDS Híbrido, isto é, é um produto que permite centralizar todas as aplicações de segurança, seja ele *open-source* ou proprietário. Para conseguir esta tarefa, o Prelude confia no padrão do IETF de IDMEF (formato da troca da mensagem da detecção de Intrusão), uma linguagem padrão que é entendida por diferentes sensores (PRELUDE,2006).

2.3. – *Firewall*

Segundo Cheswick (1994), firewall é:

> Um *FIREWALL* é DEFINIDO como uma coleção de componentes, colocada entre duas redes, que coletivamente possua as seguintes propriedades: Todo o tráfego de dentro pra fora, e vice-versa, passa pelo firewall; só o tráfego autorizado pela política de segurança pode atravessar o firewall e o firewall deve ser a prova de violações.

A função básica de um *firewall* é proteger os dados e os recursos de hardware e software de uma rede privada de ameaças externas. Ele é a porta de comunicação entre a rede interna com o mundo externo. A vantagem é que centraliza todas as ações de segurança em um único ponto, o que facilita o gerenciamento. Como desvantagem, cria-se um ponto de gargalo, ou seja, parando o *firewall*, acaba a comunicação entre essas redes.

O *firewall* consegue barrar apenas a entrada de ameaças definidas pelo administrador. Não é capaz de verificar por si mesmo, a ocorrência de ataques e se auto configurar. Não garante a integridade, confidencialidade e autenticidade das mensagens. Também não protege a rede de ameaças internas, como funcionários insatisfeitos ou com más intenções. Funciona, portanto, como a primeira linha de defesa. O acesso a serviços legítimos acessados por usuários autorizados, depois que passa pelo *firewall*, não se torna uma preocupação deste. Por isso, não é somente ele que garante a segurança dos sistemas.

O *Firewall* atua verificando o cabeçalho ou o conteúdo dos pacotes e combinam quatro técnicas para garantir a segurança:

- Controle de serviços: Determina os serviços que podem ser usados na rede. Serviços vulneráveis podem ser bloqueados pelo *firewall*.
- Controle de direção: Determina a direção que uma solicitação de serviços pode ser iniciada.
- Controle de usuários: Cada usuário acessa somente os serviços que tem

privilégios.

- Controle de comportamento: Determina o modo que cada serviço pode ser utilizado.

2.3.1. Funcionalidades do *Firewall*

Segundo Lafetá e Arruda (2005), as principais funcionalidades do *firewall* são:

- Filtro: Atua no roteamento e análise os pacotes através da leitura dos cabeçalhos ou conteúdo e decide se bloqueia ou libera a passagem deste, de acordo com as regras de filtragem estabelecidas na política de acesso.

- *Proxy*: Software utilizado para intermediar duas redes. Este assume os endereços da rede interna, ocultando com isso, informações da topologia desta, já que todos os pacotes que saem dele contém o endereço do *proxy*, e não das máquinas da rede interna.

- *Dual homed gateway*: Possui no mínimo duas interfaces de rede, e simplesmente controla o tráfego entre redes distintas, funcionando como intermediário entre elas.

- *Bastion Host*: Elo entre a rede interna e a rede externa. Por ser o único dispositivo a alcançar a rede externa, com isso, acessar a Internet, deve-se executar somente os serviços essenciais e ser protegido da melhor maneira possível.

- DMZ (*demilitarized zone*): A zona desmilitarizada é uma rede parcialmente protegida; situa-se entre a rede interna protegida e a rede externa desprotegida (Internet). Contém os serviços que precisam ser acessados pela rede pública, tais como servidores de e-mail, *FTP*, *WEB*.

- NAT (*network addresss translation*): A tradução de endereços de rede consiste em converter os endereços IPs internos e reservados para endereços IPs públicos, quando a rede externa é acessada.Comumente utilizado com serviços *proxy*.

- VPN (*virtual private network*): Rede privada virtual consiste em criar, dentro da infra-estrutura da Internet, uma rede privada, onde os usuários acessam pontos, fisicamente distantes, como se estivesse conectado localmente. Utiliza criptografia para manter a confidencialidade, integridade e autenticidade dos dados.

- Autenticação/certificação: Processos de validação da identidade de computadores ou usuários que requerem acesso à rede privada e podem ser baseados em endereços IP e portas de origem, senhas, certificados e outras formas avançadas de identificação, como a biometria.

2.3.2. Arquiteturas para *Firewall*

- *Screening router*: Arquitetura de camada única (Figura 2.2), formada por um filtro de pacotes, o roteador de triagem, responsável por todas as funções de proteção da rede interna. A comunicação entre a rede interna e a Internet é direta, apenas filtrada por regras pré-estabelecidas.

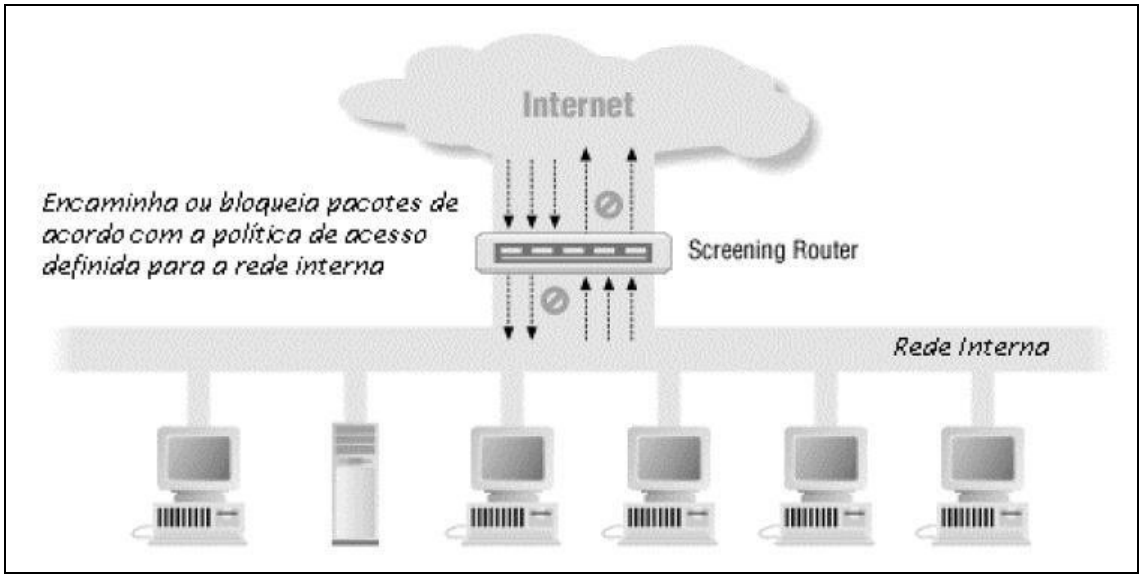

Figura 2.2 - Arquitetura *screening router* (LAFETÁ E ARRUDA, 2005)

- *Dual-homed host*: O equipamento realiza a conexão entre as redes em duas etapas: os sistemas de uma rede têm que se conectar ao *firewall* para que possa se comunicar com os da outra (Figura 2.3). Com isso, o *firewall* não realiza o roteamento de pacotes: ele os copia e então os envia para a máquina de destino. Funciona como um servidor *proxy*.

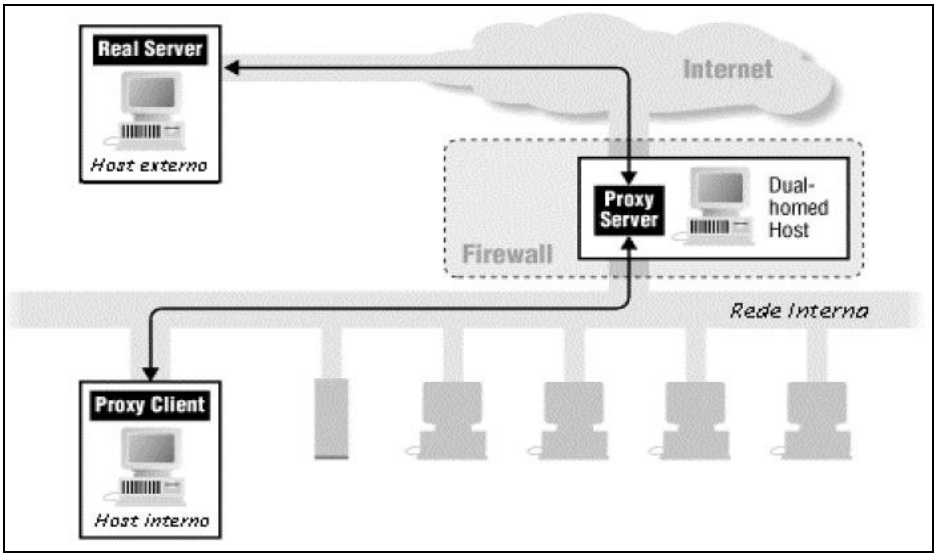

Figura 2.3 - Arquitetura *dual-homed host* (LAFETÁ E ARRUDA, 2005)

- *Screened host*: Além do roteador de triagem entre as duas redes, existe um *bastion host* localizado na rede interna. O filtro de pacotes bloqueia os pacotes que não tenham como origem ou destino o *bastion host*, que funciona como um servidor *proxy*. O problema é que se o *bastion host* for comprometido, o invasor estará dentro da rede interna (Figura 2.4).

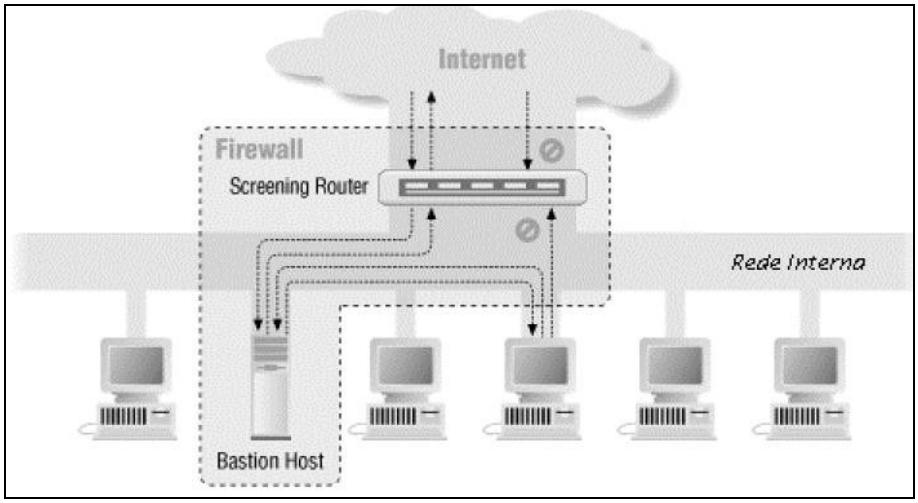

Figura 2.4 - Arquitetura *Screened Host* (LAFETÁ E ARRUDA, 2005)

- *Screened subnet*: Uma das mais seguras arquiteturas de *firewalls*, esta introduz uma zona desmilitarizada entre as duas redes para resolver o problema da anterior. Utiliza-se dois roteadores, um externo e outro interno, e o *bastion host* fica localizado dentro da DMZ (Figura 2.5). O roteador interno oferece uma camada extra de segurança, pois deixa a rede interna isolada (LAFETÁ e ARRUDA, 2005)

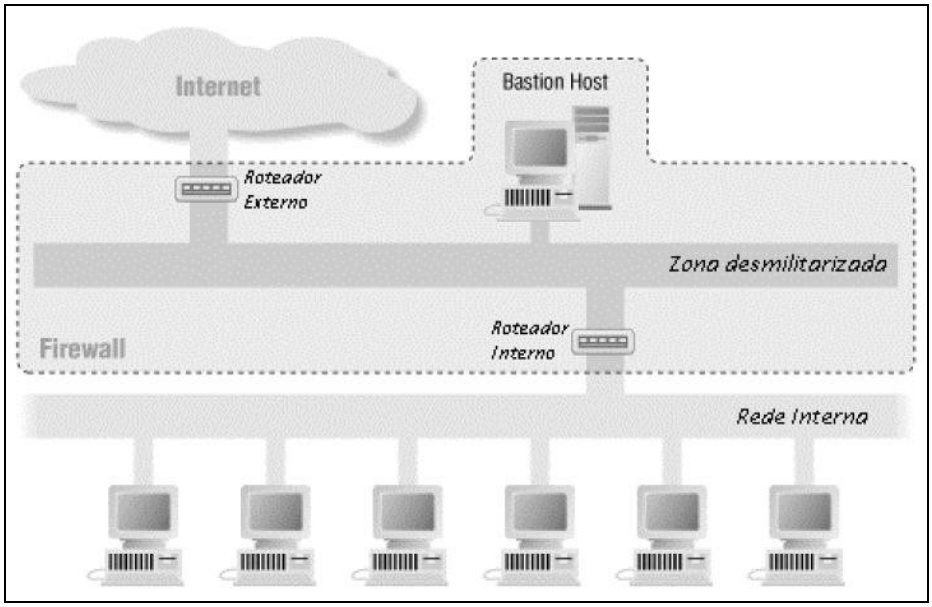

Figura 2.5 - Arquitetura *Screened subnet* (LAFETÁ E ARRUDA, 2005)

O sistema operacional *Linux* possui um *firewall* padrão: o IPTABLES.

2.3.3 Iptables

No *kernel* do sistema operacional Linux 2.4 foi introduzido um filtro de pacotes chamado *iptables* que substitui o *ipchains* do *kernel* da série 2.2.

O *iptables* é um filtro de pacotes e funciona baseado em regras que analisam o endereço e a porta de origem e destino dos pacotes.

Ele compara os pacotes com as regras, liberando ou negando os pacotes. Também

pode ser usado para modificar ou monitorar o tráfego da rede, NAT (*masquerading*, *source nat*, *destination nat*), redirecionamento de pacotes, modificar a prioridade, contagem de bytes, divisão tráfego entre máquinas, criar proteção *anti-spoofing*, *syn flood*, DoS, etc.

É necessário que o *kernel* tenha sido compilado com suporte ao *iptables*. Todo tráfego é registrado no arquivo */var/log/kern.log.*

As regras são armazenadas no *kernel*, e devem ser gravadas em um arquivo para serem lidas a cada carga do sistema.

As regras se dividem em três categorias, de acordo com a ação desejada: *INPUT, OUTPUT e FORWARD.*

Elas são organizadas em tabelas com uma determinada característica em comum. Existem três tabelas disponíveis no *iptables* (RIBEIRO, 2004):

1) filter: tabela padrão, contém 3 categorias padrões:

a) *INPUT*: Consultado para dados que chegam à máquina.

b) *OUTPUT*: Consultado para dados que saem da máquina.

c) *FORWARD*: Consultado para dados que são redirecionados para outras interfaces de rede ou outra máquina.

2) nat: Usada para dados que geram outra conexão. Possui três categorias padrões (RIBEIRO, 2004)::

a) *PREROUTING*: Consultado quando os pacotes precisam ser modificados logo que chegam.

b) *OUTPUT*: Consultado quando os pacotes gerados localmente precisam ser modificados antes de serrem redirecionados.

c)*POSTROUTING*: Consultado quando os pacotes precisam ser modificados após o tratamento de redirecionamento.

3) mangle: Utilizado para alterações especiais de pacotes, como modificar o tipo de serviço e outros detalhes. Possui duas categorias:

a) *PREROUTING*: Consultado quando os pacotes precisam ser modificados logo que chegam.

b) *OUTPUT*: Consultado quando os pacotes gerados localmente precisam ser modificados antes de serem redirecionados (RIBEIRO, 2004).

2.3.4. *Firewalls* existentes no Linux

Usando *iptables* como base, existem diversas ferramentas, para ampliar os recursos e facilitar a administração do *firewall*. A seguir, alguns exemplos:

- FIAIF: Script em bash cuja finalidade é criar um *firewall* altamente customizado.

- *Turtle Firewall*: Programa GPL escrito em PERL, cujo objetivo é facilitar o uso do *firewall*. Pode ser administrado através de interfaces web, como o Webmin, por exemplo.

- Shoreline *Firewall*: Ferramenta para simplificar a configuração do *netfilter*. Atua em conjunto com o iptables.

- Endian *Firewall*: *Firewall* que é muito fácil de instalar, usar-se e controlar, sem perder sua flexibilidade. As características incluem um firewall para inspeção do pacote, proxies ao nível-de-aplicação para os vários protocolos (HTTP, POP3, smtp) com sustentação do antivírus, vírus e spamfiltering para o tráfego do email (PNF e smtp), e uma solução de VPN hassle free" (baseada em OpenVPN).

- LutelWall *Firewall*: Ferramenta de alto-nível usada para configurar de maneira simples, porém, sem perder as opções de segurança do netfilter.

- IpCop *firewall*: Firewall desenvolvido para o mercado SOHO.

- Smoothwall: Distribuição Linux destinada a *firewall*. Graças a sua customização, não requer que os administradores tenham conhecimento em linux. (FERRAMENTAS LIVRES, 2006).

3. O LIDS – *Linux Intrusion Detection System*

Com base nas informações obtidas sobre a necessidade de desenvolver mecanismos de segurança, para uma rede computacional, tanto medidas preventivas, quanto medidas que detectem a tentativa de invasão e crie mecanismos de reação, desenvolveu-se um mecanismo de detecção de intrusão, o LIDS (Linux Intrusion Detection System) (FERREIRA, 2004) .

O LIDS é um IDS baseado em redes (NIDS), que atua como um *sniffer*, inserido em um ponto estratégico da rede, monitorando o fluxo dos dados que trafegam pela rede, e com base nas assinaturas de ataque pré-definidas, detecta a tentativa de intrusão, registrando o endereço de origem, o endereço de destino, e a assinatura detectada.

Utilizando-se de estrutura de módulos, o que facilita a flexibilização do projeto, a Figura 3.1 lista os principais módulos do LIDS:

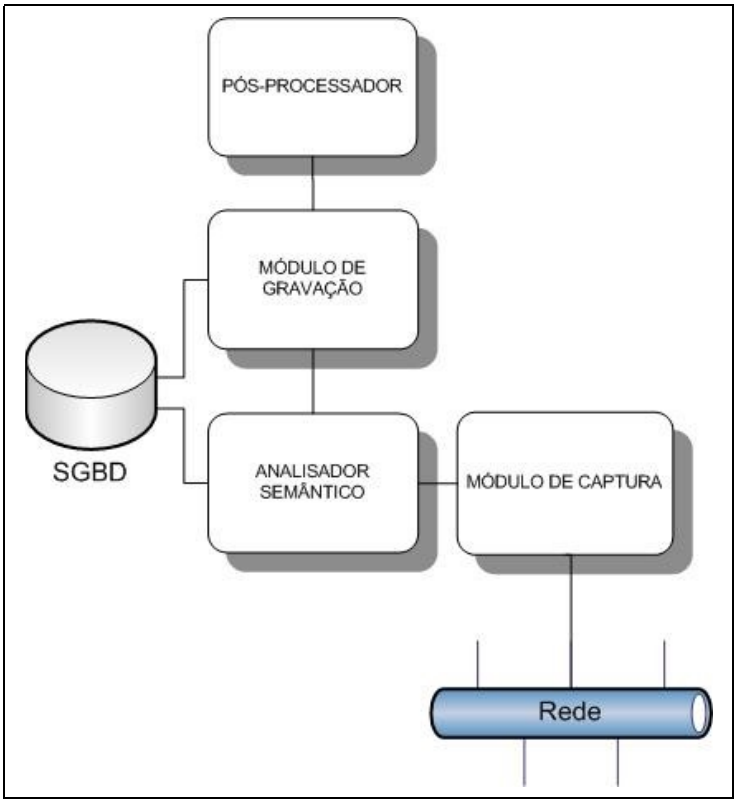

Figura 3.1– Esquema modular do LIDS

O sistema possui módulos com funções distintas, permitindo que o funcionamento de entrada e saída e a possível retro-alimentação não atrapalhe a estrutura restante do sistema.

O LIDS utiliza um banco de dados relacional tanto para as assinaturas de ataque como para o registro de tentativas de ataques. Com isto, podem-se utilizar os recursos disponíveis nestes bancos de dados para a consulta das informações registradas.

A seguir serão comentados os módulos do LIDS.

3.1. Módulo de Captura de Pacotes

O primeiro módulo tem a função de capturar os pacotes que trafegam pela rede e torná-los disponíveis para os demais módulos. Para desempenhar tal função, utiliza-se da biblioteca *libpcap*.

3.1.1. Biblioteca LIBPCAP

Libpcap é uma biblioteca para captura de pacotes, desenvolvida por Van Jacobson, Craig Leres e Steven McCanne, todos do Lawrence Berkeley National Laboratory, Universidade da Califórnia, Berkeley, CA.

A pcap é utilizada nas redes, para monitorar o tráfego, como um *sniffer*. Atua nas Camadas de Rede e Transporte do modelo de referência OSI. Utilizada como base em muitos projetos conhecidos, tais como o snort, arpwatch, fragrouter, entre outros (SANTANA, 2006).

Esta biblioteca permite capturar todos os pacotes que trafegam pela interface de rede. Possui a opção de deixar a interface de rede em modo promiscuo. Quando a interface de rede entra em modo promíscuo, é permitido capturar não apenas os pacotes destinados ou roteados pelo host, mas todos os que passam por ela. Numa rede onde os computadores são

interligados por um HUB, pode-se capturar tudo que é trafegado por esta.

Para permitir o uso, a libpcap possui uma API com diversas funções e tipos definidos, permitindo a comunicação com o filtro BPF, a recuperação de pacotes, ver estatísticas, reconhecer novos pacotes, distinguir tipos de erros.

3.1.2. Usando a LIBPCAP

Para usar a libpcap, no programa-fonte, é preciso definir a utilização dela (conforme a Figura 3.2):

```
#include <pcap.h>
```

Figura 3.2 – Bibliotecas para captura

Em seguida, definem-se as estruturas e variáveis que serão utilizadas para a captura.

```
pcap_t *cap  // descritor de captura
char *dev   // ponteiro para o dispositivo de rede (eth0, eth1)
char ebuf[PCAP_ERRBUF_SIZE] // buffer de retorno de erros
struct pcap_pkthdr pkthdr // estrutura de recuperação de pacotes
u_char *pacote //ponteiro para o pacote
```

Figura 3.3 - Estrutura de dados para o módulo de captura

A primeira função a ser utilizada é a *pcap_open_live*, que permite a abertura do descritor de captura, indicando o dispositivo, métodos de captura e tamanho do buffer, retornando, caso aconteça, mensagens de erro,, através do *char ebuff.* Veja na Figura 3.4.

```
//abertura do descritor de captura
if ((cap = pcap_open)live(dev, BUFFSIZ, 1, 1000, ebuf)) == NULL){
        fprintf(stderr,"libpcap: %s\n",ebuf);
        exit(1);
}
```

Figura 3.4 - Descritor de captura

Os pacotes capturados são armazenados em arquivo, para futura análise. Para isto, utiliza-se a função *pcap_dump_open*, responsável por criar o arquivo usado para o despejo do tráfego.

```
if ((dumpcap = pcap_dump_open)cap,warq)) == NULL)
{
fprintf(stderr, "pcap: %s\n", ebuf);
exit(1);
}
```

Figura 3.5 - Criação do arquivo para despejo de tráfego

Após a abertura do descritor cria-se um laço de repetição infinito para que os pacotes que estão trafegando na rede possam ser armazenados no arquivo criado pela *pcap_dump_open*. Para isto, utilizam-se as funções *pcap_next* e *pcap_dump.* A *pcap_next* faz a leitura do próximo pacote, enquanto a *pcap_dump* despeja o tráfego para o arquivo criado pela *pcap_dump_open*.

```
while(1)
{
pacote = (u_char *) pcap_next(cap,&pkthdr);
pcap_dump((u_char *) dumpcap. &pkthdr, pacote)
}
```

Figura 3.6 - Loop infinito de captura de pacotes

3.2. Módulo do Analisador Semântico

O Analisador Semântico é o módulo que analisa os pacotes armazenados no arquivo de despejo de tráfego, e compara com as assinaturas de ataque, procurando detectar tentativas de invasão. Registrando alguma tentativa, ativa o módulo de captura.

Para tornar-se eficaz, é necessário extrair somente as informações que interessam, pois os pacotes, além do conteúdo de dados, possuem também informações de cabeçalhos, tais como endereço IP de origem e destino, e portas de origem e destino. O primeiro passo é retirar estas informações, com isso. Por isto, utiliza-se o analisador de protocolos.

3.2.1 Analisador de Protocolos

O analisador de protocolos tem a função de filtrar os cabeçalhos dos tipos principais de protocolos pertencentes ao TCP/IP e tornar esses cabeçalhos fáceis de serem interpretados pelo administrador de rede. Para a implementação deste analisador, utilizaram-se estruturas nativas do Linux. A Figura 3.7 mostra as estruturas utilizadas.

```
#include <netinet/ether.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <netinet/icmp.h>
```

Figura 3.7 - Estrutura de dados para o analisador de protocolos

Após, criam-se manipuladores (*handles)* para cada estrutura, conforme a Figura 3.8.

```
u_int16_t handle_ethernet
(u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*  packet);
u_char* handle_IP
     (u_char *args,const struct pcap_pkthdr* pkthdr,const u_char*  packet);
```

Figura 3.8 - Manipuladores de estruturas do módulo de análise

3.2.2. Analisador de Pacotes

A função do analisador de pacotes é comparar os dados dos pacotes, com as assinaturas de ataque, verificando se nos pacotes há o conteúdo das assinaturas. As assinaturas de ataque estão armazenadas em uma tabela denominada tbl_regras, com a mostrada na Tabela 3.1:

Campo	*Descrição do campo*
Regra:	Chave primária da tabela
*Protocol*o:	Tipo de protocolo que se pretende filtrar: TCP, UDP, ICMP, NETBIOS
Descrição:	Especifica a invasão.
Comando:	A informação que será comparada com os pacotes analisados.
Porta:	A porta usada na assinatura.

Tabela 3.1 – Campos da tabela tbl_regras

Um exemplo de assinatura é apresentado na Figura 3.9:

```
INSERT INTO tbl_regras(protocolo, descricao, comando, porta) VALUES
('TCP','FTP piss scan','pass -cklaus','21');
```

Figura 3.9 - Inserção de uma regra no MySQL

A análise dos pacotes é feita no LIDS, pela função analise que recebe como argumentos a estrutura de recuperação dos pacotes, e o pacote. A Figura 3.10 mostra o cabeçalho da função análise.

```
void analise (struct pcap_pkthdr hdr, u_char * packet)
```

Figura 3.10 - Cabeçalho da função Análise

3.3. Módulo de gravação

Este módulo é responsável por efetuar o registro de todos os pacotes que coincidiram, em relação ao seu conteúdo, com alguma assinatura de ataque. Com isto, pode-se efetuar a reação necessária para impedir algum dano maior.

A gravação dos registros é feita na mesma base de dados, através da tabela tbl_invasão. A estrutura da tabela é:

Campo	*Descrição do Campo*
invasao	Chave primaria
ip_origem	Endereço de origem
ip_destino	Endereço de destino
identificação	Identificação do segmento
offset	Offset do segmento
protocolo	Tipo de protocolo (tcp, udp, etc)
checksum	Digito verificador
porta_origem	Porta origem
porta_destino	Porta destino
n_seqüência	Seqüência do segmento
ack	Dígito de reconhecimento
regra	Número da regra registrada na tbl_regras.
cadastro	Data/hora do registro

Tabela 3.2 – Campos da tabela tbl_invasao

4. IMPLEMENTAÇÃO

Após o estudo do LIDS, verificou-se a necessidade de desenvolver um módulo de bloqueio, além de aperfeiçoar os módulos existentes. Neste capítulo, mostrará o desenvolvimento destas implementações.

4.1. Objetivos do trabalho

O presente trabalho tem como objetivos:

- Utilizar multi-thread para otimização na captura e análise dos pacotes;
- Carregar as assinaturas de ataques armazenadas no banco de dados em memória para aumentar a performance do módulo de análise;
- Implementação do módulo de bloqueio, modificando as regras do firewall.

A estrutura modular do LIDS e a falta de uma reação ativa do sistema motivaram o aprimoramento do sistema, e a criação de um outro módulo chamado MÓDULO DE BLOQUEIO. A Figura 4.1 mostra o funcionamento proposto.

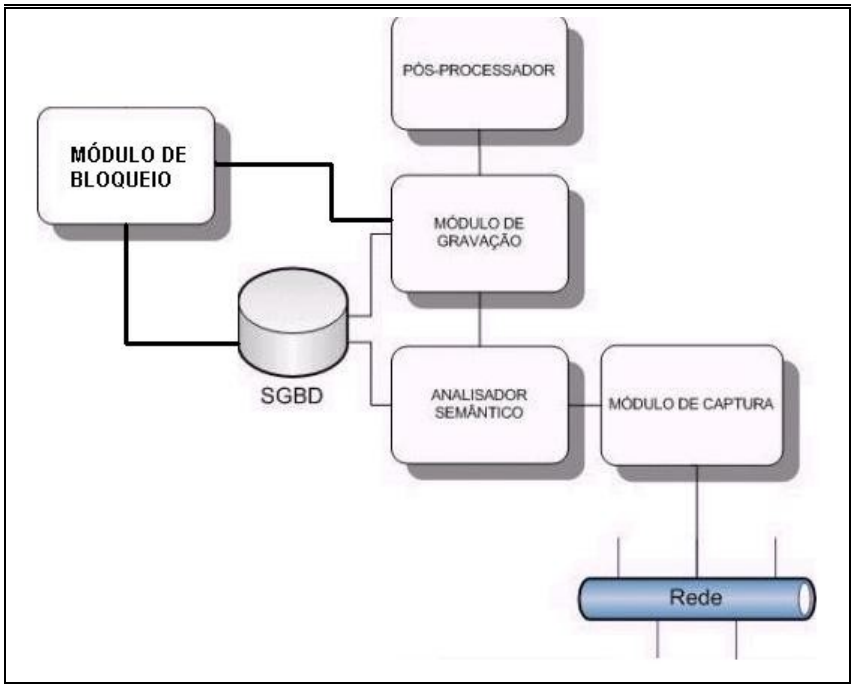

Figura 4.1 - Módulo de Bloqueio anexado ao LIDS

A proposta é permitir uma eficácia maior dos módulos de captura e análise. Para isto, reestruturaram-se alguns algoritmos, reduzindo a perda de pacotes. Para tal utilizou threads e a carga, na memória, das assinaturas de ataque.

Desenvolveu-se também, um módulo de bloqueio. Atuando junto ao firewall, permite o bloqueio do invasor, depois que se detectou a tentativa. Com isto, criou-se uma resposta ativa em relação às invasões. A Figura 4.2 ilustra seu funcionamento.

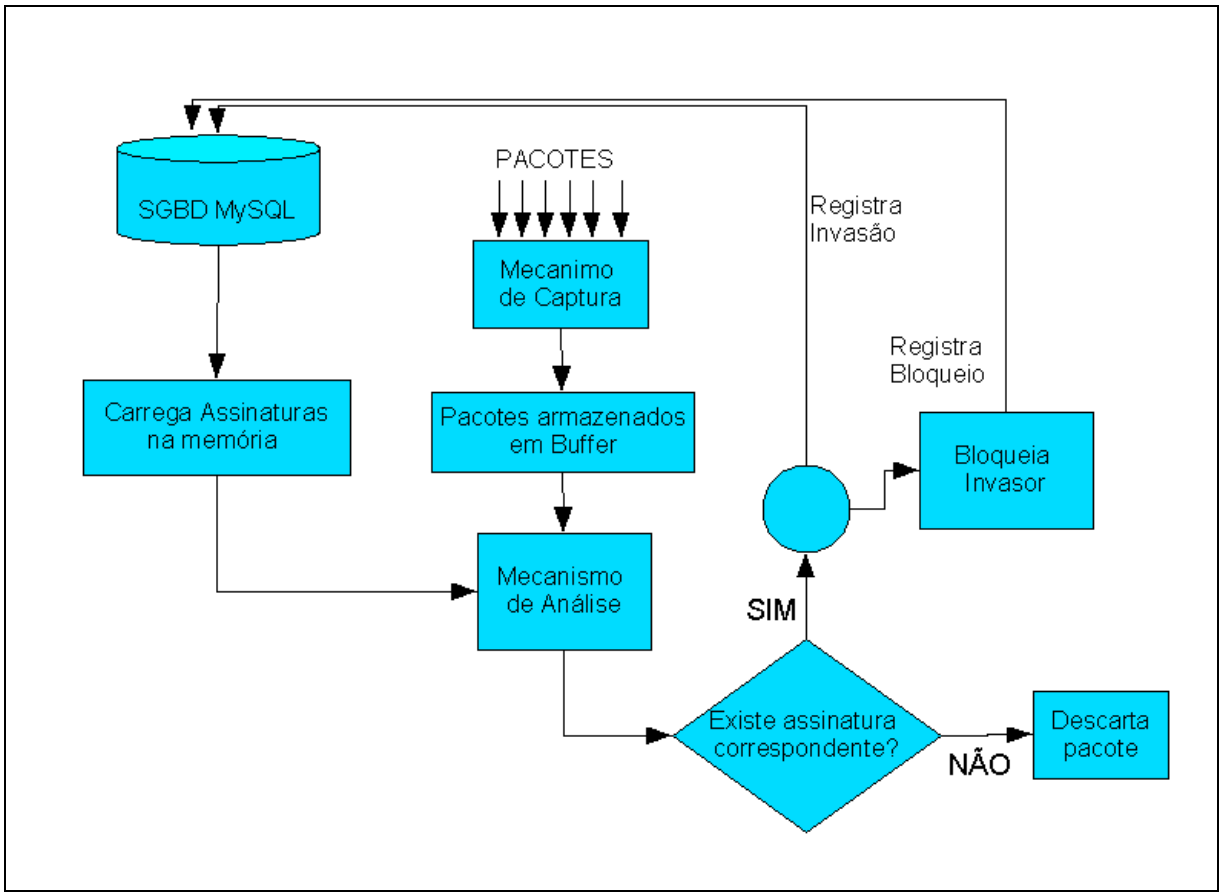

Figura 4.2 - Funcionamento proposto para o LIDS

4.2. Threads

Visando conseguir uma maior eficácia do LIDS, pesquisou-se vários métodos para a otimização do programa. O primeiro conceito que trouxe melhorias foi o uso de *threads*.

4.2.1. O que são *threads*

Segundo (TANENBAUM, 2003):

> Em sistemas operacionais tradicionais, cada processo tem um espaço de endereçamento e um único *thread* (fluxo) de controle. Na verdade, isso é quase uma definição de processos. Contudo, freqüentemente há situações em que é desejável ter múltiplos *threads* de controle no mesmo espaço de endereçamento executando em quase-paralelo, como se eles fossem processos separados (exceto para espaços de endereçamento compartilhados);

Os *threads* permitem que múltiplas execuções ocorram no mesmo ambiente do processo com um grande grau de independência uma da outra. Os vários *threads* do processo compartilham um mesmo espaço de endereçamento, arquivos abertos e outros recursos.

O termo *multithread* é usado para descrever a situação em que se permite a existência de múltiplos *threads* no mesmo processo. A CPU alterna rapidamente entre os *threads* dando a impressão de que os *threads* estão executando em paralelo. Um processo que precisa executar uma tarefa mais demorada pra depois executar outra, usando multi-threads, executam as duas como se estivesse executando-as em paralelo.

Existe dois modos principais de implementar um pacote de *thread*: no espaço do usuário ou no núcleo.

No primeiro caso, o pacote de *thread* é inserido totalmente dentro do espaço do usuário (*thread* do usuário). O núcleo não é informado sobre eles. A vantagem de utilizar este modo é poder implementar em um sistema operacional que não suporte thread. A desvantagem é que quando um *thread* é bloqueado (por exemplo, em uma E/S, um semáforo ou falta de pagina) todos os *threads* daquele processo são bloqueados, pois o núcleo pensa que existe somente um *thread* e não escalona o processo até que aquele *thread* seja liberado.

No segundo caso, o núcleo sabe dos *threads* e os gerencia. O núcleo tem uma tabela de *threads* que acompanha todos os *threads* do sistema. Quando um *thread* quer criar um novo *thread* ou destruir um já existente, ele faz uma chamada ao núcleo, que realiza então a criação ou a destruição atualizando a tabela de *threads* no núcleo

4.2.2. *Threads* no Linux: *Pthreads*

A partir do sistema operacional Unix, surgiram uma variedade enorme de versões inspiradas nele, gerando incompatibilidade entre elas. Para tornar possível escrever programas

que pudessem ser executados em qualquer sistema operacional da família *NIX, o IEEE desenvolveu um padrão para o Unix denominado POSIX (*Portable Operating System-IX* – sistema operacional portável).

As primeiras versões do Unix não tinham *threads*, sendo adicionado alguns anos depois. Quando surgiram, vieram muitos pacotes de threads em uso, e isso gerava uma enorme dificuldade na escrita de códigos portáveis. Por isso, as chamadas ao sistema usadas para gerenciar threads foram padronizadas como parte do POSIX.

A especificação do POSIX não estabelece se os threads devem ser implementados no núcleo ou no espaço do usuário

As chamadas a threads usadas para o desenvolvimento do LIDS são mostradas na Tabela 4.1:

Chamadas a threads	**Descrição**
pthread_create	Cria um novo thread no espaço de endereço do chamador
pthread_exit	Termina o thread chamador
pthread_join	Espera pelo término de um thread
pthread_mutex_init	Cria um novo mutex
pthread_mutex_destroy	Destrói um mutex
pthread_mutex_lock	Impede um mutex
pthread_mutex_unlock	Libera um mutex

Tabela 4.1 – Chamadas ao threads usadas pelo LIDS

4.2.3. Uso de *Threads*

O LIDS, na versão original, conforme a Figura 4.3, trabalhava de maneira seqüencial: Os pacotes que trafegavam pela rede eram capturados, analisados, e detectado uma invasão, registrado .

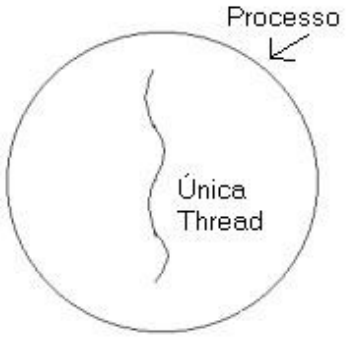

Figura 4.3 - Processo com um único *thread*

No intervalo de uma operação e outra, pacotes poderiam não ser capturados. Como o processamento da análise requer um tempo maior do que o da captura, pensou-se num mecanismo para otimizar o sistema. Para isto, utilizou-se as *threads*. O programa foi dividido em duas threads: uma para capturar, e a outra, para fazer a análise. Veja a Figura 4.4.

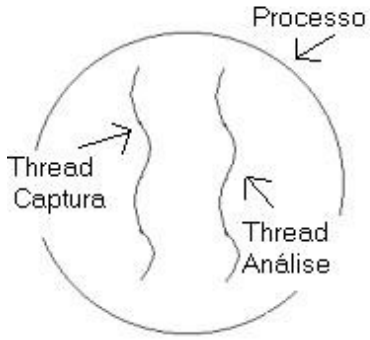

Figura 4.4 – Processo com duas threads.

Para verificar se o uso de *threads* tornaria o sistema mais eficiente, desenvolveram-se dois protótipos, denominados ids1 e ids2.

O ids1 captura e analisa os pacotes, comparando com uma palavra pré-determinada (simulando uma assinatura de ataque), exibindo a mensagem “Tentativa de invasao”, quando algum pacote analisado contém a palavra analisada. A Figura 4.5 resume o programa

```
Loop infinito
{
        pacote = captura();
       analisa(pacote);
}
```

Figura 4.5 - Esquema simplificado do ids1

O ids2 por sua vez, utiliza-se de multi-threads: Criou-se um buffer, que é manipulado em uma Região Critica, garantindo a atomicidade da operação. Com isto, a fatia de tempo destinada ao módulo de captura, tornou-se maior e diminuiu a perda de pacotes.

```
fila *fifo; //  descritor para a fila
pthread_t cap, ana; //descritor dos threads de captura e análise
fifo = queueInit (); // cria a fila do buffer
pthread_create (&cap, NULL, produtor, fifo);  // Thread da captura
pthread_create (&ana, NULL, consumidor, fifo); // Thread da analise
```

Figura 4.6 - Uso dos pthreads

4.3. Carregamento na memória das assinaturas de ataques.

O próximo passo foi a utilização das assinaturas de ataque na comparação com os pacotes capturados. Para isto, criou-se o ids3 e o ids4. Ao comparar com os ids1 e ids2, percebeu-se uma degradação significativa, ao utilizar-se o banco de dados.

Isto motivou o carregamento das assinaturas de ataque na memória do sistema. Desenvolveu-se a função CARREGA_MEMORIA: que cria um vetor dinâmico e ao iniciar o ids, o programa armazena neste todas as assinaturas, com isto, o módulo de análise utiliza este vetor, ao invés do banco de dados.

4.4. Filtragem do conteúdo dos pacotes capturados

Durante a implementação destes protótipos, verificou-se a necessidade de uma melhor filtragem dos pacotes. Para isto, desenvolveu-se a função RETIRA_ESTRANHOS, que recebe o pacote como argumento, e retorna o pacote sem os caracteres indesejáveis.

4.5. Módulo de Bloqueio

A necessidade do LIDS em ter uma resposta ativa levou ao desenvolvimento do módulo de bloqueio.

O módulo de bloqueio é ativado imediatamente após ter sido registrada a tentativa de invasão, e atua no firewall do sistema operacional, bloqueando o endereço e porta da origem,

O sistema operacional Linux, a partir da versão 2.4 de seu núcleo (kernel), utiliza como firewall padrão, o *iptables*.

O iptables é um filtro de pacotes e funciona baseado em regras pré-definidas, que analisa o endereço e a porta de origem e destino dos pacotes, liberando ou negando os pacotes. Um exemplo de regras pode ser observado na Figura 4.7:

```
iptables –A INPUT –s 192.168.1.101 –j DROP
iptables –A FORWARD –s 10.0.0.0/8 –sport 80 –d 192.168.0.0/24 –j ACCEPT
```

Figura 4.7 - Exemplos de regras no iptables

A primeira regra, o iptables bloqueia todos os pacotes oriundos da máquina 192.168.1.101 que tem como destino a máquina do firewall. Na segunda, ele libera todos os pacotes que saem da rede 10.0.0.0 com destino a rede 192.168.0.0, usando a porta 80, roteados pelo firewall.

Ao projetar um firewall, há basicamente dois modos: o primeiro, consiste em liberar todo o tráfego, bloqueando alguns específicos (como demonstrado na primeira regra), ou bloquear todo o tráfego, liberando conforme necessário (conforme a segunda regra). Para definir isto, cria-se a regra padrão, ou seja, quando o pacote é analisado, e não é encontrada nenhuma regra específica pra ele, utiliza-se a regra padrão. A Figura 4.8 mostra exemplos de regras padrões. Neste caso, a política padrão é bloquear todos os pacotes que chegam, saem ou são roteados pelo firewall.

```
iptables –P INPUT DROP
iptables –P OUTPUT DROP
iptables –P FORWARD DROP
```

Figura 4.8 - Bloqueio como regra padrão

No caso do LIDS, adotou-se como padrão, a liberação de todo tráfego, bloqueando conforme necessário. Veja na Figura 4.9:

```
iptables –P INPUT ACCEPT
iptables –P OUTPUT ACCEPT
iptables –P FORWARD ACCEPT
```

Figura 4.9 - Regra padrão do LIDS

O módulo de bloqueio ao ser ativado, verifica se o destino da tentativa de invasão é a máquina do firewall. Caso seja, utilizará o seguinte comando:

```
iptables –A INPUT –s $IP_ORIGEM –p tcp –sport $PORTA_ORIGEM –j DROP
```

Figura 4.10 - Bloqueio para pacotes endereçados ao LIDS

Caso o destino seja outra máquina e o pacote é roteado pelo firewall, o comando a ser utilizado é apresentado Figura 4.11:

```
iptables –A FORWARD –s $IP_ORIGEM –p tcp –sport $PORTA_ORIGEM –d
$IP_DESTINO  –j DROP
```

Figura 4.11 - Bloqueio para pacotes roteados pelo LIDS

Com isto, seu funcionamento é bem simples: Verifica-se na tabela tbl_invasão, os endereços e portas de origem e destino, e ativa as regras. Para que o administrador tenha conhecimento de tal ação, registra-se em uma tabela do banco de dados; a tbl_bloqueio, o evento ocorrido. A tbl_bloqueio contém os seguintes campos:

Campo	*Descrição do campo*
Bloqueio	Chave primária da tabela
Data	Data/hora da ação.
Invasão	O número da invasão registrado na tabela tbl_invasao
Comando	A regra do firewall utilizada.

Tabela 4.2 – Campos da tabela tbl_bloqueio

4.5. Testes

Desenvolvido os módulos, realizou-se testes procurando verificar seu desempenho. A seguir, será mostrado o ambiente experimental e os testes realizados.

4.5.1. Ambiente experimental

Na Figura 4.12 podemos observar o ambiente utilizado para o desenvolvimento do LIDS.

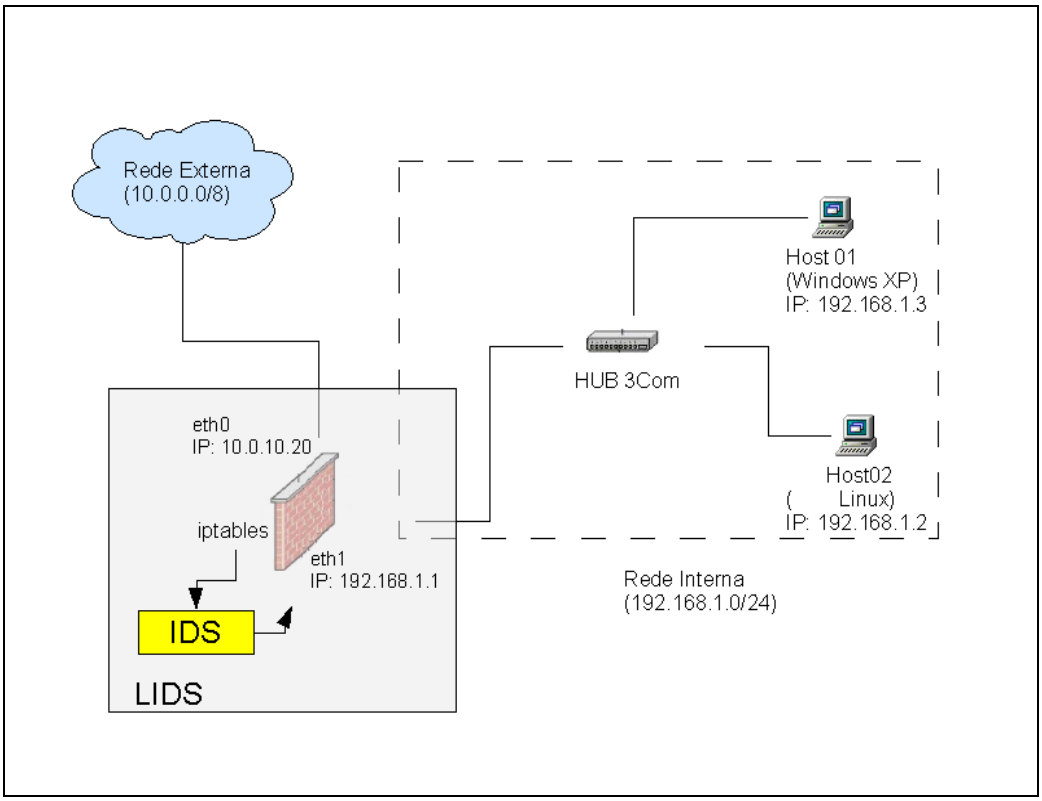

Figura 4.12 - Ambiente experimental do LIDS

O ambiente experimental utilizado foi uma rede corporativa composta de diversas máquinas. Criou-se outra rede interna, sendo a máquina onde o LIDS foi instalado, o computador intermediário entre as duas, tornando-se um gateway. Possui duas interfaces de rede: *eth0* e *eth1*. A comunicação com a rede externa, a 10.0.0.0/8, é feita através da interface de rede *eth0*, e com a rede interna, a 192.168.1.0/24, através da *eth1*. Esta rede interna possui, além do LIDS, duas máquinas, uma rodando Windows XP, e a outra, Linux DreamLinux. As máquinas da rede interna são interligadas através de um HUB 3COM de 10 Mbps, de 12 portas.

O computador hospedeiro de testes do LIDS possui a seguinte configuração: Processador Pentium 100 MHz, com 32 MB de memória RAM, e disco rígido de 3 GB. O sistema operacional utilizado foi o Linux Debian 3.1, com kernel 2.4. Escolheu-se o Debian, devido a sua robustez, e facilidade de uso. Através da ferramenta APT-GET; a instalação e

atualização de pacotes é uma atividade trivial, como veremos a seguir.

A Figura 4.13 mostra os pacotes que foram instalados, através do APT-GET:

```
# apt-get update
# apt-get install libpcap0 libpcap-dev make gcc vim apache mysql-server
mysql-client phpmyadmin
```

Figura 4.13 - Instalação pelo APT-GET

O primeiro comando atualiza a base de dados da máquina, com os repositórios dos servidores Debian. Com isto, garante-se a instalação de pacotes nas versões mais recentes.

O segundo comando instala os programas necessários para o desenvolvimento do LIDS.

O programa executável do LIDS encontra-se no diretório /usr/local/lids. O banco de dados utilizado pelo LIDS denomina-se bd_lids. Para compilar o LIDS, usa-se o comando, conforme a Figura 4.14:

```
# gcc -o lids lids.c –lpcap –lpthread -lmysqlclient
```

Figura 4.14 - Compilando o LIDS

4.5.2. Resultados obtidos

O primeiro teste realizado foi verificar qual modelo apresentaria uma eficácia maior: o mono-thread ou o mult-thread. Conforme falado anteriormente, foi desenvolvido o ids1 e o ids2, sendo o primeiro mono, e o segundo, multi-thread. Para verificar isto, desenvolveu-se o programa dispara, cujo objetivo foi de enviar para a rede, uma quantidade de pacotes pré-definidas com o conteúdo igual ao da palavra utilizada como assinatura de ataque pelos ids1 e ids2. Com isto, após o programa terminar sua operação, verificava no *log* dos ids1 e ids2 a quantidade de pacotes capturados.

Na Figura 4.15 observam-se os resultados obtidos

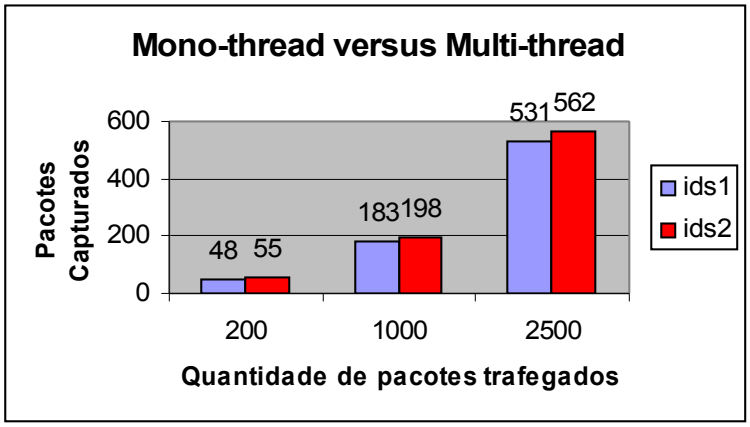

Figura 4.15 – Análise dos IDS1 e IDS2

Pode-se constatar que o protótipo executando em multi-thread apresentou uma eficácia superior em relação ao mono-thread.

O segundo teste foi verificar como o sistema se comportaria ao utilizar a base de dados para buscar as assinaturas. Com isto, utilizou-se os ids3 e ids4, programado em mono-thread e multi-thread, respectivamente. A Figura 4.16 mostra o comparativo entre os quatro protótipos:

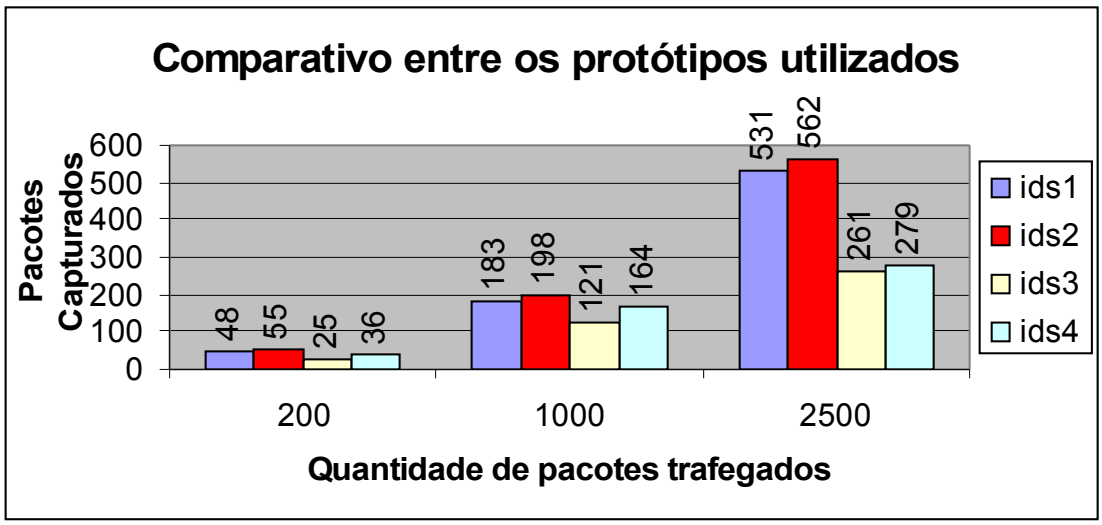

Figura 4.16 – Análise dos IDS1 ao IDS4

Observou-se uma degradação muito grande ao utilizar o banco de dados.

A partir destes testes, foi desenvolvido o lids.c. O programa ao iniciar, carrega as assinaturas de ataque para a memória, cria as duas *threads*: uma para a captura, e outra para análise dos pacotes. Registrado uma tentativa de invasão, ativa o módulo de bloqueio, e o firewall bloqueia o endereço de origem.

Para que pudesse analisar seu funcionamento, inseriu-se nas assinaturas do LIDS, algumas usadas pelo IDS *Snort*. As assinaturas do snort são baseadas em regras de condições do tipo *if-then*, ou seja, encontrada determinadas assinaturas, dispara novas ações ou novas avaliações (discutido no Capítulo 2). Precisou então, adaptá-las ao LIDS.

Ao colocar em funcionamento, percebeu-se uma enorme quantidade de falsos positivos. Isto acarretou enorme problema, principalmente ao bloquear endereços como os servidores da rede e os *hosts* da rede interna.

5. CONCLUSÕES

Após toda a pesquisa sobre a questão de segurança em redes, conhecendo algumas das técnicas usadas para invadir sistemas computacionais, e por outro lado, as ferramentas usadas para combater tais atividades, constatou-se que há uma luta incessante entre quem procura invadir ou destruir sistemas e quem precisa se proteger.

As ações para a defesa precisam ser preventivas e corretivas. A primeira, cria uma barreira, e a segunda, mecanismos para reagir quando o invasor passa pela primeira linha de defesa. As duas precisam trabalhar em conjunto.

Os sistemas de detecção de intrusão são uma ferramenta imprescindível para qualquer ambiente computacional, não importando seu tamanho ou finalidade.

A utilização do sistema operacional *Linux* facilitou bastante a implementação do LIDS, principalmente ao utilizar a distribuição Debian. A ferramenta *APT-GET* proporciona uma grande facilidade na instalação e atualização de programas e bibliotecas. Instalou-se o LIDS em uma máquina com um hardware bastante modesto, mas seu desempenho mostrou-se, ainda assim, bastante eficaz.

A ferramenta utilizada para manipular o banco de dados MySQL, foi a phpMyAdmin, uma ferramenta, feita na linguagem *php*, acessada via *browser*. Isto facilitou em muito a análise dos resultados.

Ao implementar o LIDS, verificou-se a facilidade de interceptação das mensagens que circulam pela rede. Embora não fosse o propósito do trabalho, mas ao utilizar a biblioteca *libpcap*, capturou-se mensagens e endereços de emails, sites, e conteúdo de arquivos. Com isto, torna-se de vital importância a orientação aos usuários da rede sobre como utilizar seus recursos computacionais.

Ao comparar os pacotes capturados com as assinaturas de ataque presentes no LIDS, primeiro constatou-se a grande quantidade de falsos positivos. Isto se deve ao fato do LIDS

ainda não ser um sistema especialista, ou seja, capaz de analisar não somente um pacote e constatar que é uma intrusão, mas sim, construir uma base de conhecimentos, para tomar uma decisão mais correta.

6. IMPLEMENTAÇÕES FUTURAS

Analisando o desempenho do LIDS, e após os testes realizados, um item a ser desenvolvido em implementações futuras, é a criação de bases de conhecimento, usando por exemplo, redes neurais, para a criação de um sistema especialista, eliminando com isto, a incidência de falsos positivos.

Uma outra implementação possível é o da descentralização dos sensores do detector de captura de pacotes com o restante do sistema. Em uma rede de computadores mais complexa, em determinados pontos, há um tipo especifico de dados a serem analisados. Com isto, adapta-se cada sensor ao ambiente especifico, e estes se comunicam com uma base centralizada de conhecimento. Com isto, centralizaria a análise em um único ponto, mas os sensores de captura seriam distribuídos conforme a necessidade. Conceitos como cliente-servidor e sistemas distribuídos podem ser empregados.

O módulo de bloqueio pode ganhar diversas melhorias. Este foi desenvolvido para trabalhar somente com o *iptables*, o firewall padrão do Linux, e poderia trabalhar com outros firewalls existentes. A criação de uma lista de máquinas que nunca podem ser bloqueadas pelo firewall, como por exemplo os servidores da rede, também seria de grande valia.

REFERÊNCIAS

CAMPELLO, Rafael Saldanha, WEBER, Raul Fernando. **Sistemas de Detecção de Intrusão.** Disponível em: <www.inf.ufrgs.br/~gseg/producao/minicurso-ids-sbrc-2001.pdf>. Acesso em: 28 mai. 2006.

CERIAS, **The Center for Education and Research in Information Assurance and Security.** Disponível em: <http://www.cerias.purdue.edu/about/history/coast_resources/intrusion_detection/>. Acesso em: 28 mai. 2006.

CHESWICK, W.R. e Bellovin, S.M. **Firewalls and Internet Security: repelling the wily hacker.** Addison-Wesley Publishing Company: 1994.

FERRAMENTAS LIVRES PARA LINUX. Disponível em: <http://www.debianhelp.co.uk/tools.htm>. Acesso em: 25 fev. 2006

FERREIRA, Thiago Barroso. **LIDS – LINUX INTRUSION DETECTION SYSTEM.** [Marília] 2004. 52 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Fundação de Ensino "Eurípides Soares da Rocha", Mantenedora do Centro Universitário Eurípides de Marília – UNIVEM, 2004.

Internacional Organization for Standardization, ISO 7498-2 Basic Reference Model for Open Systems Interconnection (OSI), Internacional Organization for Standardization, Geneva, Suíca, 1988.

LAFETÁ, Alessandra Valle; ARRUDA, Giuliano Macedo. **IMPLEMENTAÇÃO DE UM HONEYPOT COMO PARTE DO PROJETO DE HONEYPOTS DISTRIBUÍDOS E ANÁLISE ESTATÍSTICA DE ATAQUES À REDE DA UNB** [Distrito Federal] 2005. 102f. Trabalho de Conclusão de Curso (Graduação em Engenharia Elétrica) – Departamento de Engenharia Elétrica da Faculdade de Tecnologia da Universidade de Brasília, 2005.

OLIVEIRA, Frank Ned Santa Cruz de. **Ferramentas de IDS.** Disponível em: <http://www.rnp.br/newsgen/9909/ids.html#p5>. Acesso em: 29 mai. 2006.

OSSEC HIDS Home Page. Disponível em: <http://www.ossec.net/en/about.html>. Acesso em: 28 mai. 2006.

Prelude Hybrid IDS Project Home Page. Disponível em: <http://www.prelude-ids.org/>. Acesso em 01 jun. 2006.

RIBEIRO, Uirá. **Certificação Linux.** Axcel Books: 2004.

SANTANA, Daniel Menezes**. Libpcap – a Biblioteca para Análise de Rede.** Disponível em: <www.secforum.com.br/textos/Libpcap.doc>. Acesso em: 07 jun. 2006.

SILVA, Artur Renato Araujo da**. UM MODELO REPRESENTATIVO DE ASSINATURAS DE ATAQUE PARA SISTEMAS DETECTORES DE INTRUSÃO.** [São José do Rio Preto] 2002. 53 f. Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) – Departamento de Ciências da Computação e Estatísticas do Instituto de

Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista Júlio de Mesquita, 2002. Disponível em: <http://www.acmesecurity.org/publicacoes/monografias/folder.2005-12-27.0965990514/acme-pf-2002-artur-final.pdf>. Acesso em: 07 jun. 2006.

Snort Brasil. Disponível em: <http://www.clm.com.br/snort/> Acesso em: 05 jun. 2006.

Snort Home Page. Disponível em: <http://www.snort.org>. Acesso em 01 jun. 2006.

TANENBAUM, Andrew S. **Sistemas operacionais modernos**. Tradução: Ronaldo A.L. Gonçalves, Luís A. Consularo. Pearson Prentice Hall, 2003.

Wikipédia, a enciclopédia livre. Disponível em: http://pt.wikipedia.org/wiki/Firewall>. Acesso em: 06 jun. 2006.