UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE SOFTWARE PARA A MONITORAÇÃO DE PACOTES EM UMA REDE TCP/IP EM AMBIENTES LINUX

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

JORGE LUIZ POMPERMAYER JUNIOR

BLUMENAU, JUNHO/2002

2002/1-43

PROTÓTIPO DE SOFTWARE PARA A MONITORAÇÃO DE PACOTES EM UMA CONEXÃO TCP/IP EM AMBIENTES LINUX

JORGE LUIZ POMPERMAYER JUNIOR

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Francisco Adell Péricas — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Francisco Adell Péricas

_____

Prof. Sérgio Stringari

_____

Prof. Alexander Roberto Valdameri

Dedico este trabalho aos meus pais, Jorge e Eonice, à minha irmã, Tuca, pela companhia e à minha namorada, Aia, pela compreensão, paciência e incentivo.

AGRADECIMENTOS

Agradeço ao Patrão Velho lá de cima pelas oportunidades de crescimento nesta invernada terrena, pois quem não se afirma nos arreios da vida, não se estriba na proteção do céu.

Agradeço aos professores do curso de Bacharelado em Ciências da Computação desta universidade pelos ensinamentos ao decorrer no curso. Um chasque especial ao meu orientador, Francisco Adell Péricas, por toda ajuda no desenvolvimento do presente trabalho.

À Aia por todo incentivo, paciência e compreensão que se mostraram essenciais para o sucesso alcançado.

Aos meus familiares, pela base pessoal ensinada, e aos amigos, pelas dicas, apoio e risadas trocadas nos momentos de dificuldade. Para o Bento e Teddy, muito obrigado.

SUMÁRIO

LISTA DE SIGLAS E ABREVIATURAS

ACL – *Access Control List*

ARP – *Address Resolution Protocol*

ARPANET – *Advanced Research Projects Agency Network*

DNS – *Domain Name Server*

DoS – *Denial of Service*

FAT – *File Access Table*

FTP – *File Transfer Protocol*

HTTP – *Hyper Text Transfer Protocol*

ICMP – *Internet Control Message Protocol*

IP – *Internet Protocol*

RSA – *Rivest, Shamir and Adleman*

SMTP – *Simple Mail Transfer Protocol*

TCP – *Transmission Control Protocol*

TCP/IP – *Transmission Control Protocol/Internet Protocol*

TelNet – *Telecommunications Network*

UDP – *User Datagram Protocol*

WWW – *World Wide Web*

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Este trabalho apresenta a especificação e implementação de um protótipo de software para a monitoração de pacotes em um computador conectado a uma rede *Ethernet.* Apresenta também um estudo sobre a segurança em redes de computadores. O protótipo foi desenvolvido para ambientes Linux.

ABSTRACT

The main ideia of this article is to implement and especify a software prototipe for monitoring the TCP/IP packets, on a computer conected on Ehternet network. This job shows too a study about de security on computer networks. The software prototipe, was designed for Linux environments.

1 INTRODUÇÃO

Com a constante evolução das tecnologias da informação, tem havido um crescimento proporcional na necessidade da disponibilização de dados de forma rápida e segura. Dados que são considerados mais valiosos do que os próprios equipamentos.

A questão da rapidez foi solucionada mas não a da segurança. Sabendo que seus dados encontram-se em áreas abertas, as empresas preocuparam-se em ampliar seus métodos de proteção desses dados contra possíveis espiões industriais, contra seus concorrentes e, principalmente, contra a ação de *hackers*.

Os dados, para trafegarem em uma rede, passam por um processo de codificação, passando de informações alfanuméricas para *bits*. Esses *bits* ficam agrupados em forma de pacotes e então são transmitidos pela rede.

Sendo assim, não se tem mais o controle de quais pacotes estão trafegando, qual o destino dos pacotes, qual sua origem e que dado está trafegando.

Segundo Silva (2001), ferramentas de monitoração são importantes, pois através delas pode-se determinar com grande rapidez se houve realmente uma invasão à rede e quais os danos causados por esta invasão, auxiliando o administrador de rede a tomar providências para que os danos causados pela invasão sejam contidos o mais rápido possível.

Monitores, segundo McClure (2000), capturam, interpretam e armazenam, para análise posterior, pacotes que viajam por uma rede. Ferramentas como esta são de grande valia para o administrador da rede, pois possibilitam uma série de informações e diagnósticos do que está trafegando pela rede. Ainda segundo McClure (2000), um *sniffer* é um programa monitor que funciona em conjunto com a placa de rede para extrair todo o tráfego da rede, em vez de somente o tráfego endereçado ao *host* onde ele estiver instalado.

O protótipo desenvolvido neste trabalho é um monitor de pacotes, para o ambiente Linux, aplicado a um *host* conectado a uma rede *Ethernet*.

1.1 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho consiste em especificar e desenvolver um protótipo de software de monitoração de pacotes que trafegam em uma rede *Ethernet* e alertar o usuário quando ocorrerem situações suspeitas.

Os objetivos específicos do trabalho são:

a) interceptar e interpretar pacotes TCP/IP;

b) analisar dados contidos nos pacotes;

c) identificar situações suspeitas;

d) armazenar as informações monitoradas.

1.2 ESTRUTURA DO TRABALHO

O trabalho está organizado em cinco capítulos, estando assim distribuídos:

No capítulo 1, encontra-se a introdução, os objetivos e a organização do trabalho.

No capítulo 2, são apresentados fundamentos sobre a Internet e o protocolo TCP/IP, necessários para o desenvolvimento deste trabalho.

No capítulo 3, é apresentado um estudo sobre a segurança em redes de computadores, abordando problemas e soluções.

No capítulo 4, é apresentado o Linux, bem como algumas características e tecnologias que serão utilizadas no protótipo.

O capítulo 5 apresenta a especificação e implementação do protótipo.

Após o capítulo 5 são apresentadas conclusões, sugestões para trabalhos futuros e referências bibliográficas utilizadas para a elaboração deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados os fundamentos e os conceitos das tecnologias que servem de base para o desenvolvimento do trabalho. Estas tecnologias são importantes para que sejam alcançados os objetivos do trabalho e o conseqüente entendimento destes objetivos.

2.1 CONSIDERAÇÕES SOBRE TRABALHOS ANTERIORES

O presente trabalho dispõe-se a fazer uma continuação, ou melhor, cumprir uma das extensões sugeridas por Silva (2001): “desenvolver o suporte a outros sistemas operacionais, como Windows NT/2000 e Linux”. Neste caso, foi implementado um protótipo para o sistema operacional Linux para a filtragem de pacotes trafegando em redes *Ethernet*.

O trabalho desenvolvido em Silva (2001) trata da especificação e desenvolvimento de um protótipo de software de monitoração de pacotes, que tem a capacidade de monitorar os pacotes que trafegam entre um *host* e a Internet e alertar o usuário quando correrem situações suspeitas.

2.2 A INTERNET – UM POUCO DA HISTÓRIA

Grande parte da Internet resultou da "evolução" de um sistema criado em 1969. Nesta época, uma divisão do Departamento de Defesa dos EUA - a Agência de Projetos de Pesquisa Avançada em Defesa (DARPA) – concluiu que o país precisava criar um modo fácil de trocar informações militares entre cientistas e pesquisadores localizados em diferentes regiões geográficas.

O objetivo, segundo Stang (1994), era desenvolver um conjunto de protocolos de comunicação, os quais permitiriam computadores ligados em rede comunicarem-se de forma transparente entre várias redes diferentes. O sistema de protocolos que foi criado chamou-se TCP/IP.

Uma rede simples de quatro computadores, conhecida como DARPANET, foi desenvolvida. Algum tempo depois foi rebatizada de ARPANET e, em 1972, cresceu a ponto de incluir 37 computadores e, ao mesmo tempo, o modo de utilização da rede começou a mudar. Além de ser empregada para trocar informações importantes, sobre atividades militares, os usuários da ARPANET, começaram a enviar mensagens eletrônicas por meio de caixas de correio pessoais.

Em 1990 a Internet comercial começou a funcionar. Desde então, o crescimento da internet tem sido simplesmente um fenômeno. O número de usuários saltou de 5.000 pessoas para cerca de 30 milhões em apenas dez anos, resultando em crescimento vertiginoso de 6.000%.

2.2.1 ORGANIZAÇÃO

Segundo Silva (2001), a estrutura da Internet é formada basicamente por três figuras principais: os *backbones*, os provedores de acesso e os usuários finais.

Backbones, segundo Silva (2001), são grandes redes de computadores de alta velocidade, projetadas para a transmissão de um grande volume de dados. Pode-se dizer que os *backbones* são a espinha dorsal das redes de computadores. A Internet é formada por vários *backbones* espalhados pelo mundo, sendo que seu *backbone* principal encontra-se nos EUA.

Um provedor de acesso conecta-se diretamente aos *backbones* da Internet, formando conexões dedicadas e permanentes. O objetivo do provedor de acesso é fornecer aos usuários finais acesso à Internet e a outros serviços correlatos.

Os usuários finais podem ser particulares ou empresas que desejam obter acesso à Internet. Os usuários finais conectam-se aos provedores de acesso através de uma rede local ou da rede telefônica, via modem, e podem utilizar todos os recursos disponíveis na Internet.

A estrutura dos *backbones*, provedores de acesso ou usuários finais na Internet é formada por *hosts*, que são computadores que estão conectados a uma única rede física e têm o objetivo de executar programas aplicativos, e roteadores, que são computadores conectados a duas ou mais redes físicas e são responsáveis pela transmissão entre as redes.

2.2.2 OS RECURSOS DA INTERNET

Mesmo possuindo recursos ilimitados, a cada dia surgem novas possibilidades e tecnologias de utilização da rede. Cita-se os principais:

a) **correio eletrônico:** baseado no protocolo de aplicação *Simple Mail Transfer Protocol* (SMTP), é um recurso que permite a troca de mensagens entre usuários. A troca das mensagens é baseada em endereços no formato usuário@domínio, onde “usuário” corresponde a um único usuário ou a um conjunto de usuários, e

"domínio" corresponde ao nome do domínio de uma rede onde está localizada a caixa postal do usuário;

b) **telnet (*Telecomunications Network*):** trata-se de um emulador de sistemas que, através de um programa "cliente telnet", permite ao usuário conectar-se a outro *host* e, dependendo das suas permissões, executar aplicativos deste *host*, como se estivesse trabalhando nele localmente;

c) ***network news*:** baseado no protocolo de aplicação *Network News Transfer Protocol* (NNTP), fornece artigos referentes a vários assuntos, que são agrupadas em categorias denominadas *newsgroups*. Através de um programa de leitura de *news* o usuário pode conectar-se a um servidor de *network news* e ler os artigos disponíveis;

d) **transferência de arquivos:** baseado no protocolo de aplicação *File Transfer Protocol* (FTP) é um recurso especializado na transmissão de arquivos pela Internet. Através de um cliente de FTP o usuário pode conectar-se a um *host*, analisar o sistema de arquivos em que está conectado e escolher quais arquivos deseja pegar ou enviar;

e) **www (*World Wide Web*):** baseado no protocolo de aplicação *Hipertext Transfer Protocol* (HTTP) é um sistema de busca de informações em que a passagem de um documento para outro está embutida no próprio documento. A este mecanismo é dado o nome de navegação por hipertexto. A www permite que sejam incorporados aos documentos recursos de multimídia e imagens. O usuário utiliza um programa denominado *browser* para ler os documentos da www.

2.3 *ETHERNET* E IEEE 802.3

A *Ethernet* é a tecnologia de rede local (LAN) mais amplamente usada. A *Ethernet* foi projetada para ocupar o espaço entre as redes de longa distância, com baixa velocidade e as redes especializadas de sala de computação que transportam dados em alta velocidade por distâncias muito limitadas. A *Ethernet* é bem adequada a aplicativos em que um meio de comunicação local deva transportar tráfego esporádico, ocasionalmente intenso, a altas taxas de dados.

A arquitetura de rede *Ethernet,* segundo Tanenbaum (1997), tem suas origens nos anos 60, na Universidade do Havaí, onde o método de acesso que é usado pela *Ethernet*,

carrier sense multiple access/collision detection (CSMA/CD), foi desenvolvido. O *Palo Alto Research Center* (PARC), da *Xerox Corporation*, desenvolveu o primeiro sistema *Ethernet* experimental no início dos anos 70. Isso foi usado como base para a especificação 802.3 do *Institute of Electrical and Electronic Engineers* (IEEE), lançada em 1980.

Logo após a especificação 802.3 de 1980 da IEEE, a *Digital Equipment Corporation*, a *Intel Corporation* e a *Xerox Corporation* desenvolveram conjuntamente e lançaram uma especificação *Ethernet*, versão 2.0, que foi substancialmente compatível com a IEEE 802.3. Juntas, a *Ethernet* e a IEEE 802.3 detêm atualmente a maior fatia de mercado de todos os protocolos LAN. Hoje, o termo *Ethernet* é freqüentemente usado para se referir a todas as LANs baseadas em CSMA/CD que normalmente estão em conformidade com as especificações *Ethernet*, incluindo a especificação IEEE 802.3.

A *Ethernet* e a IEEE 802.3 especificam tecnologias similares: ambas são LANs baseadas em CSMA/CD. Estações em uma LAN CSMA/CD podem acessar a rede a qualquer momento. Antes de enviar dados, as estações CSMA/CD “escutam” a rede para determinar se ela já está em uso. Se estiver, então elas aguardam. Se a rede não estiver em uso, as estações transmitem. Uma colisão ocorre quando duas ou mais estações “escutam” o tráfego da rede, não ouvem nada e transmitem simultaneamente. Neste caso, as transmissões são prejudicadas e as estações devem retransmitir mais tarde. Algoritmos de recuo determinam quando as estações que colidiram podem retransmitir. As estações CSMA/CD devem detectar colisões, assim, elas sabem quando devem parar a transmissão para retransmitir mais tarde.

Tanenbaum (1997) diz que ambas as LANs *Ethernet* e IEEE 802.3 são redes de *broadcast*. Isso significa que todas as estações de uma LAN podem ver todos os quadros, independentemente de serem ou não o destino daqueles dados. Cada estação deve examinar os quadros recebidos para determinar se ela é o destino. Se for, o quadro é passado a um protocolo de camada mais alto dentro da estação para processamento apropriado.

As diferenças entre as LANs *Ethernet* e IEEE 802.3 são sutis. A *Ethernet* fornece serviços correspondentes às camadas 1 e 2 do modelo de referência OSI, enquanto a IEEE 802.3 especifica a camada física, a camada 1, e a parte de acesso a canais da camada de enlace, a camada 2, mas não define um protocolo de controle de enlace lógico, o que é feito pelo IEEE 802.2 (*Logical Link Control* - LLC). Ambas as LANs *Ethernet* e IEEE 802.3 são implementadas através de hardware. Normalmente, a parte física desses protocolos é uma

placa de interface em um *host* ou um conjunto de circuitos em uma placa de circuitos principal no *host*.

2.3.1 ÁRVORE DA FAMÍLIA *ETHERNET*

Segundo Tanenbaum (1997), existem pelo menos 18 variedades de *Ethernet* especificadas ou em processo de especificação. O Quadro 1 realça algumas das mais comuns e mais importantes tecnologias *Ethernet*.

Quadro 1: Tecnologias do padrão *Ethernet*

Tipo	Meio	Taxa máxima	Comprimento máximo do segmento	Topologia física	Topologia lógica
10Base-T	UTP Cat 5	10Mbps	100m	Estrela	Barramento
10Base-FL	Fibra óptica	10Mbps	2000m	Estrela	Barramento
100Base-TX	UTP Cat 5	100Mbps	100m	Estrela	Barramento
100Base-FX	Fibra óptica	100Mbps	2000m	Estrela	Barramento

2.4 INTRODUÇÃO AO TCP/IP

O termo TCP/IP refere-se aos dois principais protocolos que o constituem: *Transmission Control Protocol* (TCP) e o *Internet Protocol* (IP). Existem inúmeros outros protocolos que permitem a comunicação entre computadores, sendo o TCP/IP o mais difundido atualmente.

O início se deu quando o Departamento Americano de Defesa, no final da década de 60, resolveu criar um *software* de comunicação entre computadores de forma que fosse possível a comunicação remota ou local, entre sistemas operacionais iguais ou diferentes utilizando ou não o mesmo tipo de hardware. O protocolo permitiu que, no começo da década de 70, os computadores pudessem comunicar-se remotamente, sendo de suma importância em navios de guerra ou entre porta-aviões.

Segundo Starlin (1999), para que isto fosse possível, foi necessário criar um sistema de comunicação onde a rede como um todo fosse dividida em “pequenas” redes independentes, passando a usar como padrão de interconexão o TCP/IP.

Como o Departamento Americano de Defesa sempre foi um dos maiores compradores de sistemas de informática, comunicou que somente compraria as tecnologias que tivessem incorporado o TCP/IP. Desde então, todos os sistemas operacionais e produtos trazem implementado o conjunto TCP/IP.

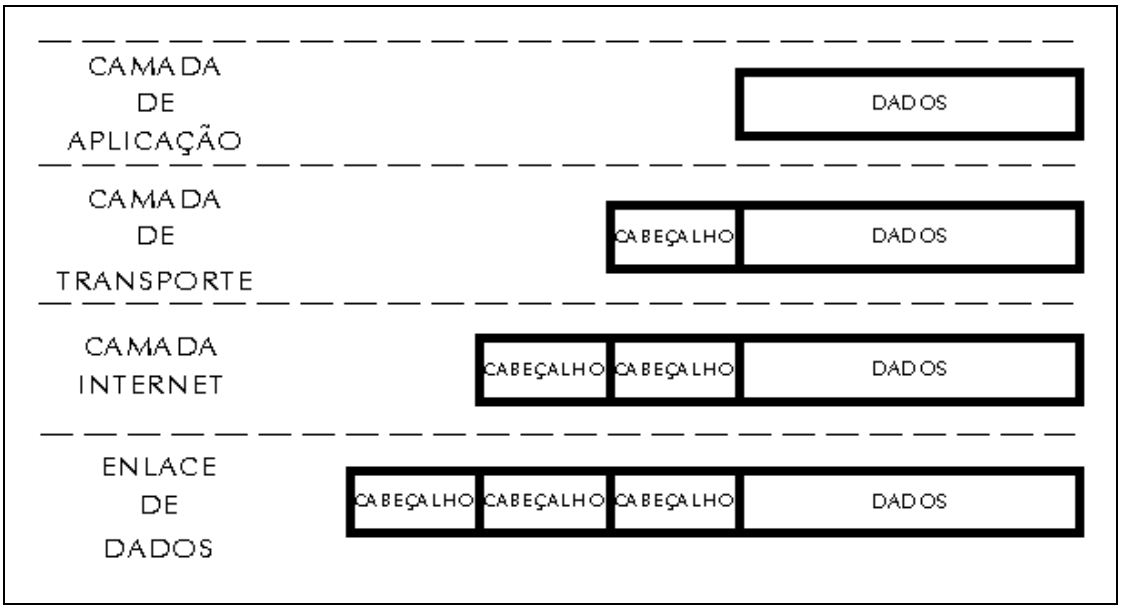
O TCP/IP possui como algumas de suas características básicas a independência tecnológica, interconexão universal, comunicação fim-a-fim, padrões de protocolos para as diversas aplicações. Uma das características mais importantes é a flexibilidade de adaptação às tecnologias de redes existentes e futuras, pois o TCP/IP foi concebido de forma independente das tecnologias de redes.

A comutação por pacote (*packet-switching)*, segundo Redes (2001), é a base de sua tecnologia de comunicação e utiliza o pacote (*datagram*) como unidade de transmissão de dados. Cada *host* conectado à rede possui um endereço único, possibilitando que a máquina emitente reconheça a máquina receptora.

2.4.1 ARQUITETURA TCP/IP

Segundo Comer (1998), a arquitetura TCP/IP, apresentada na Figura 1, trata de um conjunto de protocolos divididos em quatro camadas, sendo que cada uma destas camadas possui funções bem definidas.

Figura 1: Camadas da arquitetura TCP/IP



Conforme a Figura 1, as camadas são:

a) camada de aplicação: segundo Tanenbaum (1997) é nesta camada que estão os protocolos de alto nível como terminal virtual (TELNET), protocolo de transferência de arquivos (FTP), protocolo de envio de correio eletrônico (SMTP) entre outros. É nesta camada que estão os programas dos usuários.
b) camada de transporte: responsável por uma comunicação entre dois *hosts* fim a fim, podendo oferecer comunicações orientadas ou não orientadas à conexão. Fazem parte desta camada os protocolos *Transmission Control* Protocol (TCP) e *User Datagram Protocol* (UDP).
c) camada inter-redes ou internet: é onde está implementado o protocolo *Internet Protocol* (IP). Equivalente à camada de Rede no modelo OSI, é onde é feito o roteamento e a entrega dos pacotes IP.
d) camada de enlace: segundo Silva (2001), é responsável por encapsular os pacotes da camada inter-redes no formato específico da rede associada e extrair os pacotes dos quadros vindos da rede e encaminhá-los à camada Inter-redes. O protocolo ARP encontra-se nessa camada.

2.4.2 O PROTOCOLO IP

O *Internet Protocol* (IP) teve origem em 1970 no desenvolvimento da ARPANET, a qual foi depois interligada a outras redes formando em 1980 um vasto conjunto que passou a ser conhecido por Internet. Com a inclusão do protocolo IP no UNIX, em 1982, um grande número de universidades passou a formar as suas redes que por sua vez também foram ligadas à Internet.

O *Internet Protocol* (IP) é designado para o uso em sistemas de computadores interconectados (redes), utilizando troca de pacotes. O IP fornece transmissão de blocos de dados chamados datagramas entre origem e destino, sendo que origem e destino são *hosts* identificados por um endereço de tamanho fixo.

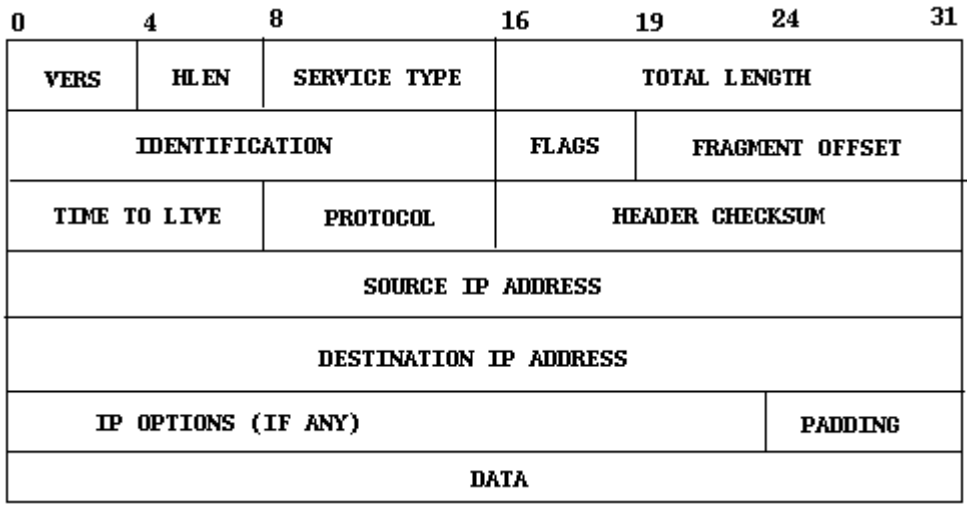
Outra função do IP, definido na RFC 791 (2001), é a fragmentação e desfragmentação de datagramas longos, se necessário, para transmissão através de redes que apresentam um desempenho melhor com pacotes menores.

O protocolo IP é especificamente limitado em escopo para fornecer as funções necessárias para entregar um pacote de bits (um datagrama internet) de uma origem para um destino em uma rede de computadores. Não há mecanismos para aumentar a confiança fim-a-fim dos dados, assegurar a seqüência dos datagramas, ou outros serviços comumente encontrados em outros protocolos *host-to-host*.

Segundo Comer (1998), o pacote pode ser perdido, reproduzido, atrasar-se ou ser entregue com problemas, mas o serviço não detectará tais condições, nem informará isso ao transmissor nem ao receptor. Também é considerado sem conexão porque cada pacote, segundo Comer (1998), é independente dos outros. Uma seqüência de pacotes enviados de um computador a outro pode trafegar por caminhos diferentes, ou alguns podem ser perdidos enquanto outros são entregues.

O formato do datagrama IP está ilustrado na Figura 2:

Figura 2: Formato do datagrama IPv4



fonte: http://penta.ufrgs.br/Esmilda/datagram.html

Os campos do datagrama são descritos como segue:

a) VERS: campo de 4 *bits* contendo a versão do protocolo IP que foi utilizada para criar o datagrama. Pode ser o IPv4 (versão 4 do protocolo IP) ou IPv6 (versão 6 do protocolo IP);
b) HLEN: contém o comprimento do cabeçalho IP em palavras de 32 *bits*. Campo de 4 *bits*;
c) SERVICE TYPE: especifica como o datagrama poderia ser manejado, sendo

subdividido em:

- precedence: campo de 3 *bits*, indica a precedência de datagramas com valores desde 0 (precedência normal) até 7 (controle de rede), com estes *bits* permite-se ao transmissor indicar a importância de cada datagrama que ele está enviando.
- bits D, T, R: indicam o tipo de transporte que o datagrama deseja: baixo retardo (D), alta capacidade de processamento (T) e alta confiabilidade (R);

d) TOTAL LENGTH: neste campo está contido o comprimento do datagrama medido em bytes, incluindo cabeçalho e dados;

e) IDENTIFICATION, FLAGS e FRAGMENTS: estes três campos controlam a fragmentação e a união dos datagramas.

- IDENTIFICATION: contém um único inteiro que identifica o datagrama, é um campo muito importante porque quando um *gateway* fragmenta um datagrama, ele copia a maioria dos campos do cabeçalho do datagrama em cada fragmento, então a identificação também deve ser copiada, com o propósito de que o destino saiba quais fragmentos pertencem a quais datagramas. Cada fragmento tem o mesmo formato que um datagrama completo;
- FRAGMENT OFFSET: especifica o início do datagrama original dos dados que estão sendo transportados no fragmento. É medido em unidades de 8 bytes;
- FLAG: controla a fragmentação;

f) TTL (*Time To Live*): especifica o tempo em que o datagrama está permitido a permanecer no sistema Internet. *Gateways* e *hosts* que processam o datagrama devem decrementar o campo TTL cada vez que um datagrama passa por eles e devem removê-lo quando seu tempo expirar (TTL=0);

g) PROTOCOL: especifica qual protocolo de alto nível foi usado para criar a mensagem que está sendo transportada na área de dados do datagrama;

h) HEADER-CHECKSUM: assegura integridade dos valores do cabeçalho;

i) SOURCE e DESTINATION IP ADDRESS: especifica o endereço IP de 32 bits do remetente e receptor;

j) OPTIONS: é um campo opcional. Este campo varia em comprimento dependendo de quais opções estão sendo usadas. Algumas opções são de um byte, e neste caso este campo é chamado de *Option Code* , e está dividido em três campos:

- COPY: (1 bit) controla a forma em que o *gateway* trata as opções durante a fragmentação:

  - l: a opção deve ser copiada em todos os fragmentos;
  - 0: a opção deve ser copiada somente no primeiro fragmento;
- CLASS (2 bits): especifica a classe geral da opção;
- OPTION NUMBER(): especifica uma opção na classe determinada no campo CLASS.

Informações mais detalhadas podem ser encontradas em Comer (1998).

2.4.3 O PROTOCOLO ICMP

O *Internet Control Message Protocol* (ICMP) é obrigatório em implementações da camada IP. Essencialmente, o ICMP é um protocolo usado na transferência de mensagens de *gateways* e estações para uma estação da rede internet. Na sua maioria, essas mensagens indicam a ocorrência de problemas no transporte de algum datagrama ou servem a operações de controle.

Quando ocorre algum problema previsto pelo ICMP, a mensagem contendo a descrição da situação é entregue à camada IP para o envio da mensagem.

Por utilizar o IP para o transporte de mensagens, o ICMP não oferece garantia de entrega.

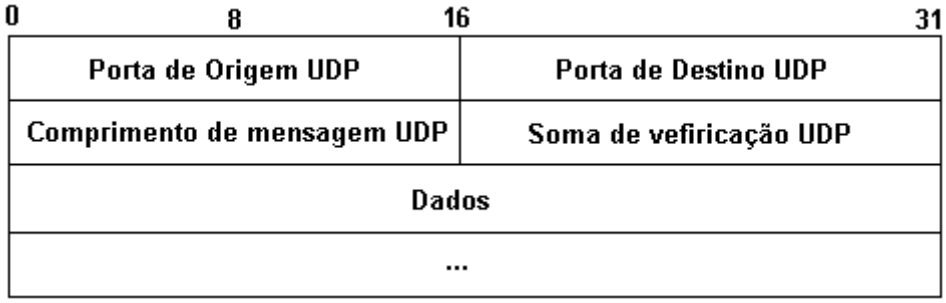
Informações mais detalhadas podem ser encontradas em Comer (1998).

2.4.4 O PROTOCOLO UDP

Na pilha de protocolos TCP/IP, o *User Datagram Protocol* (UDP) fornece o mecanismo principal utilizado pelos programas aplicativos para enviar datagramas a outros programas iguais. Segundo Redes (2001), o UDP possui como características básicas, assim como o protocolo IP, a mesma conotação não confiável de transmissão de datagramas sem conexão. Não usa confirmação para certificar-se de que as mensagens chegaram, não ordena mensagens recebidas e não fornece informação para controlar a velocidade com que as informações fluem entre as máquinas. Isso favorece que mensagens sejam perdidas, duplicadas ou que chegam com problemas.

Cada mensagem UDP é conhecida como datagrama de usuário, sendo composto por um cabeçalho UDP e uma área de dados. O formato do datagrama está representado na Figura 3.

Figura 3: Formato do datagrama UDP



O cabeçalho está subdivido em quatro campos, de 16 bits:

a) porta de origem e porta de destino UDP: contém os números de portas do protocolo UDP de 16 bits usados para demultiplexar os datagramas entre os processos que esperam para recebê-lo, tendo a porta de origem como opcional. Se não usada, deverá ser zero;
b) comprimento de mensagem UDP: possui como valor mínimo 8 bits, que é o tamanho apenas do cabeçalho. O valor total é a contagem de octetos do cabeçalho e dos dados de usuário;
c) soma de verificação UDP: campo opcional. Se não for usado significa que a soma de verificação não foi calculada.

Segundo Redes (2001), existe uma forte interação entre UDP e IP, violando a premissa básica de que a colocação em camadas reflete separação de funcionalidade. O UDP tem estado estreitamente integrado ao protocolo IP. Isso é aceitável pois é impossível identificar inteiramente um programa aplicativo de destino sem especificar a máquina de destino.

A definição de números de porta, segundo Redes (2001), é fundamental para permitir mais do que um destino em cada máquina. Num sistema multi-processo, cada processo que utiliza a rede requisita uma porta que lhe fica associada. Todos os dados que chegam à máquina com essa porta de destino são encaminhados para esse processo.

O Quadro 2 mostra alguns números de portas utilizada pelos processos no Linux.

Quadro 2: Processos e suas portas no Linux

Processo	Porta
Echo	7
ftp data	20
ftp	21
telnet	23
smtp	25
http	80
pop3	110

Informações mais detalhadas podem ser encontradas em Comer (1998).

2.4.5 O PROTOCOLO TCP

O *Transfer Control Protocol* (TCP) é um protocolo de transporte orientado à conexão. Ele garante a confiabilidade da comunicação entre pares de processos localizados em máquinas ligadas ou não a uma mesma rede.

Possui como características principais o estabelecimento e a liberação de conexões em três fases (*handshake* triplo); a manipulação da entrega confiável dos pacotes; a entrega seqüencial dos pacotes (na ordem em que foram enviados); controle do fluxo de dados, impedindo que pacotes sejam processados em uma velocidade superior à capacidade do aplicativo; a recuperação de erros no caso de perda, alteração ou duplicação de pacotes; a demultiplexação do tráfego de entrada entre as múltiplas aplicações usuárias.

O protocolo TCP especifica o formato dos dados e das confirmações que dois computadores trocam para oferecer uma transferência confiável e, também, os procedimentos de que se valem os computadores para assegurar que os dados cheguem corretamente. Especifica ainda como o software TCP confirma os múltiplos destinos em determinada máquina e como as máquinas recuperam-se de erros como pacotes duplicados ou perdidos.

O TCP trata o fluxo de dados recebidos da aplicação como uma cadeia (*stream*) de bytes ou octetos, que vão sendo agrupados dentro do *buffer* de transmissão. Periodicamente o TCP separa um determinado conjunto de dados a adiciona um cabeçalho, formando o que é chamado de segmento, sendo este a unidade básica de transferência de dados.

O TCP permite que programas aplicativos múltiplos, de determinada máquina, comuniquem-se simultaneamente. Como o UDP, o TCP utiliza números de porta de protocolo

para identificar o destino final em uma máquina. Os projetistas preferiram utilizar o mesmo número de portas para qualquer serviço que seja acessível pelo TCP e UDP, tornando-os assim independentes.

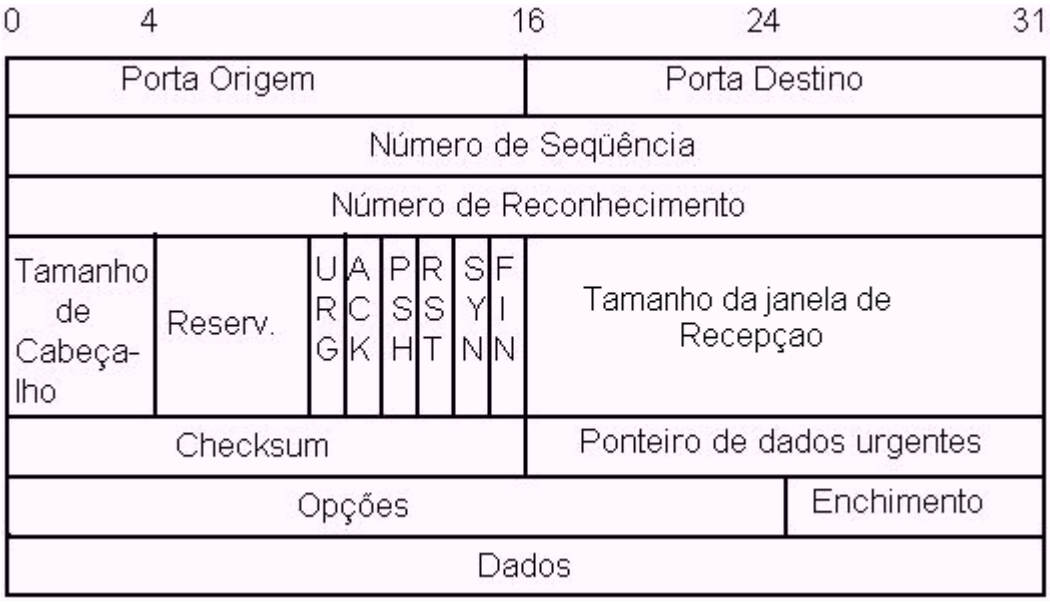
O Quadro 3 mostra alguns números de portas atribuídas para o TCP e UDP.

Quadro 3: Processos no Linux e respectivas portas para o TCP e UDP

Processo	UDP	TCP
echo	7	7
ftp data	20	20
ftp	21	21
telnet	23	23
smtp	25	25
http	80	80
pop3	110	110

Os segmentos são trocados para estabelecer conexões, transferir dados, enviar confirmações, informar tamanho de janelas e encerrar conexões. O formato do segmento TCP pode ser visualizado na Figura 4.

Figura 4: Formato do segmento TCP



Os segmentos são divididos em cabeçalho e dados. O cabeçalho transporta a identificação esperada e as informações de controle:

a) porta origem e porta destino: contém os números de portas TCP que identificam os programas aplicativos nas extremidades da conexão;
b) número de seqüência: identifica a posição na cadeia de *bytes* do transmissor dos

dados no segmento. Também refere-se ao *stream* que segue na mesma direção que o segmento;

c) número de reconhecimento: possui o número do octeto que a origem espera receber depois. Ao contrário do número de seqüência, este refere-se ao *stream* que segue em direção oposta ao segmento;
d) tamanho do cabeçalho: contém o número inteiro que especifica o comprimento do cabeçalho do segmento (múltiplo de 32 *bits*). É necessário porque o campo opções pode variar em comprimento. Ou seja, o tamanho do cabeçalho TCP varia de acordo com as opções selecionadas;
e) reservado: campo de seis *bits*, é reservado para utilização futura;
f) Alguns segmentos transportam apenas uma confirmação, enquanto alguns transportam dados. Outros transportam solicitações para estabelecer ou encerrar uma conexão. O *software* TCP utiliza o campo de seis *bits* denominado *CODE BITS* para determinar a finalidade e o conteúdo do segmento. Os seis bits indicam como interpretar outros campos do cabeçalho:
   - URG: campo de ponteiro urgente é válido;
   - ACK: campo de reconhecimento é válido;
   - PSH: esse requerimento requer *push*;
   - RST: restabelecer a conexão;
   - SYN: sincronizar os números de seqüência;
   - FIN: o emissor atingiu o final do fluxo de *bytes*;
g) tamanho da janela de recepção: indica o número de octetos, incluindo o octeto do campo de reconhecimento que o receptor pode aceitar;
h) *checksum*: permite a verificação da integridade do segmento recebido;
i) enchimento: preenchido com *bits* 0, usado para assegurar que o cabeçalho tenha tamanho múltiplo de 32 *bits*.

Os protocolos TCP de ambas as pontas precisam concordar sobre o segmento máximo que irão transmitir, pois nem todas as redes têm o mesmo tamanho de segmento. Essa negociação é feita utilizando o campo opções do segmento, especificado pelo MSS (*maximum segment size*). O tamanho ideal é o que permite que os datagramas IP que transportam os segmentos sejam os maiores possíveis, sem exigir a fragmentação em qualquer ponto ao longo do caminho.

Para que seja possível que as diversas aplicações que utilizam o protocolo TCP comuniquem-se entre si, é necessário que ocorra um estabelecimento de conexão. Neste momento às máquinas passam a se conhecer trocando diversas informações de controle e realizando uma verificação de autenticidade entre elas.

2.4.5.1 SERVIÇOS

Para garantir a transferência confiável de dados, seu principal propósito, o TCP deve oferecer alguns serviços, tais como:

a) transferência de dados: pode ser feita tanto no modo *full-duplex* quanto *half-duplex*. Em cada conexão, é o protocolo que determina o momento mais conveniente de bloquear ou liberar o dado para transmissão. O processo do ajuste de tempo para a liberação é feito pela diferença do tempo em que foi enviado o segmento e a chegada da confirmação de recebimento desse segmento;
b) confiabilidade: o TCP garante a correção dos dados nos casos de alteração, perda, duplicação ou entrega fora de seqüência. Esse controle é feito basicamente com o número de seqüência. Pode ocorrer a perda de dados, segmentos chegarem fora de ordem, duplicação de segmentos, erros de conteúdo;
c) controle de fluxo: mecanismo chamado de janela deslizante de tamanho variável ou janela de crédito, que permite à estação receptora controlar a quantidade de dados enviados pela estação transmissora;
d) multiplexação: cada máquina possui um conjunto de portas e isto permite que vários processos dentro de uma mesma máquina façam uso simultâneo dos serviços de comunicação do protocolo;
e) conexão: o TCP, por ser orientado à conexão, caracteriza três fases de funcionamento: estabelecimento da conexão, onde são iniciadas as opções de *status* e a sincronização dos números de seqüência; transferência de dados, ocorrendo a verificação da recepção correta dos dados; e liberação da conexão, que deve ser feita para liberar recursos para outros usuários;
f) relatório de erros: erros não recuperáveis relacionados a pedidos de serviço ou a problemas ocasionados pelo funcionamento deficiente do ambiente interno da rede são relatados ao processo usuário pelo TCP.

O TCP é um protocolo complexo que promove a comunicação através de uma grande variedade de tecnologias de rede. Segundo Redes (2001), a generalidade do TCP não parece interferir no seu desempenho.

Informações mais detalhadas podem ser encontradas em Comer (1998).

3 SEGURANÇA EM REDES DE COMPUTADORES

A todo momento é feita a descoberta de uma nova vulnerabilidade ou falha de segurança no que diz respeito a sistemas de informática e, em especial, redes de computadores. De um lado da muralha estão os *hackers*, o tempo todo em busca de brechas e rachaduras nessa muralha. Do outro lado estão os administradores de sistemas e redes, preocupados em tapar e consertar as brechas desta muralha, numa luta incansável para proteger os dados e sua integridade.

Técnicas de invasão surgem a todo momento, as quais exploram falhas ou vulnerabilidades nos softwares, e, talvez numa velocidade inferior, surgem as correções para tais falhas e as defesas para se fazer a proteção dos sistemas.

Inúmeras são as formas utilizadas para invadir sistemas. Formas que vão desde a engenharia social (descoberta de senhas através de conversas com funcionários, procura em latas de lixo, entre outras) até a exploração de códigos mal escritos ou falhas de hardware. Faz-se, dessa forma, imprescindível ao administrador de sistemas e de redes, atualizar constantemente os softwares e ficar “por dentro” do mundo *underground*.

Vários são os motivos que levam *hackers* e *crackers* a invadir sistemas. Existe uma diferença entre *hackers* e *crackers*: os primeiros invadem sistemas apenas para o aprimoramento dos seus conhecimentos ou para testar a segurança dos softwares e buscar vulnerabilidades. Não há o roubo de informações ou qualquer outro dano. Já *crackers* invadem com o objetivo de danificar e/ou roubar informações, projetos ou verbas.

Segundo Bernstein (1997), *crackers* revelaram os seguintes motivos para seus atos:

a) ganhos financeiros: geralmente os intrusos são funcionários que obtém acesso a sistemas financeiros para roubar dinheiro através de desvios ou para reunir informações que poderão ser vendidas;
b) vingança: funcionários descontentes com sua situação na empresa ou que foram demitidos podem ser os responsáveis por danos físicos ou dos dados;
c) necessidade de aceitação ou respeito: raramente ataques com esse motivo acarretam perdas. Geralmente são efetuados como teste para entrar em grupos de *hackers* ou para promover seu conhecimento aos olhos alheios;
d) idealismo: alguns intrusos acham-se como heróis protegendo o mundo do governo

ou sentem-se verdadeiros “*Robin Wood”* virtuais;

e) curiosidade ou busca de emoção: alguns intrusos simplesmente querem saber “o que tem dentro”, ou são viciados na adrenalina que o ato ilegal causa;
f) anarquia: têm como objetivo promover a discórdia ou a confusão e são motivados pelo ato ilegal de tentar invadir sistemas;
g) aprendizado: invadem sistemas para aprimorar suas técnicas e conhecimentos sobre os diversos sistemas e formas de proteção;
h) ignorância: alguns intrusos não têm consciência da ilegalidade dos seus atos, nem que podem ser punidos por suas ações;
i) espionagem industrial: ocorre quando uma empresa ou organização tem como objetivo atacar outras empresas ou organizações com o objetivo de roubar informações confidenciais ou até mesmo produtos;
j) espionagem internacional: semelhante à espionagem industrial, com a diferença de que os ataques são contra outros países. As informações adquiridas podem ser de natureza estratégica, onde guerras e conflitos podem ser descobertos.

3.1 VULNERABILIDADES

Segundo Silva (2001), uma vulnerabilidade ou brecha é qualquer problema ou configuração encontrados em um hardware ou software que facilite ou ofereça a possibilidade de um ataque a uma rede remota, a obtenção de acesso não permitido ou qualquer tipo ilícito de operação na rede. As vulnerabilidades ou brechas podem estar presentes em roteadores, softwares cliente-servidor, sistemas operacionais e *firewalls*.

A cada dia aparecem novas vulnerabilidades, descobertas por *hackers*, pelos desenvolvedores dos produtos e pelos administradores que usufruem de ferramentas específicas para este propósito. Essas falhas podem colocar em risco a segurança das informações da empresa, bem como abrir portas para futuras invasões ou até prejudicar o funcionamento do sistema.

3.1.1 FALTA DE PADRÕES TÉCNICOS

À medida que a internet passou a ser mais comercial, a *Internet Engineering Task Force* (IETF) tornou-se mais indefinida. Esse órgão é responsável por definir diretrizes para a

internet. Cada vez mais fornecedores estão propondo suas versões proprietárias de produtos e tecnologias, sem que haja uma homologação oficial da IETF.

O objetivo dessas empresas é colocar no mercado produtos com tecnologias proprietárias, o que garante um grande retorno financeiro. Pode-se citar como exemplos dessas tecnologias os protocolos de comunicação segura como o PCT (*Private Communications Technology*), STT (*Secure Transaction Tecnology*), e SEPP (*Secure Eletronic Payment Protocol*).

3.1.2 VULNERABILIDADES DOS PRODUTOS

Muitos produtos já vêm com falhas ainda não documentadas, sejam softwares ou hardwares, o que pode vir a facilitar as tentativas de invasão. As falhas podem ser nos projetos dos produtos, onde pode não ter havido uma preocupação com a segurança.

Vários softwares vêm com vulnerabilidades em seus códigos, como o TCP/IP que permite a falsificação do endereço IP do receptor durante trocas de pacotes. Essa técnica é conhecida como *Spoofing de IP* e será abordada mais adiante.

3.1.3 VULNERABILIDADES NAS CONFIGURAÇÕES

Muitos administradores optam pelas instalações padrão dos softwares, sem se darem conta das possíveis falhas que essa instalação pode ter, como por exemplo, habilitar portas ou dar privilégios a usuários que não os deveriam possuir.

Outro problema é a complexidade da instalação e da configuração, o que pode confundir ou até mesmo passar despercebido pelo administrador, com detalhes que seriam de vital importância para a segurança do sistema. Deve-se ficar atento à documentação que geralmente acompanha a distribuição dos produtos.

Uma das vulnerabilidades consideradas mais sérias são as causadas pelos usuários. Senhas simples, de fácil adivinhação, nomes de parentes ou até senhas repetidas, são algumas das vulnerabilidades. Contas inativas e não eliminadas do sistema podem ser pratos cheios para *hackers*.

3.1.4 POLÍTICAS DE OPERAÇÃO

Políticas de operação dizem respeito às formas adotadas pela empresa em relação às falhas e/ou vulnerabilidades que possam vir a ocorrer em seus sistemas, tanto em se tratando dos softwares quanto das formas como os usuários utilizam os sistemas.

Deve haver políticas para as regras de formação de senhas pelos usuários evitando senhas fáceis e óbvias ou senhas muito curtas, bem como a restrição do acesso físico por parte dos usuários a áreas restritas.

Outro ponto a ser observado é a atualização dos softwares: a periodicidade e a procedências dos mesmos deve ser levada em consideração. As pessoas que farão as atualizações devem ter um conhecimento do que estão fazendo e ter uma responsabilidade assumida em dar a manutenção nos sistemas.

As atitudes que serão adotadas pela empresa ao terem seus sistemas invadidos deve estar bem definidas: órgãos que deverão ser consultados, a quem deverá ser feita a denúncia, como fazer o levantamento da localização das vulnerabilidades.

3.1.5 TÉCNICAS DE INVASÃO

Segundo Segurança (2000), um ataque é qualquer ação não autorizada empreendida com a intenção de impedir, danificar, incapacitar ou quebrar a segurança de um servidor conectado à internet. O ataque pode ser uma simples recusa de serviços ou até a captura ou destruição de todas as informações contidas no servidor.

3.1.5.1 ATAQUES DE FORÇA BRUTA

Este tipo de ataque consiste em tentar adivinhar as senhas por tentativa e erro. Segundo McClure (2000), um ataque de força bruta nada mais é do que adivinhar uma combinação de usuários/senhas em um serviço que tenta autenticar o usuário antes do acesso ser concedido. Pode-se citar como exemplos TELNET, POP, FTP, SSH, entre outros.

A melhor defesa para os ataques de força bruta é usar senhas menos óbvias, com um número mínimo de caracteres que exijam a combinação de letras, números e sinais gráficos, forçar a expiração das senhas, obrigando o usuário a fazer a alteração.

Além dessas medidas, é conveniente utilizar ferramentas de composição de senha que impeçam que um usuário escolha uma senha ruim, assegurando de que contas padrão não tenham senhas padrão.

3.1.5.2 *BUFFER OVERFLOW*

Segundo McClure (2000), uma condição de estouro de *buffer* ocorre quando um usuário ou processo tenta colocar em um *buffer* (ou *array* fixo) mais dados do que o espaço originalmente alocado permite.

Esse tipo de problema pode ser explorado para ganhar acesso privilegiado ao sistema. Todos os dias aparecem correções e boletins de alerta dos distribuidores de software indicando que seja feita a atualização dos softwares o mais rápido possível.

Ao promover um estouro de pilha, o atacante pode fazer com que o sistema execute algum comando específico ou até execute algum programa implantado no sistema pelo intruso. Ao causar o estouro de pilha, é possível tirar do ar algum serviço que é executado com permissões de superusuário (*root*), podendo fazer-se valer das permissões do serviço derrubado.

A melhor medida de segurança é adotar práticas de programação seguras, como, por exemplo, ter em mente, desde o começo do projeto, a segurança como um dos objetivos principais e utilizar compiladores seguros, que se preocupam em imunizar os programas em tempo de compilação.

3.1.5.3 CAVALOS DE TRÓIA

O uso de cavalos de Tróia consiste em falsificar programas utilizados normalmente como *su, telnet, ftp, passwd, ifconfig* e assim por diante. O intruso, após conseguir acesso de superusuário, troca um dos programas utilizados por uma versão modificada que pode desde roubar senhas até apropriar-se indevidamente de dados.

Outro modo de utilização dos cavalos de Tróia é fazer com que os *logs* sejam modificados, apagando assim, seus rastros deixados no sistema.

O número de técnicas de cavalos de Tróia tem seu limite na imaginação do atacante. A melhor forma de proteger-se contra esse tipo de ataque é através da utilização de ferramentas

apropriadas, como as que fornecem assinaturas exclusivas aos arquivos. As assinaturas deverão ser armazenadas em locais seguros, fora do servidor.

3.1.5.4 *SNIFFERS*

Segundo McClure (2000), *sniffers* são conseqüência da necessidade de uma ferramenta para depurar problemas de rede. Eles capturam, interpretam e armazenam todos os pacotes trafegando em uma rede para uma análise posterior. De posse desses dados, o administrador da rede tem a possibilidade de diagnosticar possíveis problemas em sua rede, analisar os tipos de pacotes que estão trafegando e identificar gargalos.

Para que seja possível capturar todos os pacotes trafegando na rede, é necessário colocar a placa de rede em modo promíscuo, ou seja, ela passará a receber todos os pacotes da rede. Uma vez estando em modo promíscuo, o *sniffer* poderá capturar e analisar qualquer tráfego que passe no segmento em que se encontra instalado. Em redes *Ethernet*, a análise fica limitada ao segmento em que está conectado, pois não conseguirá monitorar o tráfego fora do domínio de colisão.

Algumas medidas para a proteção contra *sniffers* consistem em fazer a segmentação da rede, criando, assim, vários domínios de colisão; e a utilização de softwares que fazem a criptografia dos dados antes de enviá-los, como o *Secure Shell* (SSH) e Ip *Security Protocol* (IPSec).

3.1.5.5 DOS (*DENIAL OF SERVICE*)

Segundo McClure (2000), um ataque DoS interrompe ou nega completamente serviço a usuários legítimos, redes, sistemas ou outros recursos. O objetivo de tais ataques é normalmente de natureza mal-intensionada e muitas vezes exige pouca habilidade, pois as ferramentas que os possibilitam são facilmente encontradas.

É muito mais fácil interromper a operação de um sistema do que efetivamente invadí-lo. Existem alguns programas *hackers* que, para serem executados, precisam que o sistema alvo seja reinicializado. Isso se aplica mais a sistemas como o *Microsoft Windows* NT, que necessita ser reinicializado para que alterações tenham efeito.

O consumo da largura de banda é outro tipo de DoS, que consiste em consumir toda a largura de banda de uma rede específica. Esse tipo de ataque pode ser executado se o invasor

possui uma largura de banda maior que a da vítima ou pode utilizar o sistema da vítima como um “amplificador”, ou seja, usando mais de um sistema para inundar outro sistema.

Uma das medidas de segurança é desabilitar o *broadcast* direcionado do roteador de fronteira ou utilizar regras de filtragem chamadas *ipchains* ou *iptables* em sistemas Linux.

3.1.5.6 *SPOOFING* DE IP

Segundo Segurança (2000), *spoofing* de IP é uma técnica sofisticada de autenticar uma máquina para outra forjando pacotes de um endereço de origem confiável. Ou seja, é a troca de um IP pelo outro, onde a máquina do invasor faz-se passar por um outro *host*.

Para ocorrer um ataque de *spoofing* de IP é necessário que haja uma relação de confiança entre dois *hosts*, ou seja, quando não é necessária a autenticação de senhas entre ambos e a autenticação é feita através do endereço IP de origem dos datagramas. Essa autenticação é feita apenas no momento do estabelecimento da conexão.

Essa técnica é aplicada explorando as vulnerabilidades existentes nos protocolos TCP, TELNET, UDP, DNS e outros da arquitetura TCP/IP.

O ataque ocorre em dois ambientes diferentes. O primeiro ambiente é constituído por um computador pertencente a uma rede externa, ou seja, um computador que se localize em uma rede fisicamente separada da máquina vítima. O segundo ambiente compreende a rede interna na qual a máquina que será alvo dos ataques se encontra, essa rede pode ser caracterizada como uma rede interna ou local somente em relação às outras máquinas que estejam ligadas no mesmo barramento que ela.

Neste tipo de ataque, o intruso transmite pacotes a partir da rede externa que fingem ser originários de uma máquina interna - os pacotes falsificados contém o endereço IP de origem que especifica uma máquina interna da rede. O intruso, então, espera que uma verificação simples de endereço seja feita, de modo a permitir que pacotes de máquinas internas sejam aceitos ao mesmo tempo em que pacotes de outras máquinas sejam descartados.

Criam-se dois problemas. Em primeiro lugar, um *host* “B” irá receber a resposta de um pedido de conexão que não fez, se isto ocorrer “B” irá negar o pedido de conexão e o ataque irá falhar. E em segundo lugar, o *host* “C” não pode dar continuidade ao processo de conexão,

pois não consegue responder à confirmação enviada pelo *host* “A” já que não sabe o número de seqüência que foi enviado.

O primeiro problema é resolvido consumindo os recursos de “B”, de forma que não possa mais responder ou receber pacotes de “A”. O segundo problema é mais complexo, pois “C” terá que adivinhar o número de seqüência estabelecido por “A”. Para resolver esse problema, o *hacker* faz várias conexões legítimas com “A” para determinar o padrão do número de seqüência. Conhecendo o padrão, “C” tem uma conexão de confiança com “A”.

3.2 FORMAS DE DEFESA

Segundo Oliveira (2000), não existem sistemas 100% seguros. Alguma falha ou vulnerabilidade sempre estará presente em um sistema. O único sistema totalmente seguro é aquele que está trancado em uma sala, sem conexão física alguma e a única pessoa que tem a chave da sala morreu semana passada.

As formas de defesa mais eficientes dizem respeito à implementação e adoção de políticas de segurança, utilização de *firewalls*, criptografia ou outro esquema que sirva para proteger ao máximo o sistema. Mas as dificuldades não param por aí. Além de implementar as técnicas de segurança, faz-se necessário identificar onde está a vulnerabilidade para que seja possível corrigí-la, fazer o levantamento dos possíveis prejuízos e identificar os invasores.

Ao se implementar uma política de segurança, segundo Silva (2001), deve-se visar principalmente três aspectos:

a) confidencialidade: os dados ou serviços devem ser protegidos contra acessos de pessoas não autorizadas;
b) integridade: modificações não autorizadas, perdas, inconsistência dos dados devem ser previstas e evitadas;
c) disponibilidade: assegurar que estão disponíveis a qualquer momento as informações ou serviços a serem utilizados por pessoas devidamente autorizada.

3.2.1 *FIREWALLS*

Segundo Oliveira (2000), pode ser chamado de *firewall* qualquer dispositivo que impeça que estranhos tenham acesso a rede. Eles comumente auxiliam no planejamento e nas regras da rede. Um *firewall* é um dispositivo de hardware (roteadores, servidores) ou

software. A localização do *firewall* depende do projeto da rede que se quer proteger, ficando entre a rede interna e a internet.

O papel fundamental de um *firewall* é monitorar todo tráfego que flui entre duas redes e bloquear certos tipos de tráfegos completamente: por exemplo, pode-se configurar o *firewall* para bloquear protocolos como o UDP, limitar o acesso às portas do servidor, limitar a saída e a entrada dos dados.

Algumas regras são possíveis de serem implementadas em roteadores, onde se criam listas de acesso com permissões para protocolos ou portas que se deseja impedir que sejam acessados ou utilizados.

Um *firewall* é o ponto de entrada de uma rede e por isso, quando tem suas regras bem implementadas e mantidas, torna-se uma barreira extremamente funcional contra ataques. Mas os invasores sabem disso e a alternativa é tentar entrar por algum outro caminho que não passe pelo *firewall*, como um acesso discado.

3.2.2 CRIPTOGRAFIA

Criptografia, segundo Oliveira (2000), é a arte ou ciência de escrever em cifra ou em códigos, ou seja, um conjunto de técnicas que tornam uma mensagem incompreensível permitindo apenas que o destinatário, que conhece a chave de encriptação, consiga desencriptar e ler a mensagem com clareza. É uma das melhores formas de segurança de dados e uma possibilidade que não deve ser esquecida, principalmente se trabalharmos com dados confidenciais e envio de mensagens via internet/intranet.

A criptografia dos dados é feita através de algoritmos que codificam e decodificam os dados, enviando-os de forma ilegível para que, se forem detectados por algum *sniffer* ou algum outro dispositivo que intercepte os pacotes, seja praticamente impossível a sua decodificação. Os algoritmos devem ser conhecidos e testados, e a segurança reside totalmente na chave secreta, a qual deve ter tamanho suficiente para evitar sua descoberta por teste exaustivo. Estes algoritmos podem ser classificados em:

a) algoritmos simétricos: esses algoritmos usam a mesma chave para encriptar e decriptar. Pode-se citar o DES (*Data Encryption Standard*);
b) algoritmos assimétricos: são utilizadas chaves diferentes para a encriptação e decriptação dos dados, existindo uma relação entre as chaves. Como exemplo de

algoritmo assimétrico temos o RSA (*Rivest, Shamir and Adleman*).

Dependendo do tipo de encriptação que está sendo utilizado, pode-se ter uma queda na performance do sistema, pois quanto mais complexo for o algoritmo de criptografia utilizado, mais processamento irá exigir da máquina.

3.2.3 OUTRAS MEDIDAS DE SEGURANÇA

Por maior que seja o aparato de equipamentos e de softwares de segurança de uma empresa, algumas medidas consideradas simples podem ser adotadas para dificultar o acesso não autorizado às informações e sistemas. Algumas dessas medidas são:

a) fazer com que os usuários passem a utilizar e criar senhas mais seguras, não sendo tão óbvias e promover a troca periódica das senhas;
b) deixar claro o perigo contido em arquivos anexados em e-mails de procedência desconhecida e a instalação de qualquer tipo de *software;*
c) manter uma atualização constante de *drivers* e aplicativos, buscando versões com correções contra vulnerabilidades nas versões antigas;
d) eliminar serviços que não sejam realmente necessários para a empresa;
e) fazer um levantamento periódico de contas de usuários inativos e eliminá-las;
f) estar sempre atualizado em questões de segurança inscrevendo-se em listas de discussão sobre o assunto e fazer visitas periódicas a endereços na internet que tratam do assunto.

4 LINUX

Nos últimos anos, um nome de sistema operacional vem sendo falado com mais freqüência pelos usuários de informática: Linux. Mas o que esse sistema operacional tem de tão especial? Funciona em qualquer computador? O que ele faz de diferente? Estas são apenas algumas das perguntas em relação ao Linux.

O Linux, ao contrário de outros sistemas operacionais, não pertence a uma empresa ou uma pessoa: ele é o fruto do trabalho voluntário e gratuito de milhares de desenvolvedores do mundo inteiro.

4.1 UM POUCO DE HISTORIA

Segundo Sonnino (2001), as origens do Linux vêm de 1969, com a criação do *Unix* nos laboratórios *Bell*, da *AT*&T, por *Ken Thompson*, para um computador DEC-PDP-7. Inicialmente, ele foi escrito em *Assembly*, tendo sido posteriormente reescrito em C.

Até então o sistema era distribuído quase livremente, sendo gratuito para universidades e instituições de pesquisa e pago para uso comercial. Outras distribuições, com o passar do tempo, foram aparecendo no mercado, sendo oferecidas por empresas como *Sun*, *IBM*, *Silicon Graphics* entre outras.

Em 1991, começaram a aparecer mensagens nos grupos de notícias da Internet, vindas de um estudante da universidade de Helsinque, chamado Linus Torvalds: ele queria desenvolver um *Kernel*, o núcleo básico de um sistema operacional. O *Kernel* que ele queria desenvolver era baseado no *Minix*, um sistema operacional voltado ao ensino e utilizado em processadores 80386 da Intel.

Em 1992 foi lançada a primeira versão do *Kernel* do Linux. A partir desse ponto, o Linux vem evoluindo rapidamente sempre com a colaboração de desenvolvedores de diversas regiões do mundo.

4.2 ALGUMAS CARACTERÍSTICAS DO LINUX

O Linux é um sistema operacional multitarefa, multiusuário e portável. Como multitarefa, ele é capaz de executar várias tarefas simultaneamente, da mesma maneira que o Windows95 / 98 ou o NT / 2000. Diferentemente do Windows95 / 98 (mas não do NT /

2000), ele é capaz de utilizar múltiplos processadores simultaneamente. Outra capacidade do Linux é a utilização do processamento distribuído, onde uma tarefa é dividia entre vários computadores interligados.

Por ser um sistema operacional multiusuário, o Linux permite que várias máquinas / usuários estejam conectados nele ao mesmo tempo, tendo aplicações executando em um servidor Linux e apenas os resultados ou telas sendo mostrados nas máquinas clientes. Essa característica viabiliza a utilização de máquinas consideradas obsoletas, como os antigos 386 e 486, reduzindo custos de atualização de computadores.

Mas uma das características mais marcantes do Linux é ser um sistema operacional de código aberto, onde milhares de pessoas no mundo todo contribuem com o desenvolvimento e aprimoramento do sistema. Com o desenvolvimento aberto, é possível corrigir *bugs* encontrados no sistema ou modificá-lo para suprir suas necessidades específicas.

O Linux é hoje um sistema estável e maduro, onde cada vez mais as empresas vêem nele uma alternativa viável, apresentando confiabilidade e segurança invejáveis.

4.3 *DEVICE DRIVERS*

Muito do Linux é independente do hardware onde ele está rodando, permitindo que os usuários não precisem tomar conhecimento do mesmo. Mas para cada componente de hardware estar funcionando corretamente no Linux, alguém precisou escrever um *driver* para permitir seu funcionamento. Sem os *device drivers* (*drivers* de dispositivo) não há um funcionamento adequado do sistema.

Segundo Rubini (2001), os *device drivers* têm um papel importante no *Kernel* do Linux: eles são “caixas brancas” que fazem com que um item de hardware em particular (placa de rede, vídeo, modem, entre outros) responda corretamente a um software. Eles escondem completamente os detalhes de como o dispositivo trabalha.

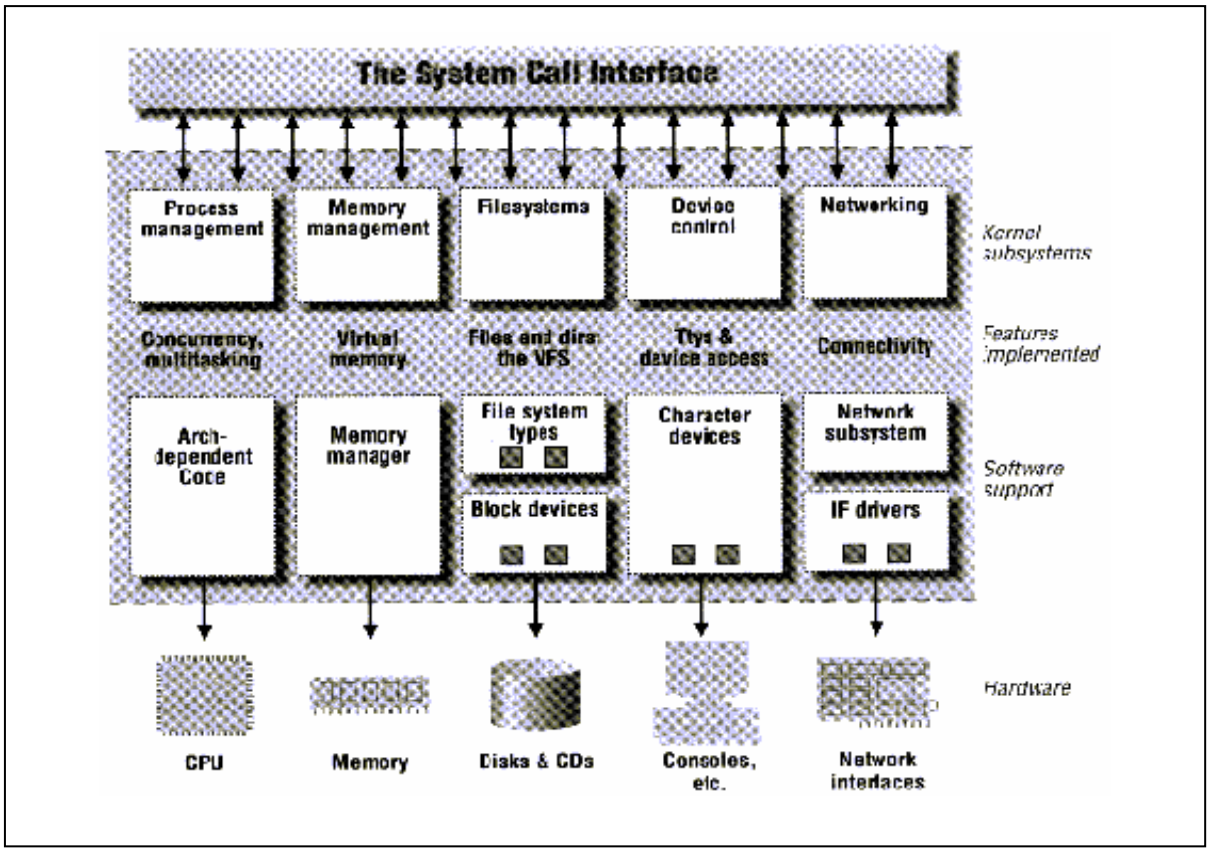
As atividades dos usuários são apresentadas por meios de um conjunto de chamadas padronizadas, sendo independentes de um *driver* específico; traçar aquelas chamadas às operações específicas do dispositivo que agem no hardware real é então papel do *device driver*. Essa programação de interface é tal que *drivers* podem ser construídos separados do

resto do kernel. Essa modularidade faz com que os drivers para o Linux possam ser fáceis de ser implementados e mantidos.

4.3.1 DIVIDINDO O *KERNEL*

Em um sistema Linux, vários processos concorrentes atendem a diferentes requisições. Cada processo chama por um recurso do sistema, como memória, vídeo, conectividade de rede ou outro recurso. O *kernel* é o maior código em execução que atende a todas as requisições. O papel do *kernel* pode ser dividido como mostrado na Figura 5:

Figura 5: Camadas do *Kernel* do Linux



Fonte: Rubini (2001)

A função dos principais subsistemas do *kernel* (*kernel subsistems*) são:

a) *process management*: o *kernel* tem a função de criação e destruição de processos e o tratamento de suas conexões (entrada/saída). A comunicação entre diferentes processos (através de *pipes*, sinais ou primitivas de comunicação interprocessos) é o básico para o funcionamento geral do sistema;

b) *memory management*: a memória do computador é o maior recurso, e as políticas usadas para compartilhá-la são pontos críticos para o bom desempenho do sistema.

Diferentes partes do *kernel* interagem com o subsistema de gerenciamento de memória através de um conjunto de chamadas de funções;

c) *filesystems:* o Linux é pesadamente baseado no conceito de *filesystems.* Quase tudo no Linux pode ser tratado como um arquivo. Além disso, há o suporte a vários tipos de *filesystems*, ou seja, diferentes meios de organizar dados no meio físico, por exemplo, disquetes podem ser formatados tanto como ext2 (padrão do Linux) como FAT (padrão do Windows);

d) *device control:* quase toda operação do sistema eventualmente é mapeada para um dispositivo físico. Com exceção do processador, memória e muitas outras entidades, toda e qualquer operação de controle de dispositivos são executados por um código que é específico para o dispositivo ser endereçado. Esse código é chamado de *device driver*;

e) *networking:* a *networking* precisa ser gerenciada pelo sistema operacional porque muitas operações de rede não são específicas para um processo específico: a entrada de pacotes são eventos assíncronos. Os pacotes precisam ser coletados, identificados e despachados antes que um processo ache que o pacote foi perdido. O sistema tem a função de entregar pacotes de dados através dos programas e das interfaces de rede, e precisa controlar a execução dos programas de acordo com suas atividades de rede. Além disso, todo o roteamento e a resolução de endereços é implementado dentro do *kernel*;

Uma ótima característica do Linux é a habilidade de estender, em tempo de execução, o conjunto de características oferecidas pelo *kernel*, ou seja, é possível adicionar funcionalidades ao *kernel* enquanto o sistema está rodando. A essas funcionalidades chamamos de módulos.

4.3.2 CLASSES DE *DEVICES* E MÓDULOS

O Linux possui três tipos de módulos: módulos de caracter (console, portas seriais, impressoras, etc), módulos de bloco (disquetes, unidades de disco, etc) e módulos de rede (interfaces de rede). Será abordado mais profundamente neste tópico o módulo de rede, tendo em vista o objetivo do trabalho.

4.3.2.1 MÓDULO DE REDE

Qualquer transação de rede é feita através de uma interface, ou seja, um dispositivo que está habilitado a trocar dados com outros *hosts*. Usualmente, uma interface é um dispositivo de hardware, mas ele precisa também ser um dispositivo de software, como a interface de *loopback*. Uma interface de rede tem a função de enviar e receber pacotes de dados, comandado pelo subsistema de rede do *kernel*, sem o conhecimento de qual transação atual está sendo mapeada para ser transmitida. Por exemplo, apesar das conexões via *Telnet e FTP* serem conexões orientadas por *streaming* (seqüência de caracteres), os pacotes são transmitidos usando-se a mesma interface. A interface não vê individualmente cada *stream*, mas apenas os pacotes de dados.

5 DESENVOLVIMENTO DO PROTÓTIPO

O objetivo do protótipo é ser uma ferramenta para a monitoração de datagramas IP. O monitor estará instalado em um *host* conectado a uma rede *Ethernet* utilizando o protocolo TCP/IP. O monitor apresentará filtros para gerar alertas adequados para o administrador da rede, os quais serão armazenados em um *log*.

Neste capítulo serão abordados os detalhes relevantes sobre o desenvolvimento do protótipo.

5.1 REQUISITOS PRINCIPAIS DO PROTÓTIPO

Foram levantados alguns requisitos que devem estar presentes no protótipo. Estes requisitos demonstram algumas características que o protótipo precisa ter para que se alcance o resulta final desejado.

Os requisitos principais do protótipo são:

a) operar no sistema operacional Linux;
b) monitorar uma rede *Ethernet*, significando que os quadros capturados estarão no formato *Ethernet;*
c) monitorar os protocolos da arquitetura TCP/IP;
d) capturar e analisar todos os pacotes que passarem pela camada de interface de rede do *host* em que o protótipo está instalado;
e) configurar situações de alerta através de filtros, cujas opções deverão ser:
   - protocolos: ICMP, TCP, UDP;
   - com o ICMP será possível escolher: IP Origem e IP Destino;
   - com o TCP e o UDP será possível escolher: IP Origem e IP Destino; Porta Origem e Porta Destino;
f) mostrar todos os pacotes capturados ao administrador;
g) mostrar e armazenar nos *logs os* pacotes considerados suspeitos em local separado;
h) permitir a alteração dos filtros poderão ser a qualquer momento durante a execução do monitor.

5.2 ESPECIFICAÇÃO DO PROTÓTIPO

A especificação do protótipo foi realizada visando a divisão do mesmo em dois módulos com funções bem distintas:

a) módulo *sniffer*: responsável por interagir com o sistema operacional, de forma a receber os pacotes de rede e encaminha-los ao módulo aplicação;
b) módulo aplicação: responsável pelo tratamento e controle dos pacotes recebidos e pela interface com o usuário.

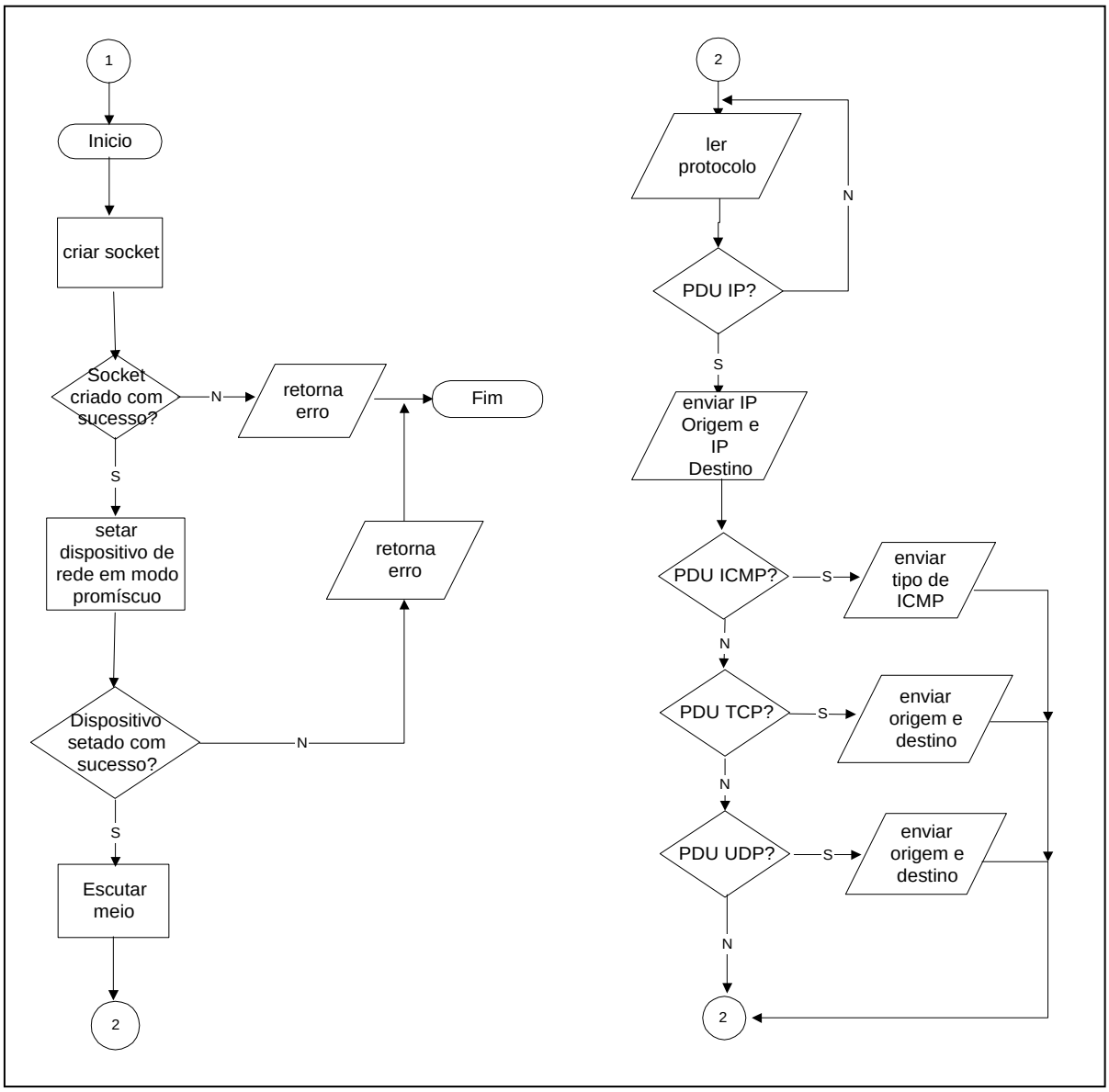
A metodologia de especificação utilizada para representar os processos que formam o protótipo foi a fluxogramação. Para auxiliar na construção dos fluxogramas utilizados na especificação do protótipo foi utilizada a ferramenta Microsoft Visio 4.0.

A seguir será apresentada a especificação primeiramente do módulo *sniffer* e posteriormente do módulo aplicação.

5.2.1 ESPECIFICAÇÃO DO MÓDULO *SNIFFER*

O módulo *sniffer* intercepta os pacotes na rede, analisá-os e envia-os para o módulo aplicação, de onde serão filtrados. Na Figura 6 está apresentada a especificação macro do módulo *sniffer*.

Figura 6: Especificação do módulo *Sniffer*



Os procedimentos de criação do *socket* e de setar o dispositivo de rede em modo promíscuo (*promiscuous mode*), dependem do sistema operacional em que serão desenvolvidos.

5.2.2 ESPECIFICAÇÃO DO MÓDULO APLICAÇÃO

O módulo aplicação é o responsável pelo tratamento dos pacotes interceptados pelo módulo *sniffer*. No módulo aplicação são configuradas as situações de alerta definidas pelo administrador. As situações poderão ser:

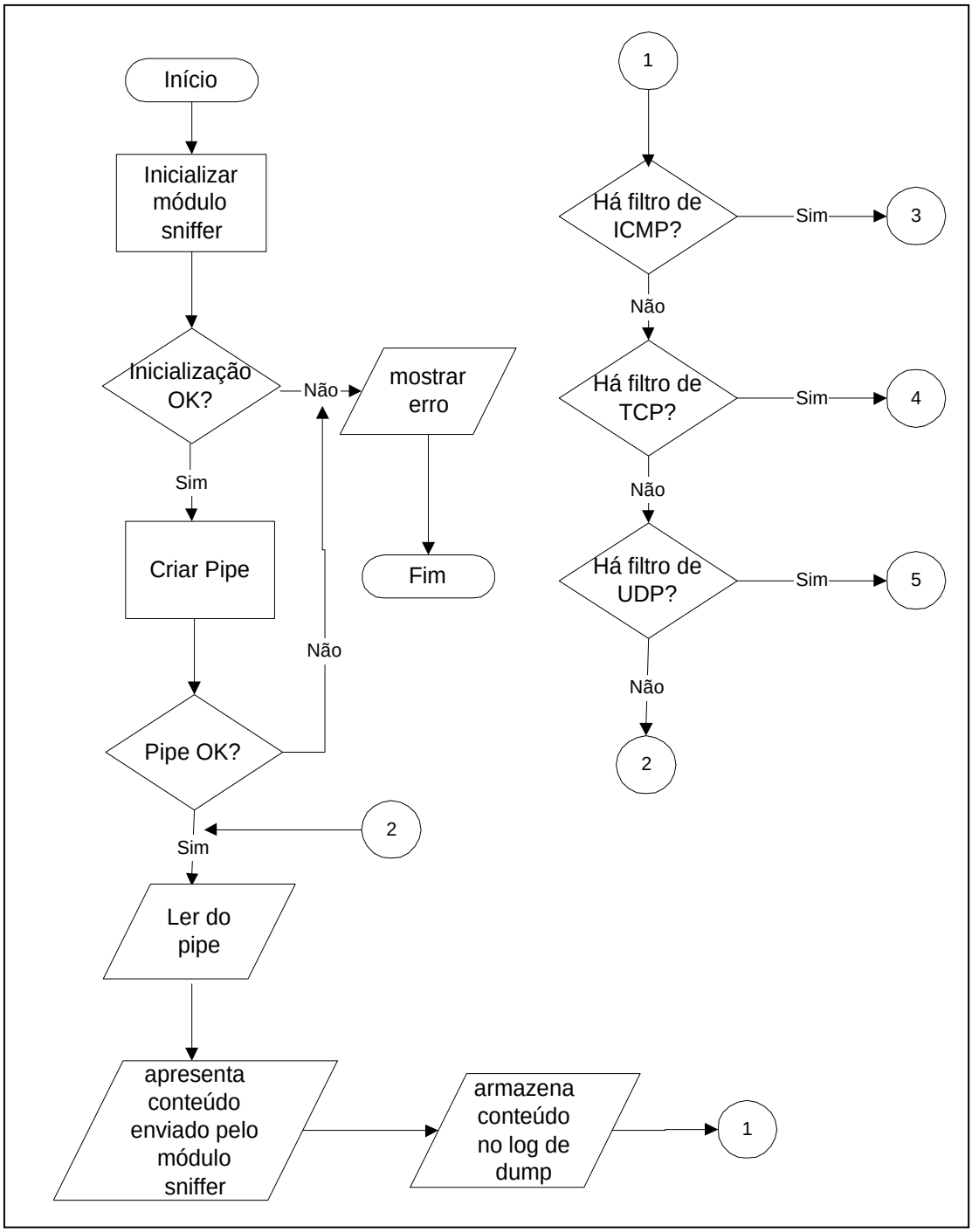
a) protocolo: ICMP, TCP, UDP;

b) o protocolo ICMP permite filtrar por: porta de origem e/ou porta de destino;

c) o protocolo TCP e UDP permitem filtrar por: porta de origem e/ou porta de destino e/ou IP de origem e/ou IP de destino;

A Figura 7 mostra a especificação macro do módulo aplicação.

Figura 7: Especificação do módulo aplicação

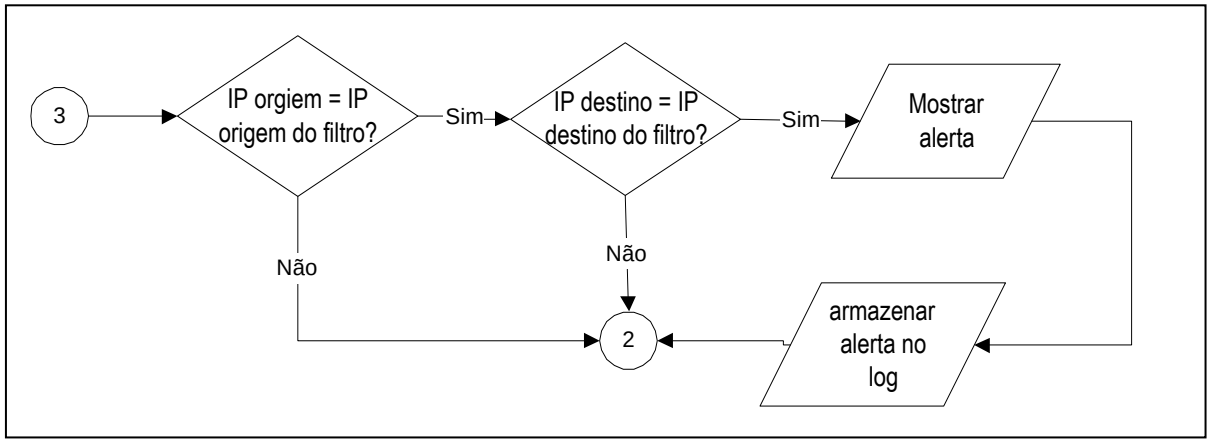


Cada um dos processos de filtragem (ICMP, TCP, UDP) representam uma seqüência de testes para averiguar se o pacote contém as informações selecionadas nos campos de definição dos filtros. Caso isto ocorra, o pacote é apresentado no campo de alerta e é armazenado em um arquivo de *log*.

Na seqüência estão as especificações dos filtros que serão aplicados ao se escolher o protocolo para os alertas.

Na Figura 8 está a especificação do filtro por protocolo ICMP

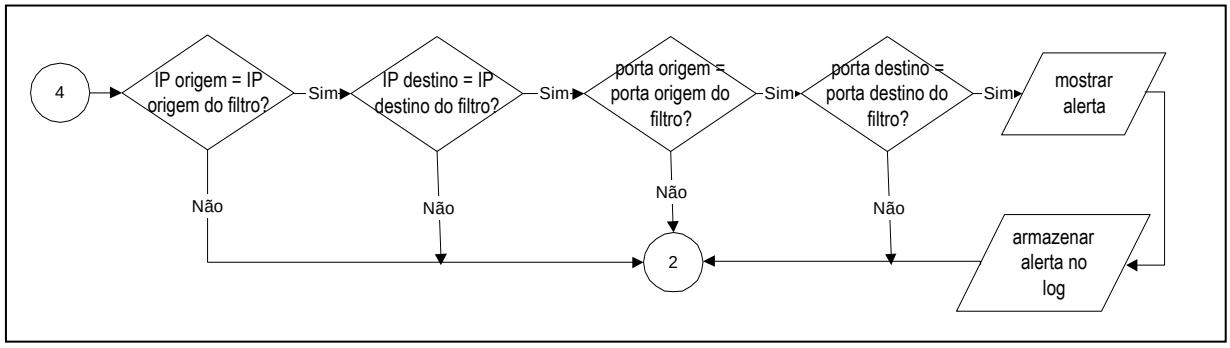
Figura 8: Especificação do filtro por ICMP



As únicas informações que estão sendo disponibilizadas pelo protocolo ICMP são o IP de origem e o IP de destino do pacote.

Os filtros por TCP pode ser visualizado na Figura 9:

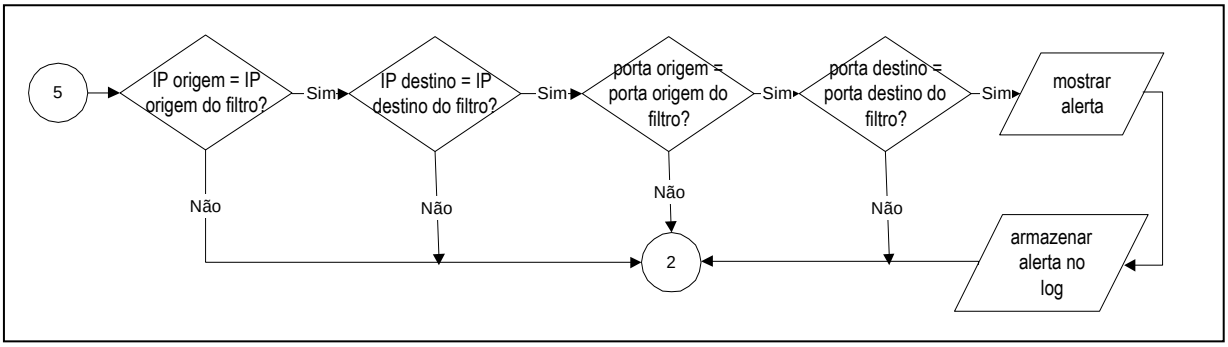
Figura 9: Especificação do filtro por TCP



O protocolo TCP apresenta várias informações nos seus segmentos. Na Figura 9 é possível visualizar a checagem dos IP's de origem e destino, bem como as portas de origem e destino.

O protocolo UDP apresenta quase as mesma informações que o TCP, podendo ser possível utilizar os mesmos filtros, conforme Figura 10.

Figura 10: Especificação do filtro por UDP



5.3 IMPLEMENTAÇÃO

A seguir são apresentadas considerações sobre a implementação do protótipo, como também as ferramentas que foram utilizadas na implementação, tanto do módulo *sniffer* quanto do módulo aplicação. Também são apresentadas as principais funções que compõem o protótipo e o funcionamento do mesmo.

5.3.1 FERRAMENTAS UTILIZADAS NA IMPLEMENTAÇÃO

A implementação do módulo *sniffer* foi desenvolvida na linguagem C, utilizando o ambiente padrão do Linux para a implementação.

A implementação do módulo aplicação foi desenvolvida na linguagem *Object Pascal*, utilizando para tal o ambiente *Kylix*, que é a versão do ambiente *Delphi* para o Linux.

Como os módulos foram desenvolvidos em linguagens diferentes e para que houvesse uma comunicação entre ambos, foi utilizado o recurso de *pipe*, que nada mais é que a criação de um "duto" de comunicação entre os processos (módulos).

O módulo aplicação irá inicializar o *sniffer* e criar o *pipe*, de onde serão coletadas as informações apresentadas pelo *sniffer*.

Todas as informações coletadas são comparadas com as situações de alerta configuradas pelo administrador. Todos os pacotes são mostrados ao administrador, mas apenas os que se encaixarem nas situações configuradas pelo administrador serão mostrados como situações de alerta e armazenados em um arquivo de *log*. Os *logs* são armazenados em arquivos sem tipo, onde podem ser visualizados pelo protótipo.

5.3.2 PRINCIPAIS FUNÇÕES DO MÓDULO *SNIFFER*

O módulo *sniffer* cria um *socket*, coloca o dispositivo de rede em módulo promíscuo e recebe todos os pacotes trafegando na rede. Os pacotes são analisados e mostrados na tela. As funções mostradas na seqüência demonstram esses processos:

Quadro 4: Criação do *socket*

```
int init_socket (char *dev)
{
  sock = socket(PF_PACKET, SOCK_RAW, htons (ETH_P_ALL));
  if (sock < 0 )
  {
    printf ("Erro ao criar socket: %s\n", strerror (errno));
    exit (1);
  }
  return sock;
}
```

A implementação apresentada no Quadro 4 é utilizada para a criação do *socket*, que faz a comunicação com o dispositivo de rede.

Quadro 5: Setar dispositivo em modo promíscuo

```
void promisc_on (int sock, char *dev)
{
  int res;
  struct ifreq ifr;
  /* pegar flags da interface
  strcpy (ifr.ifr_name, dev);
  res = ioctl (sock, SIOCGIFFLAGS, &ifr);
  if (res < 0 )
  {
    close (sock);
    printf ("Erro na chamada ioctl: %s\n", strerror (errno));
    exit (1);
  }
  /* colocar interface em modo promiscuo
  ifr.ifr_flags |= IFF_PROMISC;
  res = ioctl (sock, SIOCSIFFLAGS, &ifr);
  if (res < 0)
   printf ("Erro ao colocar dispositivo em modo promiscuo: %s\n", strerror (errno));
}
```

A implementação apresentada no Quadro 5, é utilizada pelo *sniffer* para setar o dispositivo em modo promíscuo, fazendo com que ele receba todos os pacotes trafegando na rede. Para tanto, ele recebe o *socket* criado anteriormente.

Quadro 6: Funções de tratamento dos pacotes

```
/*************** TRATAMENTO DO FRAME ICMP ***************/
void analyse_icmp_frame (eth_ip_icmp_frame *eth_pkt)
{
  struct icmphdr *icmp;
   icmp = (struct icmphdr *) (((unsigned long) &eth_pkt->icmp) -2);
  printf ("\Cabecalho ICMP: %s\n", icmp_type (icmp->type));
}
/***************** TRATAMENTO DO FRAME TCP ****************/
void analyse_tcp_frame (eth_ip_tcp_frame *eth_pkt)
{
  struct tcphdr *tcp;
  tcp = (struct tcphdr *) (((unsigned long) &eth_pkt->tcp) -2);
  printf ("\Cabecalho TCP: %d -> %d\n", tcp->source, tcp->dest);
}
/************ TRATAMENTO DO FRAME UDP *********************/
void analyse_udp_frame (eth_ip_udp_frame *eth_pkt)
{
  struct udphdr *udp;
  udp = (struct udphdr *) (((unsigned long) &eth_pkt->udp) -2);
  printf ("\tCabecalho UDP: %d -> %d\n", udp->source, udp->dest);
}
```

O Quadro 6 descreve as funções que tratam os pacotes recebidos da rede. Cada pacote é aberto e verificado o tipo de mensagem: ICMP (*icmp_type (icmp->type)*) se for um pacote ICMP; as portas origem e destino (tcp->source, tcp->dest / udp->source, udp->dest) no caso do TCP e do UDP. Todas essas informações são enviadas ao módulo aplicação.

5.3.3 PRINCIPAIS FUNÇÕES DO MÓDULO APLICAÇÃO

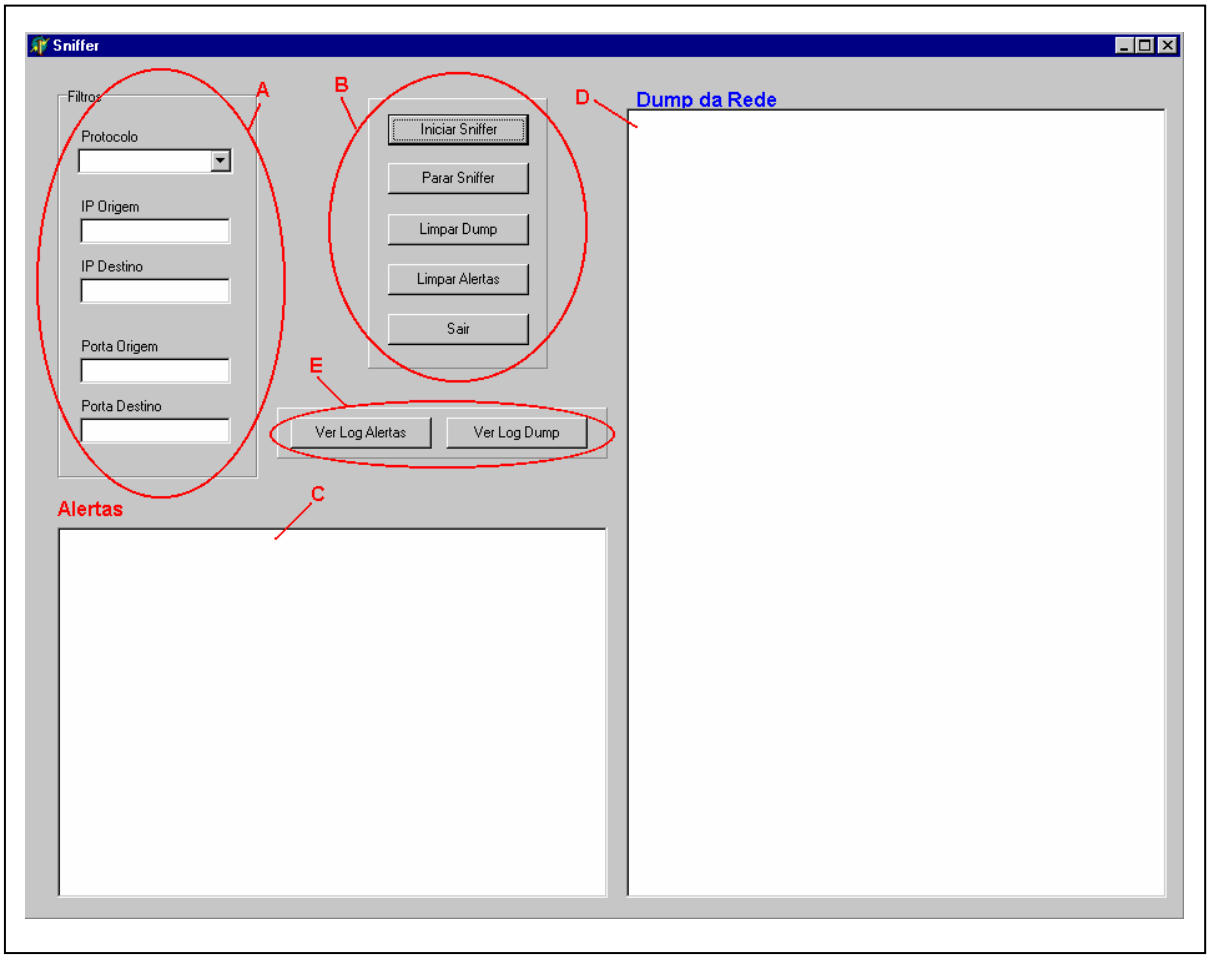
O módulo aplicação é o responsável pela inicialização do módulo *sniffer*. É na aplicação que será criado o *pipe* para a coleta dos dados informados pelo *sniffer*. O módulo aplicação recebe todos os dados, fará a comparação com as situações de alertas configuradas pelo administrador e mostrará os dados de duas formas distintas:

a) como uma amostragem geral da rede, onde todas as informações serão apresentadas ao administrador;

b) como um alerta, onde apenas as informações que forem verificadas com as configurações do administrador são mostradas.

Na Figura 11 está representada a tela principal do módulo aplicação e nos tópicos seguintes segue-se com a especificação das funcionalidades.

Figura 11: Tela do módulo aplicação

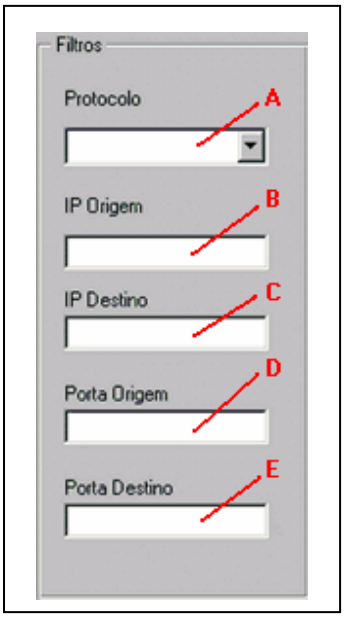


É no módulo aplicação que o administrador irá interagir com o protótipo, configurando situações de alerta e visualizando o que está acontecendo na rede.

5.3.3.1 FILTROS

Conforme apresentado no destaque “A” da Figura 11, será aqui onde o administrador irá configurar as situações de alertas para a monitoração. Essas configurações poderão ser alteradas em qualquer momento da monitoração. Na Figura 12 é possível visualizar as opções de filtragem.

Figura 12: Filtros



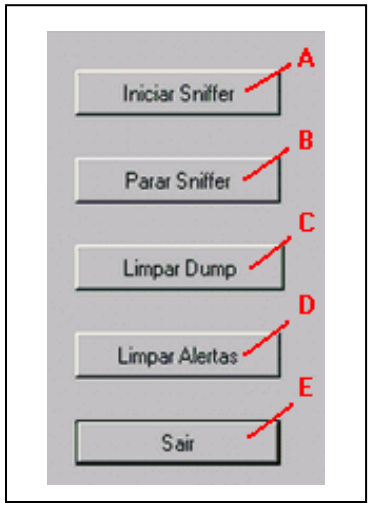
A) protocolo: possibilita a escolha do protocolo para a monitoração: ICMP, TCP, UDP;
B) IP origem: verificação de qual o IP de origem do pacote analisado;
C) IP destino: verificação de qual o IP de destino do pacote analisado;
D) porta origem: a aplicação irá verificar qual a porta de origem do pacote;
E) porta destino: será verificado qual a porta de destino para a qual será endereçado o pacote.

Ao optar pelo protocolo ICMP, apenas as opções “B” e “C” serão habilitadas, devido às características do protocolo. Tanto o UDP quanto o TCP possibilitam a configuração de todas as opções. Não é necessário a configuração de todas as opções, visto que a filtragem será feita levando-se em consideração apenas os itens preenchidos. Essa característica é válida também para os protocolos.

5.3.3.2 BOTÕES

A seguir, uma explanação sobre as funções dos botões da aplicação, conforme apresentado no destaque “B” da Figura 11.

Figura 13: Botões



A) Iniciar Sniffer: este botão tem a função de iniciar o módulo s*niffer* e criar o *pipe* entre o *sniffer* e a aplicação. A partir daqui serão lidos os pacotes e feitas as comparações com os filtros configurados pelo administrador;
B) Parar Sniffer: é interrompida a exibição dos logs, para possibilitar a análise de alguma situação no momento em que ela ocorrer ou para alterar as configurações dos filtros;
C) Limpar Dump: limpa o conteúdo mostrado na tela de Dump de Rede;
D) Limpar Alertas: limpa o conteúdo mostrado na tela de Alertas;
E) Sair: encerra o aplicativo, fecha o *pipe* e fecha o *sniffer*.

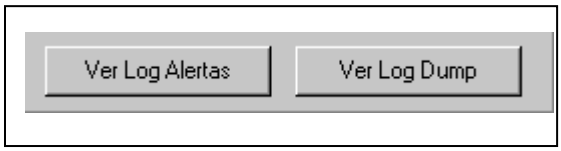
5.3.3.3 ALERTAS E DUMP DA REDE

Conforme os destaques “C” e “D”, apresentados na Figura 11, estão, respectivamente, o espaço onde serão exibidos os alertas gerados pela comparação dos pacotes lidos com as configurações dos filtros e o local onde serão exibidos todos os pacotes trafegando na rede.

5.3.3.4 BOTÕES DE LOG

Nos botões, conforme Figura 14, o administrador poderá visualizar os logs gerados, tanto os de alerta quanto os de todos os pacotes trafegando na rede.

Figura 14: Botões para visualização de logs



6 CONCLUSÕES

O crescimento das redes de computadores e a necessidade de disponibilizar informações é diretamente proporcional ao risco que a empresa corre ao tomar essas providências, podendo sofrer ataques e invasões ou ter suas informações roubadas, alteradas ou danificadas.

As maneiras de se efetuar ataques e invasões são inúmeras, indo desde a utilização de ferramentas desenvolvidas para esse propósito até uma simples conversa com funcionários desavisados (engenharia social). A cada dia surgem ferramentas para a proteção e detecção de tais tentativas, citando-se equipamentos de *firewall*, softwares cada vez mais sofisticados, liberação de correções por parte dos fabricantes e, como demonstrou esse trabalho, a utilização de monitores de pacotes (*sniffers*).

O protótipo desenvolvido nesse trabalho demonstrou ser uma ferramenta eficaz na monitoração de uma rede *Ethernet*, utilizando o protocolo TCP/IP, sendo capaz de monitorar pacotes trafegando na rede e extrair informações úteis no diagnóstico de tentativas de invasão, como pacotes de origens suspeitas com destinos não autorizados. O armazenamento em arquivos de *log* das informações extraídas permite ao administrador uma posterior análise dos acontecimentos, podendo decidir que atitude tomar e possibilitando a identificação das vulnerabilidades, como portas abertas ou sistemas desprotegidos.

Como toda pesquisa, foi possível aplicar na prática conceitos que até então eram vistos apenas em teorias e livros. Ao implementar o protótipo, foi possível entender o funcionamento dos protocolos, bem como sua forma de encapsulamento dos dados e a forma como estão disponibilizadas as informações dentro do mesmo.

A utilização de duas linguagens de programação fez-se necessária para permitir um acesso mais direto das informações, no caso do Linux, até porque as ferramentas de desenvolvimento para o Linux ainda mostram-se um tanto complicadas, demandando um maior período para a adaptação e o aprendizado. Já a possibilidade de se utilizar um ambiente como o *Kylix* permitiu uma maior desenvoltura do desenvolvimento, visto que o mesmo é um "irmão gêmeo" do tão conhecido *Borland Delphi*.

Para o desenvolvimento do módulo *sniffer*, optou-se pela linguagem C pelas facilidades e características que apresenta para a implementação, sem deixar de levar em consideração que o Linux foi originalmente desenvolvido em C, possuindo seu *kernel* implementado nesta linguagem.

É necessário ao administrador de redes ter em mente que todo sistema é totalmente seguro, até ser invadido pela primeira vez e que não existe sistema 100% seguro até que se prove o contrário.

6.1 DIFICULDADES ENCONTRADAS

Foram enfrentadas algumas dificuldades no decorrer do trabalho, podendo-se destacar:

a) peculiaridades no funcionamento dos protocolos, motivo pelo qual alguns erros eram apresentados sem uma causa aparente;
b) documentação sobre *kylix* incompleta, faltando manuais avançados da linguagem, sendo que o que existe até o momento mostra o básico;
c) dificuldades na sincronização de informações entre os módulos.

6.2 LIMITAÇÕES

O protótipo apresenta algumas limitações, podendo-se destacar:

a) o protótipo reconhece apenas cabeçalhos dos protocolos IP, TCP, UDP e ICMP;
b) as opções de alertas limitam-se nos campos de endereçamento IP e das portas de origem e destino;

6.3 EXTENSÕES

Algumas idéias para a extensão do protótipo foram surgindo no decorrer do desenvolvimento do mesmo:

a) monitoração dos dados (conteúdo) contidos dentro dos pacotes;
b) permitir a filtragem de outros protocolos, como ARP, IGMP, RARP;
c) implementar o reconhecimento de protocolos de aplicação como HTTP, FTP, SMTP e Telnet.

7 REFERÊNCIAS BIBLIOGRÁFICAS

BERNSTEIN, Terry; et. Alli; **Segurança na Internet**. Rio de Janeiro: Campus, 1997.

COMER, Douglas E; **Interligação em rede com TCP/IP**. Rio de Janeiro: Campus, 1998.

MCCLURE, Stuart; SCAMBRAY, Joel; KURTZ, George. **Hackers expostos:** segredos e soluções para a segurança de redes. São Paulo: Makron Books, 2000.

OLIVERIA, Wilson José de; **Hacker:** invasão e proteção. Florianópolis: Visual Books, 2000.

REDES, Curso de. **Trabalho sobre as camadas de rede.** Proença, mar [2001?]. Disponível em: <http://proenca.uel.br/curso-redes-graduacao/1998/trab-08>. Acesso em: 25 mar. 2002

RFC 791. **Internet Protocol**. Califórnia, mar [2000?]. Disponível em <http://www.netsys.com/rfc/rfc791.txt>. Acesso em: 12 mar. 2002.

RUBINI, Alessandro; CORBET, Jonathan; **LINUX *device drivers*.** Sebastopol: O'Reilly, 2001.

SILVA, Paulo Fernando da. **Protótipo de software de segurança em redes para a monitoração de pacotes em uma conexão TCP/IP**. 2001. 110 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

SONNINO, Bruno. **Kylix – Delphi para Linux**: guia prático de programação. São Paulo: Makron Books, 2001.

STANG, David J; MONN, Sylvia. **Segredos de segurança em rede.** Rio de Janeiro: Berkeley, 1994.

STARLIN, Gorki. **Manual Completo do *Hacker*:** como ser e evita-los. Rio de Janeiro: Book Express, 1999.

TANENBAUM, Andrew S. **Redes de computadores**. Rio de Janeiro: Campus, 1997.