UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
ESCOLA POLITÉCNICA
DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

Projeto e Implementação de uma Zona Desmilitarizada

Autor:

____

Rafael Pinaud Laufer

Orientador:

____

Prof. Otto Carlos Muniz Bandeira Duarte, Dr.Ing.

Examinadores:

____

Célio Vinicius Neves de Albuquerque, Ph.D.

____

Luís Henrique Maciel Kosmalski Costa, Dr.

____

Prof. Marcelo Gonçalves Rubinstein, D.Sc.

____

Prof. Marcelo Luiz Drumond Lanza, M.Sc.

DEL

Março de 2004

À minha família.

Agradecimentos

Agradeço aos meus pais, Julio e Klaudia, por todo amor, incentivo e orientação passados em todos os anos de minha vida. Aos meus irmãos, Gustavo e Gabriela, por reagirem bravamente aos meus implicantes comentários. A minha avó, Maria da Penha, pelos grandes cuidados com seu neto. A minha namorada Flavia, pelo amor, carinho e enorme compreensão nas horas de estudo. Sem eles, a conclusão dessa importante fase nunca aconteceria.

Ao professor e orientador Otto, responsável por grande parte da minha formação pessoal, acadêmica e profissional, por seus conselhos e orientação. Ao doutor Luís Henrique e aos professores Rezende e Rubi, pela amizade, ensinamentos e dicas.

Aos amigos e colegas de graduação, Bruno, Igor e Marco, pela ajuda na implementação e pelos bons momentos passados tanto nas horas de trabalho quanto nas de lazer. Aos amigos e colegas de trabalho, Aurélio, Bernardo, Daniel, Luiz Gustavo, Miguel e Pedro, pelas risadas e pela troca de experiências. Ao meu grande amigo Felipe, pelas inesquecíveis idas e voltas do Fundão.

Aos professores do Departamento de Eletrônica e de Computação, por me passarem um pouco do seu conhecimento. Agradeço em particular aos doutores Luís Henrique Maciel Kosmalski Costa e Célio Vinicius Neves de Albuquerque e aos professores Otto Carlos Muniz Bandeira Duarte, Marcelo Luiz Drumond Lanza e Marcelo Gonçalves Rubinstein, pela presença na banca examinadora e contribuição neste trabalho.

Resumo do Projeto Final apresentado ao DEL/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletrônico e de Computação.

Projeto e Implementação de uma Zona Desmilitarizada

Rafael Pinaud Laufer

Março de 2004

Orientador: Otto Carlos Muniz Bandeira Duarte

Departamento: Engenharia Eletrônica e de Computação

A maioria dos sítios ainda hoje apresenta uma topologia que consiste apenas de uma única rede protegida por um filtro de pacotes no roteador. Essa topologia é útil e segura para redes domésticas, onde não há necessidade de se oferecer nenhum serviço através da rede. Porém, para redes que necessitam prover serviços, essa topologia não é adequada. Caso ocorra alguma invasão passando pelo filtro e atingindo um dos servidores, a segurança de toda a rede está comprometida, pois não existe nenhum tipo de proteção entre os servidores e as outras estações que impeça o invasor de um ataque geral à rede. Logo, uma maneira de impossibilitar que as estações de trabalho sejam invadidas se torna atrativa. A utilização de uma zona desmilitarizada resolve esse problema, diminuindo o impacto de uma invasão através da segmentação da rede. O objetivo deste trabalho é justamente projetar e implementar uma zona desmilitarizada em uma rede de produção científica. Para isso, é realizada toda uma reestruturação lógica e física da topologia da rede de forma a limitar as possíveis proporções alcançadas por uma invasão e, ao mesmo tempo, não impedir a disponibilização dos seus serviços.

Palavras Chave

Redes de computadores

Internet

Segurança

Zona Desmilitarizada

Lista de Acrônimos

BASH: *Bourne Again Shell*;

COPPE: Coordenação dos Programas de Pós-graduação de Engenharia;

DF: *Don't Fragment*;

DHCP: *Dynamic Host Configuration Protocol*;

DNS: *Domain Name System*;

DoS: *Denial of Service*;

FTP: *File Transfer Protocol*;

GNU: *GNU's Not Unix*;

GPL: *General Public License*;

GTA: Grupo de Teleinformática e Automação;

HLEN: *Header Length*;

HTTP: *Hypertext Transfer Protocol*;

ICMP: *Internet Control Message Protocol*;

IDS: *Intrusion Detection System*;

IGMP: *Internet Group Management Protocol*;

IHL: *IP Header Length*;

IMAP: *Internet Message Access Protocol*;

IP: *Internet Protocol*;

IPv4: *Internet Protocol version 4*;

ISO: *International Organization for Standardization*;

MF: *More Fragments*;

NAT: *Network Address Translation*;

NFS: *Network File System*;

NIS: *Network Information Service*;

OSI: *Open Systems Interconnection*;

OSPF: *Open Shortest Path First*;

PAM: *Pluggable Authentication Modules*;

PID: *Process Identifier*;

PIM: *Protocol Independent Multicast*;

POP3: *Post Office Protocol version 3*;

QoS: *Quality of Service*;

RIP: *Routing Information Protocol*;

SFTP: *Secure FTP*;

SMTP: *Simple Mail Transfer Protocol*;

SSH: *Secure Shell*;

TCP: *Transfer Control Protocol*;

ToS: *Type of Service*;

TTL: *Time To Live*;

UDP: *User Datagram Protocol*;

UFRJ: Universidade Federal do Rio de Janeiro;

VLAN: *Virtual Local Area Network*;

VPN: *Virtual Private Network*;

ZDM: Zona Desmilitarizada.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo I

Introdução

Ao mesmo tempo em que ocorreu um enorme avanço na comunicação entre redes, tornando possível o compartilhamento globalizado de seus recursos e informações, a segurança destas redes passou a ser crítica. Se por um lado este compartilhamento permite aumentar o dinamismo das comunicações e a troca de informações, por outro, ele possibilita que pessoas não autorizadas obtenham acesso a dados sigilosos. Por isso, a segurança é considerada essencial no projeto de uma rede, seja esta corporativa, acadêmica ou até mesmo pessoal. As razões dessa nova preocupação estão relacionadas diretamente com o valor dos dados armazenados e ainda com a necessidade de se garantir diferentes níveis de privacidade, integridade e autenticidade para cada tipo de dado.

Para minimizar as chances de que dados sejam comprometidos, um dos passos a serem tomados é analisar a topologia da rede a procura de possíveis falhas de segurança. A adoção de uma topologia que contenha uma zona desmilitarizada vem justamente acabar com uma grave falha que existe em diversas redes, conforme explicado a seguir.

I.1 Motivação

A principal motivação para o projeto e a implementação de uma zona desmilitarizada é que a grande maioria dos sítios ainda hoje apresenta uma topologia que consiste apenas de uma única rede protegida por um filtro de pacotes no roteador. Essa topologia é útil e segura para redes domésticas, onde não há necessidade de se oferecer nenhum serviço através da rede. Porém, para redes que necessitam prover serviços, essa topologia não é adequada. O provimento de serviços a usuários de redes externas se torna problemático à medida que é necessário liberar a passagem de pacotes destinados a essas aplicações no filtro do roteador. Com esta liberação, possíveis vulnerabilidades presentes na implementação do serviço podem ser exploradas por usuários mal intencionados com o objetivo de comprometer o servidor. Diversos tipos de vulnerabilidade existem e são regularmente exploradas nas aplicações.

Entretanto, o risco de comprometimento do servidor sempre existirá. Se um serviço precisa ser disponibilizado, é porque é necessário que usuários o utilizem ou o acessem. Logo, é preciso que o acesso destes usuários ao servidor seja liberado de alguma forma. Seria, no mínimo, contraditório se existisse um serviço oferecido e nenhum usuário tivesse acesso ao mesmo. É claro que deste modo estaria assegurado que o servidor nunca seria comprometido, mas ele também não teria funcionalidade alguma. O problema de uma invasão se agrava ainda mais quando a topologia usada consiste de apenas uma rede onde se encontram tanto os servidores quanto as estações de trabalho. Caso ocorra alguma invasão passando pelo filtro e atingindo um dos servidores, a segurança de toda a rede está comprometida, pois não existe nenhum tipo de proteção entre os servidores e as outras estações que impeça o invasor de um ataque geral à rede. Desse modo, uma vez comprometido um dos servidores, é criado um ponto de partida para ataques futuros em máquinas internas.

A utilização de uma zona desmilitarizada é uma alternativa para o problema descrito acima. Esta solução segmenta a rede em duas sub-redes ou zonas: a zona desmilitarizada - uma rede exclusiva de servidores que provêem serviços externos - e a rede interna - onde todas as estações de trabalho estarão situadas. Ao colocar os servidores em um segmento de rede separado, pacotes destinados às estações de trabalho vindos tanto da zona desmilitarizada quanto de redes externas precisam necessariamente ser roteados. Sendo o roteador equipado com um filtro adequado de pacotes, é possível agora impedir que máquinas da rede interna venham a ser comprometidas, mesmo que algum servidor seja invadido.

A rede do Grupo de Teleinformática e Automação (GTA) da COPPE/UFRJ utiliza a topologia de uma única rede, o que a torna vulnerável a ataques partidos do servidor. Este projeto é proposto com o objetivo de implementar uma nova topologia, de maneira que nela seja incluída uma zona desmilitarizada. Para isso, é realizada toda uma reestruturação lógica e física da topologia da rede de forma a limitar as possíveis proporções alcançadas por uma invasão e, ao mesmo tempo, não impedir a disponibilização dos seus serviços.

I.2 Organização

Este projeto está organizado da seguinte forma. No Capítulo II, são revisados os conceitos de rede necessários para o entendimento do projeto. No Capítulo III, a segurança de rede é abordada explicando o funcionamento dos filtros de pacotes (*firewalls*) e ainda apresentando com mais detalhes as topologias mais usadas, dentre elas, a zona desmilitarizada. No Capítulo IV, é apresentada a topologia da rede do laboratório e o projeto proposto. No Capítulo V, é mostrado o funcionamento da ferramenta **iptables** e como foi feita a implementação da zona desmilitarizada utilizando esta ferramenta. Por fim, o Capítulo VI apresenta as conclusões tiradas.

Capítulo II

O Modelo de Referência TCP/IP

A ARPANET, uma das primeiras redes de computadores, surgiu inicialmente como uma rede de pesquisa do Departamento de Defesa dos Estados Unidos (*U.S. Department of Defense*). Aos poucos, universidades e agências do governo americano foram se conectando e novos enlaces de rádio e satélite foram adicionados, alterando de forma significativa o comportamento do tráfego. Com isso, protocolos então existentes começaram a apresentar problemas ao interconectar todos os integrantes e uma nova arquitetura de comunicação se tornou necessária. O requisito principal dessa nova arquitetura era que a interconexão de diversas redes fosse independente do tipo de tecnologia empregada. Além disso, a preocupação com que esta nova rede continuasse funcional mesmo que parte dela fosse destruída era uma outra meta a ser alcançada. Outro ponto importante seria que os diferentes requisitos das aplicações que ali trafegariam obrigavam a construção de uma arquitetura flexível o bastante para suportá-las satisfatoriamente. Essa arquitetura mais tarde foi nomeada como modelo de referência TCP/IP (*Transmission Control Protocol / Internet Protocol*), devido aos dois protocolos principais que a compõem.

II.1 As Camadas do TCP/IP

Assim como o modelo de referência OSI (*Open Systems Interconnection*) da ISO (*International Organization for Standardization*), o modelo TCP/IP também é composto por camadas. Existem quatro camadas que o compõem, como pode ser visto na Figura II.1.

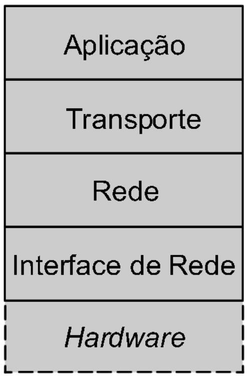

Figura II.1: Camadas do modelo TCP/IP.

Cada camada tem uma funcionalidade específica:

- **Camada de Aplicação**

  A camada de aplicação contém diversos protocolos que são usados pelos usuários. Um exemplo desses protocolos é o HTTP (*Hypertext Transfer Protocol*), o qual é usado para a transferência de páginas *web*. Usando esse protocolo, o navegador envia o nome da página requisitada pelo usuário para o servidor e este, em resposta, envia de volta a página solicitada ao navegador. Além do HTTP, existem outros diversos protocolos que têm aplicações diferentes como o FTP (*File Transfer Protocol*), que serve para a transferência de arquivos, ou ainda o SMTP (*Simple Mail Transfer Protocol*), criado para a troca de mensagens de correio eletrônico.

  Denomina-se programa de aplicação aquele responsável por acessar ou disponibilizar serviços da camada de aplicação em uma rede. No exemplo anterior, tanto o navegador (cliente) quanto o servidor atuam neste papel. Este programa de aplicação interage diretamente com um dos protocolos de transporte para enviar ou receber dados. Dependendo do tipo de aplicação, é escolhido um protocolo adequado de transporte, que pode ser uma seqüência de mensagens isoladas ou ainda um fluxo contínuo de mensagens de uma conexão previamente estabelecida.

- **Camada de Transporte**

  O principal objetivo da camada de transporte é permitir a comunicação fim-a-fim, ou seja, entre dois programas de aplicação distintos. Porém, existem duas maneiras que podem ser usadas para que essa comunicação seja realizada. A primeira maneira é através do TCP (*Transport Control Protocol*), um protocolo confiável e orientado a conexão que permite que uma cadeia de bytes originados em uma máquina seja entregue sem erros para qualquer outra máquina da rede. Para isso, é necessário que o receptor envie reconhecimentos ao emissor, informando-o sobre uma eventual perda de mensagens, e também que o emissor retransmita as mensagens perdidas. Além disso, se necessário, o TCP também fragmenta a cadeia de bytes recebida da aplicação em diversas mensagens de tamanho menor e as repassa para a camada de rede. Então, no destino, o processo de recepção do TCP remonta as mensagens recebidas formando novamente a cadeia de bytes original e a repassa para a aplicação. Outra funcionalidade do TCP é o controle de fluxo, utilizado para diminuir ou aumentar a taxa com que as mensagens são transmitidas. Desse modo, é possível controlar a taxa de transmissão do emissor para que um receptor mais lento não seja inundado com mensagens que ele não consegue tratar a tempo [3].

  A outra maneira para a realização da comunicação através da camada de transporte é através do UDP (*User Datagram Protocol*). Ao contrário do TCP, o UDP é um protocolo não confiável e, portanto, não possui nenhuma garantia de entrega das mensagens enviadas. Por isso, agora o receptor não retorna mais uma mensagem de reconhecimento de volta para o emissor. Além disso, o UDP não possui um estabelecimento prévio de conexão antes do envio de qualquer mensagem, sendo cada uma tratada independentemente das demais. Uma outra característica desse protocolo é que ele também não dispõe de qualquer tipo de controle de fluxo. A ausência dessas funcionalidades faz com que o UDP seja um protocolo simples. Entretanto, necessitando-se alguma função inexistente no protocolo, é preciso implementá-la na camada de aplicação.

Dado que os protocolos de transporte e os programas de aplicação da fonte e do destino se comunicam diretamente, essas camadas são denominadas de comunicação fim-a-fim. Isso quer dizer que os processos responsáveis pelas camadas de transporte e aplicação da fonte se comunicam com esses mesmos processos do destino final, através dos respectivos protocolos. Em contrapartida, na camada de rede, a comunicação é feita de uma outra maneira, chamada nó-a-nó. Nessas camadas, os protocolos trocam suas mensagens entre cada máquina e sua vizinha física, e não mais entre a fonte e o destino, que podem estar separadas por várias máquinas, denominadas roteadores. A Figura II.2 ilustra o conceito da troca de mensagens em uma comunicação com um roteador, mostrando ainda o modo de comunicação e o protocolo de cada camada.

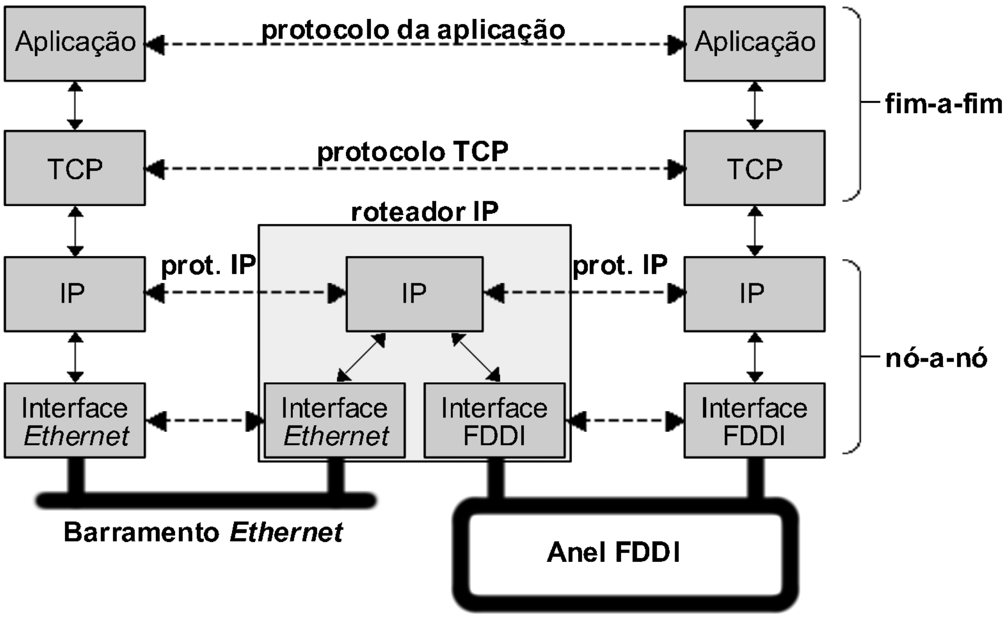

Figura II.2: Troca de mensagens em uma comunicação com um roteador.

- **Camada de Rede**

A camada de rede tem como objetivo principal permitir que todos os computadores sejam capazes de injetar pacotes a partir de qualquer rede e fazer com que cada um deles seja roteado de forma autônoma até alcançar o destino final. Para isso, ela aceita um pedido para enviar um pacote da camada de transporte, junto com um identificador do destinatário. A camada de rede então adiciona seu cabeçalho à unidade de dados recebida, consulta a tabela de rotas para determinar qual é a próxima máquina que deve recebê-lo e por qual interface de saída aquele datagrama deve ser transmitido e, por fim, passa os dados para a interface de rede escolhida. Outra função dessa camada é tratar os datagramas recebidos, checando sua validade e decidindo se ele deve ser processado localmente ou reencaminhado. Também é função da camada de rede permitir a interconexão de redes heterogêneas, ou seja, possibilitar que mesmo máquinas que possuem tecnologias de rede totalmente distintas ainda possam se comunicar. Para exercer essas funções, foi definido um protocolo oficial que é usado nesta camada, o IP (*Internet Protocol*).

- **Camada de Interface de Rede**

  Esta camada é responsável por aceitar os datagramas IP provenientes da camada superior e transmití-los na rede. Sendo esta camada a responsável pela transmissão sobre o meio físico, tanto o *driver* quanto os protocolos aqui usados dependem fortemente da tecnologia de rede empregada, que pode ser, por exemplo, o *Ethernet*, *Frame Relay* ou *Token Ring*. Cada uma dessas tecnologias possui um padrão diferente que define diversas características usadas na transmissão, como qual voltagem representa o bit 1 e qual representa o bit 0, o tempo de duração de um bit, como as máquinas se conectam fisicamente e logicamente, qual o formato dos endereços e ainda como é feito o método de acesso ao meio. Obviamente, todas essas características precisam ser consideradas durante a transmissão.

Assim, pode-se definir finalmente uma arquitetura do modelo TCP/IP baseando-se nas camadas e protocolos descritos. A Figura II.3 mostra essa arquitetura, exemplificando através das setas o modo como os protocolos podem ser empilhados.

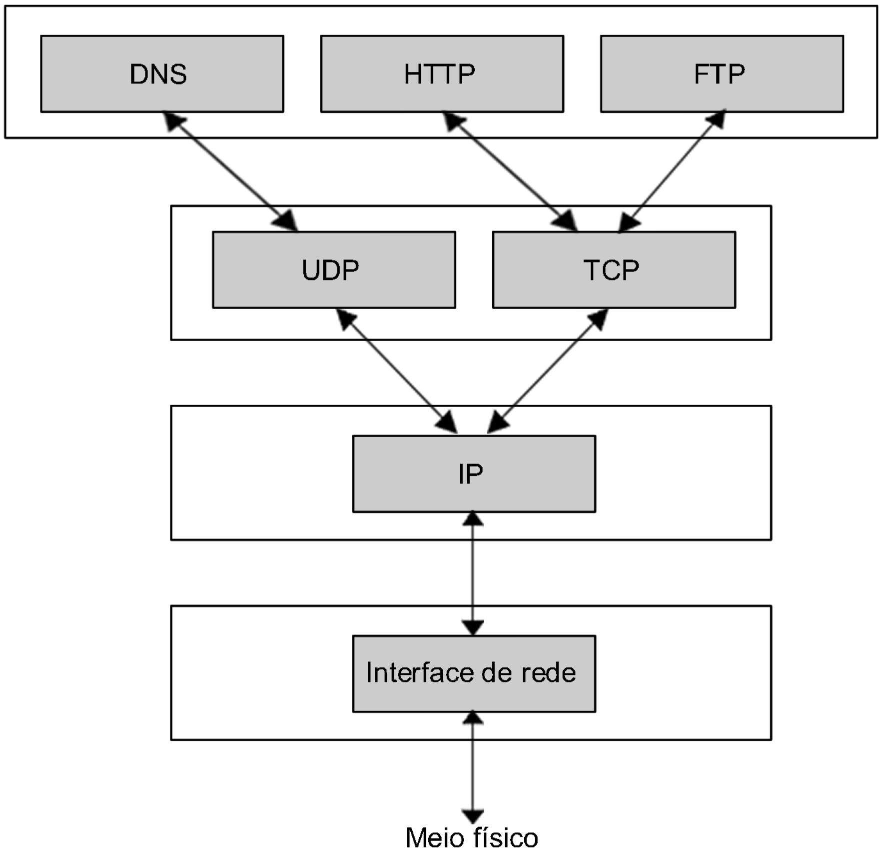

Figura II.3: A arquitetura TCP/IP.

II.2 *Internet Protocol* - IP

Um dos serviços mais fundamentais para a interconexão de redes é o sistema de entrega de pacotes. De acordo com o modelo de serviço IP, o sistema de entrega de pacotes deve ser não confiável, não deve estabelecer conexão e deve usar a entrega de melhor esforço. O serviço é não confiável porque não existe nenhuma garantia de que os pacotes enviados pela fonte irão realmente chegar ao destino. Diversos problemas podem surgir ao longo da comunicação, pois os pacotes podem ser perdidos, duplicados, atrasados ou ainda chegar fora de ordem, mas não cabe ao serviço detectar nem corrigir esses problemas. Essa função é inteiramente deixada para as camadas superiores, mais especificamente para a camada de transporte. Além disso, o serviço não estabelece conexão porque, ao contrário da técnica de circuito virtual, a técnica de datagramas utilizada pelo IP adota que cada pacote deve ser tratado de forma autônoma. Devido a isso, dentro de uma seqüência de pacotes, enviados de uma determinada origem para um mesmo destino, alguns pacotes podem chegar fora de ordem ou ainda serem perdidos, enquanto outros são entregues sem problemas. Também é dito que o serviço usa a entrega de melhor esforço porque todos os dispositivos que constituem a rede se esforçam ao máximo para que todos os pacotes alcancem seu destino final. Isso quer dizer que os pacotes não são descartados seletivamente. Apenas em caso de escassez de recursos, sejam eles memória, processamento ou banda passante, os pacotes que não podem ser tratados a tempo são jogados fora.

O protocolo responsável pela execução dessas três tarefas é o Protocolo de Inter-rede, ou *Internet Protocol* (IP). Esse protocolo tem alguns papéis fundamentais para o funcionamento da rede. Primeiro, o IP define o formato da unidade de dados básica que é usada pela camada de rede. Ou seja, qualquer dado transferido através de uma rede TCP/IP leva consigo no cabeçalho alguns dados de controle que são utilizados pelo IP. Segundo, esse protocolo também exerce a função importante de roteamento, que permite toda comunicação dentro da rede. O roteamento é um processo que acontece em máquinas responsáveis por interconectar redes distintas, os chamados roteadores. Nestes roteadores, o endereço de destino de cada pacote recebido é analisado e, a partir dele, é escolhida uma interface de saída para retransmitir o pacote para o próximo roteador. Esse processo é repetido por todos os roteadores do caminho até o pacote finalmente alcançar o seu destino. Terceiro, o IP estabelece um conjunto de regras responsáveis na prática pela entrega não confiável de pacotes, pelo não estabelecimento de conexão e ainda pelo serviço de melhor esforço. As regras definem o comportamento dos roteadores e estações finais em relação ao processamento de pacotes, à geração de mensagens de erros e ainda ao descarte de pacotes. O nome da unidade de dados correspondente ao protocolo IP é datagrama. O datagrama IP é formado por um cabeçalho e pelos dados do usuário. O cabeçalho, conhecido como cabeçalho IP, da versão 4 do protocolo (IPv4) tem uma parte fixa de 20 bytes e uma parte opcional de tamanho variável, como pode ser visto na Figura II.4.

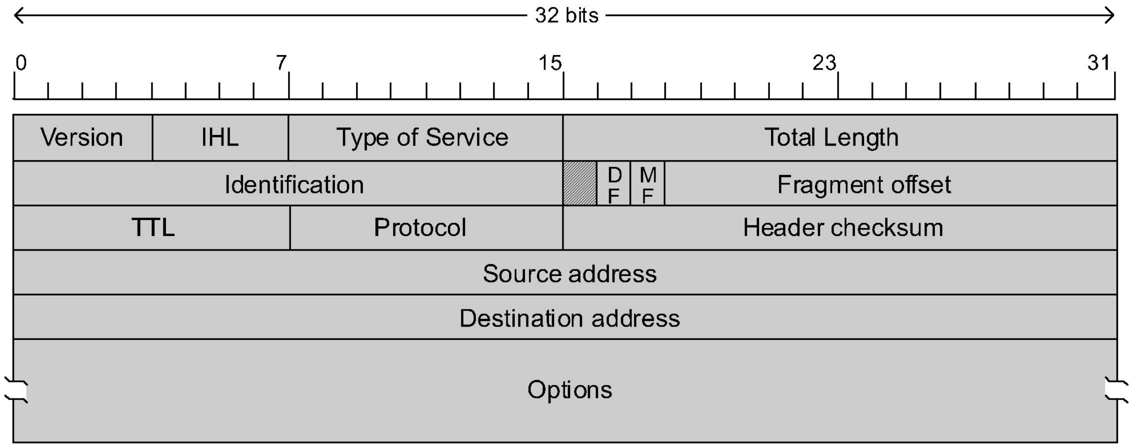

Figura II.4: O cabeçalho da versão 4 do protocolo IP (IPv4).

O campo *Version*, formado por 4 bits, é o responsável por informar a versão do protocolo usada para criar o datagrama. Desse modo, o roteador ou a estação final já sabe qual o formato esperado do cabeçalho e dos dados e, assim, pode tratar corretamente o pacote. O objetivo principal desse campo é permitir a transição entre versões diferentes do protocolo.

Dado que o cabeçalho não tem tamanho fixo, o campo IHL (*IP Header Length*) informa o seu tamanho em palavras de 32 bits. Como o tamanho mínimo do cabeçalho é 20 bytes, o valor mínimo desse campo é 5. Seu valor máximo é 15, o que limita o cabeçalho a 60 bytes, dos quais 40 bytes são destinados às opções. Apesar de aparentar muito, para algumas opções, como a que registra a rota pela qual o pacote percorre, esse tamanho ainda é considerado pequeno.

Alguns serviços são sensíveis ao atraso, à variação do atraso ou ainda à confiabilidade. O campo ToS (*Type of Service*) foi designado para fazer uma diferenciação dos pacotes, de forma que fosse possível tratá-los diferentemente de acordo com sua classificação. Assim, por exemplo, o tráfego de pacotes de voz e vídeo poderia ser tratado com prioridade em detrimento do tráfego de dados.

O campo *Total Length* informa o tamanho total do datagrama, incluindo tanto cabeçalho quanto dados. Com 16 bits, o tamanho máximo do pacote fica limitado a 64.535 bytes, o que pode ser um problema para as futuras redes gigabit [3].

Os campos *Identification*, DF, MF e *Fragment offset* são os responsáveis por possibilitar a fragmentação do pacote. A fragmentação é necessária quando o tamanho do pacote é grande demais para ser entregue pela tecnologia de rede empregada. Para cada tecnologia de rede, existe um tamanho máximo de quadro que não deve ser ultrapassado. No caso do *Ethernet*, por exemplo, o valor máximo padrão é 1500 bytes, bem abaixo dos 64 kbytes do IP. Logo, cada campo tem sua função para permitir que o pacote possa ser fragmentado e, ao alcançar seu destino, ser remontado novamente. O campo *Identification* permite ao destino identificar a qual datagrama pertence aquele fragmento. Todos os fragmentos de um mesmo datagrama têm o mesmo valor de identificação. O campo DF (*Don't Fragment*), de somente 1 bit, serve para avisar aos roteadores que o pacote não pode ser fragmentado porque a estação final não é capaz de remontar o pacote. O campo MF (*More Fragments*), também de 1 bit, indica que ainda existem mais fragmentos para chegar. O último fragmento apresenta este bit zerado para avisar que não há mais nenhum pacote faltando. Por fim, o campo *Fragment offset* indica aonde dentro do datagrama aquele fragmento se localiza, medido em unidade de 8 bytes. Todos os fragmentos, menos o último, devem ser múltiplos de 8 bytes. Para a remontagem, o destino deve obter todos os fragmentos e concatená-los, iniciando daquele cujo deslocamento é zero até o último fragmento.

O campo TTL (*Time to live*) tem a finalidade de limitar o tempo de vida dos pacotes. Ele foi inicialmente criado para contar o tempo em segundos, sendo 255 segundos o tempo de vida máximo de um pacote. Porém, na prática, cada roteador decrementa o TTL de um à medida que o pacote o atravessa. Quando o valor chega a zero, uma mensagem de erro é enviada de volta para a fonte avisando-a de que o pacote foi descartado devido ao tempo de vida ter expirado. Esse mecanismo previne que os pacotes fiquem trafegando indefinidamente pela rede, caso exista alguma instabilidade no roteamento, como um *loop*.

Após a recepção do pacote, é preciso identificar o que está sendo levado pela parte de dados do datagrama. O campo *Protocol* serve justamente para informar qual é o protocolo usado na camada diretamente superior, sendo assim possível entregar os dados para o processo correto de tratamento desses dados. Basicamente, é ele o encarregado da multiplexação e demultiplexação. Os protocolos mais comuns que são carregados pelo IP são os protocolos de transporte TCP e o UDP. Para a correta interpretação desse campo em toda a rede, existe um órgão responsável pela atribuição de um número para cada protocolo.

Com a finalidade de detectar erros, foi adicionado ao cabeçalho o campo *Header checksum*. Esse campo contém o resultado de uma operação que é feita com todos os campos do cabeçalho. Dado que o cabeçalho é modificado a cada salto devido ao decremento do campo TTL, a soma precisa sempre ser recalculada antes do roteador retransmitir o pacote pela interface de saída.

Os campos *Source Address* e *Destination Address* indicam os endereços da fonte e do destino, respectivamente. Esses campos são formados por endereços de 32 bits que nunca são mudados ao longo do roteamento do pacote. Durante esse processo desde a fonte até o destino final, somente o endereço de destino é analisado por cada roteador. Usando este endereço, a tabela de roteamento é consultada para saber por qual interface o pacote deve ser retransmitido.

Por fim, ainda é possível incluir no cabeçalho algumas opções que servem principalmente para testes. O tamanho do campo de opções varia de acordo com aquelas selecionadas. Algumas têm o tamanho de um simples byte, enquanto outras têm tamanho variável.

II.3 *User Datagram Protocol* - UDP

O protocolo de transporte da arquitetura TCP/IP que oferece um mecanismo simples para o envio e recebimento de segmentos[1] é o UDP (*User Datagram Protocol*). Ele usa os serviços da camada inferior para o transporte de mensagens de uma comunicação

[1]Segmento é o nome da unidade de dados da camada de transporte.

fim-a-fim e provê exatamente o mesmo tipo de entrega não confiável e sem estabelecimento prévio de conexão que o IP. Nenhum controle de erro, controle de fluxo ou procedimento de retransmissão existe neste protocolo. Sua única função é possibilitar a troca de mensagens entre processos rodando em máquinas diferentes. Para isso, dado que em um sistema existem diversos processos concorrentes, é necessária uma forma de identificar o processo que deve receber a informação carregada por um datagrama recém-chegado. A forma escolhida foi inserir bits extras no cabeçalho do UDP que fossem responsáveis por essa identificação, as chamadas portas. Basicamente, as portas têm o objetivo de fazer a multiplexação e a demultiplexação ao nível de processos. Além da porta usada no destino, também deve ser incluída no cabeçalho a porta usada na origem, de tal forma que seja possível o envio de uma resposta para o processo iniciador da conversação. Como o identificador do processo que é usado pelo sistema operacional, o PID (*Process Identifier*), é diferente do número da porta usada durante a comunicação, o sistema operacional deve ser avisado que todos os dados recebidos para uma determinada porta devem ser encaminhados para um processo específico.

Como o UDP possui poucas funcionalidades, seu cabeçalho é bem pequeno. A Figura II.5 mostra o seu formato, composto por quatro campos de 16 bits. Ele é formado pelas portas de origem e destino, o tamanho do cabeçalho mais os dados e, ainda, uma soma de verificação que inclui alguns campos do cabeçalho IP, UDP e a parte de dados do usuário.

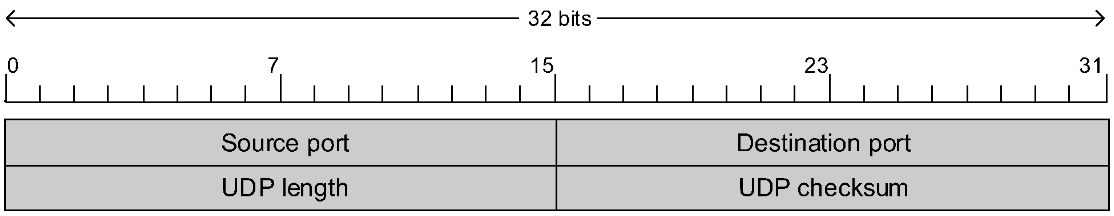

Figura II.5: O cabeçalho UDP.

Uma das aplicações mais conhecidas que usa o UDP é o Sistema de Nomes de Domínio (*Domain Name System* - DNS). Tal aplicação é responsável pela resolução de nomes, como `www.ufrj.br`, em endereços IP. Assim, uma estação que necessita o endereço de um determinado nome envia um pacote UDP para um servidor de DNS, requisitando sua resolução. O servidor então envia uma resposta contendo o endereço IP solicitado.

II.4 *Transmission Control Protocol* - TCP

Ao contrário do UDP, o TCP (*Transmission Control Protocol*) é um protocolo orientado a conexão que oferece serviços de entrega confiável, controle de erro e controle de fluxo. Além disso, este protocolo também torna possível a transmissão de longas cadeias de bytes, através de procedimentos de segmentação e remontagem. Isso torna o protocolo mais complexo se comparado ao UDP.

Para realizar o controle de fluxo, o TCP utiliza um mecanismo de janela deslizante de tamanho variável. Para seu funcionamento, os dois extremos da comunicação informam o tamanho da sua janela no cabeçalho do protocolo. O tamanho da janela indica quantos bytes podem ser enviados em avanço antes de se receber um reconhecimento confirmando a chegada correta desses bytes. Através da informação enviada, é possível realizar o controle de fluxo. Este controle é feito de acordo com a ocupação dos *buffers* de cada máquina. Se em uma delas os *buffers* estiverem quase cheios, ela pode reduzir o tamanho da janela, inibindo a transmissão e, em caso contrário, ela pode aumentá-la.

O controle de erro é efetuado pela verificação do seqüenciamento dos pacotes. Tendo cada segmento um número de seqüência, é possível agora saber se os pacotes são perdidos, duplicados ou chegam desordenados. Além disso, a possibilidade de que mensagens maiores do que os limites das camadas inferiores sejam transferidas separadamente e depois remontadas também só é possível devido ao seqüenciamento. Como a transferência de dados no TCP é bidirecional, é necessário um número de seqüência para cada fluxo unidirecional para que seja possível a recuperação de dados enviados em qualquer sentido. Além da numeração dos segmentos, para o controle de erro também são usados temporizadores que controlam procedimentos relacionados à retransmissão.

O cabeçalho do TCP é mostrado na Figura II.6. Todo segmento TCP começa com um cabeçalho fixo de, no mínimo, 20 bytes. Depois do cabeçalho fixo, podem existir opções do protocolo, assim como ocorre com o protocolo IP. Após as opções, podem estar até 65.535 - 20 - 20 = 65.495 bytes, sendo os 20 primeiros bytes devido ao cabeçalho IP e os 20 seguintes correspondentes ao cabeçalho TCP. Segmentos sem nenhum dado podem existir e servem para reconhecimentos e mensagens de controle.

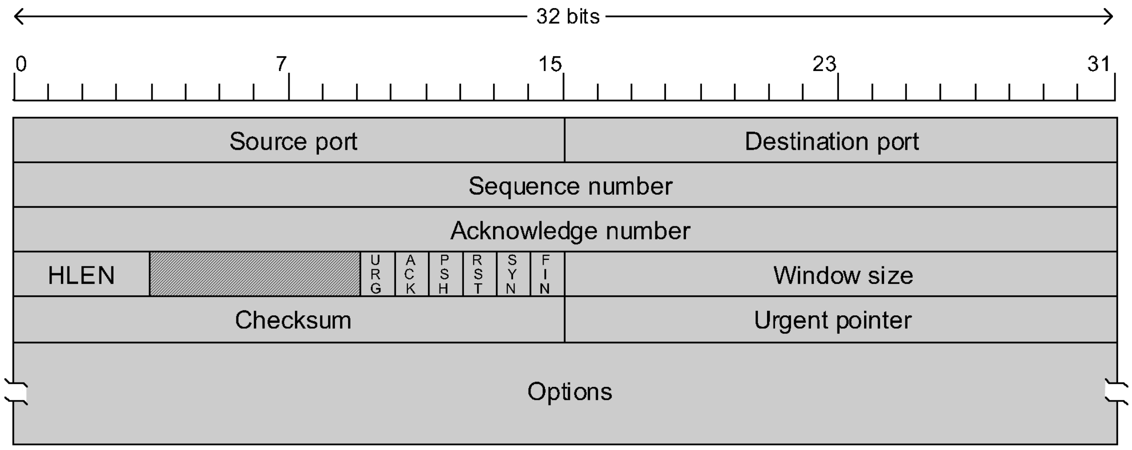

Figura II.6: O cabeçalho TCP.

Os campos *Source Port* e *Destination Port* identificam as portas de origem e destino, respectivamente. Elas têm a mesma função apresentada no protocolo UDP: permitir a multiplexação e demultiplexação ao nível de processos. Mais especificamente, a partir de um segmento recebido, o sistema operacional verifica se a porta de destino está associada a algum processo. Caso esteja, os dados recebidos são encaminhados para esse processo para devido tratamento. Ao enviar uma resposta, as portas de origem e destino do segmento recebido são trocadas e o novo segmento é enviado de volta. O mesmo processo se repete nas duas máquinas até a conexão se encerrar.

O campo *Sequence number* é o número de seqüência do segmento e o campo *Acknowledge number* é o número de seqüência do próximo byte esperado no outro sentido da comunicação. Cada byte de uma cadeia enviada pelo TCP é numerado e, portanto, os dois números representam bytes e não segmentos.

Para especificar o tamanho do cabeçalho do segmento, o campo HLEN (*Header Length*) é utilizado. A sua presença se torna necessária porque o tamanho do cabeçalho não é fixo, podendo variar devido à presença de opções. De acordo com a especificação, este campo representa o deslocamento da área de dados em relação ao início do segmento, contado em palavras de 32 bits.

Após um espaço de 6 bits não utilizados, tem-se 6 flags de 1 bit cada que têm funções específicas. O bit URG em 1 indica que o ponteiro presente no campo *Urgent pointer* é válido. Esse ponteiro é utilizado para indicar o deslocamento a partir do número de seqüência atual onde se encontram dados considerados urgentes. O bit ACK em 1 indica que o conteúdo do campo *Acknowledge number* é válido. O bit PSH indica que os dados devem ser repassados diretamente para a aplicação, e não armazenados esperando o *buffer* ficar cheio. O bit RST é usado para reiniciar uma conexão na qual ocorreu algum erro. Ele também é usado para rejeitar segmentos inválidos ou tentativas de abertura de conexão em portas não associadas a processos. O bit SYN é usado durante o estabelecimento de conexão. O pedido de abertura de conexão apresenta o bit SYN em 1 e todos os outros em 0; a resposta ao pedido apresenta o bit SYN em 1 e o bit ACK também em 1, já que ele reconhece o pedido prévio. O bit FIN é usado para liberar uma conexão previamente estabelecida.

Para estabelecer uma conexão, o TCP usa um mecanismo denominado *three-way handshake*. O processo de estabelecimento da conexão é mostrado na Figura II.7a. O primeiro segmento do estabelecimento da conexão pode ser identificado devido ao bit SYN em 1. O segmento de resposta apresenta tanto o bit SYN quanto o bit ACK ativos em 1, indicando que o primeiro segmento foi reconhecido e que o processo de estabelecimento da conexão foi iniciado no destino. Por fim, um último segmento somente de reconhecimento é enviado para informar que os dois lados concordam com a abertura da conexão.

Depois de estabelecida a conexão, ocorre o processo de troca de dados, sendo geralmente trocados somente pacotes de dados e reconhecimento, que apresentam o bit ACK ativados. A Figura II.7b mostra esta etapa da conexão. Por fim, um processo de fechamento de conexão acontece, usando um mecanismo semelhante ao de estabelecimento de conexão, porém ligeiramente modificado, conforme mostrado na Figura II.7c. Quando uma das partes acaba de enviar todos os seus dados, ela envia um pacote com o bit FIN ativo. O outro lado da comunicação informa a aplicação que não existem mais dados a serem recebidos e reconhece o fechamento da conexão em uma direção. Porém, como as conexões TCP são bidirecionais, ainda se pode transferir dados no sentido oposto sem problemas. Quando então o outro lado terminar de enviar os seus dados, ele repete o mesmo processo feito anteriormente e a conexão é fechada.

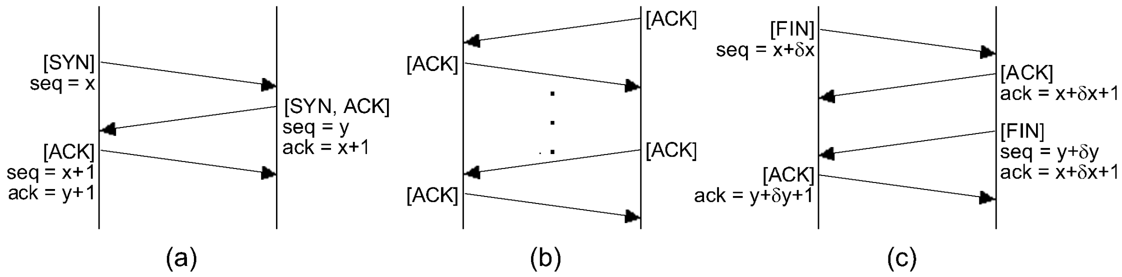

Figura II.7: A conexão TCP: (a)Estabelecimento, (b)Troca de dados, (c)Fechamento.

O campo *Window size* é o responsável pelo controle de fluxo e pelo uso eficiente do meio, como já dito anteriormente. Ele informa quantos bytes, a partir do byte reconhecido pelo campo ACK, podem ser enviados em avanço.

O campo *Checksum* é o responsável pela integridade dos dados do segmento. Assim como no UDP, para sua computação, são incluídos campos do cabeçalho IP, TCP e dados do usuário.

Após os campos fixos, podem ainda existir opções que cobrem algumas funcionalidades, assim como é feito no IP. Uma das opções mais usadas é aquela que permite especificar qual o tamanho máximo da parte de dados que cada máquina está disposta a receber.

Capítulo III

A Zona Desmilitarizada

Hoje em dia, diversas organizações já possuem uma enorme quantidade de informação armazenada digitalmente. Entretanto, nem toda informação deve estar disponível para todos conectados à rede. Com o objetivo de livrar uma rede ou uma organização inteira de comunicações indesejadas, um mecanismo de controle de acesso se torna necessário. Através dele, é possível dificultar a ação de intrusos em obter ou corromper informações, interferir nas comunicações internas ou mesmo usar as estações da rede para uso próprio. Porém, diferentemente de outros mecanismos como autenticação, adicionados aos programas da camada de aplicação, o controle de acesso à rede necessita modificações na infra-estrutura da rede [3]. Essas modificações irão levar à construção de uma nova topologia, incluindo uma rede exclusiva de servidores denominada zona desmilitarizada. A construção dessa tal zona só é possível através da existência de filtros de pacotes, que limitem de forma significativa o acesso à rede. Esse processo de filtragem é feito por dispositivos chamados *firewalls*, que serão explicados em seguida.

III.1 Os *Firewalls* e a Filtragem de Pacotes

Do inglês, *fire wall* pode ser traduzido como uma parede corta-fogo que é usada como barreira para evitar que um incêndio em um dos lados se espalhe para o outro [4]. De acordo com o significado da palavra, já se pode ter uma idéia básica do que vêm a ser os *firewalls*. Na analogia, o lado onde existe o incêndio representa uma rede não confiável, ou até mesmo hostil, enquanto que o outro lado da "barreira" representa a rede que se quer proteger. Essa parede corta-fogo é na verdade algum dispositivo ou software que tem como objetivo limitar o acesso entre uma rede considerada protegida e as outras redes. Na verdade, esse controle não é restrito somente a redes diferentes. Dependendo da camada de atuação do *firewall*, esse controle pode ser feito até mesmo dentro de uma mesma rede. Um exemplo típico deste último pode ser um *firewall* que atua na camada de interface de rede em um comutador, que é um dispositivo responsável pela interconexão de diversas máquinas em uma mesma rede. Neste caso, o papel do *firewall* é limitar cada uma das estações a se comunicar somente com o roteador, impedindo assim uma comunicação direta entre estações da rede.

Mesmo podendo existir em diversos dispositivos, como modens, comutadores ou pontos de acesso sem fio, os *firewalls* são encontrados mais freqüentemente nos roteadores. Como visto no Capítulo II, cada pacote entregue à rede vai sendo passado de roteador em roteador até alcançar o seu destino final. Logo, basicamente, a função de cada roteador é tomar uma decisão sobre como encaminhar cada pacote recebido, de forma que ele se aproxime cada vez mais do seu destino. Porém, o pacote leva consigo nada mais do que o endereço do destinatário como informação para o seu encaminhamento, ou seja, ele carrega o endereço de onde quer chegar mas não como [5]. Cabe, portanto, aos roteadores a troca das informações necessárias para que a entrega de todos os pacotes seja bem sucedida. Essa troca acontece através dos chamados protocolos de roteamento, que são os grandes responsáveis pela disseminação de rotas da rede. A partir das informações recebidas por estes protocolos, cada roteador pode então montar a sua tabela de rotas, a qual especifica como cada pacote deve ser roteado em função do seu endereço de destino. Assim, para cada pacote recebido, o roteador procura uma entrada na tabela que contenha informações sobre como alcançar o seu destino e, a partir dessas informações, o pacote pode então ser encaminhado. Enquanto no roteamento o foco principal é o modo como os pacotes são entregues, em um outro procedimento chamado filtragem de pacotes, a preocupação principal é se os pacotes devem ser encaminhados ou descartados [5].

Mais especificamente, a filtragem de pacotes é um processo de escolha seletiva de quais pacotes podem ou não ser encaminhados, de acordo com a política de segurança. O seu funcionamento é bem simples e acontece durante o processo de roteamento, quando todas as informações do pacote são acessíveis ao roteador. Essas informações podem então ser inspecionadas para verificar se atendem a determinadas regras pré-estabelecidas que espelham a política de segurança. Atendendo a essas regras, o pacote pode ser encaminhado normalmente ou, em caso contrário, descartado. A Figura III.1 mostra um exemplo de um filtro de pacotes implementado em um roteador responsável pela interconexão entre duas redes. A restrição é que somente pacotes destinados ao servidor podem passar pelo filtro, impedindo o acesso às outras estações da rede.

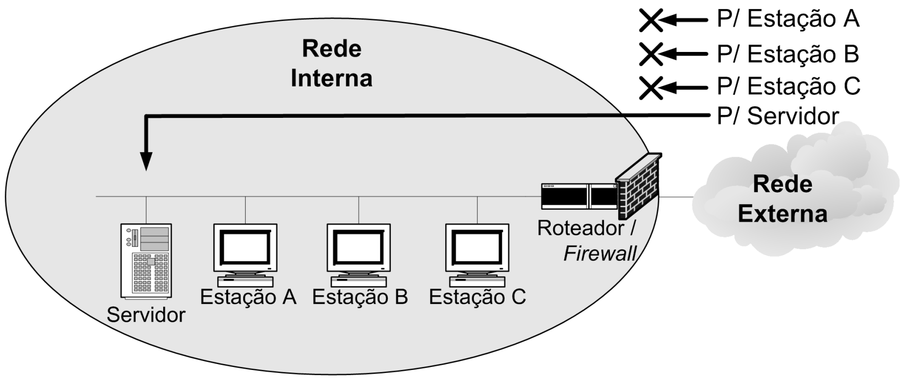

Figura III.1: Filtragem de pacotes em um roteador.

Conforme visto no Capítulo II, cada pacote contém um cabeçalho de acordo com os protocolos usados em cada camada (Figuras II.4, II.5 e II.6). Os cabeçalhos são os responsáveis por levar a informação de controle da comunicação, como endereços de rede e de transporte, sinalização para abertura e fechamento de conexão e ainda reconhecimento de pacotes recebidos. O objetivo dessas informações é permitir que uma comunicação seja realizada de forma satisfatória para ambos os lados. Sendo essas informações as responsáveis por especificar diversos atributos da comunicação, elas podem ser usadas justamente para controlar o acesso à rede, através da filtragem de pacotes. Todas essas informações são passíveis de serem usadas, porém as mais importantes são:

- os endereços IP de origem e destino, responsáveis pela identificação das estações que se comunicam. Através deles, é possível impedir o acesso a servidores da rede interna, ou até mesmo impedir que usuários internos acessem determinados servidores externos;

- as portas TCP/UDP de origem e destino, responsáveis pela identificação dos processos rodando em cada máquina. De acordo com o número da porta, é possível barrar diversos serviços, dado que alguns deles têm um protocolo e uma porta associada. Por exemplo, o HTTP (*Hypertext Transfer Protocol*), o protocolo que lida com a troca de páginas *web*, tem a porta 80 do TCP associada por padrão. Ou seja, os servidores de páginas geralmente estão associados à porta 80 do TCP e qualquer segmento a ela destinado é reencaminhado ao processo do servidor. Assim, através das portas, é possível controlar o acesso aos serviços da rede. Entretanto, estas associações não são obrigatórias e dependem da configuração de cada servidor;

- o protocolo carregado pelo IP, identificado por um dos campos do cabeçalho. Esse campo ajuda a impedir que certos protocolos trafeguem na rede. Além dos protocolos de transporte, existe uma série de outros que possuem funções específicas e que são carregados pelo IP, como o ICMP (*Internet Control Message Protocol*) e o OSPF (*Open Shortest Path First*). O primeiro é o responsável pelo envio de diversas mensagens de aviso, depuração e teste na rede, relacionadas ao IP. O segundo é um dos protocolos de roteamento mais usados atualmente. É importante deixar claro que, apesar de se localizarem acima do IP na pilha de protocolos, eles não são protocolos de transporte.

Além dessas, existem outras informações conhecidas pelos roteadores que não fazem parte do cabeçalho dos pacotes e que também podem ser usadas durante o processo de filtragem. Um exemplo é a interface por onde o roteador recebeu o pacote e a interface por onde o pacote será retransmitido (interfaces de entrada e saída, respectivamente) [5]. Baseado nessas informações e nas suas configurações, o *firewall* sabe como lidar com cada pacote. A opção escolhida, entretanto, dependerá exclusivamente da política de segurança, aplicada na forma de regras. Algumas ações que podem ser tomadas são:

- aceitar o pacote, reencaminhado-o normalmente;
- descartar o pacote, sem nenhuma notificação ao remetente;
- rejeitar o pacote, avisando o remetente que aquele pacote não foi encaminhado;
- registrar a passagem do pacote pelo roteador, armazenando dados do seu cabeçalho que podem ser usados para depuração e testes;
- alertar alguém sobre aquele pacote;
- alterar alguns campos do pacote; necessário, por exemplo, quando se quer prover Qualidade de Serviço (*Quality of Service* - QoS) em uma rede. Ou ainda quando se quer redirecionar pacotes a máquinas diferentes daquelas inicialmente destinadas, inclusive com a possibilidade de alterar as portas;
- marcar o pacote; útil, por exemplo, quando usado em conjunto com alguma ferramenta de roteamento avançado, para permitir balanceamento de carga. Desse modo, pacotes para o mesmo destino poderiam seguir rotas diferentes, aliviando enlaces sobrecarregados;
- modificar as regras de filtragem, fazendo uma auto-configuração de acordo com os pacotes recebidos.

Evidentemente, a filtragem de pacotes também apresenta suas vantagens e desvantagens. Uma grande vantagem é o fato de que a filtragem de pacotes básica consegue ser extremamente eficiente. Ela apenas requer a comparação de alguns dados dos cabeçalhos para conseguir assegurar toda uma rede de forma satisfatória. Esta eficiência é comprovada, por exemplo, quando um serviço qualquer da rede precisa ser desabilitado. Sem os filtros de pacotes, é necessário remover o serviço de todos os servidores da rede e ainda se preocupar sempre com a possibilidade dele ser reabilitado por algum usuário ou por uma eventual reinstalação. Caso exista um filtro no roteador, qualquer tentativa de acesso ao serviço é barrada, independente dele estar ativo ou não. Além disso, a configuração de um roteador com filtro de pacotes pode proteger toda uma rede contra acessos indevidos e diversas tentativas de ataques, especialmente se a interconexão da rede for feita por este roteador. Uma outra vantagem que pode ser citada é o fato de que, hoje em dia, essa tecnologia já está amplamente difundida, tendo sido implementada sob a forma de diversas ferramentas e em diversos sistemas operacionais [7, 8, 9, 10].

Como desvantagem, pode-se dizer que a filtragem de pacotes diminui um pouco o desempenho do roteador, dado que exige processamento. Sem o filtro, o trabalho do roteador se resume a decidir como melhor encaminhar cada pacote. Agora, além dessa tarefa, é necessário fazer algumas verificações em cada pacote para ter certeza de que ele atende a política de segurança especificada. De qualquer maneira, é mais seguro estabelecer um controle de acesso apropriado à rede em detrimento de um maior processamento no roteador. Entretanto, quanto mais complexo for o filtro, mais processamento será necessário. Por isso, a forma como as regras de configuração são montadas é extremamente importante, influenciando diretamente o desempenho do roteador. Quanto menos verificações forem feitas para cada pacote, menor é o processamento e melhor é o desempenho. Uma outra desvantagem é que os filtros de pacotes não oferecem proteção contra ataques originados internamente. O roteador com o filtro se limita a analisar somente os pacotes por ele roteados. Já que comunicações entre máquinas da rede interna não passam pelo roteador, elas não são verificadas. Para resolver esse problema, existem os chamados Sistemas de Detecção de Intrusão (*Intrusion Detection Systems* - IDS) [11]. O intuito desses sistemas é monitorar todos os pacotes que passam na rede, procurando por algum conteúdo suspeito. Quando detectada uma intrusão, uma notificação é enviada para o administrador da rede, avisando-o sobre o ocorrido. Uma outra possível solução é o uso de *firewalls* distribuídos [12]. Neste caso, cada estação da rede possui seu próprio *firewall* e todas seguem a política de segurança ditada por uma máquina central. Além disso, existem políticas que não podem ser implementadas com os filtros de pacotes. Por exemplo, é possível controlar o acesso de endereços IP, mas não de máquinas e muito menos de usuários. Uma máquina pode ter mais de um endereço IP e a informação relacionada com o usuário não está contida no pacote. Outro exemplo é que só é possível controlar o acesso a portas, não o acesso a aplicações. Apesar de cada porta estar associada a um serviço provido por um certo protocolo, essa associação não é obrigatória e nem sempre é verdadeira. Um servidor SSH poderia facilmente atender a requisições enviadas para a porta 80, desde que devidamente configurado para isso.

III.1.1 Filtragem de Pacotes Básica ou sem Estado

Existem duas formas conhecidas de se filtrar pacotes. A primeira delas é a chamada "sem estado" porque, assim como a técnica do datagrama, esse tipo de filtragem trata cada pacote independentemente dos demais. Ou seja, nenhum histórico ou passado é conhecido sobre outros pacotes relacionados àquele analisado e, portanto, todos os pacotes anteriores não são levados em consideração durante a filtragem. A verificação se limita às informações contidas em cada pacote.

Usando-se este tipo de filtragem é possível bloquear ou liberar portas TCP/UDP usadas por determinados serviços ou ainda impedir que endereços IP específicos tenham acesso à rede, dentre outras ações. Uma política de segurança que permite somente o acesso remoto a estações da rede através do SSH (*Secure Shell*), um protocolo que transmite dados criptografados, pode ser facilmente implementada com um filtro de pacotes sem estado. É necessário verificar se cada pacote proveniente da rede externa está endereçado à porta de destino 22 do TCP. A porta 22 é aquela associada com o protocolo SSH, por padrão. Um outro exemplo mais comum é liberar um conjunto de serviços em um mesmo servidor, barrando todos os outros. Para isso, para cada pacote destinado ao servidor, são feitas uma série de comparações da porta de destino de cada pacote com as portas autorizadas. Caso a porta do pacote seja igual a alguma das portas autorizadas, o pacote é aceito e o encaminhamento é permitido. Em caso contrário, o pacote é descartado. A Figura III.2 ilustra um processo de filtragem de pacotes em um roteador, permitindo a passagem de pacotes destinados ao serviço de acesso remoto SSH e ao serviço de páginas *web* através do HTTP.

É importante lembrar que a filtragem de endereços IP é feita baseando-se no conteúdo dos campos de endereço do cabeçalho. Entretanto, dado que a camada de rede é baseada no uso de datagramas, o endereço de origem pode ser forjado sem dificuldades. Como

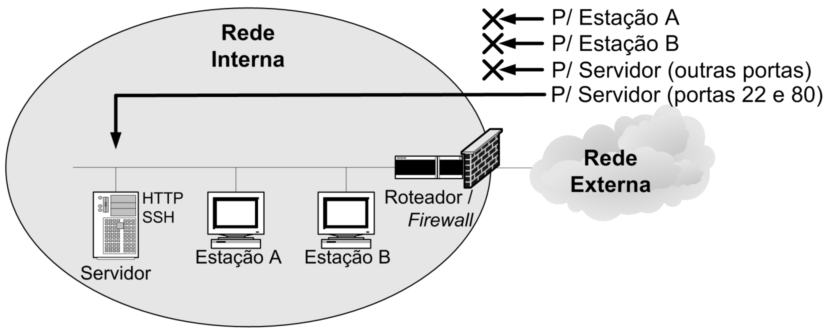

Figura III.2: Filtragem de pacotes sem estados.

somente o endereço de destino é analisado durante o roteamento, pacotes com endereço de origem falsificado geralmente alcançam seus destinos. Esses tipos de pacote são muito usados quando não existe a necessidade de uma comunicação bidirecional, como é o caso dos ataques de negação de serviço (*Denial of Service* - DoS) [6]. Esses ataques têm o objetivo de enviar tantos pacotes para uma máquina, um enlace ou uma rede, de tal forma que serviços antes providos tornem-se indisponíveis. Portanto, restringir o acesso baseando-se no endereço de origem do pacote não é uma tática eficaz. Para resolver este problema, já existe uma proposta interessante que possibilita o rastreamento da origem de cada pacote IP, através do armazenamento de informações em cada roteador atravessado [13, 14].

Ainda que a instalação de um filtro de pacotes básico já aprimore em muito a segurança, ele ainda possui algumas limitações. Primeiramente, a filtragem se restringe basicamente a informações dos cabeçalhos das camadas de rede e transporte, não analisando o protocolo usado na aplicação, como já foi dito anteriormente. Essa falta de verificação pode levar a falsas impressões de segurança, dado que através de uma porta liberada pode estar trafegando, por exemplo, um protocolo considerado proibido. Além disso, esse tipo de filtragem não considera outros pacotes senão o analisado durante a verificação, ou seja, não considera o estado dos pacotes. Com a noção de estado, uma filtragem que use as informações de um fluxo de pacotes pode ser implementada, como explicado logo a seguir.

III.1.2 Filtragem de Pacotes Dinâmica ou com Estado

Com esse tipo de filtragem, surge o conceito de estado. Uma das possíveis definições do estado de uma comunicação pode ser a situação na qual esta se encontra no momento. Durante uma troca de informações, existem alguns estados possíveis. Um exemplo bastante simples é um protocolo através do qual um cliente envia uma mensagem requisitando alguma informação a um certo servidor, e este, por sua vez, responde com os dados pedidos. Logo após o envio da requisição, o cliente entra em um estado de espera de resposta e fica neste estado até que ela seja recebida, ou até que o tempo máximo de resposta seja excedido. Por outro lado, logo antes de receber a requisição, o servidor está em um estado de espera de requisições, estado para o qual ele volta após responder ao cliente.

A filtragem dinâmica ou com estado possibilita uma discriminação dos pacotes de acordo com o estado da comunicação de cada um deles. Com ela, agora é possível, por exemplo, liberar a entrada de pacotes TCP na rede somente quando estes pertencerem a uma conexão previamente estabelecida, ou ainda, permitir a entrada de pacotes ICMP que sejam respostas de solicitações internas ou que estejam relacionados a alguma conexão já existente. Seu funcionamento é bem parecido com o de um filtro de pacotes básico, porém a sua grande vantagem é permitir o uso dos estados durante a filtragem. Para isso, o roteador segue o rastro deixado pelos pacotes que passam por ele e armazena os estados do fluxo em uma tabela, chamada tabela de estados. A Figura III.3 mostra o modo como esta tabela é montada à medida que os pacotes vão atravessando o roteador. Essa tabela é consultada posteriormente para determinar o destino de cada pacote individualmente. Assim, este tipo de filtragem é chamada com estado porque sua decisão é baseada na situação momentânea da comunicação, indicada pelo seu estado; e é chamada dinâmica porque as regras de filtragem não são mais estáticas, elas mudam em função dos pacotes roteados.

A mais óbvia vantagem desse novo tipo de filtragem é impedir que pacotes que atendam a determinados requisitos no cabeçalho penetrem na rede, como acontece na filtragem de pacotes sem estados. Nessa filtragem, o modo usado para barrar conexões TCP não

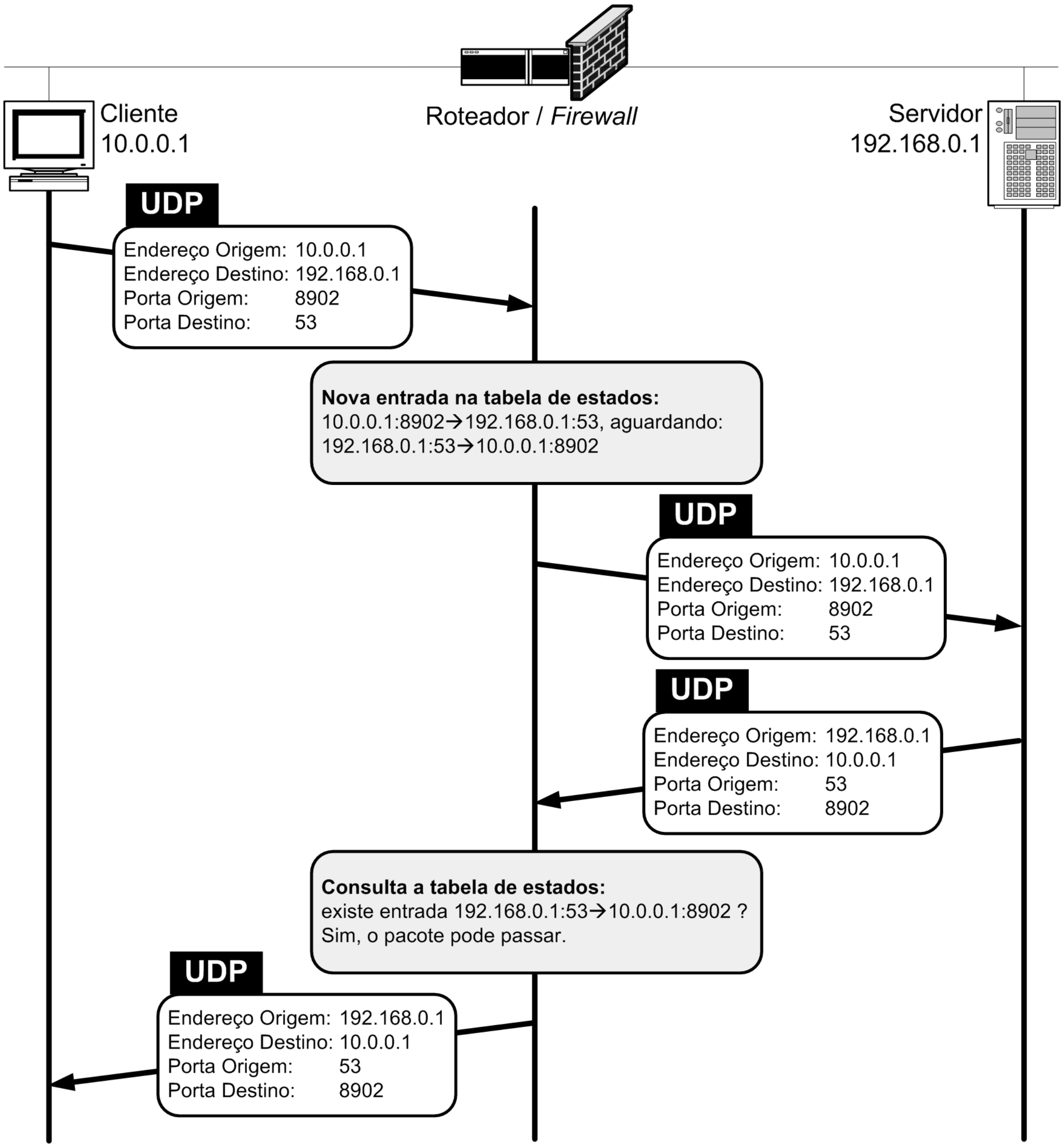

Figura III.3: Montagem e consulta da tabela de estados.

desejadas é descartar o primeiro pacote do estabelecimento da conexão, o pacote com somente o bit SYN ativado [15]. Assim, qualquer pacote com este bit ativado que não seja destinado para os serviços explicitamente liberados é rejeitado. Porém, pacotes com os outros bits do TCP ativados precisam ter sua entrada liberada porque podem pertencer a conexões previamente iniciadas por usuários da rede interna. Devido a isso, pacotes que apresentem outros bits ativados que não o SYN podem entrar sem dificuldade alguma na rede. Dessa forma, ataques como o de negação de serviço (DoS) e varredura de portas que usem esses tipos de pacotes podem ser facilmente realizados. A varredura de portas é usada para descobrir quais portas em um servidor estão esperando novos pedidos de conexão (no caso do TCP), ou seja, quais portas estão abertas. O modo mais comum de se realizar uma varredura de portas é tentar abrir uma conexão com todas as portas do servidor, de maneira normal. Entretanto, existem outras formas mais obscuras que exploram falhas do TCP/IP e de sua implementação para saber se a porta está aberta. Mais informações podem ser recolhidas em [16].

O mesmo conceito pode ser aplicado para o outro protocolo de transporte, o UDP. Entretanto, como não existe conexão no UDP, é necessário permitir a entrada pacotes UDP com determinadas portas de origem para que serviços transportados pelo UDP sejam acessíveis aos usuários da rede interna. Como exemplo, usando-se a filtragem de pacotes básica, geralmente é permitida a entrada de pacotes UDP com a porta de origem 53, já que esta porta é a usada pelo serviço de resolução de nomes, o DNS. Caso esta regra não exista, as respostas das requisições feitas pelos usuários internos são barradas ao tentar entrar na rede. Por outro lado, ela abre as mesmas vulnerabilidades na segurança da rede presentes no caso do TCP. A Figura III.4 ilustra o problema dos filtros de pacotes sem o conceito de estado.

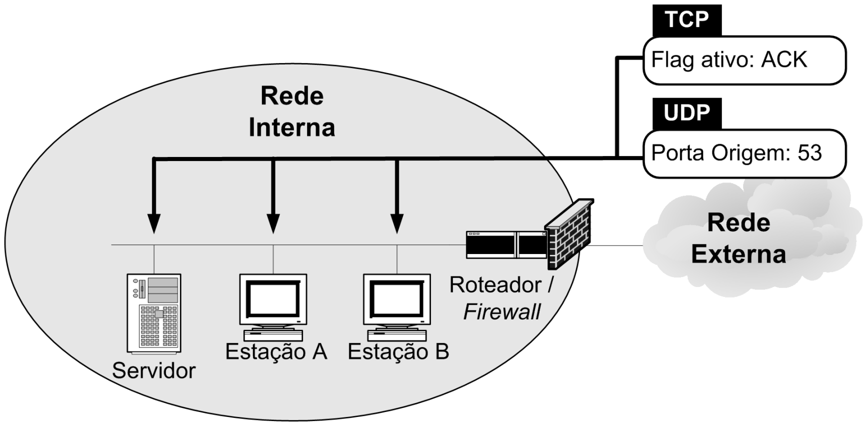

Figura III.4: Problema dos filtros de pacotes sem estado.

Uma outra vantagem dos filtros com estado é que alguns deles também interpretam alguns protocolos de aplicação com o objetivo de alterar o estado de alguns fluxos. O protocolo mais comum que necessita desta interpretação é o FTP (*File Transfer Protocol*). Este protocolo usa uma conexão para o envio de informação de controle e outra para o envio de dados. Porém, a porta usada para a transmissão de dados é negociada anteriormente através do canal de controle. Interpretando essa informação, o filtro libera o pedido de conexão para a máquina em questão na porta especificada. Com o filtro de pacotes sem estado, para se resolver o problema, o que se faz é aceitar todos os pedidos de conexões acima da porta 1023 (portas não-privilegiadas).

Ainda que apresente novas possibilidades para a melhora da segurança, os filtros de pacotes com estado também apresentam desvantagens em relação aos filtros básicos. A mais evidente é a sua maior complexidade. Enquanto na filtragem básica toda informação necessária para a verificação está no pacote, na filtragem com estados, uma tabela para controlar o estado da comunicação de cada pacote passado precisa ser montada e gerenciada. Ainda é preciso que as entradas desta tabela sejam temporizadas, para que sejam removidas caso pacotes daqueles fluxos não atravessem mais o roteador. Além de aumentar ainda mais o processamento, se mal implementada, essa filtragem ainda pode deixá-lo vulnerável a ataques de negação de serviço. Além disso, caso o roteador seja reiniciado, sua tabela de estados é apagada implicando no descarte de pacotes que na verdade deveriam ser aceitos [5].

Outro grande problema dos filtros com inspeção de estados existe caso alguma parte do fluxo, tanto de ida quanto de volta, não passe pelo mesmo filtro. Isto acontece, por exemplo, quando dois roteadores interconectam um mesmo domínio a outras redes, como mostra a Figura III.5. Se as rotas da comunicação são assimétricas, há um problema com pacotes de ida e volta passando por roteadores distintos. No caso, o pacote inicial é enviado sem problemas, criando uma entrada na tabela de estados do roteador passado. Porém, o pacote de retorno é barrado, já que ele passa pelo filtro cuja tabela não possui uma entrada para aquele fluxo. Uma possível solução é forçar cada roteador a compartilhar sua tabela de estados com os outros, o que também pode se tornar complexo [4].

Enquanto alguns autores se limitam a comentar sobre a filtragem de pacotes [2, 3, 4],

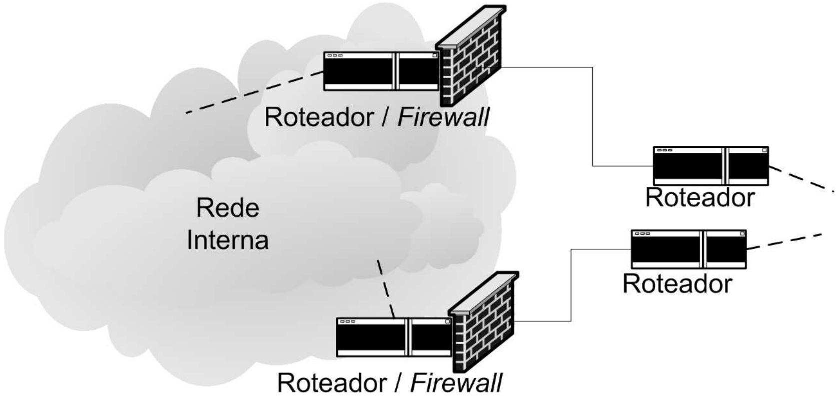

Figura III.5: Rede interconectada por mais de um roteador.

outros defendem que mais algumas tecnologias também estão associadas aos *firewalls* [5, 6]. Estas tecnologias são a Tradução de Endereços de Rede (*Network Address Translation* - NAT), Serviços de Procura (*Proxy Services*), e ainda as Redes Privadas Virtuais (*Virtual Private Networks* - VPN). Dado que nenhuma dessas outras tecnologias é usada neste projeto, elas não são, portanto, aqui detalhadas.

III.2 Topologias de Rede

Uma vez tendo definido os conceitos de filtragem de pacotes, agora é necessário determinar uma topologia de rede de forma que seja possível ao mesmo tempo oferecer serviços publicamente e ainda proteger a rede interna contra ataques. Algumas topologias são capazes de cumprir essas tarefas, cada uma à sua maneira. Entretanto, a adoção de uma em detrimento de outra depende do nível de segurança que se quer (e que se pode) alcançar.

III.2.1 Única Rede

A topologia mais simples que usa um *firewall* é aquela em que existem uma única rede e um filtro de pacotes integrado em um roteador. Uma das vantagens desta topologia é a existência de um único ponto de controle onde todos os esforços precisam ser concentrados.

Além disso, ela é a mais barata, mais simples de se entender e gerenciar [5]. Por outro lado, a sua desvantagem é que toda a segurança da rede depende de um só dispositivo, existindo um ponto de falha único na rede.

A topologia usando-se uma única rede é aquela que vem sendo usada na maioria dos exemplos e se encontra na Figura III.6. Geralmente, o filtro de pacotes é integrado com o próprio roteador. Como pelo menos um roteador já era necessário anteriormente para realizar a interconexão da rede, esta solução é de baixo custo sendo necessária apenas uma atualização de *software* para possibilitar o processo de filtragem. O filtro de pacotes é configurado de tal maneira que somente o servidor pode ser acessado por sistemas externos. Porém, ainda existem restrições quanto a este acesso, sendo que apenas determinados serviços (portas) estão liberados. Ao barrar no filtro um serviço que esteja funcionando no servidor, é possível disponibilizar serviços apenas para usuários da rede interna. Além disso, para acessar qualquer outra estação na rede interna, é necessário antes se conectar ao servidor e a partir dele realizar o outro acesso.

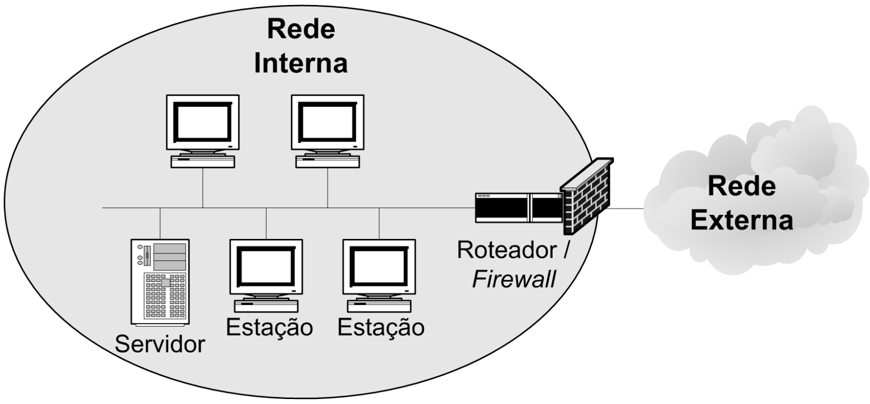

Figura III.6: Topologia usando-se uma única rede.

O grande problema com topologias que utilizam uma única rede é que ela não proporciona diferentes níveis de segurança. Caso se queira oferecer algum serviço a partir da rede interna, é necessário liberar no *firewall* o acesso àquela porta específica do servidor. Ao permitir o acesso externo ao servidor, vulnerabilidades existentes na implementação da pilha de protocolos TCP/IP ou na própria aplicação podem ser um meio para o seu comprometimento. A pior desvantagem disso, além da revelação dos dados presentes neste servidor, é que não existe nenhuma outra proteção que impeça o invasor de atacar todos os outros computadores da rede. Não há como o *firewall* ter conhecimento de que pacotes destinados à aplicação estão sendo usados com outras finalidades, como para a exploração de uma vulnerabilidade desta própria aplicação, porque estes protocolos não são geralmente checados. Mesmo os filtros de pacotes com estado não chegam a checar os protocolos usado nas camadas superiores. Geralmente, eles se limitam a analisar os dados da camada de interface de rede até a camada de transporte, salvo alguns protocolos conforme explicado na Seção III.1.2. Assim, um invasor que explorasse alguma vulnerabilidade de uma aplicação do servidor teria seus pacotes liberados pelo filtro, desde que destinados àquela aplicação. Desse modo, é possível por exemplo que um invasor tenha sucesso ao explorar uma vulnerabilidade de um serviço, conseguindo executar comandos remotamente no servidor, sem que o *firewall* tenha como barrá-lo. A partir daí, nada o impede de tentar burlar a segurança de todas as outras máquinas da rede. Ele pode atacá-las sem nenhuma dificuldade dado que este tráfego não é roteado e, portanto, não é filtrado.

Além do servidor, o roteador também é uma peça-chave na segurança da rede e precisa apresentar uma forte segurança. Se invadido, o acesso às outras máquinas pode ser feito sem qualquer restrição.

III.2.2 A Zona Desmilitarizada

"Em termos militares, uma zona desmilitarizada (ZDM) é uma área, geralmente a divisa ou fronteira entre dois ou mais grupos, onde atividade militar não é permitida, geralmente por algum tratado ou acordo. (...) A zona desmilitarizada da Coréia é uma faixa de terra que atravessa a Península da Coréia, servindo com uma área de divisão entre as Coréias do Norte e Sul. (...) Devido à situação, um grande número de tropas ainda está situada ao longo de ambos os lados da zona, cada lado prevenindo-se contra o que aparentar ser uma agressão potencial feita pelo outro." [17] Em redes de computadores, uma zona desmilitarizada é uma rede situada entre a rede interna de uma organização e uma rede externa, de tal forma que nela estejam todos os servidores que necessitam ser acessados externamente, como pode ser visualizado na Figura III.7. Analogamente ao termo militar, a zona desmilitarizada fica situada exatamente no meio de duas redes que, se estivessem colocadas em contato direto, poderiam causar resultados não muito agradáveis, conforme explicado na seção anterior. Seu objetivo é fornecer um nível extra de segurança na rede, de tal forma que a invasão dos servidores dentro da zona desmilitarizada ou do roteador externo não afete em nada a segurança dos sistemas da rede interna.

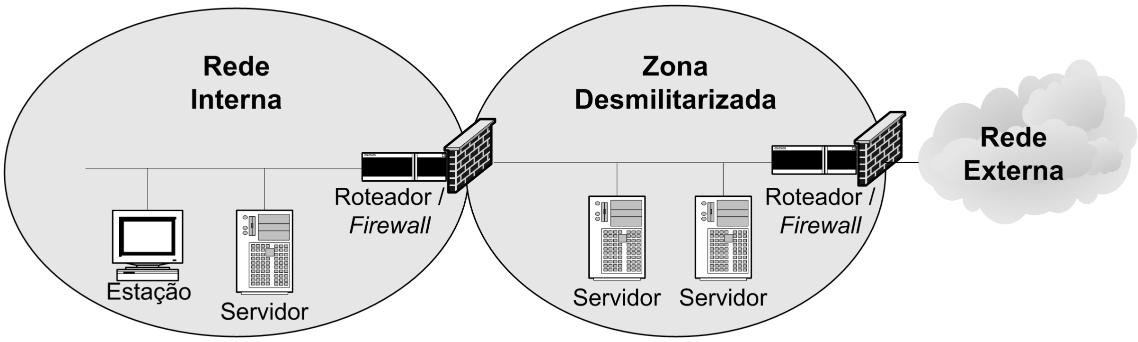

Figura III.7: Topologia usando-se uma zona desmilitarizada.

A principal motivação para a sua construção é exatamente a grande desvantagem que existe na topologia anterior. Naquele caso, existindo uma invasão no servidor, toda a segurança da rede interna está comprometida devido à inexistência de um filtro de pacotes entre o servidor e as outras máquinas da rede. Agora, estes servidores acessados por sistemas externos estão situados fora da rede interna, em uma outra rede, ou seja, na zona desmilitarizada. Logo, qualquer tráfego entre os servidores e as outras máquinas da rede interna necessariamente passa por um roteador. Este roteador, equipado com um filtro de pacotes adequado, previne que fluxos originados nos servidores entrem na rede interna. Assim, sendo os servidores comprometidos, eles não colocam mais em risco a segurança das estações de trabalho da rede interna.

A idéia principal é assumir que os servidores são um risco potencial para a segurança da rede e, portanto, não devem ser considerados confiáveis. Logo, a zona desmilitarizada não é considerada uma rede confiável e se pode considerar que toda máquina colocada nesta rede está correndo risco de ser invadida. Mas, como existem serviços que necessitam ser acessados externamente, é melhor que estes servidores não fiquem dentro da rede interna. "A razão pela qual isso (colocar servidores na zona desmilitarizada, mesmo sabendo que é arriscado) é um compromisso aceitável é porque é melhor ter uma máquina invadida na ZDM do que uma invadida na rede interna." [18] Como grande parte das corporações possuem informações que não devem ser expostas abertamente, uma máquina da rede interna invadida pode vir a trazer maiores prejuízos do que uma na zona desmilitarizada.

É importante frisar que neste caso a segurança oferecida aos servidores muda um pouco do caso quando tínhamos uma única rede. Uma vantagem oferecida com a introdução de uma zona desmilitarizada é que, se os servidores que não têm necessidade de serem acessados externamente forem mantidos dentro da rede interna, a segurança deles está garantida contra invasões externas. De acordo com a estratégia da ZDM, os únicos que podem acessar estes servidores são usuários da rede interna, sendo bloqueada pelo *firewall* qualquer tentativa de acesso originada tanto na rede externa quanto na ZDM.

Outra vantagem que esta topologia possui em relação à única rede é a inexistência de um ponto único de falha que comprometa a segurança da rede interna. Agora existem dois roteadores na topologia, sendo um responsável por interconectar a zona desmilitarizada com a rede externa e o outro responsável por interconectar a rede interna com a zona desmilitarizada. Logo, para conseguir acesso à rede interna, é necessário que os dois roteadores sejam comprometidos (isso se o acesso da rede externa ao roteador interno for barrado pelo roteador externo, o que é fortemente recomendável). Mesmo que o servidor situado na zona desmilitarizada seja invadido, é também necessário passar pelo roteador interno para obter acesso à rede interna.

O papel de cada um dos roteadores, entretanto, é diferente. O roteador mais externo precisa fazer a filtragem de todo pacote se comunicando com a zona desmilitarizada. É fundamental que exista neste roteador um filtro limitando o acesso somente às portas específicas dos servidores. Caso contrário, eles estão ainda mais expostos a ataques originados por sistemas externos. Entretanto, vale a pena ressaltar aqui que a segurança fornecida aos servidores situados na zona desmilitarizada é exatamente a mesma que existe na topologia anterior. A principal vantagem de se usar uma ZDM é a melhor segurança provida a máquinas e servidores internos, ou seja, aqueles situados fora da ZDM. Além disso, é função do roteador externo impedir que pacotes entrando na rede com endereço de origem forjados, se dizendo vir da rede interna, sejam encaminhados. A filtragem de pacotes saindo da rede com endereços de origem forjados também pode ser feita nesse roteador. Ainda que não apresentem riscos à segurança interna, estes últimos podem estar sendo usados em ataques contra terceiros.

O objetivo principal do roteador interno é proteger a rede interna tanto da rede externa quanto da zona desmilitarizada. Sua principal função é impedir que qualquer tentativa de comunicação originada por estas duas redes com máquinas da rede interna seja iniciada. Devido a isso, se um filtro de pacotes com inspeção de estados for usado, a filtragem é muito simples. Basta barrar toda comunicação que não tiver sido iniciada previamente por usuários da rede interna. Além disso, o roteador também pode ser configurado de maneira a prevenir a realização de ataques internos, ou seja, ataques ao servidor da zona desmilitarizada originados na rede interna. Neste caso, ele faria um papel semelhante ao que o roteador externo faz com a rede externa, restringindo o acesso dos usuários internos ao servidor.

Além das soluções apresentadas aqui, existem outras topologias que poderiam ser usadas de forma a prover o nível de segurança requisitado. Mais informações podem ser encontradas em [5].

Capítulo IV

A Topologia Proposta

Devido às desvantagens existentes na topologia do laboratório do Grupo de Teleinformática e Automação (GTA) da COPPE/UFRJ, uma nova topologia com uma zona desmilitarizada é atrativa. Para seu projeto, é feita antes uma análise da situação do laboratório levantando os serviços disponíveis em seus servidores, com o objetivo de saber o que precisa ser mudado. Em seguida, a nova topologia do laboratório com uma zona desmilitarizada é abordada, incluindo as mudanças necessárias na infra-estrutura da rede.

IV.1 A Topologia do GTA

A topologia do GTA é baseada em uma única rede, tendo ainda algumas máquinas que atuam tanto como servidores internos quanto externos. Ou seja, existem alguns serviços que estão disponíveis somente para usuários internos e outros que estão disponíveis para usuários externos e internos. Isso pode ser um problema à medida que, se estes servidores são comprometidos, ambos serviços internos e externos correm sérios riscos de disponibilidade e privacidade. Além disso, como já foi enfatizado anteriormente, no caso de uma invasão, não existe nenhuma proteção entre os servidores e as outras máquinas da rede. Logo, tendo penetrado em algum dos servidores, um invasor pode ter acesso amplo a toda rede. A topologia em questão se encontra na Figura IV.1, que apresenta os servidores presentes com os serviços rodando em cada um deles. A função de cada serviço é explicada logo a seguir. Também pode ser visto na figura que toda a faixa de endereços IP destinada ao laboratório, `146.164.69.0` até `146.164.69.255`, está sendo usada por esta única rede.

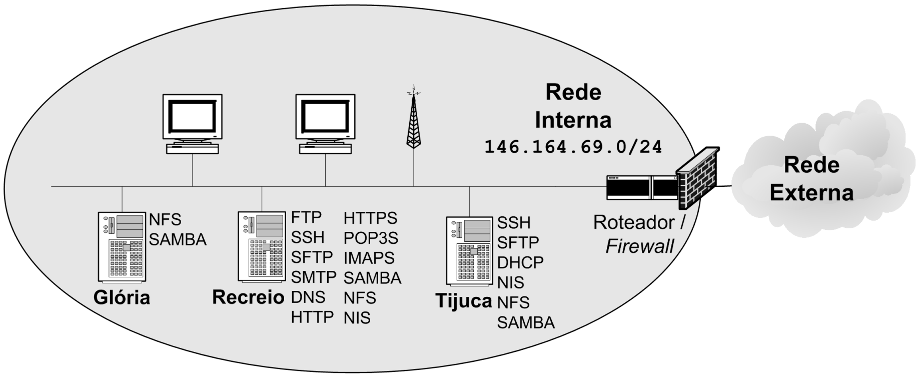

Figura IV.1: A topologia do laboratório do GTA/COPPE/UFRJ.

Uma outra falha que esta topologia apresenta é a localização inapropriada do ponto de acesso sem fio. A função de um ponto de acesso é disponibilizar conectividade a uma rede cabeada para os dispositivos móveis. Ou seja, eles são a interface entre a rede cabeada e a rede sem fio. Entretanto, a maior vantagem das redes sem fio é justamente uma de suas maiores preocupações. A mobilidade que veio com essas redes trouxe também problemas na área de segurança. O principal problema reside no fato de que o meio físico usado em redes sem fio não é enclausurado, como um fio de cobre ou uma fibra ótica, o que dificulta a violação da comunicação. Devido a isso, qualquer um que esteja dentro da área de alcance dos transmissores consegue captar os pacotes trocados no meio. Além disso, existe também o problema do controle de acesso, ou seja, como restringir que somente determinados usuários ou dispositivos acessem a rede. Por isso, mecanismos de autenticação e criptografia foram criados para assegurar a comunicação sem fio, porém a segurança dessas redes ainda é assunto de pesquisa. Portanto, colocar o ponto de acesso sem fio dentro da rede interna pode tornar vulnerável a segurança da rede.

Um dos passos importantes para reprojetar a topologia da rede é entender a função de cada serviço. A partir disso, é possível distinguir quais serviços devem ficar na rede interna e quais devem ficar na zona desmilitarizada. A idéia principal é que todo serviço que não precisa ser acessado externamente fique dentro da rede interna e, precisando-se acessá-lo de outras redes, ele deve permanecer na zona desmilitarizada. Os serviços ou protocolos que rodam nos servidores são basicamente:

- **IMAPS e POP3S**: os protocolos IMAPS e POP3S são versões seguras dos protocolos IMAP (*Internet Message Acess Protocol*) e POP3 (*Post Office Protocol*, versão 3), respectivamente. Os dois protocolos têm a função de permitir que um usuário acesse a sua caixa postal remotamente. As versões seguras trabalham com mensagens criptografadas devido à necessidade de trocar informações de autenticação pela rede. Mensagens de autenticação geralmente carregam dados sigilosos, como nome e senha do usuário, e, portanto, não devem passar na rede sem proteção.

- **SMTP e SMTPS**: o protocolo SMTP (*Single Mail Transfer Protocol*) é usado para a transferência de mensagens eletrônicas na rede. O servidor SMTP é o responsável por reenviar mensagens dos usuários para os servidores de cada destinatário da mensagem. Além disso, ele também recebe de outros servidores as mensagens destinadas aos usuários da rede e as armazena para futura visualização. O SMTPS é o protocolo SMTP protegido por criptografia.

- **HTTP e HTTPS**: o HTTP (*Hypertext Transfer Protocol*) é o protocolo que realiza a transferência de páginas *web* entre os clientes e os servidores. O HTTPS é a versão segura desse protocolo que utiliza criptografia para proteger a privacidade, autenticidade e integridade dos dados em trânsito. No laboratório, a versão segura do protocolo é usada especificamente para acesso ao *webmail*, onde a autenticação é necessária.

- **FTP**: o protocolo FTP (*File Transfer Protocol*) é usado somente para a transferência de arquivos entre máquinas. Aos poucos, seu funcionamento vem sendo substituído pelo HTTP que também é usado para transferir arquivos. No laboratório, este serviço é usado para disponibilizar a produção científica para os usuários da *Internet*.

- **SSH e SFTP**: o SSH (*Secure Shell*) é utilizado para acessar máquinas remotamente. Seu funcionamento é bem parecido com o do Telnet, porém possui a vantagem de utilizar criptografia na troca das mensagens. O SFTP (*Secure FTP*) é análogo ao FTP, mas também usa a criptografia para proteger suas mensagens.

- **DNS**: o DNS (*Domain Name System*) é um sistema de resolução de nomes em endereços IP. A razão para existir um sistema como este é a maior facilidade das pessoas em decorar nomes do que números. Sua utilidade é parecida com a de uma agenda telefônica, onde a partir do nome de uma pessoa se chega ao número, que é o necessário para se estabelecer a comunicação.

- **NIS**: o NIS (*Network Information Service*) é um serviço de banco de dados distribuído que simplifica o gerenciamento dos arquivos administrativos mais importantes, como os arquivos de contas e senhas de usuários. Ao usá-lo, é possível modificar estes arquivos em um servidor central e propagar as mudanças para o resto da rede. No GTA, o NIS é usado especificamente para distribuir os arquivos relacionados com as contas dos usuários para todas as máquinas da rede.

- **NFS**: o NFS (*Network File System*) é um sistema que possibilita o acesso de arquivos situados em um servidor central como se eles estivessem na própria máquina, sendo transparente para o usuário que na realidade ele está acessando outra máquina. No laboratório, existem dois servidores que possuem os arquivos de todos os usuários e eles "exportam" estes arquivos para toda a rede.

- **SAMBA**: o SAMBA faz em um ambiente Linux as mesmas funções que um servidor Windows faria: serviço de arquivos e impressão, autenticação e autorização, resolução de nomes e anúncio de serviços. Desse modo, é transparente para um cliente Windows qual o sistema operacional que o servidor usa.

- **DHCP**: o DHCP (*Dynamic Host Configuration Protocol*) é um protocolo que aloca endereços IP dinamicamente a máquinas conectadas a rede. Assim que uma máquina é conectada, ela pede um endereço IP para usá-lo durante o tempo que ela estiver na rede. O servidor então lhe envia um endereço, junto com algumas informações importantes, como o roteador da rede e o servidor de DNS, e então a máquina é configurada. Depois da máquina deixar a rede, aquele endereço é liberado e pode ser alocado por outra estação.

Para redes externas, somente os servidores Recreio e Tijuca estão liberados a disponibilizar serviços através do *firewall* com estados. A Recreio oferece os serviços de FTP, SSH/SFTP, SMTP/SMTPS, DNS, HTTP/HTTPS, IMAPS e POP3S enquanto que a Tijuca só oferece os serviços de SSH/SFTP. Internamente, a Recreio ainda disponibiliza serviços de NFS, NIS e SAMBA, a Tijuca disponibiliza serviços de NFS, NIS, SAMBA e ainda DHCP, e a Glória é responsável também por NFS e SAMBA.

Por mais que se use um *firewall* com inspeção de estados, o problema da falta de proteção entre servidores e estações de trabalho persiste. Com o objetivo de eliminar esse problema, uma nova topologia é proposta, como mostra a próxima seção.

IV.2 O Projeto da Zona Desmilitarizada

A nova topologia que inclui uma zona desmilitarizada se encontra na Figura IV.2. A primeira diferença visível que existe entre essa topologia e a estudada na Seção III.2.2 é que um roteador está conectado ao mesmo tempo à rede interna, à zona desmilitarizada e à rede de interconexão dos roteadores. O motivo que levou à construção de tal modo é que não existe nenhuma atualização de *software* para o roteador mais externo que o habilite a filtrar pacotes com inspeção de estado. Nele, só se consegue criar um filtro de pacotes básico, o qual apresenta todas as desvantagens já expostas no capítulo passado. Usar a rede de interconexão dos roteadores como zona desmilitarizada é arriscado, já que os servidores acessados externamente estarão bem nesta rede e eles não estariam tão bem protegidos quanto se estivessem atrás de um filtro de pacotes com estados. Essa rede tem um papel importante para o ponto de acesso sem fio, como é visto mais adiante. Entretanto, a grande desvantagem de se usar somente um roteador para a configuração da zona desmilitarizada é que é incluído um ponto de falha único na rede. Sendo o roteador comprometido através de alguma falha na implementação dos protocolos ou de aplicações, toda a segurança da rede está vulnerável [4].

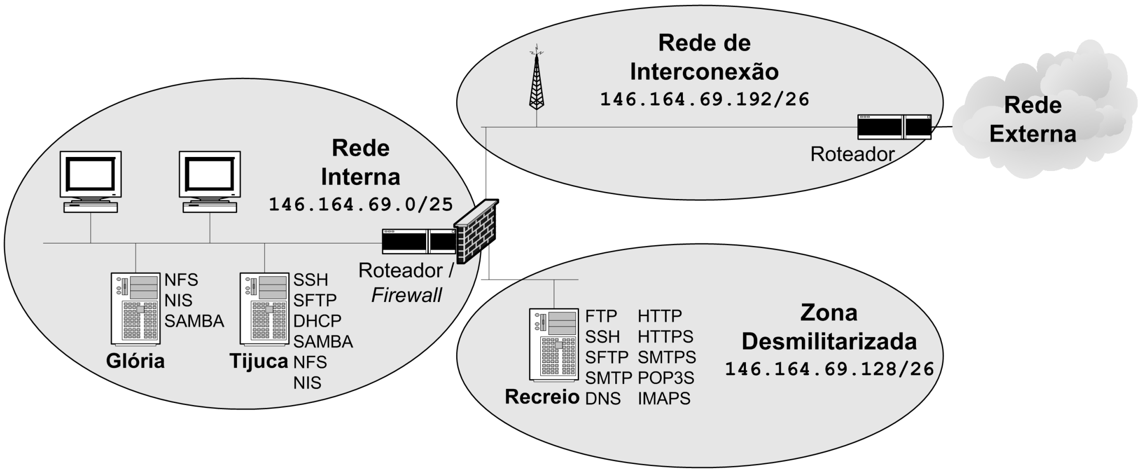

Figura IV.2: A topologia do projeto da zona desmilitarizada.

Um dos requisitos básicos para a construção de uma zona desmilitarizada é a segmentação da rede. Isso quer dizer que o que antes era somente uma rede agora será dividido em no mínimo duas. Como conseqüência, a faixa de endereçamento também precisa sofrer alterações. Como agora existem três redes (interna, ZDM e a de interconexão dos roteadores), estabeleceu-se que a faixa de endereços ficaria alocada da forma mostrada na Tabela IV.1.

Tabela IV.1: Faixa de endereço realocada para cada rede.

Rede	**Endereço IP inicial**	**Endereço IP final**
Rede interna	146.164.69.0	146.164.69.127
Zona Desmilitarizada	146.164.69.128	146.164.69.191
Rede de interconexão	146.164.69.192	146.164.69.255

Uma característica dessa nova topologia é que, agora, o ponto de acesso sem fio está situado na rede que interconecta os dois roteadores. Quando o ponto de acesso estava dentro da rede interna, existia uma falha na segurança de rede que permitia que um usuário sem fio tivesse amplo acesso à rede, inclusive aos servidores e roteadores, sem nenhuma filtragem de pacotes. Se, por outro lado, o ponto de acesso fosse colocado dentro da zona desmilitarizada, também não existiria nenhum filtro entre os usuários móveis e os servidores da ZDM, possibilitando que algum usuário mal intencionado testasse a segurança dos servidores sem restrições. Logo, a melhor escolha é colocar o ponto de acesso tanto fora da rede interna quanto fora da zona desmilitarizada, colocando-o na rede de interconexão dos roteadores. Desse modo, para acessar qualquer serviço disponível na zona desmilitarizada, é necessário passar antes pelo filtro de pacotes. Basicamente, os usuários da rede sem fio têm agora o mesmo tratamento recebido pelos usuários externos.

Os serviços foram separados de acordo com a função e a necessidade de disponibilização externa de cada um deles. Assim, serviços de HTTP e FTP necessitam permanecer na zona desmilitarizada porque o motivo da existência desses serviços é justamente disponibilizar para usuários da *Internet* informações relativas às atividades do laboratório e a sua produção científica. O HTTPS também precisa permanecer fora da rede interna porque seu papel é basicamente proteger os dados trocados pelo serviço de *webmail*. O objetivo deste último é possibilitar ao usuário checar sua caixa postal de qualquer lugar, inclusive redes externas, via *web*. Assim como o *webmail*, o papel dos protocolos IMAPS e POP3S é permitir a visualização de mensagens eletrônicas para usuários fora da rede do laboratório, daí sua necessidade de estar na zona desmilitarizada. O uso destes protocolos geralmente é feito através de um programa cliente de *e-mail*, como o *Netscape*, *Mozilla* ou *Microsoft Outlook*. Outro serviço que necessariamente precisa estar fora da rede interna é o DNS. Para consultas feitas pelos usuários internos ao serviço de nomes, ele poderia perfeitamente estar situado na rede interna, já que nessa situação ele faria o papel de cliente requisitando aos outros servidores a resolução dos nomes. Porém, quando algum usuário externo quer se comunicar com alguma máquina do laboratório usando o nome `máquina.gta.ufrj.br`, o servidor DNS precisa ser acessado. É ele que diz os endereços IPs correspondentes aos nomes das máquinas do laboratório. Com funcionamento análogo, o SMTP e o SMTPS também precisam estar fora da rede interna para receber as mensagens destinadas aos usuários do GTA vindas de outros servidores SMTP. O serviço de SSH precisa estar localizado tanto na zona desmilitarizada quanto na rede interna, como é visto mais adiante. Os serviços de disco pela rede, NFS e SAMBA, necessariamente ficam na rede interna para que as contas dos usuários sejam protegidas. Logo, não é possível que os usuários acessem seus dados a partir da ZDM, com a exceção da página *web* pessoal de cada um que precisa estar no servidor HTTP da zona desmilitarizada para que usuários externos possam visualizá-la. Além disso, também rodam internamente serviços de autenticação pela rede, NIS e SAMBA, para que todas as máquinas da rede utilizem uma máquina central para autenticação. Na ZDM, isso não é necessário devido a só existir um servidor. Ele próprio contém toda informação necessária para autenticar seus usuários.

Muitos usuários do laboratório armazenam alguns de seus arquivos em suas próprias máquinas para não sobrecarregar o servidor de disco. Porém, de acordo com a filosofia da zona desmilitarizada, as estações de trabalho da rede interna não são acessíveis tanto da rede externa quanto da própria ZDM. Logo, o uso de tal filosofia poderia causar uma grande limitação para os usuários do laboratório que quisessem trabalhar de casa. É então adotada a solução dada pelo termo "túnel coreano", introduzido aqui pela primeira vez. A adoção de tal termo decorre dos túneis cavados abaixo da zona desmilitarizada da Coréia: "Alguns túneis foram descobertos conectando a Coréia do Norte à Coréia do Sul por baixo da zona desmilitarizada, os quais autoridades do sul alegaram ser caminhos para esconder acessos feitos do norte para o sul." [17] Ele é aqui empregado para representar uma forma de acessar computadores da rede interna, mesmo que "ilegalmente", a partir da rede externa.

Um túnel coreano para um servidor de SSH da rede interna é usado para eliminar com a limitação de que usuários conectando-se de redes externas não podem acessar as máquinas internas. O túnel se estabelece da seguinte forma: primeiramente, usuários externos necessitam acessar via SSH o servidor da zona desmilitarizada, se autenticando corretamente, e, em seguida, eles podem tentar o acesso ao servidor interno, como mostra a Figura IV.3. Porém, é importante frisar que somente o acesso partindo do servidor da zona desmilitarizada com destino ao servidor da rede interna via SSH é permitido no filtro de pacotes do roteador. Tentativas de acesso ao servidor de SSH interno diretamente da rede externa são barradas.

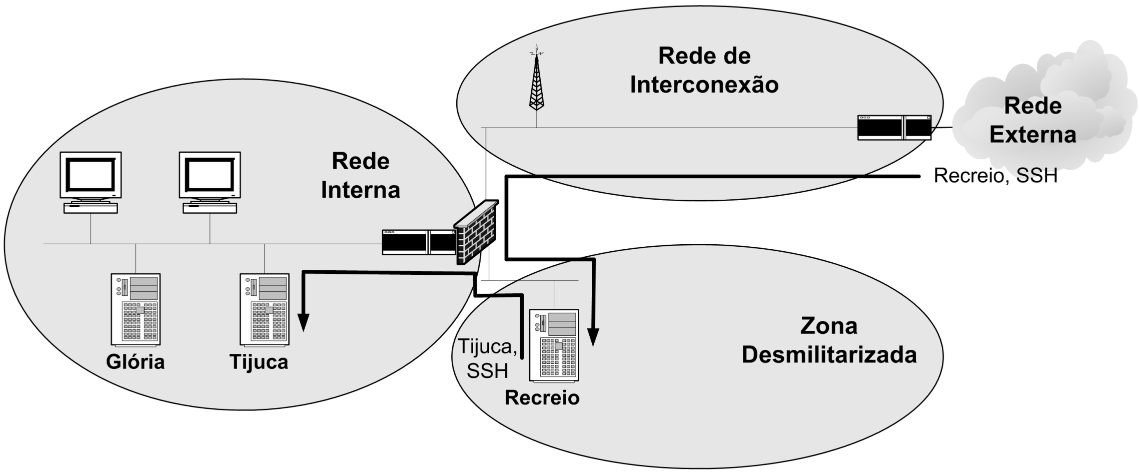

Figura IV.3: Funcionamento do túnel coreano.

Apesar do túnel coreano resolver algumas limitações, ele também insere algumas falhas na segurança da rede. Primeiro, ele não respeita a filosofia da zona desmilitarizada ao permitir que, mesmo de maneira restrita, servidores internos sejam acessíveis para usuários da rede externa. Segundo, ele abre a rede interna para que vulnerabilidades no serviço de SSH sejam exploradas, podendo comprometer a rede inteira. Existindo alguma vulnerabilidade no serviço de SSH, um invasor poderia explorar primeiro o servidor da zona desmilitarizada e em seguida repetir o ataque no servidor interno com sucesso. O mesmo problema quando se usa uma única rede persiste aqui ao se colocar servidores acessíveis externamente dentro da rede interna. A vantagem é que esse problema agora está limitado somente ao serviço de SSH, enquanto que, quando uma única rede é usada, uma vulnerabilidade em qualquer um dos serviços disponibilizados pode colocar em risco toda a rede.

Além disso, uma outra medida tomada para assegurar ainda mais a rede interna é dividir os domínios de autenticação da rede interna e da zona desmilitarizada. Ou seja, a autenticação dos usuários em cada uma das redes é feita por servidores distintos. Dentro da rede interna, existe um servidor com as contas dos usuários que é responsável somente pela autenticação interna enquanto que, na zona desmilitarizada, o próprio servidor é o responsável pela autenticação dos usuários que utilizam os seus serviços. São dois bancos de contas diferentes onde os usuários têm senhas distintas em cada um deles. Assim, o comprometimento do banco de contas do servidor da zona desmilitarizada através de uma invasão afetaria pouco a segurança da rede interna. Isso acontece por dois motivos principais:

- os usuários tanto na rede interna quanto na zona desmilitarizada são obrigados a seguir a política de usar senha de no mínimo 8 caracteres, no mínimo 1 letra maiúscula, no mínimo 1 letra minúscula e no mínimo um caractere especial (ex.: !@#$%.,<>:;/?{}). Na verdade, as senhas não ficam armazenadas no servidor, o que fica é o resultado de sua passagem por uma função *hash*. Tais funções são inversíveis e têm como saída uma cadeia de bits de tamanho fixo. Desse modo, para se adivinhar a senha usada, é necessário escolher uma senha, usá-la como entrada para a mesma função *hash* e, por fim, comparar o resultado com aquele armazenado no servidor. Fazer isso repetidamente até encontrar a senha caracteriza o chamado método da força bruta. A adoção dessa política dificulta em muito a quebra das senhas por força bruta, porque tais métodos geralmente usam palavras de dicionário e combinações dos dados do usuário como senhas de teste;

- por mais que as senhas das contas dos usuários da ZDM sejam quebradas por força bruta, isso afeta pouco a segurança da rede interna, já que as senhas usadas pelo mesmo usuário nas duas redes é necessariamente diferente. Assim, mesmo que um invasor consiga quebrar a senha de um usuário na zona desmilitarizada, sua entrada via SSH na rede interna com essa mesma senha não é aceita.

Uma vez levantados todos os problemas e propondo um solução usando uma zona desmilitarizada, o capítulo seguinte descreve e analisa a implementação da proposta.

Capítulo V

Implementação

Para a implementação da zona desmilitarizada, foi preciso configurar o *firewall* e reconfigurar os servidores de tal modo que eles continuassem realizando suas funções corretamente. O filtro de pacotes foi configurado usando-se a ferramenta `iptables`, a qual já vem instalada na distribuição da Red Hat 9 do Linux. Hoje em dia, ela é uma das poucas ferramentas disponíveis livres com inspeção de estados, sendo este o motivo principal da sua escolha. Os serviços tanto da zona desmilitarizada quanto da rede interna foram configurados de acordo com o projeto. Neste capítulo, é mostrado o funcionamento do `iptables`, a implementação do filtro de pacotes e do mecanismo usado para garantir a robustez das senhas.

V.1 A Ferramenta `iptables`

O `iptables` é uma ferramenta usada para configuração do filtro de pacotes IPv4 (IP versão 4) no *kernel* do Linux. O filtro é formado por tabelas que contêm um número determinado de cadeias pré-construídas. Cada cadeia, por sua vez, é formada por uma lista de regras nas quais vários pacotes podem se encaixar. A regra contém detalhes dos dados do pacote ou do seu estado que são os responsáveis pela sua seleção ou não. Além disso, elas também especificam o que fazer com o pacote que atende aos seus critérios, o chamado "alvo" ou "salto". Ele pode ser uma determinada ação, como aceitar o pacote, descartá-lo, ou ainda, redirecioná-lo para uma outra cadeia definida pelo usuário. Assim, resumidamente, cada regra é composta por um critério de seleção e um alvo. Se um pacote não se encaixa no critério de uma regra, a próxima é então analisada. E isso vai acontecendo até que o pacote se encaixe em uma das regras passadas. Neste caso, o que se deve fazer com o pacote é especificado pelo valor do alvo [19].

V.1.1 Tabelas e Cadeias

Existem três tabelas responsáveis por exercer as funções de *firewall* e ainda de roteamento avançado no `iptables`:

- *filter*: esta tabela é a responsável pelo processo de filtragem de pacotes. A partir dela, é possível estabelecer uma grande diversidade de critérios para que os pacotes se encaixem e ainda filtrar os pacotes da maneira mais adequada. Toda ação tomada contra pacotes que não seguem a política de segurança do sítio é feita nesta tabela. Ela é o principal foco deste projeto, já que uma zona desmilitarizada depende fortemente da filtragem de pacotes.

- *nat*: esta tabela é basicamente usada para Tradução de Endereços de Rede (*Network Address Translation* - NAT). Esta tradução é feita pelo roteador para que máquinas com endereços de rede reservados possam se comunicar com todas as outras máquinas da Internet.

- *mangle*: o uso desta tabela é específico para atividades como roteamento avançado e provimento de Qualidade de Serviço (*Quality of Service* - QoS). Ela permite mudar determinados campos do cabeçalho IP, como o ToS (*Type of Service*) e o TTL (*Time To Live*), e ainda marcar os pacotes para que sejam usados por outro programa como o `iproute2`.

Cada tabela possui algumas cadeias pré-construídas que são responsáveis por uma atividade da tabela. Em função dessas atividades, as cadeias apresentam uma ordem determinada para a passagem dos pacotes, ou seja, os pacotes percorrem as cadeias em uma ordem específica. Esta ordem é mostrada na Figura V.1 enquanto que a função de cada cadeia é especificada na Tabela V.1.

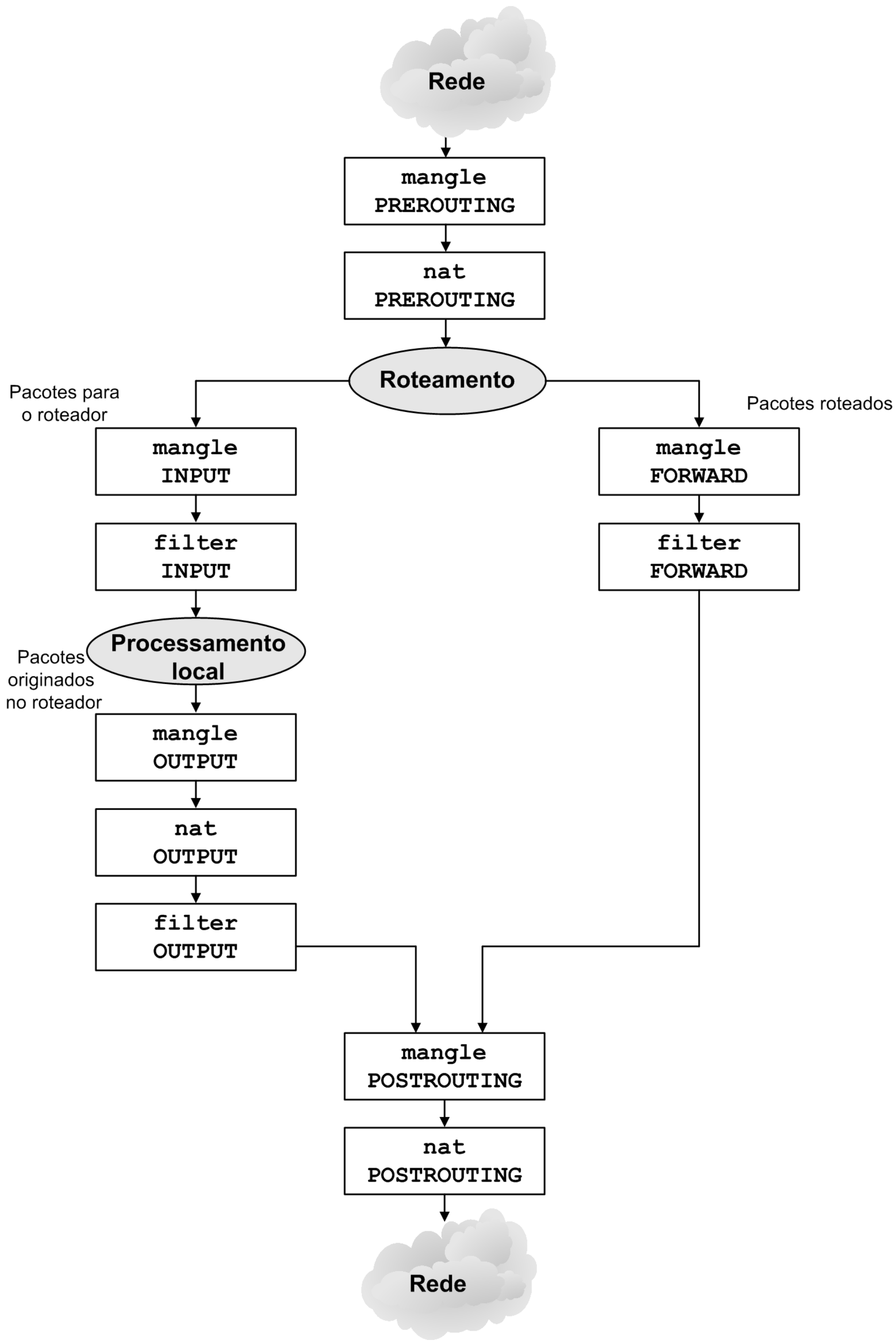

Figura V.1: Ordem que as cadeias são atravessadas (figura adaptada de [1]).

Tabela V.1: Função das cadeias pré-construídas no `iptables`.

Tabela	Cadeia	Função
filter	INPUT	Filtrar pacotes destinados ao roteador.
	OUTPUT	Filtrar pacotes originados pelo roteador.
	FORWARD	Filtrar pacotes roteados.
nat	PREROUTING	Fazer o DNAT (*Destination* NAT), ou seja, mudar o endereço de destino dos pacotes.
	POSTROUTING	Fazer o SNAT (*Source* NAT), ou seja, mudar o endereço de origem dos pacotes.
	OUTPUT	Traduzir os endereços dos pacotes originados pelo próprio roteador.
mangle	PREROUTING	Marcar pacotes ou alterar campos do seu cabeçalho antes mesmo de serem roteados.
	POSTROUTING	Marcar pacotes ou alterar campos do seu cabeçalho depois do roteamento.
	INPUT	Marcar ou alterar pacotes depois do roteamento, antes de serem entregues ao processamento local. Assim como na tabela *filter*, a cadeia INPUT só trata os pacotes destinados ao roteador.
	OUTPUT	Marcar ou alterar pacotes originados pelo próprio roteador.
	FORWARD	Marcar ou alterar pacotes roteados.

V.1.2 Máquina de Estados

Para que um filtro de pacotes com inspeção de estados seja construído, é preciso definir os possíveis estados de cada fluxo de pacotes. Na implementação do `iptables`, existem quatro estados que um fluxo pode se encontrar:

- **NEW**: estado atribuído a um pacote se ele é o primeiro daquele fluxo. Todo primeiro pacote de um fluxo se encaixa em uma regra que verifica se o estado do pacote é novo.

- **ESTABLISHED**: este estado existe quando já foi trocado tráfego nos dois sentidos. Este estado é alcançado quando um pacote é retornado em resposta a um outro previamente enviado. Até a passagem da resposta pelo roteador, o estado daquele fluxo ainda era considerado novo.

- **RELATED**: estado usado para designar um fluxo relacionado com algum outro fluxo. Um exemplo de conexão que pode ser considerada relacionada é aquela de transferência de dados do FTP (*File Transfer Protocol*). Neste protocolo, é negociado anteriormente pela conexão de controle os parâmetros da transmissão dos dados, ou seja, o envio é feito em uma outra conexão TCP. Esta é considerada relacionada com a primeira. Outro exemplo seria os pacotes de erro ICMP gerados a partir de uma outra conexão.

- **INVALID**: estado atribuído a pacotes cujo estado não pode ser identificado ou simplesmente não existe. Os pacotes de erro ICMP que não estão relacionados com nenhum outro fluxo já existente são um exemplo.

Para cada protocolo acima do IP (TCP, UDP e ICMP), os estados são tratados diferentemente. No caso do TCP, por exemplo, quando os pacotes de pedido de abertura de conexão (com somente o bit SYN ativado) passam pelo *firewall*, uma nova entrada na tabela de estados é criada e aquela conexão entra no estado NEW. Quando o pacote de resposta (com os bits SYN e ACK ativados) atravessa o roteador, aquela conexão entra no estado ESTABLISHED. Desse modo, é possível liberar no filtro a saída de todos os pacotes nos estados NEW ou ESTABLISHED e só permitir a entrada de pacotes que pertençam a conexões previamente estabelecidas (ESTABLISHED) se na rede não existir nenhum servidor para sistemas externos. A conexão permanece neste estado até que o último pacote do mecanismo de fechamento de conexão seja trocado.

No caso do UDP, um mecanismo análogo é usado. Quando um pacote UDP de um fluxo novo passa pela primeira vez no roteador, é atribuído a este fluxo o estado NEW. A medida que um pacote de resposta atravessa o filtro de pacotes, o estado muda para ESTABLISHED e, a partir daí, todo pacote trocado se encontra neste estado. Como um fluxo UDP não possui conexão, se dentro de determinado tempo nenhum pacote for trocado, sua entrada na tabela de estados é retirada.

Para o protocolo ICMP, existem algumas diferenças devido às próprias singularidades deste protocolo. Pacotes ICMP de requisição e resposta, como os pacotes de eco (*echo* *request / reply*), são um exemplo dessa diferença. Na primeira vez que um pacote ICMP de requisição passa pelo roteador, é atribuído o estado NEW para aquele novo fluxo. A resposta daquele pedido é considerada como sendo de um fluxo já estabelecido e, portanto, é considerada do estado ESTABLISHED. Depois que a resposta a um pedido prévio passa pelo *firewall*, aquela entrada é retirada da tabela já que não existem mais pacotes pertencentes àquele fluxo.

Um outro possível estado de pacotes ICMP pode ocorrer quando ele é usado para informar sobre algum possível erro na comunicação. Neste caso, quando pacotes ICMP são retornados em virtude de erros acontecidos, eles são considerados relacionados (RELATED) com o fluxo que o gerou. Na Figura V.2 são mostrados alguns exemplos do comportamento da máquina de estados com os protocolos mencionados à medida que os pacotes atravessam o roteador.

V.1.3 Construindo Regras

Conforme já foi dito anteriormente, uma regra é composta por alguns parâmetros, que permitem a seleção dos pacotes, e por um alvo. Se o pacote satisfaz o critério, o alvo é então executado pelo *kernel*. Cada regra é criada com o comando `iptables` seguindo a seguinte sintaxe:

`iptables` [-t <u>tabela</u>] <u>comando</u> [<u>critério de seleção</u>] [-j <u>alvo</u>]

A primeira parte diz a qual tabela aquela regra está se referindo, podendo ser uma das três tabelas já detalhadas: *filter*, *nat* e *mangle*. Ela não é obrigatória e, não sendo especificada, a tabela *filter* é usada. O comando indica ao programa o que fazer com a regra: inserí-la em alguma cadeia ou apagá-la, por exemplo. Os comandos mais comuns e suas funções podem ser vistos na Tabela V.2. O critério de seleção indica quais os parâmetros que um pacote deve ter para que aquele alvo seja aplicado. Exemplos desse critério podem ser endereços IP e portas de origem e/ou destino, o estado daquele pacote dentro do fluxo ou qual o protocolo utilizado. Exemplos dos critérios de seleção mais comuns estão descritos na Tabela V.3. Por fim, o alvo indica o que fazer com o pacote, ou

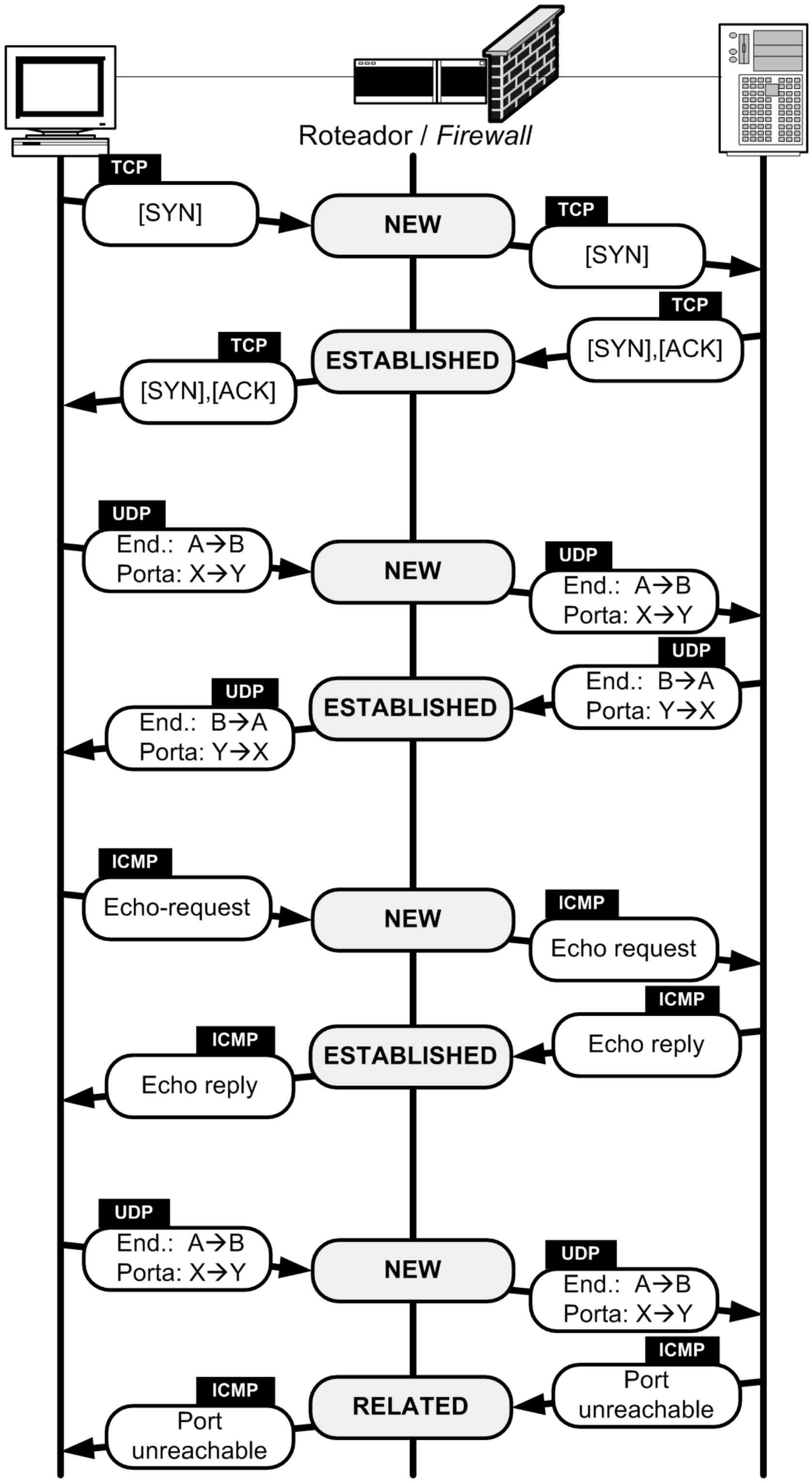

Figura V.2: Estado dos fluxos dos protocolos TCP, UDP e ICMP.

qual destino ele deve ter. O pacote pode, por exemplo, ser aceito, descartado, ou enviado para uma outra cadeia para mais verificações. A Tabela V.4 mostra alguns possíveis alvos disponíveis.

Tabela V.2: Comandos mais comuns e suas funções.

Comando		Função
-A, --append	<CADEIA>	Adicionar uma regra no final da cadeia.
-L, --list	<CADEIA>	Listar todas as regras existentes em uma cadeia.
-F, --flush	[CADEIA]	Apagar todas as regras de uma cadeia específica. Caso não seja especificada uma cadeia, todas as regras de todas as cadeias da tabela em questão são apagadas.
-Z, --zero	[CADEIA]	Zerar os contadores da cadeia especificada. Caso não seja especificada uma cadeia, os contadores de todas as cadeias da tabela em questão são apagados.
-N, --new-chain	<CADEIA>	Criar uma nova cadeia na tabela especificada.
-X, --delete-chain	[CADEIA]	Apagar a cadeia especificada da tabela. Se usado sem a especificação da cadeia, todas as cadeias definidas pelo usuário da tabela são apagadas.
-P, --policy	<CADEIA> <POLÍTICA>	Especificar um alvo padrão para aqueles pacotes que não se encaixam nos critérios de nenhuma regra. Esses pacotes têm como alvo o que for especificado como política.

Tabela V.3: Critérios de seleção mais comuns e suas funções.

Critério de Seleção	Função
`-p, --protocol <PROTOCOLO>`	Especificar o protocolo levado pelo IP. Alguns exemplos podem ser `tcp`, `udp`, `icmp`.
`-s, --src, --source <ENDEREÇO_IP>`	Especificar o endereço de origem que o pacote deve ter para se encaixar nessa regra. Pode ser usada ainda a notação `/XX` ou `/MÁSCARA` para especificar uma faixa de endereços.
`-d, --dst, --destination <ENDEREÇO_IP>`	Especificar o endereço de destino do pacote. Pode ser usada ainda a notação `/XX` ou `/MÁSCARA` para especificar uma faixa de endereços.
`-i, --in-interface <INTERFACE>`	Selecionar os pacotes baseando-se na sua interface de entrada.
`-o, --out-interface <INTERFACE>`	Selecionar os pacotes baseando-se na sua interface de saída.
`--sport, --source-port <PORTA>`	Especificar a porta de origem do pacote.
`--dport, --destination-port <PORTA>`	Especificar a porta de destino do pacote.
`--syn`	Selecionar o pacote se ele tiver o bit SYN do cabeçalho TCP ativado e os bits ACK e RST desativados.
`--icmp-type <NÚMERO_TIPO>`	Especificar o tipo de pacote ICMP. Alguns tipos são 0 (`echo-request`) e 8 (`echo-reply`).
`--limit <VEZES/UNIDADE_TEMPO>`	Limitar quantas vezes um pacote pode se encaixar em uma regra durante certo intervalo de tempo. Muito útil para não gerar arquivos de registros (*log*) muito grandes.
`--state <ESTADO>[,ESTADO]...`	Especificar o(s) possível(is) estado(s) do fluxo daquele pacote.

Tabela V.4: Alvos mais comuns e suas funções.

Alvo	Função
`ACCEPT`	Aceitar o pacote, desde que atenda aos critérios de seleção.
`DROP`	Descartar o pacote sem enviar nenhuma resposta de alerta ao emissor.
`LOG`	Registrar em um arquivo alguns detalhes do cabeçalho do pacote. Algumas opções podem ser especificadas neste alvo.
`REJECT`	Descartar o pacote e enviar uma mensagem ICMP de volta ao emissor. Além de poder enviar um pacote ICMP, uma outra opção é enviar um pacote TCP com o bit RST ativado.
`RETURN`	Fazer o pacote parar de atravessar a cadeia na qual estava a regra. Caso esta seja uma cadeia chamada previamente por outra, o pacote volta para a cadeia que inicialmente a chamou e a próxima regra dessa cadeia é analisada. Caso este alvo seja usado por uma cadeia pré-construída, o destino do pacote dependerá da política padrão definida.

V.2 Implementação

Dado que a configuração do filtro de pacotes é feita através de uma seqüência de comandos do `iptables`, um *script* na linguagem Bash (*Bourne Again Shell*) foi feito de modo a automatizar a configuração do *firewall*. O Apêndice A apresenta o código na sua forma integral. A seguir são apresentados e descritos o seu funcionamento e o modo como as cadeias do filtro de pacotes foram organizadas.

Para forçar os usuários a trocar a senha da zona desmilitarizada, foi usado um mecanismo específico. O Apêndice B apresenta o código executado na primeira vez que o usuário acessa o servidor da zona desmilitarizada remotamente. Na Seção V.2.2, este mecanismo é explicado com mais detalhes.

V.2.1 O Filtro de Pacotes

O código do filtro de pacotes é descrito no Apêndice A. O código está sob os termos da GPL (GNU *General Public License*) indicando que ele é livre para modificação e/ou redistribuição.

A Figura V.3 mostra a topologia lógica da rede do laboratório do GTA, incluindo os dois roteadores, suas redes e as respectivas faixas de endereços. A rede interna é aquela onde estão os computadores do laboratório, sendo a rede que exige maior proteção por estar armazenando todos os dados dos usuários. A zona desmilitarizada consiste de um servidor que agrupa os serviços que necessariamente precisam ser acessados externamente. Este servidor também é usado como ponto de partida para os usuários que queiram acessar a rede interna. A rede de interconexão é a que liga os dois roteadores e a que disponibiliza o acesso aos usuários móveis através do ponto de acesso sem fio.

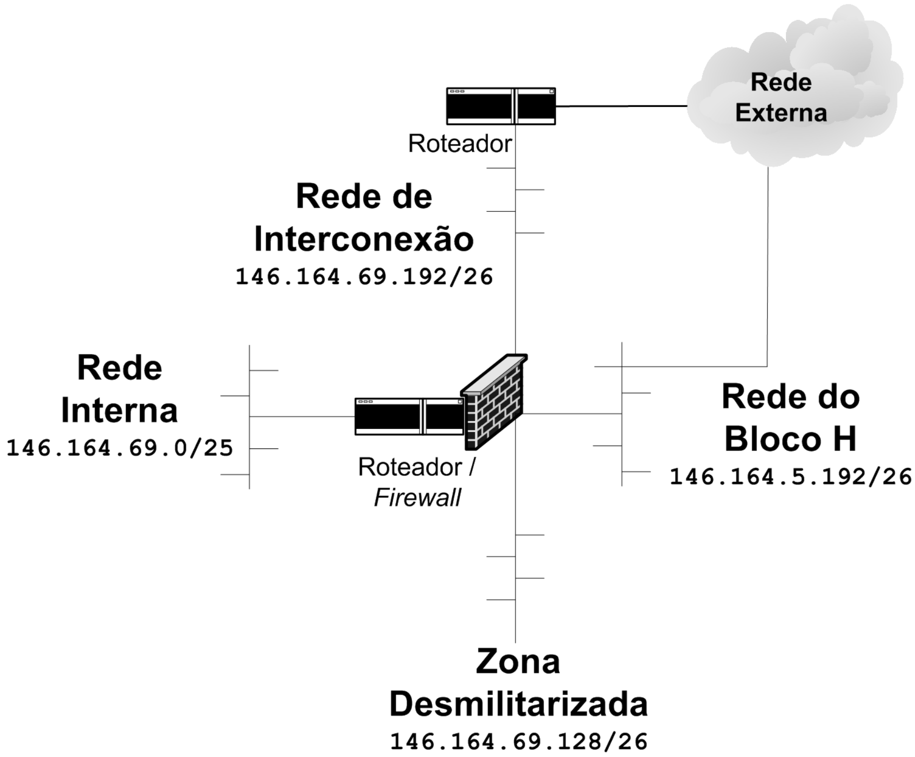

Figura V.3: A topologia lógica da rede do laboratório.

A comunicação entre todos os dispositivos destas três redes é feita por somente um comutador (*switch*). O papel do comutador é análogo ao de um roteador, com a diferença que o roteador trabalha na camada de rede enquanto o comutador trabalha na camada abaixo, a de interface de rede. No cabeçalho do protocolo da camada interface de rede, também existem endereços de origem e destino, denominados endereços físicos que somente são usados no enlace local. A partir de um quadro recebido, o comutador analisa o endereço físico do destinatário, consulta uma tabela para saber em qual interface aquele destinatário está conectado e, por fim, retransmite o pacote.

Entretanto, para que máquinas pertencentes a redes diferentes se conectem a um mesmo comutador, é necessário que este possua a capacidade de criar VLANs (*Virtual Local Area Networks*). As VLANs são redes lógicas formadas por somente uma parte restrita dos dispositivos conectados ao comutador. A necessidade desse mecanismo adicional surge porque máquinas pertencentes a redes distintas não se comunicam diretamente pela camada de interface de rede. Os pacotes destinados a outras redes devem necessariamente ser enviados a roteadores. Ao se usar VLANs, pacotes de uma rede não interferem em outra, principalmente os de difusão (*broadcast*), que são enviados a todos da rede. Porém, para que uma rede se comunique com outra, um roteador precisa estar conectado às duas VLANs.

Uma das possíveis maneiras de se estabelecer uma VLAN é configurar o comutador de forma que cada uma de suas interfaces fique associada a somente uma VLAN. Ou seja, cada interface do comutador pertence a uma determinada rede. Esse foi o método utilizado para que as máquinas das três redes do laboratório pudessem se comunicar usando somente um comutador. Assim, algumas portas foram reservadas para a zona desmilitarizada, algumas para a rede de interconexão e a maioria foi liberada para a rede interna, como pode ser visto na Figura V.4. É importante observar que o roteador, com cada uma de suas interfaces conectada a uma rede diferente, pertence às três VLANs, com uma interface física em cada VLAN.

O filtro define algumas variáveis que contêm principalmente o nome das interfaces de rede do roteador, seus endereços IP e a faixa de endereços de cada rede. O objetivo de definir tais variáveis é para facilitar a manutenção do código, dado que todas as regras do filtro fazem referência a estas variáveis. Também são definidas algumas funções que são utilizadas ao longo do código. As funções `success()` e `failure()` somente imprimem na tela "`[ OK ]`" e "`[FAILED]`", respectivamente. Elas servem para avisar ao usuário que

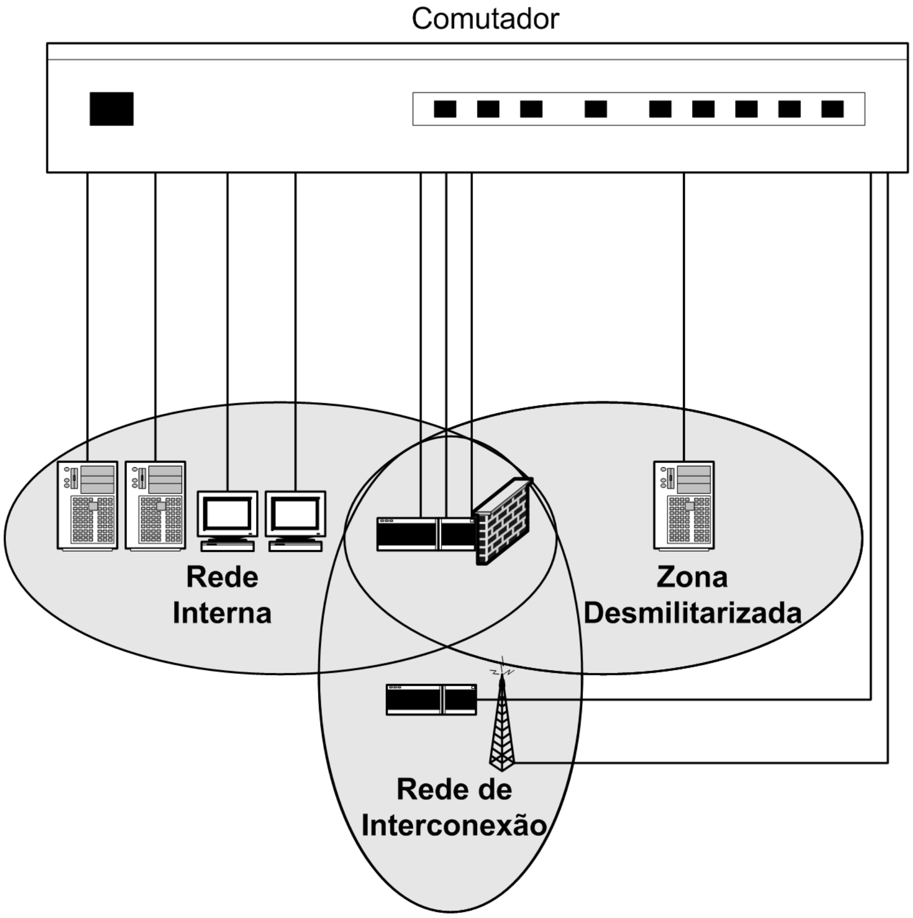

Figura V.4: A comunicação física entre as máquinas da rede.

alguma rotina interna foi executada com sucesso ou que algum erro ocorreu, extremamente útil durante o processo de depuração e testes. A função `set_policy()` somente configura a política padrão de todas as cadeias de todas as tabelas para o que lhe for passado como argumento. As funções `flush_chains()`, `rm_chains()` e `zero_counter()` atuam em todas as cadeias das três tabelas e são elas que, respectivamente, apagam todas as suas regras, removem as cadeias definidas pelo usuário e zeram seus contadores. Por fim, a função `check()` é a responsável por fazer uma checagem inicial e é chamada sempre que o *script* é executado. Ela realiza quatro verificações principais: se o usuário que rodou o *script* é o super-usuário (`root`), se o binário do `iptables` existe e é executável, se a versão do *kernel* é adequada e, por fim, se o `ipchains` não está carregado. O `ipchains` é o antecessor do `iptables`, e a grande diferença entre os dois é que o primeiro não fazia filtragem com inspeção de estados.

A função `stop()` tem o papel de parar o serviço de filtragem. Para isso, é preciso apagar as regras de todas as cadeias, remover as cadeias definidas pelo usuário, zerar todos os contadores e, finalmente, configurar as políticas padrão de todas as cadeias para aceitar qualquer pacote. Feito isso, nenhum pacote mais é barrado, o que pode ser indesejável. Por isso, é preciso cuidado ao parar o serviço de filtragem.

A função `start()` é a responsável por iniciar o serviço de filtragem de pacotes no roteador e contém todas as regras para isso. A primeira tarefa da função é carregar alguns módulos para que algumas funcionalidades que não estejam compiladas estaticamente no *kernel* possam ser utilizadas. Esses módulos seriam carregados sob demanda à medida que fossem necessários; porém, aqueles que são usados nas regras já foram pré-carregados manualmente.

Na configuração das regras, deve-se primeiro limpar toda a configuração já existente. Isso é feito através do apagamento das regras de todas as cadeias, da remoção das cadeias definidas pelo usuário e ainda da ação de zerar os contadores. É importante frisar que as outras tabelas (*nat* e *mangle*) não são utilizadas neste projeto devido a não ser necessária a tradução de endereços de rede e a alteração dos pacotes. O projeto se baseia na construção de uma zona desmilitarizada e, portanto, se restringe somente à filtragem dos pacotes com a tabela *filter*. Devido a isso, a travessia dos pacotes pelas cadeias se limita ao mostrado na Figura V.5. Todas as outras cadeias não têm nenhuma regra e aceitam todos os pacotes como política padrão.

O objetivo principal ao se montar o filtro foi economizar o máximo de processamento do roteador e ainda manter os requisitos de segurança para o funcionamento adequado da zona desmilitarizada. A idéia norteadora do projeto do filtro é não gastar processamento com regras desnecessárias. Um exemplo muito comum de como isso acontece é quando os pacotes UDP são inspecionados somente depois de terem passado por todas as regras dos pacotes TCP. Geralmente, o que é feito é colocar todas as regras em uma só cadeia fazendo com que pacotes passem por diversas regras até que se encaixem em alguma. Assim, a

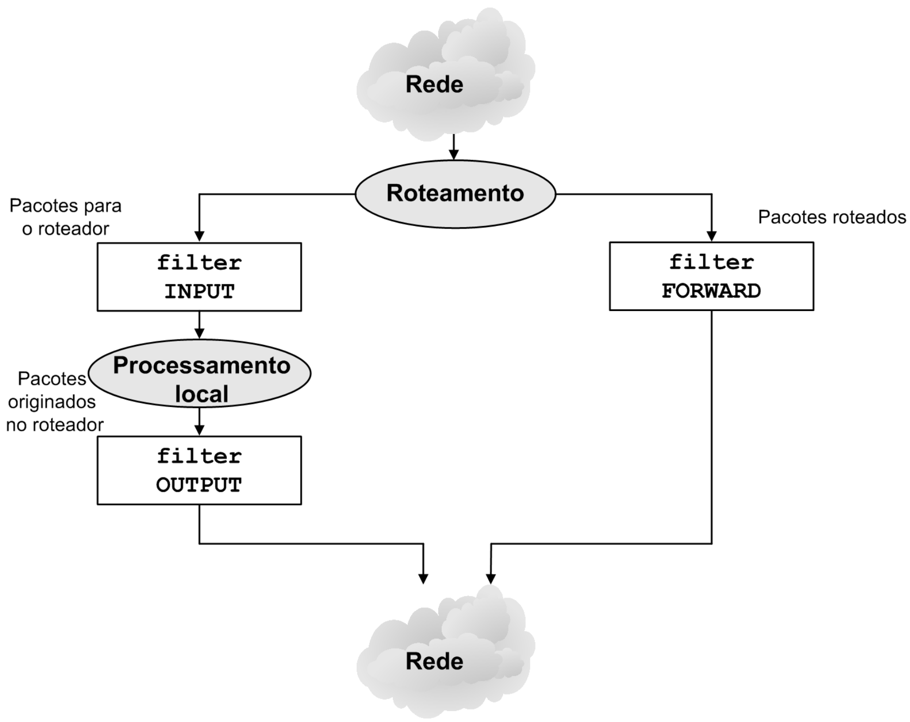

Figura V.5: Travessia dos pacotes pelas cadeias da tabela *filter*.

primeira regra usada tanto na cadeia de entrada (INPUT) quanto na de encaminhamento (FORWARD) é aceitar todos os pacotes pertencentes aos estados ESTABLISHED ou RELATED. O motivo principal é que, com a exceção dos primeiros pacotes trocados em uma comunicação, a maioria dos pacotes se encaixa no estado de comunicação previamente estabelecida. Através de um filtro de pacotes com inspeção de estados, é possível liberar todos os pacotes que pertençam a algum fluxo já estabelecido ou relacionado sem se preocupar com a segurança, desde que se faça um controle rígido sobre os pacotes iniciais da comunicação. Dessa maneira, ao invés de analisar todo pacote trocado no fluxo de pacotes, somente os pacotes iniciais são analisados e todos os outros que passam pelo filtro são liberados depois de uma consulta à tabela de estados. Portanto, colocando essa regra como a primeira em cada cadeia, quase todos os pacotes que atravessam o roteador não passam por nenhuma outra regra senão esta.

Nas mesmas duas cadeias, um grande esforço é então colocado no tratamento dos pacotes iniciais, pertencentes ao estado NEW. Entretanto, devido às diferentes funções de cada uma (a cadeia INPUT faz a filtragem dos pacotes destinados ao roteador enquanto que a cadeia FORWARD filtra os pacotes roteados), esse tratamento não é feito da mesma maneira nestas cadeias. O esquema e as cadeias auxiliares criadas para a cadeia INPUT lidar com esses pacotes é mostrado na Figura V.6. Depois de liberar os pacotes de fluxos estabelecidos ou relacionados, todos os pacotes iniciais são redirecionados para uma outra cadeia definida como `input_new`, que novamente redireciona cada pacote de acordo com o protocolo levado acima do IP. Porém, antes disso, algumas verificações são feitas na cadeia `spoofing_check` para checar por pacotes com endereço de origem forjados. Foram criadas as cadeias `input_new_tcp`, `input_new_udp`, `input_new_icmp`, `input_new_pim` para lidar com os protocolos TCP, UDP, ICMP e PIM (*Protocol Independent Multicast*), respectivamente. Este último protocolo é responsável pela troca de informações sobre roteamento *multicast*. A cadeia `input_new_all_lo` foi criada para liberar aqueles pacotes originados no próprio roteador, pela interface de *loopback*. Todas as cadeias a chamam para se ter certeza de que nenhum pacote criado pelo roteador está sendo descartado.

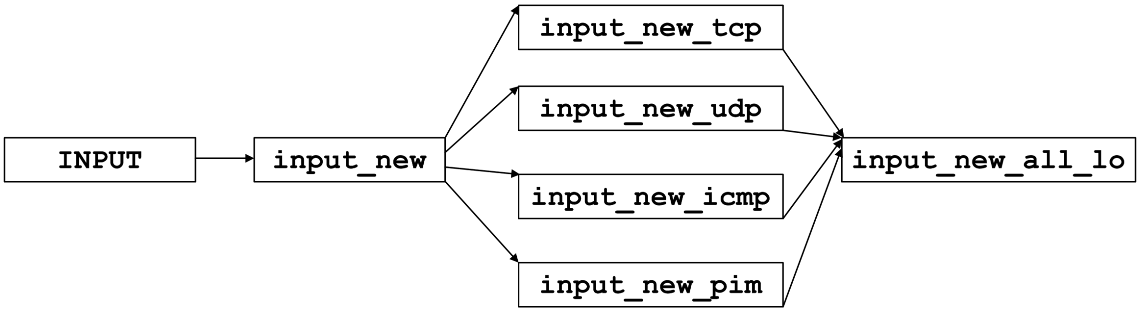

Figura V.6: Relação entre as cadeias criadas para pacotes destinados ao roteador.

As cadeias criadas e o esquema da cadeia FORWARD se encontram na Figura V.7. Assim como a cadeia de entrada, a cadeia de encaminhamento também deixa todos os pacotes pertencentes aos estados ESTABLISHED e RELATED passarem, redirecionando todos do estado NEW para a cadeia `forward_new`. Esta, por sua vez, verifica se o endereço de origem é forjado e redireciona mais uma vez o pacote de acordo com o seu protocolo. As cadeias `forward_new_tcp`, `forward_new_udp` e `forward_new_icmp` foram criadas com o propósito de filtrarem os protocolos TCP, UDP e ICMP, respectivamente. A cadeia `forward_new_all` foi criada para liberar os pacotes que não tenham se encaixado em nenhuma das regras mas que não ferem a política de segurança. Todas as cadeias a chamam depois que um pacote as atravessa por completo sem se encaixar em nenhuma regra.

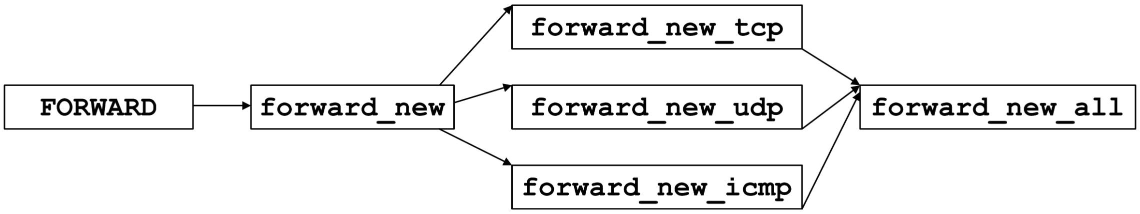

Figura V.7: Relação entre as cadeias criadas para os pacotes roteados.

Depois de carregar os módulos necessários, as políticas padrão de cada cadeia são definidas. As cadeias INPUT e FORWARD têm a política de descartar (DROP) todo pacote que não se encaixar em nenhuma das regras presentes nessas cadeias. A cadeia OUTPUT, por sua vez, apresenta a política de aceitar todo pacote sem restrições. Desse modo, nenhum pacote originado no roteador é barrado.

Após a definição das políticas, todas as cadeias auxiliares são criadas através do comando **-N**. A princípio, elas são criadas vazias e são preenchidas mais adiante. A primeira cadeia a ser preenchida é a `spoofing_check`, que apenas checa se os pacotes que chegam pela interface da rede do Bloco H contêm o endereço de origem da rede do laboratório. Neste caso, eles são descartados. A partir deste roteador, só é possível se prevenir contra pacotes com endereços falsificados vindos da interface conectada à rede do Bloco H. O segundo roteador é responsável pela checagem de endereços de origem forjados no outro ponto de entrada da rede.

A segunda cadeia a ser preenchida é a `input_new`, responsável pelo redirecionamento dos pacotes às cadeias de tratamento de cada protocolo. Depois de verificar se o pacote tem o endereço de origem forjado, esta cadeia verifica na seqüência se o protocolo do pacote se encaixa em algum dos listados e o encaminha para a cadeia específica. No final, se o pacote não se encaixar em nenhum protocolo listado, ele é registrado. É permitido que, no máximo, três pacotes sejam registrados por minuto com um limite de rajada também de três. Isso quer dizer que se três pacotes são registrados em um segundo, o próximo pacote que se encaixar nessa regra só é registrado depois de um minuto. Tal limitação serve para não encher demais os arquivos de registro (*log*). Além disso, o nível do registro também é configurado e um prefixo (“`input_new packet died: `”) é colocado à frente do registro daquele pacote. As cadeias realizam este registro para depuração e verificação de erros no filtro.

A próxima cadeia é a `input_new_all_lo`, cuja função é liberar todos os pacotes criados pela interface de *loopback*, desde que tenham os endereços de origem de alguma interface do roteador. Essa cadeia é chamada por todas as outras antes de descartar os pacotes, de forma que nenhum pacote criado no próprio roteador é barrado. Ao invés de colocá-la como uma das primeiras regras, foi escolhido que ela seria chamada pelas cadeias específicas por que acredita-se que a maioria dos pacotes de entrada não vem do próprio roteador. Desse modo, evita-se verificar se todos os pacotes vieram da interface de *loopback*.

A cadeia `input_new_tcp` libera os pacotes destinados ao SSH no roteador, somente para as interfaces da rede interna e da zona desmilitarizada. Somente máquinas destas redes podem acessar o roteador. A cadeia `input_new_udp` deixa passar os pacotes trocados pelo protocolo de roteamento RIP (*Routing Information Protocol*). Essa informação precisa chegar ao roteador porque é através dela que ele constrói sua tabela de rotas, usada no encaminhamento dos pacotes. A cadeia `input_new_icmp` libera somente os novos pacotes ICMP `echo-request` para a entrada do roteador. A próxima cadeia, `input_new_pim`, deixa os pacotes PIM chegarem até o roteador, desde que venham pelas interfaces externas (BL_H e CON). Por fim, a cadeia `input_new_igmp` libera pacotes IGMP (*Internet Group Management Protocol*) se estes vierem pela interface da rede interna. Esses pacotes servem para avisar ao roteador que existem estações de trabalho interessadas em receber pacotes de um determinado grupo *multicast*. Tendo esse conhecimento, o roteador se afilia ao grupo desejado.

A função da cadeia `forward_new` é análoga à exercida pela cadeia `input_new`, pois verifica se o pacote tem o endereço de origem forjado e o redireciona de acordo com o protocolo. Ao final desta cadeia, se o pacote não se encaixar em nenhum dos protocolos TCP, UDP ou ICMP, ele vai para a cadeia `forward_new_all`. Esta cadeia libera todos os pacotes que saem pelas interfaces externas e rejeita todos aqueles que seriam roteados para a rede interna com um pacote ICMP (`icmp-net-prohibited`). Ela é chamada ao final de todas as cadeias, caso o pacote não se encaixe nas suas regras.

A cadeia `forward_new_tcp` é a que trata de todos os pacote TCP roteados. Ela libera pacotes iniciais de abertura de conexão TCP destinados ao servidor da zona desmilitarizada, desde que para as portas dos serviços FTP, SSH, SMTP, DNS, HTTP, HTTPS, SMTPS, IMAPS e POP3S. Além disso, são permitidas conexões de saída da zona desmilitarizada somente para os serviços de SMTP e DNS. O primeiro necessita ter a saída liberada para o reenvio de mensagens eletrônicas aos servidores e o segundo utiliza o TCP para a troca de informações entre domínios de maneira confiável. Além disso, é permitido que o servidor envie pacotes para o servidor SSH da rede interna para o estabelecimento do túnel coreano, conforme explicado anteriormente. Todos os outros pacotes recebidos pela interface da ZDM são descartados.

No caso do UDP, a cadeia `forward_new_udp` deixa que pacotes de consulta ao serviço DNS passem pelo filtro. A saída de pacotes da zona desmilitarizada com destino a esses serviços também é liberada, desde que não seja para a rede interna, de forma que as consultas a servidores externos funcionem. Assim como no caso do TCP, todos os outros pacotes recebidos pela interface da zona desmilitarizada são descartados.

A cadeia `forward_new_icmp` tem a função de liberar os pacotes ICMP `echo-request` desde que não direcionados para a rede interna. As cadeias INPUT e FORWARD liberam pacotes de fluxos já estabelecidos ou relacionados, redirecionam os iniciais para respectivo tratamento e registram os pacotes que não atingirem nenhuma regra. As outras cadeias não apresentam regras por não terem objetivo algum neste projeto.

Existem sete opções de uso do *script*, são elas: `start`, `stop`, `restart`, `status`, `panic`, `save` e `restore`. A opção `start` serve para iniciar o serviço de filtragem de pacotes, ela simplesmente chama a função `start()` definida no código. A opção `stop` chama a função `stop()` para parar o filtro e aceitar todos os pacotes. Como a opção `start` já pára o serviço (se ele estiver rodando) e o reinicia, a opção `restart` faz exatamente o mesmo que a opção `start` e está aqui somente para usuários acostumados a ela. A opção `status` imprime na tela as regras vigentes no filtro. A opção `panic` serve para bloquear todo pacote de entrada ou roteado. Ela limpa todas as cadeias e configura a política padrão de cada uma para descartar os pacotes. As opções `save` e `restore` utilizam os programas `iptables-save` e `iptables-restore` para, respectivamente, salvar as configurações do filtro em um outro arquivo e recarregá-las quando necessário. Ainda existe o modo `debug`, especificamente para depuração, que imprime mensagens na tela à medida que os comandos são bem ou mal sucedidos, permitindo ao usuário saber onde existe algum problema.

V.2.2 Garantindo a Robustez de Senhas

Para garantir a robustez das senhas segundo uma política de construção de senhas, foi criado um mecanismo que força a mudança de senha e a impede caso não esteja dentro dos padrões requeridos. Este mecanismo foi criado para que todos os usuários que já possuem conta no laboratório fossem obrigados a usar uma senha diferente daquela usada na rede interna.

Para forçar a mudança de senha, o *script* descrito no Apêndice B foi criado. Para que a mudança seja feita no primeiro acesso à máquina via SSH, a configuração do *shell* de cada usuário foi alterada. O *shell* é o interpretador de comando que é executado logo após uma autenticação bem sucedida. Substituindo o *shell* de todos os usuários presentes no arquivo `/etc/passwd` pelo *script* criado, este último é executado logo depois que o usuário se autentica.

Depois de imprimir uma mensagem na tela que mostra os requisitos que a senha deve ter, o *script* tenta modificar a senha do usuário através do comando `/usr/bin/passwd`. Caso a mudança falhe, o usuário é desconectado e terá que mudar a senha no próximo acesso. Caso a mudança seja bem sucedida, um menu apresentando os *shells* disponíveis para o usuário é mostrado na tela e ele necessariamente escolhe o seu preferido. Ao fazer isso, a sua configuração no arquivo `/etc/passwd` é novamente alterada, colocando de volta o seu *shell* no lugar do *script*. Desse modo, somente uma vez será pedido que o usuário mude sua senha.

Para impedir que senhas que não atendem aos requisitos sejam colocadas, o código fonte de um módulo PAM (*Pluggable Authentication Modules*) foi alterado através de um *patch* encontrado em [20]. O *patch* se encontra no Apêndice C. Estes módulos são usados para processos relacionados a autenticação. Com a aplicação do *patch* e a recompilação do módulo `pam_cracklib`, é possível especificar qual o mínimo de caracteres que uma senha deve ter, quantas minúsculas, maiúsculas, dígitos e, ainda, o número mínimo de caracteres especiais. Os requisitos são configurados no arquivo `/etc/pam.d/system-auth` através da linha:

```
password required /lib/security/$ISA/pam_cracklib.so retry=3 type=
ucredit=-1 lcredit=-1 ocredit=-1 dcredit=0 minlen=8 debug
```

Nessa linha, os parâmetros `ucredit`, `lcredit` e `ocredit` especificam, em números negativos, o mínimo que uma senha deve possuir de letras maiúsculas, minúsculas e caracteres especiais, respectivamente. O parâmetro `dcredit` não é usado, mas serviria para especificar o mínimo de números que uma senha deve ter. Depois, o parâmetro `minlen` é o responsável pela especificação do tamanho mínimo de caracteres da senha. Com a configuração adequada destes requisitos e forçando os usuários a modificar suas senhas no primeiro acesso, é possível a existência de dois domínios de autenticação independentes.

Capítulo VI

Conclusões

O objetivo deste trabalho foi projetar e implementar uma nova topologia para a rede do Grupo de Teleinformática e Automação (GTA) da COPPE/UFRJ, de maneira que nela fosse incluída uma zona desmilitarizada. Para isso, antes foi feita uma análise da situação vigente no laboratório, de sua topologia e dos serviços prestados tanto interna quanto externamente, para somente depois propor as mudanças necessárias na infra-estrutura da rede.

Apesar do uso de uma zona desmilitarizada melhorar a segurança da rede, ela é também muito restritiva. Para acessos de fora da rede, nenhum serviço, em teoria, pode ser acessado senão aqueles disponibilizados na própria zona. Como diversos usuários do laboratório acessam suas máquinas de casa, foi preciso flexibilizar um pouco essa rígida definição. A solução adotada foi criar um túnel coreano conectando usuários externos ao servidor SSH interno através do servidor da zona desmilitarizada. Desse modo, foi assumido o risco de ter a rede interna comprometida, caso alguma vulnerabilidade do SSH fosse explorada, em detrimento de uma maior liberdade para os usuários. Mesmo com esta flexibilização, esse método ainda apresenta a vantagem de correr este risco somente para o serviço de SSH. Usando-se uma única rede, qualquer um dos serviços poderia comprometer a rede. Também devido a esta flexibilização, foi escolhido separar os domínios de autenticação. Desse modo, a autenticação da rede interna é feita por um servidor interno enquanto que a autenticação da zona desmilitarizada é feita pelo próprio servidor. Além disso, os usuários tem senhas diferentes em cada um dos domínios de forma que, se o servidor da zona desmilitarizada for comprometido, a segurança da rede interna se mantém intacta.

Na implementação, a adoção de VLANs no comutador proporcionou uma economia no projeto. Caso o comutador não apresentasse esta característica, seriam necessários três repetidores ou comutadores, um para cada rede. Como o comutador do laboratório permite o uso de VLANs, apenas um comutador foi necessário para a realização de todo projeto.

Este projeto detalha uma sistemática do filtro de maneira a evitar ao máximo o processamento desnecessário e, ao mesmo tempo, atender aos requisitos de segurança. A forma escolhida foi liberar os pacotes pertencentes a fluxos previamente estabelecidos ou relacionados enquanto os pacotes iniciais são rigidamente controlados. Dessa maneira, somente os primeiros pacotes dos fluxos são inspecionados e, sendo liberados nesta fase inicial, todos os pacotes daquele fluxo têm a passagem permitida.

Referências Bibliográficas

[1] O. Andreasson, "Iptables Tutorial", 2003. Versão 1.1.19. http://iptables-tutorial.frozentux.net.

[2] D. E. Comer, *Internetworking with TCP/IP: Principles, Protocols and Architecture*, vol. 1. Englewood Cliffs, NJ, EUA: Prentice Hall, third ed., 1995.

[3] A. S. Tanenbaum, *Computer Networks*. Upper Saddle River, NJ, EUA: Prentice Hall PTR, fourth ed., 2002.

[4] W. R. Cheswick, S. M. Bellovin e A. D. Rubin, *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley Professional Computing Series, Addison-Wesley, second ed., 2003.

[5] E. D. Zwicky, S. Cooper e D. B. Chapman, *Building Internet Firewalls*. Sebastopol, CA, EUA: O'Reilly & Associates, Inc., second ed., 2000.

[6] M. Strebe e C. Perkins, *Firewalls*. São Paulo, SP, Brasil: Makron Books, 2002.

[7] "The netfilter/iptables Project". http://www.netfilter.org.

[8] "IP Filter". http://www.ipfilter.org.

[9] "Norton Personal Firewall". http://www.symantec.com/sabu/nis/npf.

[10] "Zone Labs, Inc.". http://www.zonelabs.com.

[11] R. P. Laufer, "Introdução a sistemas de detecção de intrusão", julho de 2003. Trabalho da disciplina "Redes de Computadores I", do curso de Engenharia Eletrônica e de Computação da Universidade Federal do Rio de Janeiro (UFRJ). http://www.gta.ufrj.br/~rlaufer/production/undergraduate/redesI.

[12] S. M. Bellovin, "Distributed firewalls", *;login:*, vol. 24, pp. 39–47, novembro de 1999.

[13] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, B. Schwartz, S. T. Kent e W. T. Strayer, "Single-packet IP traceback", *IEEE/ACM Transactions on Networking*, no. 6, pp. 721–734, dezembro de 2002.

[14] A. C. Snoeren, C. Partridge, L. A. Sanchez, C. E. Jones, F. Tchakountio, S. T. Kent e W. T. Strayer, "Hash-based IP traceback", *SIGCOMM'01*, San Diego, Califórnia, EUA, agosto de 2001.

[15] R. Russel, "Linux IPCHAINS HOWTO", julho de 2000. Versão 1.0.8. http://www.linux.org/docs/ldp/howto/IPCHAINS-HOWTO.html.

[16] Fyodor, "The Art of Port Scanning", setembro de 1997. http://www.insecure.org/nmap/nmap_doc.html.

[17] "Wikipedia, The Free Encyclopedia". http://www.wikipedia.org.

[18] J. Kemp, "Basic Firewall Concepts". http://cc.uoregon.edu/cnews/spring2002/firewall.html.

[19] H. Eychenne, *Man page of Iptables*, março de 2002.

[20] W. Puschitz, "Procedure for Patching pam_cracklib.c", dezembro de 2002. http://www.puschitz.com/pam_cracklib_patch.shtml.

[21] A. T. Pedroza, "Notas de aula da disciplina 'Arquitetura e Projeto de Protocolos'", julho de 2003. Lecionada no mestrado do Programa de Engenharia Elétrica (PEE) da COPPE/UFRJ.

Apêndice A

O Filtro de Pacotes

```
#!/bin/bash
# /etc/init.d/iptables - iptables firewall configuration script
#
#    based on rc.DMZ.firewall from Oskar Andreasson
#    <blueflux@koffein.net> and on the Red Hat iptables startup
#    scritp by bero@redhat.com, which was based on the ipchains
#    script written by Joshua Jensen <joshua@redhat.com>,
#    -- hacked up by gafton with help from notting, modified by Anton
#    Altaparmakov <aia21@cam.ac.uk>, modified by Nils
#    Philippsen <nils@redhat.de>
#
# Copyright (C) 2003 Grupo de Teleinformática e Automação <staff@gta.ufrj.br>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; version 2 of the License.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.

# You should have received a copy of the GNU General Public License
# along with this program or from the site that you downloaded it
# from; if not, write to the Free Software Foundation, Inc., 59 Temple
# Place, Suite 330, Boston, MA  02111-1307   USA
```

```
#########################################################################
# 1. Topologia da rede do laboratório
#
#                                     |    Anel FDDI
#                                     |    146.164.1.0/26
#                                     |
#                                     | .25
#                                     |(fddi0)
#                               ______|______
#                              |   FUNDÃO    |
#                              |_____________|
#                                     | .254
#                                     |(fa0)
#                                    _|
#                                     |_   Rede de Interconexão (CON)
#                                    _|    146.164.69.192/26
#                                     |_
#                                     | .253
#       |_                            |(eth1)                          |_
#      _|                       ______|______                         _|
#       |______________________|    ANGRA    |_________________________|_
#      _|                   .1 |_____________| .210                   _|
#       |_               (eth2)       |       (eth0)                   |_
#                                     | .129
#      Rede interna (INT)             |(eth3)  Backbone do Bloco H (BL_H)
#       146.164.69.0/25              _|             146.164.5.192/26
#                                     |_
#                                    _|
#                                     |_
#                          Zona Desmilitarizada (ZDM)
#                              146.164.69.128/26

#########################################################################
# 2. Opções de configuração
#

#
# 2.1 Configuração da Rede interna
#

INT_IFACE="eth2"
INT_IP="146.164.69.1"
INT_REDE="146.164.69.0/25"
INT_SERVIDOR_SSH="146.164.69.2"

#
# 2.2 Configuração da Zona Desmilitarizada
#

ZDM_IFACE="eth3"
ZDM_IP="146.164.69.129"
ZDM_REDE="146.164.69.128/26"
ZDM_SERVIDOR="146.164.69.130"

#
```

```
# 2.3 Configuração do Backbone do Bloco H
#

BL_H_IFACE="eth0"
BL_H_IP="146.164.5.210"
BL_H_REDE="146.164.5.192/26"

#
# 2.4 Configuração da Rede externa
#

CON_IFACE="eth1"
CON_IP="146.164.69.253"
CON_REDE="146.164.69.192/26"

#
# 2.5 Configuração da Estação local
#

LO_IFACE="lo"
LO_IP="127.0.0.1"
LO_REDE="127.0.0.0/8"

#
# 2.6 Configuração do Iptables
#

IPTABLES="/sbin/iptables"
IPTABLES_CONFIG="/etc/sysconfig/iptables"

#
# 2.7 Outras configurações
#

GTA_REDE="146.164.69.0/24"
IP_MULTICAST="224.0.0.0/4"

##########################################################################
# 3. Algumas funções úteis
#

#
# success() - imprime "[  OK  ]" no canto direito da tela, em cor verde
#
success() {
    echo -en \\033[60G
    echo -n "[  "
    echo -en "\\033[1;32m"
    echo -n $"OK"
    echo -en "\\033[0;39m"
    echo -n "  ]"
    echo -ne "\r"
    echo
    return 0
}
```

```
#
# failure() - imprime "[FAILED]" no canto direito da tela, em cor
#             vermelha
#
failure() {
    echo -en \\033[60G
    echo -n "["
    echo -en "\\033[1;31m"
    echo -n $"FAILED"
    echo -en "\\033[0;39m"
    echo -n "]"
    echo -ne "\r"
    echo
    return 1
}

#
# set_policy() - ajusta a política das cadeias para o
#                primeiro argumento passado para a função.
#
set_policy() {
    $IPTABLES -t filter -P INPUT       $1 && \
    $IPTABLES -t filter -P OUTPUT      $1 && \
    $IPTABLES -t filter -P FORWARD     $1 && \
    $IPTABLES -t nat    -P PREROUTING  $1 && \
    $IPTABLES -t nat    -P POSTROUTING $1 && \
    $IPTABLES -t nat    -P OUTPUT      $1 && \
    $IPTABLES -t mangle -P PREROUTING  $1 && \
    $IPTABLES -t mangle -P POSTROUTING $1 && \
    $IPTABLES -t mangle -P INPUT       $1 && \
    $IPTABLES -t mangle -P OUTPUT      $1 && \
    $IPTABLES -t mangle -P FORWARD     $1 && \
    return 0 || return 1
}

#
# flush_chains() - esvazia todas as cadeias de todas as tabelas.
#
flush_chains() {
    $IPTABLES -t filter -F && \
    $IPTABLES -t nat    -F && \
    $IPTABLES -t mangle -F && \
    return 0 || return 1
}

#
# rm_chains() - remove todas as cadeias definidas pelos usuários em
#               todas as tabelas.
#
rm_chains() {
    $IPTABLES -t filter -X && \
    $IPTABLES -t nat    -X && \
    $IPTABLES -t mangle -X && \
    return 0 || return 1
}
```

```
#
# zero_counter() - zera os contadores de todas as cadeias em todas as
#                  tabelas.
#
zero_counter() {
    $IPTABLES -t filter -Z && \
    $IPTABLES -t nat    -Z && \
    $IPTABLES -t mangle -Z && \
    return 0 || return 1
}

#
# check() - faz uma checagem inicial
#
check() {

    # Mostrando na tela o que estamos fazendo
    if [ $DEBUG -eq 1 ]; then
        echo -n $"Checagem inicial:"
    fi
    # Verificando versão do kernel
    KERNELMAJ=`uname -r | sed                    -e 's,\..*,,'`
    KERNELMIN=`uname -r | sed -e 's,[^\.]*\.,,' -e 's,\..*,,'`

    # Checando se o usuário é root
    [ $UID == 0 ] && \

    # Checando se o iptables existe e é executável
    [ -x $IPTABLES ] && \

    # Checando se a versão do kernel é maior que 2.3
    [ ! "$KERNELMAJ" -lt 2 ] && \
    [ ! "$KERNELMIN" -lt 3 ] && \

    # Checando se o ipchains não está carregado
    ! /sbin/lsmod 2>/dev/null |grep -q ipchains && \

    # Se nada falhar, chama success() e retorna. Se algo falhar,
    # chama failure() e sai do programa.
    if [ $DEBUG -eq 1 ]; then success; fi && return 0 || \
    { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }
}

########################################################################
# 4. Parando o iptables
#

stop() {
    # Esvaziando todas as cadeias pré-existentes
    if [ $DEBUG -eq 1 ]; then
        echo -n $"Esvaziando cadeias:"
    fi
    flush_chains && \
    if [ $DEBUG -eq 1 ]; then success; fi || \
    { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }
```

```
    # Removendo cadeias previamente definidas pelo usuário
    if [ $DEBUG -eq 1 ]; then
        echo -n $"Removendo cadeias definidas pelo usuário:"
    fi
    rm_chains && \
    if [ $DEBUG -eq 1 ]; then success; fi || \
    { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

    # Zerando os contadores de pacotes das cadeias
    if [ $DEBUG -eq 1 ]; then
        echo -n $"Zerando os contadores de pacotes:"
    fi
    zero_counter && \
    if [ $DEBUG -eq 1 ]; then success; fi || \
    { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

    # Definindo política das cadeias para ACCEPT
    if [ $DEBUG -eq 1 ]; then
        echo -n $"Definindo políticas para ACCEPT:"
    fi
    set_policy ACCEPT && \
    if [ $DEBUG -eq 1 ]; then success; fi || \
    { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }
}

##########################################################################
# 5. Iniciando o iptables
#

start() {
    ######################################################################
    # 5.1 Carregamento de módulos
    #

    if [ $DEBUG -eq 1 ]; then
        echo -n $"Carregando módulos:"
    fi
    #
    # Necessário para carregar os módulos inicialmente
    #
    /sbin/depmod -a                  && \

    #
    # 5.1.1 Módulos necessários
    #

    /sbin/modprobe ip_tables         && \
    /sbin/modprobe iptable_filter    && \
    /sbin/modprobe ip_conntrack      && \
    /sbin/modprobe ip_conntrack_ftp  && \
    /sbin/modprobe ipt_LOG           && \
    /sbin/modprobe ipt_REJECT        && \
    /sbin/modprobe ipt_limit         && \
    /sbin/modprobe ipt_state         && \
    /sbin/modprobe ipt_multiport     && \
```

```
#
# 5.1.2 Módulos desnecessários
#

#/sbin/modprobe ipt_owner        && \
#/sbin/modprobe ipt_MASQUERADE   && \
#/sbin/modprobe ip_conntrack_irc && \
#/sbin/modprobe ip_nat_ftp       && \
#/sbin/modprobe ip_nat_irc       && \
#/sbin/modprobe iptable_nat      && \
#/sbin/modprobe iptable_mangle   && \

if [ $DEBUG -eq 1 ]; then success; fi || \
{ if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

######################################################################
# 5.2. Configuração das regras
#

# Esvaziando todas as cadeias pré-existentes
if [ $DEBUG -eq 1 ]; then
    echo -n $"Esvaziando cadeias:"
fi
flush_chains && \
if [ $DEBUG -eq 1 ]; then success; fi || \
{ if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

# Removendo cadeias previamente definidas pelo usuário
if [ $DEBUG -eq 1 ]; then
    echo -n $"Removendo cadeias definidas pelo usuário:"
fi
rm_chains && \
if [ $DEBUG -eq 1 ]; then success; fi || \
{ if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

# Zerando os contadores de pacotes das cadeias
if [ $DEBUG -eq 1 ]; then
    echo -n $"Zerando os contadores de pacotes:"
fi
zero_counter && \
 if [ $DEBUG -eq 1 ]; then success; fi || \
{ if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

######
# 5.2.1 Tabela filter
#

if [ $DEBUG -eq 1 ]; then
    echo -en "\\033[1;32m"
    echo "TABELA: filter"
    echo -en "\\033[0;39m"
fi

#
# 5.2.1.1 Política
```

```
#

# Definindo política das cadeias
if [ $DEBUG -eq 1 ]; then
    echo -ne $"\tDefinindo políticas:"
fi
$IPTABLES -t filter -P INPUT   DROP   && \
$IPTABLES -t filter -P OUTPUT  ACCEPT && \
$IPTABLES -t filter -P FORWARD DROP   && \
if [ $DEBUG -eq 1 ]; then success; fi || \
{ if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

#
# 5.2.1.2 Criaçao das cadeias especificadas pelo usuário
#

# Criando cadeias definidas pelo usuário
if [ $DEBUG -eq 1 ]; then
    echo -ne $"\tCriando novas cadeias:"
fi

$IPTABLES -N spoofing_check           && \

$IPTABLES -N input_new                && \
$IPTABLES -N input_new_all_lo         && \
$IPTABLES -N input_new_tcp            && \
$IPTABLES -N input_new_udp            && \
$IPTABLES -N input_new_icmp           && \
$IPTABLES -N input_new_pim            && \
$IPTABLES -N input_new_igmp           && \

$IPTABLES -N forward_new              && \
$IPTABLES -N forward_new_all          && \
$IPTABLES -N forward_new_tcp          && \
$IPTABLES -N forward_new_udp          && \
$IPTABLES -N forward_new_icmp         && \
if [ $DEBUG -eq 1 ]; then success; fi || \
{ if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

#
# 5.2.1.3 Criação das regras das cadeias especificadas pelo usuário
#

# Criando regras das cadeis definidas pelo usuário
if [ $DEBUG -eq 1 ]; then
    echo -ne $"\tCriando regras das novas cadeias:"
fi

#
# Cadeia spoofing_check
# - A checagem da outra interface é feita pelo outro roteador
#
$IPTABLES -A spoofing_check -i $BL_H_IFACE -s $GTA_REDE -j DROP  && \

#
# Cadeia input_new
```

```
#
$IPTABLES -A input_new         -j spoofing_check                  && \
$IPTABLES -A input_new -p tcp  -j input_new_tcp                   && \
$IPTABLES -A input_new -p udp  -j input_new_udp                   && \
$IPTABLES -A input_new -p icmp -j input_new_icmp                  && \
$IPTABLES -A input_new -p pim  -j input_new_pim                   && \
$IPTABLES -A input_new -p igmp -i input_new_igmp                  && \
$IPTABLES -A input_new -m limit --limit 3/minute --limit-burst 3     \
                              -j LOG --log-level DEBUG     \
                    --log-prefix "input_new packet died: " && \
$IPTABLES -A input_new -j DROP                                    && \

#
# Cadeia input_new_all_lo
#

$IPTABLES -A input_new_all_lo -i $LO_IFACE -s $LO_IP  -j ACCEPT  && \
$IPTABLES -A input_new_all_lo -i $LO_IFACE -s $INT_IP -j ACCEPT  && \
$IPTABLES -A input_new_all_lo -i $LO_IFACE -s $ZDM_IP -j ACCEPT  && \
$IPTABLES -A input_new_all_lo -i $LO_IFACE -s $CON_IP -j ACCEPT  && \

#
# Cadeia input_new_tcp
#

$IPTABLES -A input_new_tcp -i $INT_IFACE -p tcp -s $INT_REDE        \
                      -d $INT_IP --syn --dport ssh -j ACCEPT     && \
$IPTABLES -A input_new_tcp -i $ZDM_IFACE -p tcp -s $ZDM_REDE        \
                      -d $ZDM_IP --syn --dport ssh -j ACCEPT     && \
$IPTABLES -A input_new_tcp -i $LO_IFACE -j input_new_all_lo      && \
$IPTABLES -A input_new_tcp -m limit --limit 3/minute                \
            --limit-burst 3 -j LOG --log-level DEBUG                \
            --log-prefix "input_new_tcp packet died: "           && \
$IPTABLES -A input_new_tcp -j DROP                               && \

#
# Cadeia input_new_udp
#

$IPTABLES -A input_new_udp -i $CON_IFACE  -p udp -s $CON_REDE       \
           -d $CON_IP --sport router --dport router -j ACCEPT    && \
$IPTABLES -A input_new_udp -i $BL_H_IFACE -p udp -s $BL_H_REDE      \
           -d $BL_H_IP --sport router --dport router -j ACCEPT   && \
$IPTABLES -A input_new_udp -i $LO_IFACE -j input_new_all_lo      && \
$IPTABLES -A input_new_udp -m limit --limit 3/minute                \
            --limit-burst 3 -j LOG --log-level DEBUG                \
             --log-prefix "input_new_udp packet died: "          && \
$IPTABLES -A input_new_udp -j DROP                               && \


#
# Cadeia input_new_icmp
#

$IPTABLES -A input_new_icmp -p icmp --icmp-type echo-request        \
                                               -j ACCEPT     && \
```

```
$IPTABLES -A input_new_icmp -i $LO_IFACE -j input_new_all_lo     && \
$IPTABLES -A input_new_icmp -m limit --limit 3/minute               \
             --limit-burst 3 -j LOG --log-level DEBUG               \
             --log-prefix "input_new_icmp packet died: "         && \
$IPTABLES -A input_new_icmp -j DROP                              && \

#
# Cadeia input_new_pim
#

$IPTABLES -A input_new_pim -i $BL_H_IFACE -p pim -j ACCEPT       && \
$IPTABLES -A input_new_pim -i $CON_IFACE  -p pim -j ACCEPT       && \
$IPTABLES -A input_new_pim -i $LO_IFACE -j input_new_all_lo      && \
$IPTABLES -A input_new_pim -m limit --limit 3/minute                \
            --limit-burst 3 -j LOG --log-level DEBUG                \
            --log-prefix "input_new_pim packet died: "           && \
$IPTABLES -A input_new_pim -j DROP                               && \

#
# Cadeia input_new_igmp
#
$IPTABLES -A input_new -p igmp -i $INT_IFACE -s $INT_REDE           \
                                      -d $IP_MULTICAST -j ACCEPT && \
$IPTABLES -A input_new_igmp -i $LO_IFACE -j input_new_all_lo     && \
$IPTABLES -A input_new_igmp -m limit --limit 3/minute               \
             --limit-burst 3 -j LOG --log-level DEBUG               \
             --log-prefix "input_new_igmp packet died: "         && \
$IPTABLES -A input_new_igmp -j DROP                              && \

#
# Cadeia forward_new
#

$IPTABLES -A forward_new         -j spoofing_check               && \
$IPTABLES -A forward_new -p tcp  -j forward_new_tcp              && \
$IPTABLES -A forward_new -p udp  -j forward_new_udp              && \
$IPTABLES -A forward_new -p icmp -j forward_new_icmp             && \
$IPTABLES -A forward_new         -j forward_new_all              && \

#
# Cadeia forward_new_all
#

$IPTABLES -A forward_new_all -o $CON_IFACE  -j ACCEPT            && \
$IPTABLES -A forward_new_all -o $BL_H_IFACE -j ACCEPT            && \
$IPTABLES -A forward_new_all -o $INT_IFACE  -j REJECT               \
                               --reject-with icmp-net-prohibited && \
$IPTABLES -A forward_new_all -m limit --limit 3/minute              \
              --limit-burst 3 -j LOG --log-level DEBUG              \
              --log-prefix "forward_new_all packet died: "       && \
$IPTABLES -A forward_new_all -j DROP                             && \

#
# Cadeia forward_new_tcp
#
```

```
   $IPTABLES -A forward_new_tcp -o $ZDM_IFACE -p tcp -d $ZDM_SERVIDOR  \
-m multiport --dports ftp,ssh,smtp,domain,http,https,smtps,imaps,pop3s \
                                                    --syn -j ACCEPT && \
   $IPTABLES -A forward_new_tcp -i $ZDM_IFACE -o ! $INT_IFACE -p tcp   \
                  --syn -m multiport --dports smtp,domain -j ACCEPT && \
   $IPTABLES -A forward_new_tcp -i $ZDM_IFACE -o $INT_IFACE -p tcp     \
  --syn -s $ZDM_SERVIDOR -d $INT_SERVIDOR_SSH --dport ssh -j ACCEPT && \
   $IPTABLES -A forward_new_tcp -i $ZDM_IFACE -j DROP               && \
   $IPTABLES -A forward_new_tcp -j forward_new_all                  && \

   #
   # Cadeia forward_new_udp
   #

   $IPTABLES -A forward_new_udp -o $ZDM_IFACE -p udp -d $ZDM_SERVIDOR  \
                                        --dport domain -j ACCEPT && \
   $IPTABLES -A forward_new_udp -i $ZDM_IFACE -o ! $INT_IFACE -p udp   \
                                        --dport domain -j ACCEPT && \
   $IPTABLES -A forward_new_udp -i $ZDM_IFACE -j DROP               && \
   $IPTABLES -A forward_new_udp -j forward_new_all                  && \

   #
   # Cadeia forward_new_icmp
   #

   $IPTABLES -A forward_new_icmp -o ! $INT_IFACE -p icmp               \
                              --icmp-type echo-request -j ACCEPT && \
   $IPTABLES -A forward_new_icmp -i $ZDM_IFACE -j DROP              && \
   $IPTABLES -A forward_new_icmp -j forward_new_all                 && \

   if [ $DEBUG -eq 1 ]; then success; fi || \
   { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

   #
   # 5.2.1.4 Cadeia INPUT
   #

   # Criando regras da cadeia INPUT
   if [ $DEBUG -eq 1 ]; then
       echo -ne $"\tCriando regras de INPUT:"
   fi
   $IPTABLES -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT &&\
   $IPTABLES -A INPUT -m state --state NEW -j input_new              &&\
   $IPTABLES -A INPUT -m limit --limit 3/minute --limit-burst 3 -j LOG \
                --log-level DEBUG --log-prefix "INPUT packet died: " &&\

   if [ $DEBUG -eq 1 ]; then success; fi || \
   { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

   #
   # 5.2.1.5 Cadeia FORWARD
   #

   # Criando regras da cadeia FORWARD
   if [ $DEBUG -eq 1 ]; then
       echo -ne $"\tCriando regras de FORWARD:"
```

```
fi
$IPTABLES -A FORWARD -m state --state ESTABLISHED,RELATED          \
                                               -j ACCEPT        && \
$IPTABLES -A FORWARD -m state --state NEW -j forward_new        && \
$IPTABLES -A FORWARD -m limit --limit 3/minute --limit-burst 3      \
   -j LOG --log-level DEBUG --log-prefix "FORWARD packet died: " && \

if [ $DEBUG -eq 1 ]; then success; fi || \
{ if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }

#
# 5.2.1.6 Cadeia OUTPUT
#

######
# 5.2.2 Tabela nat
#

#
# 5.2.2.1 Política
#

#
# 5.2.2.2 Criaçao das cadeias especificadas pelo usuário
#

#
# 5.2.2.3 Criação das regras das cadeias especificadas pelo usuário
#

#
# 5.2.2.4 Cadeia PREROUTING
#

#
# 5.2.2.5 Cadeia POSTROUTING
#

#
# 5.2.2.6 Cadeia OUTPUT
#

######
# 5.2.3 Tabela mangle
#

#
# 5.2.3.1 Política
#

# 5.2.3.2 Criaçao das cadeias especificadas pelo usuário
#

#
# 5.2.3.3 Criação das regras das cadeias especificadas pelo usuário
#
```

```
    #
    # 5.2.3.4 Cadeia PREROUTING
    #

    #
    # 5.2.3.5 Cadeia INPUT
    #

    #
    # 5.2.3.6 Cadeia FORWARD
    #

    #
    # 5.2.3.7 Cadeia OUTPUT
    #

    #
    # 5.2.3.8 Cadeia POSTROUTING
    #

    ######################################################################
    # 5.3 Configuração do /proc
    #

    if [ $DEBUG -eq 1 ]; then
        echo -n $"Configurando /proc:"
    fi

    #
    # 5.3.1 Configuração necessária do /proc
    #

    # Habilitando o roteamento de pacotes
    echo "1" > /proc/sys/net/ipv4/ip_forward          && \

    #
    # 5.3.2 Configuração desnecessária do /proc
    #

    #echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter && \
    #echo "1" > /proc/sys/net/ipv4/conf/all/proxy_arp && \
    #echo "1" > /proc/sys/net/ipv4/ip_dynaddr         && \
    if [ $DEBUG -eq 1 ]; then success; fi || \
    { if [ $DEBUG -eq 1 ]; then failure; fi; return 1; }
}

##########################################################################
# 6. Parte principal
#

DEBUG=0
if [ "$2" == "debug" ]; then
    DEBUG=1
fi
```

```
case "$1" in
  start)
    if [ $DEBUG -eq 0 ]; then
        echo -n $"Iniciando iptables:"
    fi
    check && \
    start && \
    if [ $DEBUG -eq 0 ]; then success; fi || \
    { if [ $DEBUG -eq 0 ]; then failure; fi; exit 1; }
    ;;

  stop)
    if [ $DEBUG -eq 0 ]; then
        echo -n $"Parando iptables:"
    fi
    check && \
    stop && \
    if [ $DEBUG -eq 0 ]; then success; fi || \
    { if [ $DEBUG -eq 0 ]; then failure; fi; exit 1; }
    ;;

  restart)
    if [ $DEBUG -eq 0 ]; then
        echo -n $"Iniciando iptables:"
    fi
    check && \
    start && \
    if [ $DEBUG -eq 0 ]; then success; fi || \
    { if [ $DEBUG -eq 0 ]; then failure; fi; exit 1; }
    ;;

  status)
      check
    tables=‘cat /proc/net/ip_tables_names 2>/dev/null‘
    for table in $tables; do
        echo
        echo -en "\\033[1;32m"
        echo $"TABELA: $table"
        echo -en "\\033[0;39m"
        $IPTABLES -t $table -L -n
    done
    ;;

  panic)
    # Definindo política das cadeias para DROP
    echo -n $"Definindo políticas para DROP:"
    set_policy DROP && success || \
    { failure; exit 1; }

    # Esvaziando todas as cadeias pré-existentes
    echo -n $"Esvaziando cadeias:"
    flush_chains && success || \
    { failure; exit 1; }

    # Removendo cadeias previamente definidas pelo usuário
    echo -n $"Removendo cadeias definidas pelo usuário:"
```

```
    rm_chains && success || \
    { failure; exit 1; }
    ;;

  save)
    echo -n $"Salvando regras atuais em $IPTABLES_CONFIG: "
    touch $IPTABLES_CONFIG                                              && \
    chmod 600 $IPTABLES_CONFIG                                          && \
    /sbin/iptables-save -c > $IPTABLES_CONFIG  2>/dev/null             && \
    success || \
    { failure; exit 1; }
    ;;

  restore)
    echo -n $"Recuperando regras de $IPTABLES_CONFIG: "
    # Checando se o IPTABLES_CONFIG existe e é legível
    [ -r $IPTABLES_CONFIG ]                                             && \
    /sbin/iptables-restore -c < $IPTABLES_CONFIG  2>/dev/null          && \
    success || \
    { failure; exit 1; }
    ;;

  *)
    echo $"Usage: $1 {start|stop|restart|status|panic|save|restore} [debug]"
    exit 1
esac
exit 0
```

Apêndice B

Script de Mudança de Senha

```
#/bin/bash
echo "##################################################################"
echo "# Por medida de segurança, é necessário que todos os usuários    #"
echo "# mudem a senha do servidor da zona desmilitarizada. Essa        #"
echo "# mudança só será feita uma vez, por isso escolha uma senha      #"
echo "# forte. Uma senha forte pode ser, por exemplo, as iniciais de   #"
echo "# uma frase fácil de ser memorizada.                             #"
echo "# Exemplo:                                                       #"
echo "#    Frase: José da Silva é o nome do vizinho !                  #"
echo "#    Senha: JSeondv!                                             #"
echo "#                                                                #"
echo "# Os requisitos para a senha são:                                #"
echo "#    - Tamanho mínimo de 8 caracteres                            #"
echo "#    - Pelo menos uma letra maiúscula                            #"
echo "#    - Pelo menos uma letra minúscula                            #"
echo "#    - Pelo menos um caracter especial (ex.: ;:!@#$%&*,.<>)      #"
echo "#                                                                #"
echo "# Não serão aceitas senhas que não atendam aos requisitos.       #"
echo "#                                                                #"
echo "# Após a mudança de senha, escolha seu interpretador de comando  #"
echo "# (shell) preferido.                                             #"
echo "##################################################################"
echo
if /usr/bin/passwd; then
    while true; do
        echo
        echo "Tipos de shell"
        echo "[1] bash"
        echo "[2] tcsh"
        echo "[3] csh"
        echo "[4] sh"
        echo -n "Escolha seu shell [1]: "
        read choice
        case "$choice" in
```

```
            1|"")
            SHL=/bin/bash
            ;;
            2)
            SHL=/bin/tcsh
            ;;
            3)
            SHL=/bin/csh
            ;;
            4)
            SHL=/bin/sh
            ;;
            *)
            continue
            ;;
        esac
        /usr/bin/chsh -s "$SHL" && break
    done
    exec "$SHL" -i
fi
```

Apêndice C

Patch do Módulo pam_cracklib

```
--- pam_cracklib.c 2002-08-18 18:27:12.000000000 -0400
+++ pam_cracklib.c 2002-08-18 18:27:22.000000000 -0400
@@ -142,19 +142,19 @@
   opt->min_length = CO_MIN_LENGTH_BASE;
   } else if (!strncmp(*argv,"dcredit=",8)) {
       opt->dig_credit = strtol(*argv+8,&ep,10);
-     if (!ep || (opt->dig_credit < 0))
+      if (!ep)
   opt->dig_credit = 0;
   } else if (!strncmp(*argv,"ucredit=",8)) {
       opt->up_credit = strtol(*argv+8,&ep,10);
-     if (!ep || (opt->up_credit < 0))
+      if (!ep)
   opt->up_credit = 0;
   } else if (!strncmp(*argv,"lcredit=",8)) {
       opt->low_credit = strtol(*argv+8,&ep,10);
-     if (!ep || (opt->low_credit < 0))
+      if (!ep)
   opt->low_credit = 0;
   } else if (!strncmp(*argv,"ocredit=",8)) {
       opt->oth_credit = strtol(*argv+8,&ep,10);
-     if (!ep || (opt->oth_credit < 0))
+      if (!ep)
   opt->oth_credit = 0;
   } else if (!strncmp(*argv,"use_authtok",11)) {
   opt->use_authtok = 1;
@@ -294,23 +294,39 @@
   * defaults cause the effect to be the same as before the change
   */

-  if (digits > opt->dig_credit)
+  if (opt->dig_credit >= 0 && digits > opt->dig_credit)
      digits = opt->dig_credit;
```

```
-  if (uppers > opt->up_credit)
+  if (opt->up_credit >= 0 && uppers > opt->up_credit)
      uppers = opt->up_credit;

-  if (lowers > opt->low_credit)
+  if (opt->low_credit >= 0 && lowers > opt->low_credit)
      lowers = opt->low_credit;

-  if (others > opt->oth_credit)
+  if (opt->oth_credit >= 0 && others > opt->oth_credit)
      others = opt->oth_credit;

   size = opt->min_length;
-  size -= digits;
-  size -= uppers;
-  size -= lowers;
-  size -= others;
+
+        if (opt->dig_credit >= 0)
+            size -= digits;
+        else if (digits < opt->dig_credit * -1)
+            return 2;
+
+        if (opt->up_credit >= 0)
+            size -= uppers;
+        else if (uppers < opt->up_credit * -1)
+            return 3;
+
+        if (opt->low_credit >= 0)
+            size -= lowers;
+        else if (lowers < opt->low_credit * -1)
+            return 4;
+
+        if (opt->oth_credit >= 0)
+            size -= others;
+        else if (others < opt->oth_credit * -1)
+            return 5;

  if (size <= i)
  return 0;
@@ -331,6 +347,7 @@
 {
  const char *msg = NULL;
  char *oldmono, *newmono, *wrapped;
+        int  simpleres;

  if (strcmp(new, old) == 0) {
         msg = "is the same as the old one";
@@ -352,8 +369,19 @@
  if (!msg && similar(opt, oldmono, newmono))
  msg = "is too similar to the old one";

-if (!msg && simple(opt, old, new))
-msg = "is too simple";
+        simpleres = simple(opt, old, new);
+        if (!msg && simpleres) {
```

```
+                if (simpleres == 2)
+                        msg = "needs numeric characters";
+                else if (simpleres == 3)
+                        msg = "needs upper case characters";
+                else if (simpleres == 4)
+                        msg = "needs lower case characters";
+                else if (simpleres == 5)
+                        msg = "needs other characters (e.g., ;:!@#$%&*,.<>)";
+                else
+                        msg = "is too Simple";
+        }

  if (!msg && strstr(wrapped, newmono))
  msg = "is rotated";
```