Implementação de um *Firewall* em Linux utilizando *Iptables*

Júnior Mateus Dias, Luís Augusto Mattos Mendes, Elio Lovisi Filho, Eduardo Macedo Bhering

Departamento de Ciência da Computação – DCC
Universidade Presidente Antônio Carlos – UNIPAC

junior.mateus@terra.com.br, lmendes@email.it, professor-elio@nextwave.com.br, bhering@unipac.br

Resumo. *Este artigo apresenta uma abordagem sobre a implementação de um firewall desenvolvido utilizando totalmente a filosofia de software livre e implementado sobre a plataforma Linux. As regras utilizadas na construção do mesmo, foram criadas para a aplicação específica de uma empresa.*

1. Introdução

O processo evolutivo na área tecnológica tem ocorrido rapidamente, fazendo com que a troca de informações em tempo real torne-se uma realidade sendo mais utilizada a cada dia. Neste contexto, a segurança das informações passou a ser a palavra chave para a maioria das empresas e pessoas que de alguma forma interagem neste ambiente.

A definição de segurança relacionada a um *firewall*, pode ser baseada em três princípios básicos como apresentado na Figura 1: confidencialidade, integridade e disponibilidade de recursos do sistema em questão. Confidencialidade requer que as informações sejam acessadas somente por pessoas com permissões para tal; integridade refere-se à permanência intacta e inalterada da informação na ocorrência de acidentes ou ataques, e disponibilidade requer que o sistema funcione de modo adequado, fornecendo os recursos necessários no momento desejado.

Figura 1 – Representação dos princípios básicos de segurança.

Desta forma um sistema computacional que não forneça os recursos necessários é considerado ineficaz. Sabe-se que a segurança é obtida a custo de conveniência e facilidade de uso dos sistemas. Pode-se afirmar que a facilidade na conectividade de computadores é inversamente proporcional ao nível de segurança [1].

Mesmo possuindo muitos pontos em comuns, o conceito de segurança em sistemas de computadores é extremamente flexível, visto que há diversos fatores e características. A segurança deve ser interpretada não como um todo, mas sim como um conjunto de ações, as quais permitem a proteção dos dados e recursos de acesso restrito. Os sistemas de filtragem, controle de acesso e auditoria, tornam-se importantes no conjunto de ações em um sistema de segurança. Este sistema integrado de proteção é denominado de "*Firewall*".

Alguns conceitos e definições importantes, usados ao longo deste texto:

- *Host*: um computador anexado a uma rede;
- Filtragem de pacotes (*packet filtering*) [2]: controle seletivo do fluxo de dados de, e para, uma rede. Filtro de pacotes permitem ou bloqueiam pacotes, geralmente enquanto são roteados de uma rede para outra (*Internet* para rede local e vice-versa). Pode-se estabelecer um conjunto de regras para se especificar que tipos de pacotes serão permitidos e/ou bloqueados. Esse tipo de filtragem pode ocorrer em um roteador, em uma *bridge*, ou em um *host*. Para construir as regras é necessário conhecer algumas informações encontradas nos cabeçalhos dos pacotes, como: endereço IP de origem e destino, protocolo (TCP, UDP, ICPM), portas de origem e destino (UDP, TCP) ou tipo de mensagem ICPM. O componente que faz a filtragem de pacotes pode utilizar também as interfaces de entrada e saída de pacotes.
- Mascaramento (*masquerading*): faz o mascaramento de endereços IP de uma rede (geralmente endereços reservados na *Internet*, ou seja, inválidos) para um único endereço IP (endereço válido na *Internet*). Com isso pode-se compartilhar uma única conexão de *Internet* com várias outras máquinas de uma rede local.

Estas são algumas definições básicas a respeito de *Firewall*. O artigo em questão tem como objetivo, a implementação de um *firewall* utilizando totalmente a idéia de *software* livre.

2. *Firewall*

Existem alguns fatores que tornam muito difícil evitar ataques a um sistema. Em geral a maioria possui algum tipo de "furo de segurança" conhecido também por "*bug*", que permitirá ataques bem sucedidos a informações confidenciais. Diante desta situação é interessante a utilização de um modelo de segurança de rede unificado (*Network Security Model*), o qual consiste em controlar o acesso de rede para vários *host's* e serviços ao mesmo tempo, ao invés de controle individualizado.

O *Firewall* é definido como uma barreira de proteção, que controla o tráfego de dados entre um computador, ou uma rede, e a *Internet*. Seu objetivo é permitir somente

a transmissão e a recepção de dados previamente autorizados. Utilizado para filtragem e NAT (*Network Adress Translation*), logicamente separa, restringe e analisa datagramas IP e podendo ser fisicamente um *hardware* dedicado, roteador, computador ou uma combinação destes.

Todo o tráfego entre a *Internet* e a rede privada pode ser direcionado ao *Firewall*. Devido a isso, o mesmo tem a capacidade de verificar se esse tráfego é aceitável ou não, conforme a política de segurança.

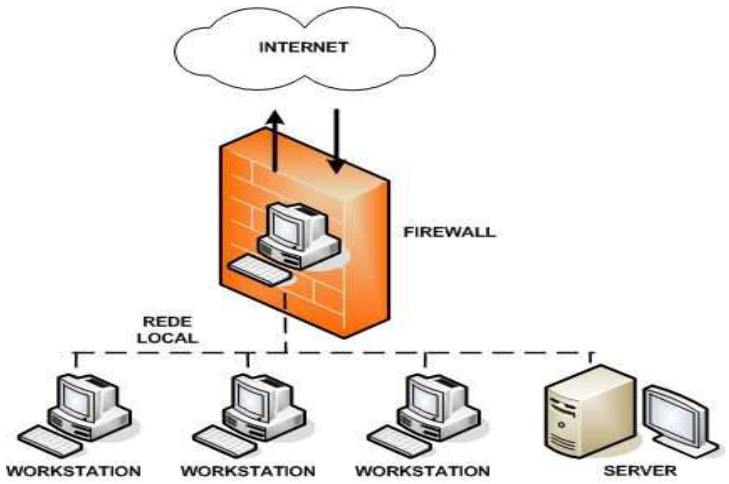

Figura 2 – Funcionamento básico de um *Firewall.*

Em síntese, o *Firewall* é um sistema que faz a intercomunicação entre duas ou mais redes, geralmente a *Internet* e a rede interna, sendo utilizado para proteger a rede como demonstrado pela Figura 2.

Temos disponíveis no mercado e na *Internet* vários mecanismos que fazem a filtragem de pacotes das mais variadas formas. Entretanto, os conceitos envolvem basicamente três tipos de filtragem:

- *Stateless* ou estática: são as filtragens mais simples e que consomem mais recursos dos dispositivos. Cada pacote é analisado de forma independente, sem nenhuma associação com possíveis pacotes que já foram processados.

- *Stateful* ou dinâmica: são filtragens mais refinadas e que oferecem um desempenho visivelmente melhor do que a filtragem anterior. Nesta filtragem há conhecimento de conexões, cada pacote é analisado e associado (ou não) a uma conexão já existente. Este processo permite que os pacotes associados a conexões estabelecidas passem automaticamente, diminuindo o *overhead*[1] de análise e ação sobre cada pacote.

[1] ***overhead***: custo adicional em processamento ou armazenamento que, como conseqüência, piora o desempenho de um programa ou dispositivo de processamento.

- ***Proxy*:** são filtragens complexas, e lentas, que atuam predominantemente nos protocolos de aplicação. A velocidade está relacionada à quantidade de demultiplexação que é necessária para análise em um simples pacote de rede.

3. *Free Softwares* para construção de *Firewalls*

A seguir são apresentados alguns *softwares* livres para construção de um *Firewall* utilizando Linux:

- ***Ipfwadm:*** o IP *firewall administration*, ou simplesmente *ipfwadm* foi a ferramenta padrão para construção de regras de *firewall*, para o *kernel* anterior à versão 2.2.0. Extremamente complexo e com regras de difícil implementação tornou-se inviável.
- ***Ipchains:*** O *ipchains* foi a solução feita para o *kernel* 2.2. A idéia do *ipchains* foi ter o poder de *ipfwadm*, mas com uma simplicidade e facilidade no que diz respeito à criação de regras. Seu objetivo era prover facilidades, e criar uma compatibilidade com o *ipfwadm* através do utilitário *ipfwadm-wrapper*.
- ***Iptables:*** A nova geração de ferramentas de *firewall* para o *kernel* 2.4 do Linux. Além de possuir as facilidades do *ipchains*, e implementar a facilidade do NAT, possui filtragens de pacotes mais flexíveis.

4. Fluxo do *Kernel* x *Netfillter*

O k*ernel* do Linux a partir da versão 2.4 sofreu uma grande. O objetivo no desenvolvimento do *Netfilter* foi criar um *firewall* menos complicado, para isso tratou de simplificar os pontos do *kernel* que dizem respeito ao processamento de datagramas e produziu um código muito mais limpo e flexível. Ele chamou essa nova estrutura de *Netfilter*.

O *Netfilter* acabou com a rigidez e a complexidade das velhas soluções de *firewall*, implementando um esqueleto genérico no *kernel*, que possibilita a alteração das políticas de filtragem sem a necessidade de se alterar o *kernel*. A Figura 3, abaixo, representa de forma detalhada o diagrama de fluxo do *Netfilter*.

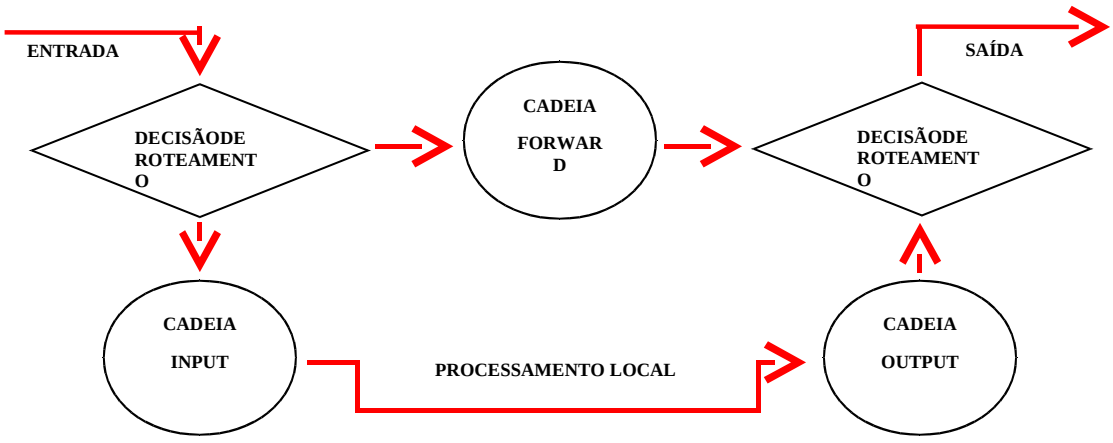

Figura 3 – Diagrama de fluxo do *Netfilter*.

O *Netfilter* é um *framework*[2] dentro do *kernel* Linux com o qual o módulo do *iptables* pode conectar-se.

Na estrutura acima, os três "círculos" representam as cadeias (*chains*). Quando o pacote atinge um círculo no diagrama, a cadeia de regras é examinada a fim de decidir o destino do pacote, ou alvo do pacote (*target*). Se o destino for para descartar (*DROP*) o pacote, ele é descartado sem envio de mensagem ICMP, se o destino for para rejeitar (*REJECT*) o pacote, ele é rejeitado e uma mensagem ICMP é enviada, mas, se o destino for para aceitar (*ACCEPT*) o pacote, ele então continua a viajar no diagrama.

Uma cadeia (*chain*) é uma lista de regras. O cabeçalho do pacote é comparado com cada regra até que encontre uma que case com seu cabeçalho, então o alvo da regra é aplicado no destino do pacote (*DROP*, *REJECT* ou *ACCEPT*). Caso não haja nenhuma regra que case com o cabeçalho do pacote, ele é submetido a política padrão do *kernel*, que por questões de segurança deve ser sempre descartar (*DROP*).

Em etapas pode-se descrever o fluxo desta forma:

1. Quando o pacote chega (pela placa *Ethernet*, por exemplo) o *kernel* analisa o destino do pacote: isso é chamado de roteamento (*routing*).

2. Se o destino do pacote é a própria máquina, o pacote desce no diagrama, indo para a *chain INPUT*. Onde então é filtrado pelas regras da cadeia *INPUT*. Caso o pacote consiga passar pelas regras da cadeia *INPUT*, então a máquina recebe o pacote.

3. Se o *kernel* não tiver suporte a encaminhamento (*forwarding*), qualquer pacote com destino diferente da própria máquina será descartado. Caso haja suporte a encaminhamento e o pacote é destinado a outra *interface* de rede (se possuir outra), o pacote vai para cadeia *FORWARD* (encaminhamento). Se o pacote for aceito por alguma regra da cadeia, ele continua seu caminho, caso contrário é descartado.

4. Qualquer pacote originário da própria máquina passa pela cadeia *OUTPUT* (saída). O cabeçalho do pacote é examinado e comparado com as regras da cadeia *OUTPUT*. Sendo aceito o pacote sai da *interface* e segue seu destino. Caso contrário é descartado.

5. *Iptables*

O *Iptables* é um *firewall* no nível de pacote que funciona através de comparação de regras para saber se um pacote tem ou não permissão de passar, também pode ser utilizado para modificar e monitorar o tráfego da rede, fazer NAT (*masquerading*, *source nat, destination nat*), redirecionamento de pacotes que chegam e saem do

[2] ***Framework***: no desenvolvimento de *software*, um *framework* é uma estrutura de suporte definida, em que um outro projeto de *software* pode ser organizado e definido.

sistema, contagem de *bytes*, dividir o tráfego entre máquinas, criar proteções anti-*spoofing*[3] e contra *syn flood, DoS*.

O filtro de pacotes do *kernel* 2.4.x e 2.6.x funcionam por meio de regras estabelecidas pelo sistema operacional. Todos os pacotes entram no *kernel* para serem analisados. As *chains* (correntes) são as situações possíveis dentro do *kernel*. Quando um pacote entra no *firewall*, o *kernel* verifica o destino dele e decide qual *chain* manipulará esse pacote. Isso é chamado de *roteamento* interno. Os tipos de *chains* irão depender da tabela que está sendo utilizada no momento.

O *iptables* é um *firewall* com estado, ou seja, um *firewall stateful*. Os anteriores eram *stateless*. O modo de filtragem *'Stateless'* tende a tratar cada pacote roteado pelo *firewall* como pacotes individuais, sendo mais simples de implementar e por terem uma resolução mais rápida que um do tipo *stateful*, podem ser usados para obterem um desempenho melhor em situações onde existem regras de nível de rede bem simples.

O tipo de filtragem *stateful* (*IPTABLES*) cria um poderoso sistema de *firewall* que se "lembra" das conexões entrantes, evitando ataques do tipo *Stealth Scans*[4], que trazem *flags* especiais para técnicas de *port scanning*.

6. Vantagens do *Kernel* 2.4 *Netfilter* e *Iptables*

O novo sistema de filtragem de pacotes, *Netfilter* [3], é o primeiro sistema de *stateful firewalling* no Linux. Entre muitos melhoramentos, permite ao *Netfilter* detectar e bloquear muitos *steath scans*, que não eram anteriormente detectados nos *firewall's* do Linux.

6.1 Stateful Firewalling

Packet filters normais, como os presentes na maioria dos roteadores e *firewall's* antigos, inspecionam cada pacote individualmente, sem o uso de memória ou verificação da relação desse pacote com conexões já existentes. Em uma conexão TCP, esses *packet filters* diferenciam pacotes de conexões já existentes, de outros que fazem parte de uma nova conexão, apenas verificando se o pacote possui a *flag* SYN ajustada. Isso significa que um computador pode criar um pacote com *flags* especificadas de forma diferente, e passar como pacote de uma conexão já existente, de forma a contornar o *firewall* e executar um *scan* na rede interna, por exemplo.

Ao contrário da situação descrita acima, um *stateful firewall* possui o controle de cada conexão que passa através dele. Portanto, se um pacote forjado tentar contornar o *firewall*, como parte de uma conexão já existente, o *firewall* irá consultar sua lista de conexões e verificar se esse pacote pertence ou não a essa conexão, realizando através deste conceito a defesa de muito ataques de *scans*.

---

[3] ***Spoofing***: ato de falsificar o remetente de um pacote de transmissão de dados, para que o receptor o trate como se fosse de um outro utilizador.

[4] ***Steath Scans***: método de escaneamento de portas em que o escaneador manda pacotes com opções que são muito menos possíveis de serem registradas por um *firewall* que pacotes SYN normais.

Stateful firewall é um grande aperfeiçoamento para os *firewall's*, mas a conveniência e o poder de possuir o controle sobre as conexões (*connection tracking*) gera a inconveniência do aumento no consumo de memória. Por isso geralmente deve-se utilizar dois mecanismos de filtragem: um dispositivo comum (roteador/*firewall*) para bloquear o tráfego que realmente não é permitido, e um *stateful firewall* para lidar com o resto.

6.2 Network Address Translation (NAT)

NAT é um processo de converter um ou mais endereços IP para outro endereço IP [5]. Um uso mais comum disso é o IP *Masquerading*, em que uma rede interna pode ser "mascarada" em um único endereço IP, que seja válido na *Internet*. O novo código de filtragem de pacotes do *Kernel* 2.4 possui uma forma mais robusta de conversão de endereços: suporta "uma-para-uma", "uma-para-muitas" e "muitas-para-muitas" conversões de portas e endereços.

Opções de NAT no *Netfilter* [5]:

- *Destination* NAT (DNAT): esse tipo de NAT especifica que endereço de destino do pacote deve ser modificado, ou seja, mudar o destino da conexão.
- *Source* NAT (SNAT): esse tipo de NAT especifica que o endereço de origem do pacote deve ser modificado, ou seja, mudar de onde a conexão esta vindo.

Pacotes que sofreram tanto DNAT como SNAT, serão novamente convertidos quando fizerem o caminho inverso, para garantir que sejam entregues de volta ao *host* com IP real.

6.3 Proteção contra Negativa de Serviço (Deny of Service – DoS)

Uma boa defesa contra certos tipos de DoS, mais especificamente os que ocorrem com inundações de pacotes SYN, seria limitar a quantidade de pacotes SYN de uma única origem. Isso torna possível ao *Netfilter* possuir implementado um sistema que pode limitar quantos pacotes são aceitos em um intervalo de tempo definido. Essa inovação chama-se *rate limiting*, e pode ser utilizada também para evitar que certo serviços como *Web services* e *E-mail services* não ocupem toda a banda.

6.4 Proteção contra Stealth Scan

Existem certo tipos de pacotes que transpassam muitos *firewall's* atingindo seus *host's* internos, sem serem detectados. Podemos definir uma combinação desses pacotes como *stealth scan*. Esses pacotes englobam tudo, desde um pacote ACK, que se passa como parte de uma conexão, pacotes XMAS, que possui todas as *flags* do TCP habilitadas, até pacotes NULL, que não possui nenhuma *flag* habilitada. Alguns desses pacotes podem não fazer nada, mas conseguem que o computador atingido responda a algum, permitindo um *scan* na rede interna, sem ser detectado. O novo *Netfilter*, do *kernel* 2.4, pode filtrar qualquer combinação de *flags* TCP, ao contrário dos antigos *kernel* 2.0 e 2.2 que conseguiam filtrar apenas pacotes SYN.

6.5 Tabelas do Netfilter

Com o *Netfilter* é introduzido o conceito de tabelas (*tables*), foram criadas três tabelas chamadas: *filter*, *nat* e *mangle*.

A) Tabela Filter

É a tabela padrão do *Netfilter* e trata das situações implementadas por um *firewall* filtro de pacotes. Essas situações são [7]:

- INPUT: Tudo que entra no *host*.
- FORWARD: tudo que chega ao *host* mas deve ser redirecionado a um *host* secundário ou outra *interface* de rede.
- OUTPUT: tudo que sai do *host*.

B) Tabela NAT

Esta tabela implementa funções de NAT ao *host firewall*. O NAT por sua vez, possui diversas utilidades, conforme abordado anteriormente. Suas situações são [7]:

- *PREROUTING*: utilizada quando há necessidade de se fazer alterações em pacotes antes que os mesmo sejam roteados.
- *OUTPUT*: trata os pacotes emitidos pelo *host firewall*.
- *POSTROUTING*: utilizado quando há necessidade de se fazer alterações em pacotes após o tratamento de roteamento.

C) Tabela Mangle

Implementa alterações especiais em pacotes em um nível mais complexo. A tabela *mangle* é capaz de alterar a prioridade de entrada e saída de um pacote baseado no tipo de serviço (TOS) o qual o pacote se destinava. Suas situações são: [7]

- *PREROUTING*: modifica pacotes antes de serem roteados, dando-lhes um tratamento especial.
- *OUTPUT*: altera pacotes de forma “especial” gerados localmente após o roteamento.

7. Trabalhando com *Iptables*

O *Iptables* é um software utilizado para analisar os pacotes que passam entre redes. A partir desse princípio podemos aceitar, rejeitar ou descartar esses pacotes. Através de um método de controle de acesso baseado em registros e tabelas presentes no *kernel*, pode-se monitorar e registrar qualquer informação que está sendo transmitida.

7.1 Configuração do *Kernel*

Ao se utilizar e implementar regras para um *firewall* em Linux utilizando *Iptables*, alguns pacotes do *kernel* dever estar disponíveis. Para que o *Iptables* funcione, é necessário ter algumas opções do *kernel* configuradas como mostra o Anexo 1:

Essas opções são escolhidas dependendo do tipo de *firewall* ou aplicação a ser construído. São necessárias que outras opções no *kernel* sejam habilitadas, como as que configuram o IP, ou as *interfaces* de rede. Elas podem ser compiladas como módulos, ou integradas ao *kernel*. Se forem compiladas como módulo, devem ser carregadas antes de usar a aplicação.

7.2 Fluxo de pacotes no *Iptables*

Para definir as regras de filtragem, tais como NAT, *masquerading*, dentre outras, é necessário saber como funciona o fluxo de pacotes no *Iptables*. O *Iptables* lida com o conceito de *firewall chains*, ou apenas *chains*, que são as listas de regras nas tabelas de filtragens, NAT e *mangle*.

O *kernel* inicia com três *chains* na tabela "*filter*", são elas: *INPUT*, *OUTPUT*, *FORWARD*. Existe também a tabela "NAT", que lida com as *chains PREROUTING*, *POSTROUTING* e *OUTPUT*, e a tabela "*mangle*" que lida com as *chains PREROUTING* e *OUTPUT*.

Essas *chains* estão relacionadas como mostrado na Figura 04:

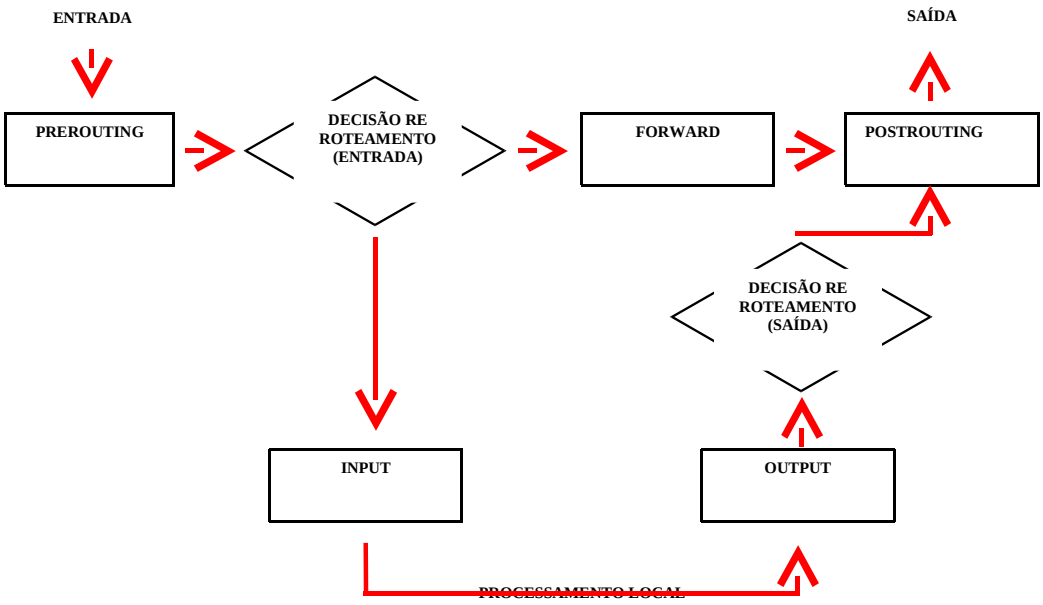

Figura 04 – Relacionamento entre as *chains* [6].

1. *PREROUTING*: por onde todos os pacotes entram no *firewall*. É válido apenas para *interfaces* de entrada.
    - *connection track*: verifica se o pacote de entrada está relacionado com algum item já existente na sua lista de conexões.
    - *Mangle*: usado para mudar informações nos pacotes que estão chegando (*Type of Service*)
    - DNAT: usado para mudar o destino do pacote. Isso só acontece com o primeiro pacote da conexão. O resto dos pacotes da conexão é tratado pelo *connection track*.
2. Decisão de roteamento (entrada): decide se o pacote é destinado para o próprio *firewall* ou se vai ser destinado para alguma *interface* de saída.
3. *INPUT*: serve para pacotes que vem da interface de entrada e que são destinados apenas ao *firewall*.
    - *filter*: possui as regras para proteger o *firewall* do mundo externo.
4. *FORWARD*: serve para pacotes que atravessam o *firewall*, mas não sendo usados atualmente dentro do próprio *firewall*.
    - *filter*: regras para proteger redes internas do mundo externo e para controlar o tráfego entre essas redes que passam pelo *firewall*.
5. *OUTPUT*: processa os pacotes que são criados no firewall.
    - *connection track*: faz o armazenamento das informações da nova conexão.
    - *mangle*: adiciona informações (TOS) aos pacotes que saem.
    - *filter*: regras de filtragem de pacotes que saem do *firewall*.
6. Decisão de roteamento (saída): "roteia" o pacote para a interface de saída apropriada. Apenas para pacotes vindos do próprio *firewall*.
7. *POSTROUTING*: por onde os pacotes deixam o *firewall*. É válido apenas para *interfaces* de saída. Toda filtragem acontece antes de chegar em *POSTROUTING*.
    - *SNAT/MASQUERADE*: o primeiro pacote de uma conexão de saída tem seu endereço: porta de origem alterada.
    - *connection track*: faz o armazenamento das informações da nova conexão.

Como foi dito anteriormente, uma *chain* é uma lista de regras. Cada regra especifica o que fazer com o pacote. Isso é feito através do cabeçalho do pacote. Se a regra não coincide com o pacote, a próxima é consultada até que não haja mais regras. Se nenhuma regra coincidiu com o pacote, então o *kernel* verifica a política da *chain* para decidir o que fazer.

Existem duas políticas: *ACCEPT* e *DROP*. Quando as *chains* são iniciadas pelo *kernel*, a política é *ACCEPT*, mas em um sistema seguro, essa política deve ser alterada para *DROP*.

7.3 Criando Regras

- ***Manipulando chains***

# *iptables* –N "*chain*"	Cria uma nova *chain*.
# *iptables* –X "*chain*"	Apaga uma *chain* vazia.
# *iptables* –P "*chain*"	Muda a política de uma *chain*.
# *iptables* –L "*chain*"	Lista as regras de uma *chain*.
# *iptables* –F "*chain*"	Apaga todas as regras de uma *chain*.
# *iptables* –Z "*chain*"	Zera contadores de pacotes de todas regras em uma *chain*.
# *iptables* –A "*chain*"	Adiciona uma nova regra em um *chain*.
# *iptables* –I "*chain*"	Insere uma nova regra em alguma posição de uma *chain*.
# *iptables* –D "*chain*"	Apaga alguma regra em uma cadeia.

Tabela 01 – Manipulação de *Chains*

8. Estudo de caso de um Firewall.

A implementação do *firewall* em questão tem como objetivo proteger uma única rede, onde há conexão direta com o *link* da EMBRATEL, e conexão de *Internet*. A Figura 05 representa o esquema proposto:

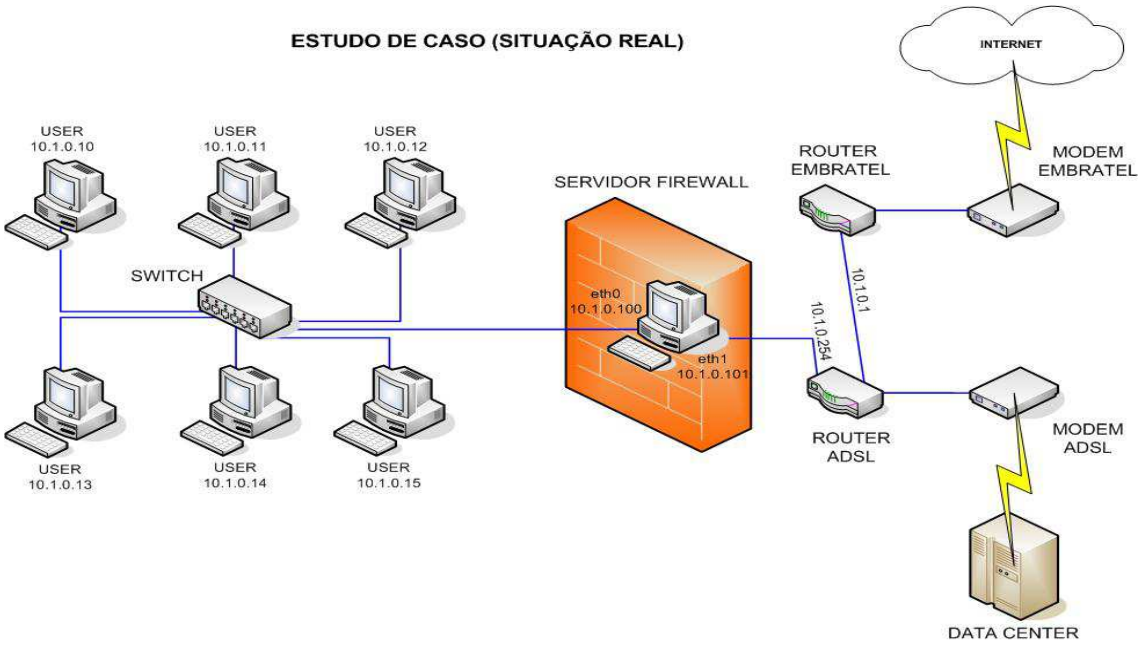

Figura 05 – Representação do problema proposto.

- ***Especificação:***

  Nesta aplicação, o servido firewall possui duas *interfaces* de rede:

  - eth0: 10.1.0.100
  - eth1: 10.1.0.101 – *Gateway* 10.1.0.254

  As portas descritas abaixo devem estar liberadas:

  - FTP (21)
  - WWW (80)
  - MAIL/SMTP (25)
  - MAIL/POP3 (110)
  - SSH (22)
  - APLICAÇAO SAP (3600)

Uma possível solução para o estudo de caso apresentado acima está apresentada no Anexo II.

10. Conclusão

Firewall's são hoje em dia ferramentas de segurança essenciais para proteger uma rede ou até mesmo computadores domésticos, pois a cada dia surgem novas técnicas de obstruir a segurança de sistemas. E na questão de dificuldade de utilização e eficiência em segurança para Linux, o *Netfilter* e *Iptables* são uma ótima escolha, pois com pequenos scripts podemos proteger um computador, ou uma rede, contra invasões, além de utilizarmos totalmente a idéia de *software* livre.

Este artigo aborda as propriedades do sistema de *Firewall* do Linux, o *Netfilter* do *Kernel* 2.4 e 2.6, e como manipular suas funções através do Iptables. Esse sistema, através de suas propriedades de *stateful firewalling* e IP *connection tracking*, aumenta consideravelmente a capacidade do *Firewall*, com eficiência, simplicidade nas regras, e principalmente, segurança.

Visto do ponto de vista teórico, a solução apresentada neste artigo vem solucionar de forma concreta a criação de um *firewall* com regras específicas. Estas regras foram implementadas de acordo com as necessidades e possibilidades de acessos existentes na empresa.

Visto que, a constante busca pela segurança faz parte do cenário atual no desenvolvimento tecnológico e a necessidade de novas ferramentas e idéias neste contexto se torna um aspecto de alta prioridade, deixo como sugestão de trabalhos futuros as seguintes propostas:

- Implementação prática desta solução com tentativas reais de invasão;
- Análise e comparação dos recursos e soluções *Iptables* x *Packet Filtering* comerciais.

11. Referências Bibliográficas

[1] Simsom Garfinkel & Gene Spafford. Pratical Unix and Internet Security – 2nd Edition. O'Reilly and Associates, Sabastopol, Califórnia, 1996.

[2] Network (In)Security Through IP Packet Filtering, D. Brent Chaoman.
http://www.greatcircle.com/pkt_filtering
Acesso em 22/10/05.

[3] Linux Kernel 2.4 Firewalling Matures: netfilter, Dave Wreski.
http://www.linuxsecurity.com/feature_stories/netfilter-print.html
Acesso em 08/10/2005.

[4] Linux 2.4 NAT HOWTO, Rusty Russel.
http://www.netfilter.org/documentation/HOWTO/pt/NAT-HOWTO.html
Acesso em 20/11/05.

[5] Linux 2.4 Packet Filtering HOWTO, Rusty Russel.
http://www.netfilter.org/documentation/HOWTO/pt/packet-filtering-HOWTO.html
Acesso em 20/11/2005

[6] Iptables Tutorial, Oskar Andreasson.
http://boingworld.com/workshops/linux/iptables-tutorial
Acesso em 20/11/2005.

[7] Neto, U., Dominando Linux Firewall Iptables, Rio de Janeiro, Editora Ciência Moderna, 2004.

[8] Tanenbaum, A. S., Redes de Computadores, Rio de Janeiro, Editora Campus, 2003.

[9] Morimoto, C. E., Redes e Servidores Linux, Porto Alegre, Sul Editores, 2005.

[10] Jang, M., Dominando Had Hat Linux 9, Rio de Janeiro, Editora Ciência Moderna, 2003.

[11] Ferreira, R. E., Linux Guia do Administrador do Sistema, São Paulo, Navotec Editora, 2003.

[12] Torres, G., Redes de Computadores Curso Completo, Rio de janeiro, Editora Axcel Books do Brasil, 2001.

Anexo I

- *CONFIG_NETFILTER*: habilita a filtragem de pacotes no *kernel*. Sem isso, não é possível usar o *Iptables*.
- *CONFIG_IP_NF_CONNTRACK*: habilita o *connection tracking*, que mantém a "trilha" (*track*) das conexões e relaciona os pacotes com as conexões. Esta opção é necessária para realizar qualquer tipo de NAT.
- *CONFIG_IP_NF_FTP*: essa opção é necessária para realizar NAT e *connection tracking* em conexões de FTP.
- *CONFIG_IP_NF_IPTABLES*: adiciona o suporte para o *Iptables*, ou seja, sem ele o *Iptables* não pode ser habilitado.
- *CONFIG_IP_NF_MATCH_LIMIT*: permite o controle da taxa em que uma regra pode ser aceita. Muito útil combinado ao serviço de *LOG* e para evitar alguns tipos de DoS.
- *CONFIG_IP_NF_MATCH_MAC*: permite que regras sejam criadas utilizando o endereço MAC de origem.
- *CONFIG_IP_NF_MATCH_MULTIPORT*: permite aceitar pacotes TCP e UDP baseados em uma série de portas de origem e destino.
- *CONFIG_IP_NF_MATCH_TOS*: permite construir regras baseadas no campo de tipo de serviço (*Type of Service*) dos pacotes IP.
- *CONFIG_IP_NF_MATCH_STATE*: adiciona a possibilidade de filtrar pacotes baseado no fato de como eles são relacionados com outras conexões.
- *CONFIG_IP_NF_FILTER*: define uma tabela "*filter*" que possui uma série de regras para filtragem de pacotes na "*local imput*", *forwarding* e "*local output*".
- *CONFIG_IP_NF_TARGET_REJECT*: permite que uma conexão negada seja respondida com ICMP *error*, ao invés de apenas negar os pacotes.
- *CONFIG_IP_NF_NAT*: permite que todos os tipos de NAT, como *port forwarding, masquerading*, etc.
- *CONFIG_IP_NF_TARGET_MASQUERADE*: torna possível mudar todas as conexões de saída de uma rede interna para um único IP. Se a *interface* de saída é desabilitada todas as conexões são perdidas. Útil para serviços *dial-up* com IP dinâmico.
- *CONFIG_IP_NF_TARGET_REDIRECT*: é um caso especial de NAT, todas as conexões que chegam são mapeadas para um endereço de *interface* de entrada. Útil para *proxies* transparentes.
- *CONFIG_IP_NF_MANGLE*: adiciona uma tabela "*mangle*" para o *Iptables*: esta tabela é utilizada para alterações de pacotes que resultam no modo como o pacote é roteado.
- *CONFIG_IP_NF_TARGET_LOG*: permite criar regras com *Iptables* que geram *logs* do cabeçalho do pacote para o *syslog*.

Anexo II

```
#!/bin/sh

# O comando a seguir ativa roteamento do kernel
# ------------------------------------------------------
echo 0 > /proc/sys/net/ipv4/ip_forward

# Definição de variáveis para simplificação dos comandos
# ------------------------------------------------------
LAN_IP_NET='10.1.0.1/24'
LAN_NIC='eth1'
WAN_IP='10.1.0.254'
WAN_NIC='eth0'
FORWARD_IP='10.1.0.100'

# Ativa módulos do kernel do Linux
# ------------------------------------------------------
/sbin/modprobe iptable_nat
/sbin/modprobe ip_conntrack
/sbin/modprobe ip_conntrack_ftp
/sbin/modprobe ip_nat_ftp
/sbin/modprobe ipt_LOG
/sbin/modprobe ipt_REJECT
/sbin/modprobe ipt_MASQUERADE

Proteção contra IP spoofing
# ------------------------------------------------------
echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter

# Zera regras das três tabelas do Netfilter
# ------------------------------------------------------
iptables -F
iptables -X
iptables -F -t nat
iptables -X -t nat
iptables -F -t mangle
iptables -X -t mangle

# Determina a política padrão das cadeias
# ------------------------------------------------------
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT
iptables -t nat -F POSTROUTING
iptables -t nat -F PREROUTING
iptables -t nat -F OUTPUT
```

 # Proteção contra syn-flood
 # ---
iptables -A FORWARD -p tcp --syn -m limit --limit 2/s -j ACCEPT

 # Habilitar Masquerade e forwarding
 # ---
iptables -t nat -A POSTROUTING -s $LAN_IP_NET -j MASQUERADE
iptables -A FORWARD -j ACCEPT -i $LAN_NIC -s $LAN_IP_NET
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT

 # Abrir as portas a serem utilizadas
 # ---
iptables -A INPUT -j ACCEPT -p tcp --dport 80
iptables -A INPUT -j ACCEPT -p tcp --dport 21
iptables -A INPUT -j ACCEPT -p tcp --dport 110
iptables -A INPUT -j ACCEPT -p tcp --dport 25
iptables -A INPUT -j ACCEPT -p tcp --dport 22
iptables -A INPUT -j ACCEPT -p tcp --dport 3600

 # Aceita os pacotes que realmente devem entrar
 # ---
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

Abrir portas para servir a rede
 # ---
iptables -A FORWARD -j ACCEPT -p tcp --dport 80
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 80 -j DNAT --to 10.1.0.100:80
iptables -A FORWARD -j ACCEPT -p tcp --dport 21
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 21 -j DNAT --to 10.1.0.100:21
iptables -A FORWARD -j ACCEPT -p tcp --dport 22
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 22 -j DNAT --to 10.1.0.100:22
iptables -A FORWARD -j ACCEPT -p tcp --dport 25
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 25 -j DNAT --to 10.1.0.100:25
iptables -A FORWARD -j ACCEPT -p tcp --dport 110
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 110 -j DNAT --to 10.1.0.100:110
iptables -A FORWARD -j ACCEPT -p tcp --dport 3600
iptables -t nat -A PREROUTING -i eth0 -p tcp --dport 110 -j DNAT --to
10.1.0.100:3600