FreeBSD como Fw/Gateway de uma rede utilizando o PACKET FILTER (Pf)

Por Ataliba de Oliveira Teixeira
http://www.ataliba.eti.br

1.FreeBSD ou Openbsd ?

O Packet Filter, conhecido como pf, é o sistema utilizado pelo OpenBSD para fazer a filtragem e NAT em TCP/IP. O PF é capaz também de fazer outros tipos de serviços como controle de banda, priorização de pacotes etc. O PF está contido nos sistemas OpenBSD desde a sua versão 3.0 .

A grande vantagem do PF, sobre outros tipos de sistemas de Firewall, é possibilidade de criação de macros (que são variáveis, como em programas, definidas pelo usuário e que podem armazenar ips, interfaces, etc), tabelas (listagem de endereços que são filtrados por uma única regra), scrub (reprocessamento de pacotes para normalização e desfragmentação), NAT e regras de filtragem que podem ser feitas de acordo com o modo em que chegam as interfaces de uma máquina.

Esta tranquilidade de criação de firewalls com o PF, deu ao OpenBSD a fama de um sistema operacional preparado para a criação de firewalls. Apesar de isto ser uma grande verdade, um dos grande problemas para criação de firewalls com o OpenBSD, para o iniciante no mundo dos BSDs seria dominar o sistema.

Pela sua maior divulgação, o FreeBSD sempre foi uma opção. Sua documentação, muito rica, fornecia possibilidades de criação de firewalls a partir do ipfw. O grande e maior problema do ipfw, é, que, por não possuir a mesma usabilidade do pf, sempre fornecia mais dificuldade na criação de um firewall para o administrador de uma rede.

A partir da versão 5.0, o FreeBSD passou a integrar em seus ports, o pf. Isto forneceu a quem é fã deste sistema operacional, a possibilidade de criar firewalls com um sistema conhecido e um bom sistema de tratamento de pacotes que é o pf.

A partir daí, a mudança para o OpenBSD, aconteceria somente, se o administrador já o dominasse por completo e quisesse criar seus firewalls em um novo sistema operacional.

Em minha opinião pessoal, o FreeBSD, bem configurado, pode funcionar tão bem como o OpenBSD, para aplicações de segurança. Até, porque, um sistema operacional só é realmente seguro, se o elemento humano, administrador, souber realmente o que está fazendo.

Portanto, deve-se escolher sempre, aquele sistema operacional que conhecemos melhor para fazer certos tipos de serviços.

2.Preparando seu FreeBSD par ser um firewall :

Não há grande diferenças entre a instalação de um FreeBSD para um firewall e a instalação de um FreeBSD para outros tipos de serviços. Deve-se, sempre, haver a preocupação, principalmente no caso de um firewall, de desabilitar todos os serviços desnecessários que possam estar rodando na máquina.

E, ainda, evitar a instalação de programas que possam ser armas na mão de um invasor. Em geral, a instalação mínima do FreeBSD pode ser a melhor escolha. Tendo a mesma instalada na máquina, você já pode efetuar a configuração de um firewall com o FreeBSD.

Após a instalação é necessário fazer algumas configurações necessárias no rc.conf do seu FreeBSD.

A primeira, e mais importante tarefa, é desabilitar todos os serviços que estão rodando desnecessariamente. No /etc/inet.conf temos ident, consat, daytime, time e outros serviços ativados por padrão. Apesar de não causarem grandes problemas, evitar dor de cabeça é uma das melhores coisas para qualquer administrador. Assim, comente as linhas que contém os mesmos e os desative.

No arquivo rc.conf é interessante também efetuar o desativamento do sendmail, ntpd e portmap, já que a máquina não vai utilizá-los. O modo de fazer o desativamente destes serviços, é através das três linhas abaixo :

```
check_quotas=NO
ntpd=NO
sendmail_flags=NO
```

Após estas configurações, adicione as linhas abaixo ao seu rc.conf :

```
gateway_enable=”YES”
kern_securelevel=”1”
kern_securelevel_enable=”YES”
pf_enable=”YES”
```

A primeira linha adicionada ao rc.conf, diz ao FreeBSD que o mesmo deve permitir o tráfego de pacotes entre as suas interfaces. Ou seja, indiferente do número de interfaces de rede que você tenha instaladas na máquina, você deve conseguir trafegar seus pacotes dentro dela.

Um dos ditos grandes trunfos do OpenBSD em cima do FreeBSD é que o mesmo não permite o carregamento de módulos, ou seja, ele tem todo seu suporte

compilado monoliticamente em seu kernel. Isto é interessante, para um firewall, mas pode não ser para um sistema mais generalista.

Como o FreeBSD foi projetado para ser um sistema de diversas aplicações, ele por padrão vem com seu nível de segurança setado em -1. Isto pode ser visto, efetuando o comando abaixo :

```
sysctl kern.securelevel
kern.securelevel: -1
```

Assim, para não permitir o carregamento de módulos dinâmicos no kernel do FreeBSD o administrador simplesmente tem que escrever as duas linhas em vermelho, no primeiro quadro.

Para maiores informações sobre os níveis de segurança do FreeBSD, você pode passar uma olhada no Apêndice 1.

E, finalmente, devemos ativar o pf, através da linha pf_enable. Assim, você irá criar o device /dev/pf, que vai ser o device responsável pela filtragem dos pacotes.

De acordo com Julio Cesar Melo, em um de seus tutoriais sobre OpenBSD, é interessante setar mais duas opções. Ele cita a ativação de alta performance na transferência de dados entre hosts de acordo com o documento "**Enabling High Performance Data Transfers on Hosts**" (www.psc.edu/networking/perf_tune.html), adicione o seguinte em /etc/sysctl.conf:

```
# 3. Increase TCP Window size for increase network
performance

net.inet.tcp.recvscape=65535

net.inet.tcp.sendspace=65535
```

4.Dicas sobre o PF :

Aqui, agora, é que começaremos realmente a criação do nosso firewall. A partir das configurações efetuadas acima, você já tem uma máquina preparada para ser o firewall de sua rede. Neste tutorial, levamos em consideração, uma máquina com duas placas de rede, fazendo a ponte entre a sua rede interna e a internet.

O pf.conf, tem uma ordem que deve ser seguida. Isto pode ser visto no arquivo exemplo que se encontra no diretório /etc . Neste pf.conf, iremos ver a seguinte

ordem :

- Options
- Scrub
- NAT e RDR
- Filter

Se não existir nenhuma regra de filtro, é só permitir tudo por padrão, estas linhas inclusive se encontram no arquivo pf.conf.

A partir daqui, iremos, a partir de um firewall do Autor Shamim Mohamed (http://www.drones.com/obsd-fw.html) , criar um pequeno gateway para um DSL ou conexões cable. Também pode ser usado como um firewall de link dedicado, logicamente, com algumas ressalvas, pois o mesmo pode não estar preparado para este fim.

Primeiramente, deve-se fazer a declaração das interfaces que temos em nossa máquina :

```
# Network interfaces (Remember, if using PPPoE the ext.
interface is tun0)
internal = "fxp0"
external = "dc0"
wireless = "rl0"
unsafe = "{ dc0, rl0 }"
```

Podemos ainda, criar algumas macros, ou seja, variáveis que serão utilizadas em todo o arquivo do pf .

```
services = "{ ssh, http, https, smtp, domain }"
nonroutable = "{ 192.168.0.0/16, 127.0.0.0/8, 172.16.0.0/12,
10.0.0.0/8,0.0.0.0/8, 169.254.0.0/16, 192.0.2.0/24, 204.152.64.0/23,
224.0.0.0/3,255.255.255.255/32 }"
```

Leve em consideração um ponto no exemplo acima. Se você possui algum serviço de rede que funciona em algum ip inválido, tire esta faixa de rede desta matriz.

As regras de nat podem ser feitas de um modo muito fácil, através de uma linha como a abaixo :

```
# NAT endereços IP internos do range 192.168.0.0/24 para o
endereço
# IP roteável da interface rl0
nat on $external from 192.168.0.0/24 to any -> $external
```

Mas há outras possibilidades de NAT, como por exemplo, a máquina sair com um ip qualquer válido. Isto pode ser feito utilizando-se a seguinte linha :

```
nat on $external from 192.168.0.0/24 to any -> 200.200.200.1
```

A criação de filtros deve ser feita seguindo a seguinte forma :

```
#  ICMP: allow incoming ping and traceroute only
#
pass in quick on $unsafe inet proto icmp from any to any icmp-type {  echorep,
echoreq, timex, unreach }
block in log quick on $unsafe inet proto icmp from any to any

#  TCP: Allow ssh, smtp, http and https incoming. Only match
#  SYN packets, and allow the state table to handle the rest of the
#  connection. ACKs and ToS "lowdelay" are given priority.
#
pass in quick on $external inet proto tcp from any to any port $services \ flags
S/SA keep state queue (default_q, highpri_q)
```

A primeira regra permite o icmp echo reply e outros tipos. Em compensação a segunda regra bloqueia todo o tráfego icmp na interface não válida, ou seja, a interface que recebe um ip dinâmico do provedor.

O pf permite também criação de controles de banda. No firewall do autor, podemos ver que ele criou um controle de banda para o tráfego normal e para o tráfego prioritário.

```
# Create two packet queues: one for regular traffic, another for
# high priority: TCP ACKs and packets with ToS 'lowdelay'
altq on $external priq bandwidth 125Kb queue { highpri_q, default_q }
queue highpri_q priority 7
queue default_q priority 1 priq(default)
```

A partir do apresentado é possível criar um pequeno firewall com o FreeBSD utilizado o pf.

5. Administrando o seu firewall :

- Apaga as regras atuais de nat e as recarrega :
  #pfctl -F nat && pfctl -N /etc/pf.conf
- Apaga as regras de filtro atuais e as recarrega :
  # pfctl -F rules && pfctl -R /etc/pf.conf
- Mostra as informações de filtro :
  # pfctl -s info

- Exibe a lista atual ativa de filtros MAP/Redirect e sessões ativas :
  # pfctl -s state
- Exibe informações detalhadas sobre cada regras no arquivo pfc.conf
  # pfctl -s rules -v
- Carrega novamente todas as regras e as tabelas atualizadas :
  # pfctl -f /etc/pf.conf

6. Conclusão :

O tutorial, apesar de pequeno, apresenta o FreeBSD como uma opção para a criação de firewalls em redes de computadores.

Apêndice 1 (Fonte : http://www.fug.com.br/content/view/99/2/) :

1. -1 Modo *default*, sem segurança de kernel. A única segurança aqui é em relação à permissões tradicionais do de sistemas Unix.
2. 0 Este nível é apenas usado quando o sistema está sendo inicializado pela primeira vez, e não possui alguma característica especial. Não há motivos especiais para utilizar este nível.
3. Aqui sim você realmente começa a ter segurança

```
* Não é possivel carregar módulos ao kernel, a não ser que o mesmo seja
recompilado. Veja:

# sysctl kern.securelevel=1
kern.securelevel: -1 -> 1
# kldload if_tap
kldload: can't load if_tap: Operation not permitted

* Nenhum programa pode escerver diretamente a memória do sistema via
/dev/mem ou /dev/kmem

* Discos que foram montados não podem ser escritos diretamente, não
sendo possível formatar partições ou usar o dd(1) nesses discos.

* O Gerenciador de Janelas X não pode ser iniciado (pois ele faz acesso
direto aos dispositivos de memória, e isso não é mais permitido nesse
nível de segurança)
```

4. As mesmas característacas do modo 1 alêm de:

```
* Impossibilidade de escrever direto a sistema de arquivos montados ou
desmontados (exceto pelo mount(2),). Impossibilita alterar a estrutura do sistema de
arquivos, redefinir ou modificar opções de montagem e também formatar newfs(8)
quando o sistema estiver em multi-user.

* A hora somente poderá ser alterada de 1 em 1 segundo.
```

5. As mesmas caracteristicas do modo 2 além proibir aletrações das definições de regras de firewall, seja o firewall ipfw(8), pfctl(8) ou ipf(8), e também proibe modificações nas disciplinas alternativas de enfileiramento, com altq(4) ou dummynet(4).

Apêndice 2 :

Firewall Completo :

```
#########################################################################
#
# IP packet filtering rules (firewall)
# Shamim Mohamed 3/2002, 5/2003

# See pf.conf(5) for syntax and examples

# If you change this file, run
#    pfctl -f /etc/pf.conf
# to update kernel tables (also run "pfctl -e" if pf was not running)

# Network interfaces (Remember, if using PPPoE the ext. interface is tun0)
internal = "fxp0"
external = "dc0"
wireless = "rl0"
unsafe = "{ dc0, rl0 }"

# Services visible from the outside – remove any you're not using
services = "{ ssh, http, https, smtp, domain }"

# The wireless interface is not allowed to send anything to the inside
# network. It can send anything out except smtp since we don't
# want being used as a spam relay. Yes, this is paranoid. Better safe
# than sorry.

# You shouldn't need to change anything below this line
#########################################################################

# Non-routable IP numbers
nonroutable = "{ 192.168.0.0/16, 127.0.0.0/8, 172.16.0.0/12, 10.0.0.0/8,
    0.0.0.0/8, 169.254.0.0/16, 192.0.2.0/24, 204.152.64.0/23, 224.0.0.0/3,
    255.255.255.255/32 }"

# All rules are "quick" so go strictly top to bottom

# Fix fragmented packets
scrub in all

# Create two packet queues: one for regular traffic, another for
# high priority: TCP ACKs and packets with ToS 'lowdelay'
altq on $external priq bandwidth 125Kb queue { highpri_q, default_q }
queue highpri_q priority 7
queue default_q priority 1 priq(default)
```

```
# NAT

# nat: packets going out through dc0 with source addr 192.168.1.0/24
# will get translated as coming from our external address. State is
# created for such packets, and incoming packets will be redirected to
# the internal address.

# I have an experimental web server on an inside machine; I can test it
# from the outside by connecting to port 8042.
rdr on $external inet proto tcp to port 8042 -> 192.168.1.12 port 80

# NAT: we need a rule for the inside network as well as the wireless.
nat on $external from 192.168.1.0/24 to any -> $external
nat on $external from 192.168.2.0/24 to any -> $external

##############################################################################

#  Don't bug loopback
#
pass out quick on lo0 from any to any
pass in quick on lo0 from any to any

#  Don't bother the inside interface either
#
pass out quick on $internal from any to any
pass in quick on $internal from any to any

##############################################################################
#
#  First, we deal with bogus packets.
#

#  Block any inherently bad packets coming in from the outside world.
#  These include ICMP redirect packets and IP fragments so short the
#  filtering rules won't be able to examine the whole UDP/TCP header.
#
block in log quick on $unsafe inet proto icmp from any to any icmp-type redir

#  Block any IP spoofing atempts.  (Packets "from" non-routable
#  addresses shouldn't be coming in from the outside).
#
block in quick on $external from $nonroutable to any

#  Don't allow non-routable packets to leave our network
#
block out quick on $external from any to $nonroutable

##############################################################################
#
# Wireless:  block SMTP from wireless - spam threat
#
block in quick on $wireless inet proto tcp from any to any port smtp
#
#
##############################################################################

##############################################################################
```

```
#
#  The normal filtering rules
#

#  ICMP: allow incoming ping and traceroute only
#
pass in quick on $unsafe inet proto icmp from any to any icmp-type { \
    echorep, echoreq, timex, unreach }
block in log quick on $unsafe inet proto icmp from any to any

#  TCP: Allow ssh, smtp, http and https incoming. Only match
#  SYN packets, and allow the state table to handle the rest of the
#  connection. ACKs and ToS "lowdelay" are given priority.
#
pass in quick on $external inet proto tcp from any to any port $services \
     flags S/SA keep state queue (default_q, highpri_q)

# UDP: allow DNS since I run a public nameserver (remove if you don't!)
pass in quick on $unsafe inet proto udp from any to any port domain

################
# Wireless
#
# allow connections from 192.168.2.0/24, the inside wired network.
pass out quick on $wireless inet proto tcp from any to any \
     flags S/SA keep state queue (default_q, highpri_q)

# Everyone is allowed to send UDP and ICMP out
pass out quick on $external inet proto udp  all keep state
pass out quick on $external inet proto icmp from any to any keep state

# Block wireless -> inside network
block in quick on $wireless from any to $nonroutable

# Everything else is ok
pass in quick on $wireless from any to any

##################
#  Of course we need to allow packets coming in as replies to our
#  connections so we keep state. Strictly speaking, with packets
#  coming from our network we don't have to only match SYN, but
#  what the hell. It allows us to put those packets in the high
#  priority queue.
#
pass out quick on $external inet proto tcp from any to any \
     flags S/SA keep state queue (default_q, highpri_q)
pass out quick on $external inet proto udp  all keep state
pass out quick on $external inet proto icmp from any to any keep state


#  End of rules. Block everything to all ports, all protocols and return
#  RST (TCP) or ICMP/port-unreachable (UDP).
#
block return-rst in log quick on $unsafe inet proto tcp from any to any
block return-icmp in log quick on $unsafe inet proto udp from any to any
block in quick on $unsafe all
```