PF
OpenBSD packet filter

Appunti su PF e OpenBSD

a cura di

Stefano Sasso

stefano(at)gnustile.net

Versione 0.9 - 24 Febbraio 2008

Indice

Parte I

PF

Capitolo 1

Introduzione a PF

1.1 Introduzione

pf è il firewall di default di OpenBSD, portato anche su FreeBSD e NetBSD.
pf, il packet filter, richiede un certo ordine per le regole:

1. definizione macros
2. definizione tables
3. definizione options
4. definizione scrub
5. definizione queueing
6. definizione translation (*nat*, *binat*, *rdr*)
7. definizione filter rules

É buona norma, prima di scrivere le istruzioni pf, descrivere a tavolino quello che vogliamo che il nostro firewall faccia.

1.2 Abilitare PF

Per abilitare PF al boot, andare a modificare **/etc/rc.conf(.local)** ed inserire

```
pf=YES
```

per abilitarlo/disabilitarlo senza rebootare:

```
# pfctl -e
```

(e = enable)

```
# pfctl -d
```

(d = disable)

La configurazione di pf è **/etc/pf.conf**. Per caricarla, provarla, etc...:

- carica la configurazione:

  ```
  # pfctl -f /etc/pf.conf
  ```

- testa la sintassi:

  ```
  # pfctl -nf /etc/pf.conf
  ```

- mostra tutta la configurazione attiva:

  ```
  # pfctl -sa
  ```

In ogni caso **man pfctl** può aiutare.

Capitolo 2

Costrutti di PF

In questa parte vedremo quali sono i costrutti di pf. Per capirne il funzionamento verranno utilizzati accoppiati ad alcune regole di filtraggio. Tali regole verranno spiegate nel dettaglio in seguito.

2.1 Liste

Una lista permette di inserire più criteri simili in una regola, ad esempio, più protocolli, più host, più porte...
ad esempio:

```
block out on fxp0 from { 192.168.0.20, 192.168.1.0/24 } to any
```

(le virgole tra gli item sono opzionali)

{ 22, 80 }

equivale a

{ 22 80 }

nelle liste esistono anche le negazioni, prese in considerazione con ordine, ad esempio:

{ 192.168.0.0/24, !192.168.0.12 }

significa tutti i *192.168.0.0/24* tranne *192.168.0.12/32*

2.2 Macro

Le macro sono delle variabili definite dall'utente, in maniera simile alle variabili degli script di shell:

```
ext_if="fxp0"
tizio="192.168.0.1"
caio="192.168.3.21"
block in on $ext_if from $tizio to $caio
```

una macro può anche essere una lista

```
allowed="{ 25 80 110 443 }"
pass in from any to any port $allowed
```

inoltre, una macro può essere una lista di macro

```
host1="192.168.0.1"
host2="192.168.0.2"
hosts="{" $host1 $host2 "}"
```

2.3 Tabelle

Una tabella è usata per contenere un gruppo di indirizzi ipv4 o ipv6. Lookup attraverso tabelle di indirizzi sono più veloci e consumano meno memoria rispetto a lookup di liste. Le tabelle sono create usando la direttiva *table*. I seguenti attributi possono essere specificati per una tabella:

- *const*: la tabella non è modificabile dopo la sua creazione
- *persist*: viene tenuta in memoria dal kernel, anche se nessuna regola la usa

ecco alcuni esempi:

```
table <goodguys> { 192.0.2.0/24 }
table <rfc1918> const { 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }
table <spammers> persist
block in on fxp0 from { <rfc1918>, <spammers> } to any
pass in on fxp0 from <goodguys> to any
```

Come in una lista, anche nelle tabelle è possibile inserire negazioni:

```
table <goodguys> { 192.0.2.0/24, !192.0.2.5 }
```

Una tabella può essere popolata partendo da un file di testo:

```
table <spammers> persist file "/etc/spammers"
block in on fxp0 from <spammers> to any
```

Il file **spammers** conterrà un elemento per riga. Le righe inizianti con # non saranno interpretate. É possibile manipolare le tabelle usando **pfctl**:

```
# pfctl -t spammers -T add 218.70.0.0/16
# pfctl -t spammers -T show
# pfctl -t spammers -T delete 218.70.0.0/16
```

I comandi sono auto-esplicativi :)

Capitolo 3

Packet Filtering

3.1 Regole di filtraggio

Le regole di filtering sono valutate tutte, dalla prima all'ultima, tranne nel caso in cui venga usata la keywork *quick*. nel caso normale la regola che vince è l'ultima.
In generale, la sintassi di una regola è questa:

> *action* [*direction*] [**log**] [**quick**] [**on** *interface*] [*af*] [**proto** *protocol*] \
> [**from** *src_addr* [**port** *src_port*]] [**to** *dst_addr* [**port** *dst_port*]] \
> [**flags** *tcp_flags*] [*state*]

dove:

- *action*: *pass* o *block*, l'azione da compiere
- *direction*: direzione del pacchetto, *in* oppure *out*
- *log*: se specificato logga questo traffico
- *quick*: la regola viene considerata la vincente, e la catena si ferma
- *interface*: l'interfaccia (o lista di interfacce)
- *af*: address family del protocollo: *inet* o *inet6*
- *protocol*: il protocollo layer 4: *tcp*, *udp*, *icmp*, *icmp6*, o un protocollo valido da **/etc/protocols**, o un numero da 0 a 255
- *src_addr*, *dst_addr*: ip sorgenti/destinazione; possono essere:
  - singolo ipv4/ipv6
  - blocco cidr[1]
  - fqdn[2] risolvibile

[1] Classless Inter-Domain Routing
[2] Fully Qualified Domain Name

  - il nome di un'interfaccia. verrà fatto il matching con tutti gli ip di quell'interfaccia
  - il nome di un'interfaccia seguito da /netmask. come sopra, anche su tutti gli indirizzi che matchano netmask
  - il nome di un'interfaccia, tra parentesi *()*: come sopra, ma dice a pf di aggiornarsi se l'ip cambia
  - il nome di un'interfaccia, seguito da questi modificatori:
    * *:network* ritorna il cidr network block
    * *:broadcast* ritorna l'indirizzo di broadcast
    * *:peer* l'indirizzo del per in caso di P-t-P[3]
    * aggiungendo un *:0*, sia dopo i modificatori che dopo l'interfaccia, non si considerano gli alias
  - una tabella
  - un oggetto di cui sopra, negato con !
  - una lista di indirizzi
  - *any*
  - *all*, che sta per *from any to any*

- *src_port*, *dst_port*: la porta sorgente o destinazione (solo tcp e udp)
  - una singola porta, tra 1 e 65535
  - un nome valido e risolvibile da **/etc/services**
  - una lista di porte
  - un range
    * ! = diverso
    * < minore
    * > maggiore
    * <= minore o uguale
    * >= maggiore o uguale
    * >< range
    * <> range inverso
    * : range inclusivo

- *tcp_flags*: (solo tcp) specifica il flag che deve essere settato nell'header tcp. I flag sono specificati come *check/mask*. Ad esempio

  flags S/SA

  controlla che ci siano S e A (SYN e ACK), fa il match se e solo se S è on

- *state*: specifica se deve essere mantenuto lo stato di questa regola
  - *keep state* (tcp, udp, icmp)

[3] Point-To-Point, PPP

- *modulate state* (tcp, udp)
- *synproxy state* (solo tcp), protegge da spoofed syn flood. pf fa da proxy per il 3-way handshake[4]. Include *modulate* e *keep*; non funziona con bridge

3.2 Default deny

Il default deny si applica così:

```
block in all
block out all
```

o più semplicemente

```
block all
```

3.3 Connection tracking

É possibile tenere in memoria lo stato delle connessioni con le direttive *state*. Vediamo subito un esempio banale: uso del keep state per proteggere l'host su cui gira pf:

```
block in all
pass out on $ext_if proto {tcp, udp, icmp} from any to any keep state
```

Il tracking può avere varie opzioni, specificabili tra *()* dopo il *keep state*:

- *max*: il numero di entry massime (ovvero il numero di connessioni verso ...)
- *source-track*: abilita il filtraggio sul numero di stati creati da singoli ip. Ha due formati:
  - *source-track rule*: il numero massimo di stati creati da questa regola, limitati dalla direttiva *max-src-nodes* e *max-src-states*
  - *source-track global*: il numero di stati creati da tutte le regole; limitati sempre dalle due direttive di cui sopra
- *max-src-nodes*: limita il numero di ip che possono creare connessioni
- *max-src-states*: limita il numero di stati per ip

Ad esempio:

```
pass in on $ext_if proto tcp to $web_server \
    port www flags S/SA keep state \
    (max 200, source-track rule, max-src-nodes 100, max-src-states 3)
```

[4] 3-way handshake: procedura di scambio dati tra client e server che consente l'inizializzazione di una connessione tcp.

Questa regola:

- limita il numero di stati assoluti per questa regola a 200
- limita la creazione di stati basati su questa regola
- limita il numero massimo di nodi a 100
- limita il numero massimo di connessione/per/nodo a 3

Le seguenti opzioni possono essere applicate solo a pacchetti tcp, in quanto si riferiscono al 3-way handshake:

- *max-src-conn*: limita il numero di connessioni che hanno completato l'handshake
- *max-src-conn-rate*: limita il numero di connessioni per intervallo di tempo

Entrambe sono incompatibili con source-track global.
Applicando queste regole a tcp, è possibile compiere azioni più aggressive;
vediamo un esempio:

```
table <abusive_hosts> persist
block in quick from <abusive_hosts>
pass in on $ext_if proto tcp to $web_server \
    port www flags S/SA keep state \
    (max-src-conn 100, max-src-conn-rate 15/5, overload <abusive_hosts> flush)
```

questa regola:

- limita il numero di connessioni per nodo a 100
- limita il numero di connessioni a 15 per 5 secondi
- inserisce l'ip che sgarra nella tabella *abusive_hosts*
- per ogni ip che sgarra cancella gli altri stati creati da questa regola

3.4 Filtraggio ICMP

Non è mai buona cosa droppare tutto il traffico ICMP... É utile infatti lasciar passare qualche tipo di pacchetto. Per questo esiste la keywork *icmp-type*, simile al *port* di tcp e udp, che consente di selezionare il tipo di traffico icmp.
Ad esempio, per lasciar passare solo i pacchetti ping e traceroute

```
pass in quick on fxp0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in quick on fxp0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on fxp0 proto icmp from any to any
```

Ovviamente è possibile mantenere lo stato anche di connessioni icmp:

```
pass out on $ext_if proto icmp from any to any icmp-type 8 keep state
```

Questa regola consentirà di pingare, e lascera passare il pacchetto pong (icmp-type 0) relativo.

3.5 I flag TCP

Ripassiamo un po': i flag TCP sono:

- *F*: fin, finish, indica la fine di una richiesta
- *S*: syn, synchronize, indica l'inizio di una richiesta
- *R*: rst, reset, cancella una connessione
- *P*: push, i pacchetti sono inviati
- *A*: ack, acknowledgement
- *U*: urg, urgent
- *E*: ece, explicit congestion notification echo
- *W*: cwr, congestion window reduced

Per ispezionare i flag tcp bisogna usare la keyword *flags check/mask*:

```
pass in on fxp0 proto tcp from any to any port ssh flags S/SA
```

I pacchetti con il flag *S* attivo, guardando solo *S* ed *A* passano.
Un pacchetto con *SE* passa, uno con *SA* non passa, uno *A* non passa, un *S* passa, etc...
I flag sono spesso usati col keeping state, per aiutarli a creare gli stati:

```
pass out on fxp0 proto tcp all flags S/SA keep state
```

ovvero crea uno stato solo per nuove connessioni (solo *S*)

3.6 Lo Scrubbing

Lo scrubbing serve a normalizzare i pacchetti in transito. Questa regola serve per normalizzare tutti i pacchetti:

```
scrub in all
```

La direttiva *scrub* è molto simile ad una direttiva di filtering, che rende semplice la selezione dei pacchetti su cui operare. In questo caso, come nel NAT, la prima regola vince.
Anteponendo un *no* allo *scrub*, non si normalizzeranno i pacchetti individuati dalla regola.
scrub ha le seguenti opzioni:

- *no-df*: cancella il bit `don't fragment` dall'intestazione del pacchetto. Ne è raccomandato l'uso insieme alla direttiva *random-id*
- *random-id*: rimpiazza l'id di un pacchetto con uno generato casualmente. Viene applicato solo ai pacchetti non frammentati

- *min-ttl*: forza un TTL[5] minimo nei pacchetti
- *max-mss*: forza un massimo MSS[6] nei pacchetti
- *fragment reassemble*: ri-assembla i pacchetti prima di passarli al filtro. É l'unica opzione di fragment che funziona con NAT.
- *fragment crop*: elimina i frammenti duplicati
- *fragment crop-ovl*: come sopra, ma elimina anche i frammenti corrispondenti
- *reassemble tcp*: normalizza le connessioni TCP. Non è necessaria l'indicazione di un verso in/out

Esempi:

```
scrub in on fxp0 all fragment reassemble min-ttl 15 max-mss 1400
scrub in on fxp0 all no-df
scrub    on fxp0 all reassemble tcp
```

3.7 Anti-spoofing

Usando la keywork *antispoof* è possibile bloccare facilmente i tentativi di spoofing:

antispoof [**log**] [**quick**] **for** *interface* [*af*]

ad esempio:

```
antispoof for fxp0 inet
```

3.8 Skip di interfacce da packet filtering

A volte può esssere utile evitare che determinate interfacce vengano filtrate (ad esempio le interfacce di loopback come lo0)

```
set skip on lo0
```

(è sempre buona cosa)

[5] Time-To-Live, tempo di vita del pacchetto
[6] Maximum Segment Size, massima grandezza di un segmento

3.9 OS fingerprinting

É possibile effettuare fingerprinting passivo sul sistema operativo che ha generato la connessione, e quindi filtrare in base a quello; ad esempio:

```
pass  in on $ext_if from any os OpenBSD keep state
block in on $ext_if from any os "Windows 2000"
pass  in on $ext_if from any os "Linux 2.4 ts"
block in on $ext_if from any os unknown
```

É possibile vedere l'elenco dei fingerprintings con

```
# pfctl -ospf
```

Capitolo 4

Network Address Translation

4.1 Source NAT

D'ora in poi con NAT intenderemo solo il Source NAT.
La sintassi di una regola di nat è pressappoco

> **nat** [**pass** [**log**]] **on** *interface* [*af*] **from** *src_addr* [**port** *src_port*] **to** \
> *dst_addr* [**port** *dst_port*] **->** *ext_addr* [*pool_type*] [**static-port**]

dove:

- *nat*: keywork che indica l'inizio di una regola di nat
- *pass*: dice di fregarsene del filtraggio successivo
- *log*: boh :)
- *interface*: l'interfaccia su cui translare i pacchetti
- *af*: address family
- *src_addr*: indirizzo sorgente, stesse regole di indirizzi di filter rules
- *src_port*: porta sorgente, stesse regole di porte di filter rules
- *dst_addr*: l'indirizzo destinazione
- *dst_port*: porta destinazione
- *ext_addr*: l'indirizzo con cui mascherare
- *pool_type*: il tipo di pool di indirizzi della translazione
- *static-port*: dice di non translare la porta sorgente

ad esempio:

```
nat on $ext_if from 192.168.1.0/24 to any -> 24.5.0.5
nat on $ext_if from $int_if:network to any -> $ext_if
```

e se $ext_if è in dhcp:

```
nat on tl0 from dc0:network to any -> (tl0)
```

ci possono essere delle eccezioni nelle regole di nat:

```
no nat on tl0 from 192.168.1.208 to any
nat on tl0 from 192.168.1.0/24 to any -> 24.2.74.79
```

ATTENZIONE: nel NAT, come nello scrubbing, la prima regola vince.

4.2 NAT bidirezionale

Con il NAT bidirezionale si vuole avere un'assegnazione di ip 1:1.
Per questo scopo bisogna usare la regola *binat*.

```
web_serv_int = "192.168.1.100"
web_serv_ext = "24.5.0.6"
binat on tl0 from $web_serv_int to any -> $web_serv_ext
```

4.3 Port Forwading

Partiamo subito vedendo un esempio:

```
rdr on tl0 proto tcp from any to any port 80 -> 192.168.1.20
```

ma il from any to any non è molto utile...
si può limitare la redirezione solo a certi ip:

```
rdr on tl0 proto tcp from 27.146.49.0/24 to any port 80 -> 192.168.1.20
```

un range di porte può essere specificato:

```
rdr on tl0 proto tcp from any to any port 5000:5500 -> 192.168.1.20
rdr on tl0 proto tcp from any to any port 5000:5500 -> 192.168.1.20 port 6000
rdr on tl0 proto tcp from any to any port 5000:5500 -> 192.168.1.20 port 7000:*
```

nella prima regola le porte vengono translate in 5000->5000, 5001->5001, etc...
nella seconda regola 5000->6000, 5001->6000, etc...
nella terza regola 5000->7000, 5001->7001, etc...
I pacchetti che hanno subito port forwarding devono comunque passare per i filtri, a meno che non ci sia *pass* dopo *rdr*

Molto spesso *rdr* serve per forwardare connessioni verso una macchina interna che offre servizi:

```
server = 192.168.1.40
rdr on $ext_if proto tcp from any to $ext_if port 80 -> $server port 80
```

ma, ovviamente, dall'interno della lan questa regola non funziona.
É comunque desiderabile che i client della lan possano comunque accedere al servizio; ci sono molti modi per fare questo:

- dns interno; si evita anche il passaggio di pacchetti verso il firewall
- usare un ulteriore segmento, DMZ
- tcp proxy nel firewall: dei semplici proxy possono essere realizzati con inetd e nc (vedi tips and tricks)
- regola nat addizionale; che ci semplifica di molto la vita :)

  ```
  rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> $server
  no nat on $int_if proto tcp from $int_if to $int_net
  nat on $int_if proto tcp from $int_net to $server port 80 -> $int_if
  ```

Capitolo 5

Opzioni Runtime di pf

block-policy

set block-policy *option*

block-policy definisce la politica da adottare quando un pacchetto viene scartato. Può essere *drop* o *return*. Il default è *drop*.

```
set block-policy return
```

debug

set debug *option*

debug setta il livello di verbosità dei messaggi di debug:

- none - nessun messaggio
- urgent - solo messaggi generati da errori gravi
- misc - messaggi da errori non gravi
- loud - messaggi generati da condizioni normali

Il default è *urgent*

fingerprints

set fingerprints *file*

fingerprints setta il file usato per il fingerprinting passivo degli OS. Il default è **/etc/pf.os**

limit

set limit *option value*

limit imposta dei limiti per determinate operazioni.

- *frags*: massimo numero di entry per il ri-assemblamento dei pacchetti. Il default è 5000
- *src-nodes*: massimo numero di entry per la tabella di memorizzazione degli ip sorgente. Il default è 10000
- *states*: massimo numero di stati memorizzabili. Il default è 10000

loginterface

set loginterface *interface*

loginterface setta l'interfaccia dalla quale recuperare statistiche sulle connessioni. Il default è *none*

optimization

set optimization *option*

optimization ottimizza pf per uno dei seguenti tipi di rete:

- *normal*: una rete normale
- *high-latency*: rete ad alta latenza, come connessioni satellitari
- *aggressive*: fa scadere aggressivamente connessioni dalla tabella degli stati
- *conservative*: conserva gli stati il più possibile

Il valore di default è *normal*.

state-policy

set state-policy *option*

state-policy modifica le opzioni per il mantenimento dello stato:

- *if-bound*: gli stati sono creati a partire dall'interfaccia
- *group-bound*: gli stati sono creati a partire dal gruppo dell'interfaccia
- *floating*: creati da qualsiasi interfaccia

Il valore di default è *floating*.

timeout

set timeout *option value*

timeout modifica le impostazioni per i vari timeout:

- *interval*: secondi tra la rimozione dello stato e tra i frammenti dei pacchetti. Il default è 10
- *frag*: secondi dopo i quali un pacchetto non ri-assemblato scade. Il default è 30
- *src.track*: secondi per i quali mantenere un indirizzo sorgente in memoria dopo la scadenza dello stato. Il default è 0

Esempi

```
set timeout interval 10
set timeout frag 30
set limit { frags 5000, states 2500 }
set optimization high-latency
set block-policy return
set loginterface dc0
set fingerprints /etc/pf.os.test
set skip on lo0
set state-policy if-bound
```

Capitolo 6

Packet Queueing e Prioritization

6.1 Introduzione agli schedulatori

Uno schedulatore decide cosa e quando processare.
Gli schedulatori supportati da pf sono:

- class based queueing (CBQ)
- priority based queueing (PRIQ)

Class based queueing (CBQ) divide la banda tra più classi o code;
ogni coda ha quindi un tot di banda assegnato a seconda di ip, porte, etc..
una coda può richiedere in prestito banda dal padre, sempre che la banda sia disponibile.
Le code CBQ hanno livello gerarchico a N sotto-alberi;

root (2Mbps)
\-queue a (1Mbps)
\-queue b (500Kbps)
\-queue c (500Kbps)

ad esempio:

root (2Mbps)
\-user 1 (1Mbps)
\-\-ssh (200Kbps)
\-\-ftp (800Kbps, borrow)
\-user 2 (1Mbps)

se ssh di user 1 ha meno traffico di 200K, ftp può prendere in prestito del traffico ssh per incrementare il suo;
se ssh poi avesse bisogno di tutta la banda, questa viene restituita

CBQ assegna anche una priorità a ciascuna coda. In caso di congestione vengono privilegiate code con maggiore priorità;
esempio:

\-\-ssh (200Kbps, prio: 5, borrow)
\-\-ftp (800Kbps, prio: 3, borrow)

Lo schedulatore basato sulla sola priorità, a differenza del CBQ ha un solo sotto-livello:

root (2Mbps)
\-queue a (prio: 1)
\-queue b (prio: 2)
\-queue c (prio: 3)

6.2 Uso del packet queueing

Per abilitare il packet queueing è sufficente usare le direttive *altq*, che ha sintassi:

altq on *interface scheduler* **bandwidth** *bw* **qlimit** *qlim* **tbrsize** *size* \
queue { *queue_list* **}**

dove:

- *interface*: l'interfaccia d'uscita su cui lavorare
- *scheduler*: il tipo di schedulatore, *cbq* o *priq*; è possibile usare un solo schedulatore per interfaccia
- *bw*: la banda totale a disposizione dello schedulatore; possono essere specificati suffissi quali b, Kb, Mb, Gb (Xbits)
- *qlim*: il massimo numero di pacchetti da tenere in coda, default è 50
- *size*: dimensione del tocket bucket regulator; se non specificato è basato automaticamente su interfaccia
- *queue_list*: elenco di code children

ad esempio:

```
altq on fxp0 cbq bandwidth 2Mb queue { std, ssh, ftp }
```

Le code chiltren si specificano con *queue*, che ha sintassi:

queue *name* [**on** *interface*] **bandwidth** *bw* [**priority** *pri*] [**qlimit** *qlim*] \
scheduler **(** *sched_options* **) {** *queue_list* **}**

dove:

- *name*: nome della coda

- *interface*: interfaccia su cui lavorare
- *bw*: banda della coda
- *pri*: priorità della coda
- *qlim*: massimo numero di pacchetti, default: 50
- *scheduler*: schedulatore
- *sched_options*: opzioni dello schedulatore:
  - *default*: indica che i pacchetti che non fanno match con altre code devono essere messi qui
  - *red*: abilita random early detection
  - *rio*: abilita red con i/o
  - *ecn*: abilita explicit congestion notification; *ecn* implica *red*
  - *borrow*: questa coda può richiedere altra banda
- *queue_list*: code children

esempio:

```
queue std bandwidth 50% cbq(default)
queue ssh bandwidth 25% { ssh_login, ssh_bulk }
  queue ssh_login bandwidth 25% priority 4 cbq(ecn)
  queue ssh_bulk bandwidth 75% cbq(ecn)
queue ftp bandwidth 500Kb priority 3 cbq(borrow red)
```

posso assegnare il traffico ad una coda usando le regole di filtraggio:

```
pass out on fxp0 from any to any port 22 queue ssh
```

Una coda può essere definita per un'interfaccia diversa da quella specificata dalla regola di filtraggio; in quel caso la coda viene usata solo se il pacchetto esce da quell'interfaccia.
Esempio: (ext_if=fxp0, int_if=dc0)

```
altq on fxp0 cbq bandwidth 2Mb queue { std, ftp }
queue std bandwidth 500Kb cbq(default)
queue ftp bandwidth 1.5Mb
pass in on dc0 from any to any port 21 queue ftp
```

Capitolo 7

Address Pool e Load Balancing

7.1 NAT address pool

Il nat address pool viene usato per mascherare in uscita con 2 o più indirizzi:

```
nat on $ext_if inet from any to any -> { 192.0.2.5, 192.0.2.10 }
```

ma per fare in modo che un client venga sempre mappato con lo stesso indirizzo:

```
nat on $ext_if inet from any to any -> 192.0.2.4/31 source-hash
```

7.2 Port forwarding con load balancing

É possibile bilanciare le connessioni in ingresso:

```
web_servers = "{ 10.0.0.10, 10.0.0.11, 10.0.0.13 }"
rdr on $ext_if proto tcp from any to $ext_if port 80 -> $web_servers \
    round-robin sticky-address
```

7.3 Bilanciamento del traffico in uscita

La direttiva *route-to* ci permette di bilanciare il traffico su due connessioni internet diverse, quando non si può usare un protocollo di routing adeguato (come BGP4):

```
lan_net = "192.168.0.0/24"
int_if  = "dc0"
ext_if1 = "fxp0"
ext_if2 = "fxp1"
ext_gw1 = "68.146.224.1"
ext_gw2 = "142.59.76.1"
my_server = "63.99.144.32"
```

```
pass in quick on $int_if route-to ($ext_if1, $ext_gw1) \
   proto tcp from $lan_net to $my_server port ssh modulate state
pass in on $int_if route-to \
   { ($ext_if1 $ext_gw1), ($ext_if2 $ext_gw2) } round-robin \
   from $lan_net to any keep state
pass out on $ext_if1 route-to ($ext_if2 $ext_gw2) from $ext_if2 \
   to any
pass out on $ext_if2 route-to ($ext_if1 $ext_gw1) from $ext_if1 \
   to any
```

e, se serve, il nat

```
nat on $ext_if1 from $lan_net to any -> ($ext_if1)
nat on $ext_if2 from $lan_net to any -> ($ext_if2)
```

Capitolo 8

Uso avanzato delle regole di filtraggio

8.1 Tagging

É possibile usare il tagging per marcare alcuni pacchetti, e quindi decidere in seguito il loro futuro; è possibile usare *tag* e *tagged* sia nelle regole di filtraggio che nel nat/rdr.

```
pass in quick on $int_if proto tcp to port 80 tag \
    INT_NET_HTTP keep state
...
pass out on $ext_if tagged INT_NET keep state
pass out on $ext_if ! tagged WIFI_NET keep state
```

8.2 Ancore

Oltre alle regole principali, pf può analizzare delle sotto-regole, che possono essere modificate al volo usando **pfctl**.
Queste sotto-regole consentono di aggiungere regole di filtraggio, nat, rdr e binat; sono collegate alle regole principali tramite delle ancore.
Ci sono 4 tipi di ancore:

1. **anchor** *name* - valuta tutte le regole di filtraggio nell'ancora
2. **binat-anchor** *name* - valuta le regole binat
3. **nat-anchor** *name* - valuta le regole nat
4. **rdr-anchor** *name* - valuta le regole rdr

Esempio:

```
ext_if = "fxp0"
block on $ext_if all
pass out on $ext_if all keep state
anchor goodguys
```

Un'ancora può essere popolata in 2 modi:

1. utilizzando una direttiva *load*, che permette di leggere le regole da un file esterno. Deve essere posizionata dopo la definizione dell'ancora

   ```
   anchor goodguys
   load anchor goodguys from "/etc/anchor-goodguys-ssh"
   ```

2. utilizzando **pfctl**

   ```
   # echo "pass in proto tcp from 192.0.2.3 to any port 22" \
      | pfctl -a goodguys -f -
   ```

   oppure

   ```
   # vi /etc/anchor-goodguys-www
   pass in proto tcp from 192.0.2.3 to any port 80
   pass in proto tcp from 192.0.2.4 to any port { 80 443 }
   # pfctl -a goodguys -f /etc/anchor-goodguys-www
   ```

Attenzione: le macro definite nella configurazione principale non sono visibili dalle ancore!

Le ancore possono essere innestate:

```
anchor "spam/*"
```

questa regola valuterà in ordine alfabetico tutti i figli dell'ancora spam.

É possibile passare delle opzioni all'ancora:

```
ext_if = "fxp0"
block on $ext_if all
pass out on $ext_if all keep state
anchor ssh in on $ext_if proto tcp from any to any port 22

# echo "pass in from 192.0.2.10 to any" | pfctl -a ssh -f -
```

La manipolazione delle ancore avviene via **pfctl**. Ecco le azioni principali

- elenca tutte le regole dell'ancora:

  ```
  # pfctl -a ssh -s rules
  ```

- flusha tutte le regole dell'ancora:

  ```
  # pfctl -a ssh -F rules
  ```

8.3 Nat e Filtraggio di connessioni FTP

the same old story... :-)

abilitare **ftp-proxy** all'avvio al boot, da **/etc/rc.conf**

```
ftpproxy_flags=""       (con -r se active mode?)
```

Nelle regole di nat e filtraggio inserire poi:

```
nat-anchor "ftp-proxy/*"
rdr-anchor "ftp-proxy/*"
rdr on $int_if proto tcp from any to any port 21 -> 127.0.0.1 \
   port 8021
anchor "ftp-proxy/*"
```

8.4 Authpf

AUTHPF, user shell for authenticating gateways, ovvero un captive portal da console :)

- la user shell dell'utente deve essere settata a **/usr/sbin/authpf**
- per configurazione completa vedere man page di authpf
- deve per forza essere presente il file **/etc/authpf/authpf.conf** (anche vuoto)
- authpf usa queste ancore:

  ```
  nat-anchor "authpf/*"
  rdr-anchor "authpf/*"
  binat-anchor "authpf/*"
  anchor "authpf/*"
  ```

- le regole sono invece caricate da 2 files:
  - **/etc/authpf/users/$USER/authpf.rules**
  - **/etc/authpf/authpf.rules**

  se esiste il primo viene caricato, altrimenti il secondo; il primo sovrascrive il secondo

Authpf ha due macro predefinite:

$user_ip

e

$user_id

Authpf inoltre definisce la tabella

<authpf_users>

che include gli ip di tutte le macchine loggate.

ATTENZIONE: ricordarsi di definire la tabella!

```
table <authpf_users> persist
pass in on $ext_if proto tcp from <authpf_users> \
    to port smtp flags S/SA keep state
```

se si vuole bannare un utente, creare **/etc/authpf/banned/$USER**, con all'interno il messaggio che comparirà.

Quando un utente effettua correttamente il login su authpf, gli viene mostrato un messaggio. Il messaggio è definibile nel file **/etc/authpf/authpf.message**, che verrà visualizzato dopo il messaggio di default (Hello charlie. You are authenticated from host 64.59.56.140)

Esempio di **/etc/authpf/authpf.rules**:

```
wifi_if = "wi0"
pass in quick on $wifi_if proto tcp from $user_ip to port { ssh, http, \
   https } flags S/SA keep state
```

Capitolo 9

Logging

Usando la direttiva *log* di pf è possibile loggare i pacchetti che transitano per il nostro firewall.
Il salvataggio dei log avviene ad opera di **pflogd**, che si occupa di restare in ascolto sull'interfaccia **pflog0** e di scrivere sul file **/var/log/pflog** in formato **tcpdump**.

9.1 Log dei pacchetti

Come detto prima, è possibile loggare i pacchetti inserendo la direttiva *log* nelle regole di filtraggio, di nat, di binat o di rdr.
É inoltre possibile passare dei parametri alla direttiva *log*, inserendoli tra *()*:

- *all*: vengono loggati tutti i pacchetti, non solo quello inziale che genera lo stato
- *user*: vengolo loggate le informazioni riguardo agli utenti locali che hanno creato i pacchetti

```
pass in log (all) on $ext_if inet proto tcp to $ext_if port 22 keep state
```

9.2 Lettura dei log

É possibile leggere i log utilizzando l'utility **tcpdump**:

```
# tcpdump -n -e -ttt -r /var/log/pflog
```

oppure

```
# tcpdump -n -e -ttt -i pflog0
```

9.3 Filtraggio dei log

Sempre usando l'utility **tcpdump** è possibile filtrare il contenuto dei log; ad esempio

```
# tcpdump -n -e -ttt -r /var/log/pflog port 80
```

visualizzerà solo i pacchetti che hanno porta 80, mentre

```
# tcpdump -n -e -ttt -r /var/log/pflog port 80 and host 192.168.1.3
```

visualizzerà i pacchetti con porta 80 e host 192.168.1.3

Capitolo 10

Bridge Firewall

Un bridging firewall può essere utile in contesti in cui il firewall deve essere trasparente.

```
block in quick on fxp0
pass in quick on fxp1 proto tcp modulate state
pass in quick on fxp1 proto {udp, icmp} keep state
```

Anche se un bridge opera a livello 2, è possibile comunque ispezionare le informazioni di livello 3, come indirizzi ip, protocolli, etc...; ad esempio, se in una zona protetta ci sono dei server che devono erogare dei servizi:

```
pass in quick on fxp1 proto tcp modulate state
pass in quick on fxp1 proto {udp, icmp} keep state
pass in quick on fxp0 from any to 20.20.20.2/32 proto udp port 53 keep state
pass in quick on fxp0 from any to 20.20.20.3/32 proto tcp port 80 flags S/SA modulate state
```

Vediamo delle regole complete: ipotizziamo di dover splittare in due una rete, tenendo da un lato i server e dall'altro i client, ovvero creiamo una DMZ con un bridge.

```
client_if="fxp0"
server_if="fxp1"
webserver="10.0.0.2"
pass in quick on $server_if all
pass out quick on $server_if all
block in on $client_if all
block out on $client_if all
#permette connessione al server
pass in quick on $client_if from any to $webserver proto tcp port 80 \
   flags S/SA modulate state
```

Capitolo 11

Cluster di firewall

11.1 Introduzione

Costruire un cluster di firewall con OpenBSD è diventato molto semplice grazie a 2 protocolli: *carp*[1] e *pfsync*. Il primo si occupa di gestire la rindondanza a livello ip, mentre il secondo si occupa di propagare le tabelle di stato di pf.

11.1.1 Il protocollo CARP

CARP deriva dal protocollo VRRP[2] di Cisco, migliorandolo sotto diversi aspetti; le differenze principali, e più interessanti, sono 4:

- CARP, a differenza di VRRP, è esente da brevetti
- si ha la possibilità di avere un singolo ip virtuale, con più MAC-address virtuali, uno per ogni host (`arp balance`)
- utilizza la crittografia SHA1-HMAC all'interno dei pacchetti di advertisement, rendendo molto difficile la creazione di pacchetti ad-hoc
- supporta sia ipv4 che ipv6

CARP è registrato con il numero di protocollo IP **112**.

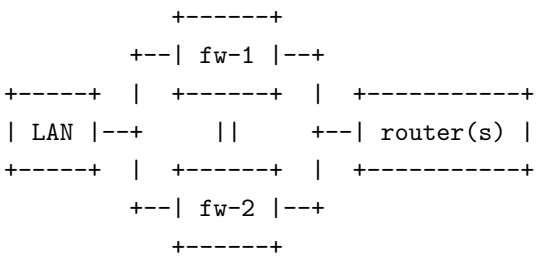
11.1.2 Il protocollo PFSYNC

PFSYNC è il protocollo (ip protocol **420**) fondamentale per gestire le tabelle di stato del nostro cluster di firewall. Ovviamente è necessario solamente se il firewall è configurato per fare lo stateful inspection, oppure in caso di NAT. In questi casi PFSYNC consente di mantenere le connessioni in atto anche in caso di fault del firewall master.
Per questioni di sicurezza è consigliabile utilizzare per PFSYNC una rete dedicata (in caso di 2

[1] Common Address Redundancy Protocol
[2] Virtual Router Redundancy Protocol

firewall basta collegarli con cavo cross). Questo per evitare sia la perdita di pacchetti che l'invio di pacchetti fasulli, forgiati ad-hoc, che potrebbero andare a modificare le tabelle di stato.

11.2 Configurazione della rete nei 2 firewall



- rete LAN:
  - rete: 192.168.0.0/24
  - firewall-1: 192.168.0.251
  - firewall-2: 192.168.0.252
  - firewall virtual ip: 192.168.0.254
- rete router(s):
  - rete: 192.168.254.0/24
  - firewall-1: 192.168.254.1
  - firewall-2: 192.168.254.2
  - firewall virtual ip: 192.168.254.10
- rete pfsync:
  - rete: 192.168.255.0/24
  - firewall-1: 192.168.255.1
  - firewall-2: 192.168.255.2

11.2.1 Configurazione firewall 1

```
# cat /etc/hostname.sis0
inet 192.168.254.1 255.255.255.0 NONE NONE
# cat /etc/hostname.sis1
inet 192.168.0.251 255.255.255.0 NONE NONE
# cat /etc/hostname.sis2
inet 192.168.255.1 255.255.255.0 NONE NONE
# cat /etc/hostname.carp0
inet 192.168.254.10 255.255.255.0 192.168.254.255 vhid 1 pass 12345
# cat /etc/hostname.carp1
```

```
inet 192.168.0.254 255.255.255.0 192.168.0.255 vhid 2 pass 67890
# cat /etc/hostname.pfsync0
up syncdev sis2
```

11.2.2 Configurazione firewall 2

```
# cat /etc/hostname.sis0
inet 192.168.254.2 255.255.255.0 NONE NONE
# cat /etc/hostname.sis1
inet 192.168.0.252 255.255.255.0 NONE NONE
# cat /etc/hostname.sis2
inet 192.168.255.2 255.255.255.0 NONE NONE
# cat /etc/hostname.carp0
inet 192.168.254.10 255.255.255.0 192.168.254.255 vhid 1 pass 12345
# cat /etc/hostname.carp1
inet 192.168.0.254 255.255.255.0 192.168.0.255 vhid 2 pass 67890
# cat /etc/hostname.pfsync0
up syncdev sis2
```

11.2.3 Regole pf

```
pass quick on sis2 proto pfsync
pass quick on { sis0 sis1 } proto carp keep state
```

11.2.4 Preemption

Nella situazione descritta sopra, all'avvio, il primo firewall che trasmette l'advertisement diventa master, e l'altro slave. Se si vuole imporre una macchina come slave (ovviamente quando il master è up) bisogna aggiungere nella configurazione di carp, nel firewall di backup, il parametro *advskew*, con un valore che può variare da 1 a 255.

```
# cat /etc/hostname.carp0
inet 192.168.254.10 255.255.255.0 192.168.254.255 vhid 1 advskew 100 pass 12345
# cat /etc/hostname.carp1
inet 192.168.0.254 255.255.255.0 192.168.0.255 vhid 2 advskew 100 pass 67890
```

Inoltre, in entrambi i firewall, deve essere abilitato in **/etc/sysctl.conf**

```
net.inet.carp.preempt=1
```

Capitolo 12

Tips & Tricks

Quando in e quando out?

Un dubbio è: quando usare *in* e quando usare *out* (es: *pass in*, *block out*)?
Il trucco è quello di lavorare sui pacchetti il prima possibile; ad esempio:

- da X verso il firewall: ovvio, uso *in*
- dal firewall verso X: ovvio, uso *out*
- da Y verso X: uso *in*, salvo casi particolari, da vedere sul momento

```
# pass tcp, udp, and icmp out on the external (Internet) interface.
# keep state on udp and icmp and modulate state on tcp.
# se si hanno più interfacce interne che vanno in internet meglio farlo in out.
# in ogni caso bisogna farlo in out per i pacchetti originati dal firewall.
# idem verso eventuale dmz
pass out on $ext_if proto tcp all modulate state flags S/SA
pass out on $ext_if proto { udp, icmp } all keep state
```

Un altro proxy ftp

(solo per vecchie versioni)
Inserire nel file **/etc/inetd.conf**

```
127.0.0.1:8021 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy -n
```

e poi, tra le regole di pf:

```
rdr on $int_if proto tcp from any to any port ftp -> 127.0.0.1 port 8021
pass in on $ext_if inet proto tcp from port ftp-data \
    to ($ext_if) user proxy flags S/SA keep state
```

Web proxy trasparente

É possibile redirigere in modo trasparente all'utente del traffico ad un servizio in esecuzione su altre macchine, ad esempio un proxy http:

```
rdr on $int_if proto tcp from $int_net to any port 80 \
                                    -> 127.0.0.1 port 3128
```

Applicationi tcp multicast o igmp

Se si hanno applicazioni che usano multicast o igmp bisogna lasciar passare le opzioni ip:

```
pass in quick on fxp0 all allow-opts
```

Una tabella contente tutti gli ip del firewall

```
table <firewall> const { self }
```

Un semplice tcp proxy

Inserire nel file **/etc/inetd.conf**

```
127.0.0.1:5000 stream tcp nowait nobody /usr/bin/nc nc -w 20 192.168.1.10 80
```

queste è poi la regola rdr:

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> 127.0.0.1 port 5000
```

Logging remoto

Con qualche trucco è possibile mandare a un server syslog il log dei pacchetti:
Innanzitutto creiamo lo script **/usr/local/bin/pflogrotate**

```
#!/bin/sh
PFLOG=/var/log/pflog
FILE=/var/log/pflog5min.$(date "+%Y%m%d%H%M")
kill -ALRM $(cat /var/run/pflogd.pid)
if [ -r $PFLOG ] && [ $(stat -f %z $PFLOG) -gt 24 ]; then
   mv $PFLOG $FILE
   kill -HUP $(cat /var/run/pflogd.pid)
   tcpdump -n -e -ttt -r $FILE | logger -t pf -p local0.info
   rm $FILE
fi
```

aggiungiamolo quindi ai cronjob di root:

```
# crontab -u root -e

# rotate pf log file every 5 minutes
0-59/5 * * * * /bin/sh /usr/local/bin/pflogrotate
```

aggiungiamo quindi le seguenti linee a **/etc/syslog.conf**

```
local0.info     /var/log/pflog.txt
```

per il logging su file ascii, e

```
local0.info     @192.168.13.24
```

per il logging su un server remoto.
Si può ora disabilitare la rotazione automatica di **/var/log/pflog**, e abilitare quella di **/var/log/pflog.txt**. Per fare questo modifichiamo il file **/etc/newsyslog.conf**

```
#/var/log/pflog    600 3 250 *  ZB /var/run/pflogd.pid
/var/log/pflog.txt 600 7 *   24
```

Anti-spam gateway

In maniera semplice ma efficace è possibile usare un firewall openbsd+pf come anti-spam gateway. Tutto questo con il demone `spamd`. Vediamo in maniera semplice il principio di funzionamento: Viene inizializzata una connessione verso il server di posta. Pf la intercetta, se il server inviante è nella sua whitelist la fa arrivare correttamente al nostro server di posta, altrimenti:

- se il server è nella blacklist procede inviando la risposta un bit al secondo, ovviamente senza accettare poi la mail, facendo così perder tempo al server sorgente :)
- altrimenti inserisce i dati nella mail nella greylist (principio già noto), e successivamente in whitelist

Iniziamo modificando **/etc/rc.conf**:

```
spamd_flags="-v -G5:4:864"
```

Modifichiamo poi la configurazione di `spamd`, **/etc/spamd.conf**:

```
all:\
:spews1:

# Mirrored from http://www.spews.org/spews_list_level1.txt
spews1:\
:black:\
```

```
:msg="SPAM. Your address %A is in the spews level 1 database\n\
See http://www.spews.org/ask.cgi?x=%A for more details":\
:method=http:\
:file=www.openbsd.org/spamd/spews_list_level1.txt.gz:

# Mirrored from http://www.spews.org/spews_list_level2.txt
spews2:\
:black:\
:msg="SPAM. Your address %A is in the spews level 2 database\n\
See http://www.spews.org/ask.cgi?x=%A for more details":\
:method=http:\
:file=www.openbsd.org/spamd/spews_list_level2.txt.gz:
```

Vediamo ora come dovrebbe essere configurato pf:

```
# grey host list
table <spamd> persist

# white host lists
table <spamd-white> persist

# This whitelist we are manually maintaining.
table <whitelist> persist file "/etc/whitelist.txt"

# The order of these rules is very, very important.
# send manually whitelisted hosts to the actual mail server
rdr on $ext_if proto tcp from <whitelist> to \
$ext_if port smtp -> $mail_host port smtp

# send all previously seen but still suspect senders the spamd daemon
rdr pass on $ext_if inet proto tcp from <spamd> to \
         $ext_if port smtp -> 127.0.0.1 port 8025

# send all unknown senders to the spamd daemon
rdr pass on $ext_if inet proto tcp from !<spamd-white> to \
         $ext_if port smtp -> 127.0.0.1 port 8025

# send whitelisted hosts to the actual mail server
rdr on $ext_if proto tcp from <spamd-white> to \
$ext_if port smtp -> $mail_host port smtp
```

Creiamo ora il file **whitelist.txt**:

```
#!/bin/sh

FILE=spamd-spf.txt
```

```
rm -f $FILE
touch $FILE

for domain in \
        aol.com \
        apple.com \
        amazon.com \
        gmx.net \
        _spf.google.com \
        spf-a.hotmail.com \
        spf-b.hotmail.com \
        spf-c.hotmail.com \
        spf-d.hotmail.com \
        _spf-a.microsoft.com \
        _spf-b.microsoft.com \
        _spf-c.microsoft.com \
        mynethost.com
do
echo \#$domain >> $FILE;
dig $domain TXT +short | tr "\ " "\n" | grep ^ip4: | cut -d: -f2 >> $FILE;
done

echo '# http://cvs.puremagic.com/viewcvs/*checkout*/greylisting/schema/whitelist_ip.txt' >>
$FILE;

ftp -o - http://cvs.puremagic.com/viewcvs/*checkout*/greylisting/schema/whitelist_ip.txt \
    | awk '/^[^#]/ {print $0}' >> $FILE;
```

Parte II

OpenBSD & Networking

Capitolo 13

Appunti sparsi su OpenBSD

13.1 Creazione ISO OpenBSD

```
$ wget -c -r -np ftp://spargel.kd85.com/pub/OpenBSD/4.0/i386
$ mv spargel.kd85.com/pub/OpenBSD/4.0 .
$ cp 4.0/i386/cd40.iso OpenBSD-4.0.iso
$ growisofs -M OpenBSD-4.0.iso -R -iso-level 3 -graft-points 4.0=4.0
$ rm -rf 4.0 spargel.kd85.com
```

13.2 Configurazione permanente interfacce di rete

```
# cat /etc/myname
koala.gnustile.lan

# cat /etc/mygate
192.168.3.254

# cat /etc/resolv.conf
lookup file bind
nameserver 192.168.3.251

# cat /etc/hostname.ep0
inet 192.168.3.23 255.255.255.0 192.168.3.255
inet alias 192.168.3.24 255.255.255.255

# cat /etc/hostname.ep1
dhcp NONE NONE NONE

# cat /etc/hostname.wi0
inet 10.0.0.1 255.255.255.0 10.0.0.255
```

```
!wicontrol  \$if -e 1 -k "secretkey" -p 1 -c 1 -s "hostname" -n "networkname" \
     -q "networkname" -f 3
```

Il comando **wicontrol** dell'interfaccia wireless imposta cifratura web (-e 1), con chiave *secretkey*. **man wicontrol** aiuta sicuramente.

13.3 Abilitazione ip forwarding

```
# vi /etc/sysctl.conf

net.inet.ip.forwarding=1
```

13.4 Creazione di un bridge

É semplicissimo creare un bridge permanente:

```
# echo up > /etc/hostname.fxp0
# echo up > /etc/hostname.fxp1
# vi /etc/bridgename.bridge0
   add fxp0
   add fxp1
   blocknonip fxp0
   blocknonip fxp1
   up
```

Può aiutare la manpage **man brconfig**.

13.5 Gestione utenti

Alcuni comandi per la gestione utenti:

```
adduser
rmuser
vipw
chsh
chpass
passwd
group*
user*
```

13.5.1 su

L'utente che esegue **su** deve appartenere al gruppo `wheel`

13.5.2 Shell ksh

```
# vi /etc/profile

export PS1='$USER@koala:$PWD $ '
```

13.6 Keyboard and Display

13.6.1 Keyboard mapping

```
# wsconsctl -w keyboard.encoding=it
```

In ogni caso **man wsconsctl** aiuta.

13.6.2 Cancellare la console al logout

```
# vi /etc/gettytab

P|Pc|Pc console:\
     :np:sp#9600:\
     :cl=\E[H\E[2J:
```

13.6.3 Console seriale

Iniziamo con le modifiche a **/etc/ttys**:
modifichiamo

```
tty00  "/usr/libexec/getty std.9600"  unknown  off
```

in

```
tty00  "/usr/libexec/getty std.9600"  vt100  on secure
```

Per abilitare le informazioni di boot via console seriale modifichiamo **/etc/boot.conf**:

```
set tty com0
```

13.7 Rimozione di un package di base

Ipotizziamo di voler rimuovere **compXX.tgz**:

```
cd /
rm -rf `tar tzf /path/di/compXX.tgz`
```

13.8 Pidof & Killall

Ecco due semplici script che assolvono i compiti di *pidof* e *killall*:

pidof

```
#!/bin/sh
ps axc -o pid,command | awk "/‘echo $1‘/ {print \$1}"
```

killall

```
#!/bin/sh
kill ‘/sbin/pidof $1‘
```

Capitolo 14

Appunti sparsi sui servizi di rete con OpenBSD

14.1 Server DHCP

Iniziamo impostando il sistema perchè faccia partire dhcpd al boot, modificando **/etc/rc.conf**:

```
dhcpd_flags=""
```

Nel file **/etc/dhcpd.interfaces** ci sarà l'elenco delle interfacce su cui **dhcpd** rimarrà in ascolto:

```
# cat /etc/dhcpd.interfaces

ep0 wi0
```

Possiamo ora editare il file di configurazione principale, **/etc/dhcpd.conf**:

```
option domain-name "gnustile.net";
option domain-name-servers 192.168.3.201, 192.168.3.202;
option default-lease-time 604800;
# lease time: 1 week

# internal net
subnet 192.168.3.0 netmask 255.255.255.0 {
     option routers 192.168.3.254;
     option broadcast-address 192.168.3.255;
     option subnet-mask 255.255.255.0;
     range 192.168.3.100 192.168.3.150;
}

subnet 10.0.0.0 netmask 255.255.255.0 {
     option routers 10.0.0.1;
```

```
     option broadcast-address 10.0.0.255;
     range 10.0.0.2 10.0.0.199;
}
```

É necessario poi dire a pf di consentire il traffico dhcp:

```
pass in quick on {ep0, wi0} proto {tcp, udp} port 67 keep state
```

14.2 Server DNS

Per prima cosa impostiamo bind per l'avvio al boot:

```
# vi /etc/rc.conf

named_flags=""
```

Poi modificando il file **/var/named/etc/named.conf**:

```
# vi /var/named/etc/named.conf

# opzioni
options {
   [...];
   forwarders { 193.23.12.34; 62.100.65.2; };
};

# acl di accesso
acl clients {
   127/8; 192.168.3/24
};

# una zona master
zone "gnustile.lan" {
     type     master;
     file     "master/gnustile.lan";
};

zone "3.168.192.in-addr.arpa" {
     type     master;
     file     "master/db.192.168.3";
};

# una zona slave
zone "funkyway.lan" {
     type     slave;
```

```
     file     "slave/funkyway.lan";
     masters  { 192.168.12.2; 192.168.12.3; };
};
```

Modifichiamo ora la nostra zona master: **/var/named/master/gnustile.lan**:

```
$TTL 3h
@ IN SOA ns1.gnustile.lan. root.gnustile.lan. (
     2007070701  ;  serial
     3h   ; refresh after 3 hours
     1h   ; retry after 1 hour
     1w   ; expire after 1 week
     1h ) ; negative caching ttl of 1 hour

  IN NS    ns1.gnustile.lan.
  IN NS    ns2.gnustile.lan.
  IN MX    10  mx1.gnustile.lan.
  IN MX    20  mx2.gnustile.lan.
  IN A     192.168.3.210

localhost IN A 127.0.0.1

mx1   IN A 192.168.3.220
mx2   IN A 192.168.3.221
ns1   IN A 192.168.3.201
ns2   IN A 192.168.3.202
www   IN A 192.168.3.210
proxy IN A 192.168.3.203
      IN A 192.168.3.204
      IN A 192.168.3.205
cache IN CNAME proxy
```

e **/var/named/master/db.192.168.3**

```
$TTL 3h
@ IN SOA ns1.gnustile.lan. root.gnustile.lan. (
     2007070701  ;  serial
     3h   ; refresh after 3 hours
     1h   ; retry after 1 hour
     1w   ; expire after 1 week
     1h ) ; negative caching ttl of 1 hour

  IN NS    ns1.gnustile.lan.
  IN NS    ns2.gnustile.lan.

1   IN PTR host001.gnustile.lan.
```

```
[...]
101 IN PTR host101.gnustile.lan.
[...]
201 IN PTR ns1.gnustile.lan.
```

É possibile separare i contesti di visibilità; ad esempio per gli ip pubblici che verranno nattati; modifichiamo quindi **named.conf**:

```
acl "internal" {
   127/8;
   192.168.3/24;
};
view "internal" {
   match-clients { "internal" };
   recursion yes; # abilita il lookup verso internet
   zone "..." {
      [...]
   };
};
view "internet" {
   match-clients { any; }
   recursion no;
   zone "..." {
      [...]
   };
};
```

14.3 OpenNTPD

14.3.1 Server

```
# vi /etc/ntpd.conf

# $OpenBSD: ntpd.conf,v 1.7 2004/07/20 17:38:35 henning Exp $
# sample ntpd configuration file, see ntpd.conf(5)
# Addresses to listen on (ntpd does not listen by default)

#listen on *
listen on ::1

# sync to a single server
#server ntp.example.org
# use a random selection of 8 public stratum 2 servers
# see http://twiki.ntp.org/bin/view/Servers/NTPPoolServers
servers pool.ntp.org
```

14.3.2 Client

```
# vi /etc/ntpd.conf

server 192.168.3.204
```

Capitolo 15

Appunti sparsi sulle VPN con OpenBSD

Attenzione: è necessario decommentare le seguenti righe in **/etc/sysctl.conf**:

```
net.inet.ip.forwarding=1
net.inet.esp.enable=1
```

Inoltre, per abilitare il demone *isakmpd* al boot, è necessario modificare in **/etc/rc.conf**

```
isakmpd_flags=""
```

15.1 VPN ipsec site-to-site con Pre-Shared Keys

La nostra rete

```
+-----+  +----+      +----+  +-----+
|lan_A|--|gw_A|--<>--|gw_B|--|lan_B|
+-----+  +----+      +----+  +-----+
```

- gateway A:
  - IP pubblico: 62.100.95.12
  - IP privato: 192.168.1.254
- gateway B:
  - IP pubblico: 62.120.43.10
  - IP privato: 192.168.2.254

Configurazione del gateway A

/etc/isakmpd/isakmpd.conf

```
[Phase 1]
62.120.43.10= peer-gwB

[Phase 2]
Connections= VPN-A-B

[peer-gwB]
Phase= 1
Transport= udp
Address= 62.120.43.10
Configuration= Default-main-mode
Authentication= change_this_to_your_secret

[VPN-A-B]
Phase= 2
ISAKMP-peer= peer-gwB
Configuration= Default-quick-mode
Local-ID= gwA-internal-network
Remote-ID= gwB-internal-network

[gwA-internal-network]
ID-type= IPV4_ADDR_SUBNET
Network= 192.168.1.0
Netmask= 255.255.255.0

[gwB-internal-network]
ID-type= IPV4_ADDR_SUBNET
Network= 192.168.2.0
Netmask= 255.255.255.0

[Default-main-mode]
DOI= IPSEC
EXCHANGE_TYPE= ID_PROT
Transforms= 3DES-SHA,BLF-SHA

[Default-quick-mode]
DOI= IPSEC
EXCHANGE_TYPE= QUICK_MODE
Suites= QM-ESP-3DES-SHA-SUITE
```

Configurazione del gateway B

/etc/isakmpd/isakmpd.conf

```
[Phase 1]
62.100.95.12= peer-gwA

[Phase 2]
Connections= VPN-B-A

[peer-gwA]
Phase= 1
Transport= udp
Address= 62.100.95.12
Configuration= Default-main-mode
Authentication= change_this_to_your_secret

[VPN-B-A]
Phase= 2
ISAKMP-peer= peer-gwA
Configuration= Default-quick-mode
Local-ID= gwB-internal-network
Remote-ID= gwA-internal-network

[gwA-internal-network]
ID-type= IPV4_ADDR_SUBNET
Network= 192.168.1.0
Netmask= 255.255.255.0

[gwB-internal-network]
ID-type= IPV4_ADDR_SUBNET
Network= 192.168.2.0
Netmask= 255.255.255.0

[Default-main-mode]
DOI= IPSEC
EXCHANGE_TYPE= ID_PROT
Transforms= 3DES-SHA,BLF-SHA

[Default-quick-mode]
DOI= IPSEC
EXCHANGE_TYPE= QUICK_MODE
Suites= QM-ESP-3DES-SHA-SUITE
```

In entrambe le macchine

/etc/isakmpd/isakmpd.policy

```
Keynote-version: 2
Authorizer: "POLICY"
Conditions: app_domain == "IPsec policy" &&
            esp_present == "yes" &&
            esp_enc_alg != "null" -> "true";
```

Esempio di configurazione di pf per gateway A

```
#macros
int_if="fxp0"
ext_if="fxp1"
vpn_if="enc0"
icmp_types = "{ echorep, echoreq, timex, unreach }"
GATEWAY_A = "62.100.95.12/32"
GATEWAY_B = "62.120.43.10/32"
NETWORK_A = "192.168.1.0/24"
NETWORK_B = "192.168.2.0/24"

#scrubbin in ingresso
scrub in all

#nat verso internet
nat on $ext_if from $int_if:network to any -> $ext_if

#regole di filtraggio
block in log on { $vpn_if, $ext_if } all
block out on { $vpn_if, $ext_if } all

pass quick on lo0 all

pass in on $ext_if inet proto icmp from any to $ext_if icmp-type $icmp_types keep state

pass in on $int_if from $int_if:network to any keep state
pass out on $int_if from any to $int_if:network keep state

pass out on $ext_if proto tcp all modulate state
pass out on $ext_if proto { udp, icmp } all keep state

#regole vpn
# passa il traffico criptato tra i 2 gw
pass in quick on $ext_if proto esp from $GATEWAY_B to $GATEWAY_A
```

```
pass out quick on $ext_if proto esp from $GATEWAY_A to $GATEWAY_B

# lascia passare il protocollo ipencap sull'interfaccia del tunnel vpn
pass in quick on $vpn_if proto ipencap all

#consente il traffico tra le 2 sottoreti della vpn
pass in quick on $vpn_if from $NETWORK_B to $NETWORK_A
pass out quick on $vpn_if from $NETWORK_A to $NETWORK_B

#consente il traffico di isakmpd
pass in quick on $ext_if proto udp from $GATEWAY_B to $GATEWAY_A port isakmpd
pass out quick on $ext_if proto udp from $GATEWAY_A to $GATEWAY_B port isakmpd
```

15.2 VPN ipsec client-to-site con Pre-Shared Keys

PS: `mai testato, non sono sicuro che funzioni :)`
Affronteremo solo la configurazione del gateway, in quanto la configurazione del client dipende dal sistema operativo. Verranno comunque forniti alcuni link che potranno aiutare nella configurazione della stessa.
Una configurazione simile a quella che vedremo può anche essere utilizzata per cifrare il traffico di una connessione wireless.
Il file **/etc/isakmpd/isakmpd.policy** è uguale a quello visto nel paragrafo precedente. Vedremo solo il file **/etc/isakmpd/isakmpd.conf**:

```
[General]
Retransmits=            5
Exchange-max-time=      120
Check-interval=         60

# IKE Phase 1 & 2
[Phase 1]
Default=                Client-phase1
[Phase 2]
Passive-Connections=    Client-phase2

# Clients, phase 1 and 2
[Client-phase1]
Phase=                  1
Transport=              udp
Local-address=          62.100.95.12
Configuration=          Default-main-mode
Authentication= change_this_to_your_secret

[Client-phase2]
```

```
Phase=                  2
Configuration=          Default-quick-mode
Local-ID=               My-network
Remote-ID=              Unknown-address

# Network details
[My-network]
ID-Type=                IPV4_ADDR_SUBNET
Network=                192.168.3.0/24
Netmask=                255.255.255.0
[Unknown-address]
ID-Type=                IPV4_ADDR
Address=                0.0.0.0

[Default-main-mode]
DOI= IPSEC
EXCHANGE_TYPE= ID_PROT
Transforms= 3DES-SHA,BLF-SHA

[Default-quick-mode]
DOI= IPSEC
EXCHANGE_TYPE= QUICK_MODE
Suites= QM-ESP-3DES-SHA-SUITE
```

Per proteggere il traffico di una rete wireless sarebbe sufficiente sostituire le relative parti con

```
[Client-phase2]
Phase=                  2
Configuration=          Default-quick-mode
Local-ID=               All-networks
Remote-ID=              Unknown-address

# Network details
[All-networks]
ID-Type=                IPV4_ADDR_SUBNET
Network=                0.0.0.0
Netmask=                0.0.0.0
```

Esempio di configurazione di pf per il gateway

```
# ESP and IKE
pass out quick on $ext_if proto esp from any to any
pass in quick on $ext_if proto esp from any to any
pass out quick on $ext_if proto udp from $ext_ip to any \
port isakmpd
```

```
pass in quick on $ext_if proto udp from any to $ext_ip \
port isakpmd
```