

# Introduzione al firewall di OpenBSD



# OpenBSD: Packet Filter

Come già accennato, pf (PacketFilter) nasce per questioni di licenza (ipf aveva cambiato licenza ed il team di OpenBSD non accettava la nuova licenza).

Questo prodotto è stato l'elemento che ha reso noto ai più, assieme ad OpenSSH, il progetto OpenBSD.

pf è stato portato su tutte le famiglie BSD (NetBSD e FreeBSD) nonché anche su altre piattaforme (per esempio windows: CoreForce), anche se le nuove features sono disponibili, di norma, solo su OpenBSD.

# pf: comandi e configurazione

pf è totalmente gestito per mezzo del comando *pfctl*.

pfctl ha 3 funzionalità che ci possono interessare su base quotidiana:

- Caricamento e validazione del file di configurazione
- Modifica delle tabelle
- Verifica dello stato del filtro e delle sessioni

# pf: comandi e configurazione

Vediamo ora alcune delle opzioni più comuni con *pfctl*

*pfctl -a <anchor name> <commands>*

Questa funzione serve per interagire con le *anchors* che vedremo più avanti.

*pfctl -d / pfctl -e*

Questa funzione serve per disabilitare o abilitare pf

*pfctl -f <nome file>*

Questa funzione serve a dire da quale file vogliamo che pf legga il nostro ruleset

# pf: comandi e configurazione

*pfctl -n*

Questa funzione istruisce pf di non attivare le regole o i comandi che gli stiamo dando. È utile, assieme a *-g* ed *-f <filename>*, per verificare che il file di regole non contenga errori di sintassi (questo però non ci garantisce che quello che abbiamo scritto sia corretto!)

*pfctl -k <host/network> / pfctl -K <host/network>*

Questa funzione è utile a terminare una specifica connessione da un host o da una rete. *k* termina la/le sessione/sessioni, *K* serve a terminare le source tracking.

# pf: comandi e configurazione

*pfctl -m*

Questa funzione per unire alle regole attive le regole che aggiungiamo (queste verranno accodate alle relative sezioni).

*pfctl -s <tipo>*

Questa funzione a mostrare lo stato di pf.

I tipi più usati sono:

# pf: comandi e configurazione

- nat: mostra il contenuto delle regole di nat/rdr
- queue: mostra le regole per il controllo della banda
- rules: mostra le regole di filtering
- Anchors: mostra le anchors attive nella configurazione
- state: mostra la tabella degli stati delle sessioni
- all: mostra lo stato di pf in tutte le sue parti (stati, regole, ottimizzazione, tables, etc)

# pf: comandi e configurazione

*pfctl -T <comando> -t <nome della table>*

Ci permette di operare nella tabella.

(vedremo più indettaglio questa parte con le tabelle)



# pf.conf

Di norma pf viene gestito per mezzo del file */etc/pf.conf*.

pf.conf è un file con una propria e precisa sintassi.

pf.conf è diviso in sezioni o *statement*.

Queste sono 7.

# pf.conf - macro

Il primo statement che andremmo a vedere sono le macro (*macros*)

Queste sono delle variabili globali di pf.conf.

Devono essere definite prima di essere utilizzate all'intero della configurazione. Non possono coesistere due macro con lo stesso nome (nemmeno una sezione stub).

Alcuni esempi:

# pf.conf - macro

external\_interface = "rl0"

all\_interface = "{ rl0, fxp0, lo0, tun0 }"

smtp\_hosts = "{ 192.168.1.25 }"

InTCP = "{ ssh,smtp,domain,www,81,https,<1023 }"

local\_net = "{ 192.168.1.0/24 }"

all\_net = "{ \$local\_net 80.23.24.0/29 }"

no\_bad\_host = "{ !192.168.5.3/32 }"

# pf.conf - tabelle

Le tabelle ( o *tables* ), concettualmente, sono oggetti **simili** alle macro.

Devono, come le macro, esser definiti prima di essere utilizzati.

A differenza delle macro il loro contenuto può essere vuoto.

Le tabelle contengono indirizzi.

Il contenuto di una tabella può esser variato dinamicamente.

# pf.conf - tabelle

Le tabelle posso essere popolate 'a mano' o per mezzo di un file.

Una tabella può essere anche statica (ovvero viene letta solo al caricamento di pf.conf)

Le tabelle sono più efficienti e flessibili, se comparate alle macro, quando si hanno in gioco molteplici indirizzi.

```
<server_virus> persist
```

```
<i_buoni> const file “/il_mio_file_sicuro”
```

```
<server_amici> { 192.168.2.0/25, 172.16.2.0/24 }
```

# pf.conf - tabelle

Le tabelle vengono popolate dinamicamente dall'amministratore di sistema per mezzo del comando `pfctl -t NomeTabella -T` [comando con possibili opzioni]

I comandi sono:

`.add`

`.show`

`.delete`

# pf.conf - opzioni

Le opzioni permettono di controllare il comportamento del filtro.

Per esempio il tempo per un timeout di sessione, il numero di stati in memoria, di ip originanti, di tabelle o di quanti dati possano esser conservati in una tabella.

# pf.conf - normalizzazione

Una feature interessante (ma al contempo potenzialmente dannosa) di pf è lo *scrub* o normalizzazione del traffico.

Questa funzionalità se correttamente implementata protegge gli host da attacchi che si basano su pacchetti apparentemente validi, ma che hanno come scopo ultimo compromettere il target.

Basti pensare a pacchetti che giungono frammentati (alcuni possono essere leciti, ma altri possono essere usati per riempire il buffer/stack).



# pf.conf - normalizzazione

Quando dico dannoso intendo che, non tutto il traffico in arrivo con il flag di pacchetto frammentato, detto debba esser ricomposto prima di essere passato all'host di destinazione.

Per esempio, posso eseguire la normalizzazione di tutti i pacchetti, escludendo quelli che abbiano il bit set "do not fragment".

*scrub in on rl0 no-df*

# pf.conf - Queueing

pf ha al suo interno anche un ottimo strumento di gestione della banda.

Lo scheduler predefinito di OpenBSD è basato su FIFO (First In, First Out).

Quest'ultimo, a volte, non è confacente alle esigenze moderne (non si può attendere, per esempio, il termine di un download da un sito ftp, per poter vedere il resto dello streaming video)

A tal fine in OpenBSD sono stati introdotti due tipi di scheduler:

- .CBQ (Class Based Queueing)

- .PRIQ (Priority Queueing)

# pf.conf - Queueing

Le FIFO sono la forma più semplice di gestione di banda.

Alla banda viene assegnato un valore ed i pacchetti vengono processati secondo la regola del primo venuto.

In caso di congestione si potranno quindi perdere dati (o flussi) anche importanti.

# pf.conf - Queueing

Le CBQ sono organizzate in maniera gerarchica.

La banda viene divisa tra più code/classi.

Ad ogni coda viene poi assegnata una certa banda minima.

Vi è anche la possibilità di prendere banda da altre code, se queste non sono in uso (*borrow*)

Da notare che CBQ utilizza le priorità per gestire il traffico. Si potranno quindi avere varie priorità (più è alto il livello di priorità maggiore sarà la precedenza assegnata a quel traffico, nel caso ci si trovi di fronte a congestione)

# pf.conf - Queueing

Root Queue (2Mbps)

    UserA (1Mbps, priority 2)

        ssh (50Kbps, priority 5)

        bulk (950Kbps, priority 1)

    UserB (1Mbps, priority 1)

        audio (250Kbps, priority 1)

        bulk (750Kbps,borrow)

            http (100Kbps)

            other (650Kbps)

# pf.conf - Queueing

Con PRIOQ si gestisce la banda ricorrendo alla sola priorità assegnata.

Ciò, implica che la coda con massima priorità, avrà la garanzia di esser sempre servita, mentre le priorità minori potrebbero, in caso di congestione, non venire servite.

# pf.conf - Queueing

Root Queue (2Mbps)

Queue A (priority 1)

Queue B (priority 2)

Queue C (priority 3)

# pf.conf - Queueing

Per gestire le congestioni pf utilizza RED ed ECN

RED è un acronimo che sta per Random Early Detection

Il sistema RED, si basa essenzialmente su due soglie di livello, che sono calcolate dinamicamente a seconda della lunghezza delle code stesse.

.minimo: ogni coda al di sotto di questo livello sarà **SEMPRE** processata

.massimo: ogni nuovo pacchetto che arrivi in questa coda verrà scartato



# pf.conf - Queueing

RED risulta quindi utile specie nelle situazioni di *global synchronization* (ovvero quando siamo di fronte ad un momento di elevata congestione, il sistema scarta tutti i pacchetti, indifferentemente dalle sessioni) in quanto permette di intervenire solo su alcune code, andando a non terminare tutte le sessioni ma solo una parte di esse.

*RED riesce a gestire, in pratica, solo connessioni che ricorrano a protocolli di trasporto che rispondano ad indicatori/trigger di congestione (TCP)*

# pf.conf - Queueing

ECN è un acronimo che sta per Explicit Congestion Notification.

ECN funziona assieme a RED. ECN avverte l'host remoto su eventuali congestioni presenti nel percorso tra il mittente ed il ricevente. Questa notifica è ottenuta per mezzo di un flag nell'header del pacchetto. Se il mittente supporta l'ECN leggerà il pacchetto del destinatario, e di conseguenza rallenterà o abbasserà la sua trasmissione.

# pf.conf - Queueing

La sintassi per gestione delle code è basata su 2 parti:

- .Definizione della coda (queue e altq)

- .Tagging dei pacchetti.

# pf.conf - Queueing

Vi sono due direttive per abilitare le code in pf:

*.altq on*

*.queue*

# pf.conf - Queueing

La sintassi per **altq** è:

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

# pf.conf - Queueing

- *interface* - l'interfaccia di rete sulla quale attivare il queueing.
- *scheduler* - lo scheduler queueing da utilizzare. E' possibile scegliere tra cbq e priq. Si può attivare un solo scheduler alla volta.
- *bw* - l'ammontare totale di banda disponibile allo scheduler. Questo può essere specificato come un valore assoluto utilizzando i suffissi b, Kb, Mb, e Gb che rappresentano rispettivamente bit, kilobit, megabit, e gigabit al secondo, o come una percentuale della banda dell' interface.
- *qlim* - il massimo numero di pacchetti da conservare nella coda. Questo parametro è opzionale. Il valore di default è 50.
- *size* - ampiezza del regolatore in byte. Se non specificato, l'ampiezza è definita in base alla banda dell'interface.
- *queue\_list* - una lista di code figlie da creare al di sotto della coda root.

# pf.conf - Queueing

La sintassi per **queue** è:

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

# pf.conf - Queueing

- *name* - il nome della coda. Questo deve corrispondere a uno dei nomi delle code definite nella direttiva *altq on* di *queue\_list*. Per *cbq* può anche corrispondere al nome di una coda in una precedente coda della direttiva *queue\_list*. Il nome della coda non può essere più grande di 15 caratteri.
- *interface* - l'interfaccia di rete sulla quale c'è una coda valida. Questo valore è opzionale, e se non specificato, la coda sarà valida su tutte le interfacce.
- *bw* - l'ammontare totale di banda disponibile sulla coda. Questo può essere specificato come un valore assoluto usando un suffisso b, Kb, Mb, e Gb per rappresentare bit, kilobit, megabit, e gigabit al secondo, oppure come una percentuale della banda della coda madre. Questo parametro è applicabile solo quando si utilizza lo scheduler *cbq*. Se non viene specificato, il valore di default è 100% della banda della coda madre.



# pf.conf - Queueing

- *pri* - la priorità della coda. Per cbq il range di priorità può variare tra 0 e 7 e per priq il range è tra 0 e 15. Il valore 0 di priorità è il minore. Quando non specificato, viene usato il valore di default 1.
- *qlim* - è il massimo numero di pacchetti da conservare nella coda. Quando non specificato viene utilizzato il valore di default pari a 50.
- *scheduler* - lo scheduler utilizzato, cbq o priq. Deve essere lo stesso della coda root.

# pf.conf - Queueing

- *sched\_options* - ulteriori opzioni possono essere passate allo scheduler per controllare il suo comportamento:
  - *default* - definisce una coda di default per tutti quei pacchetti che non hanno una corrispondenza con le altre code. Viene richiesta esattamente una coda di default.
  - *red* - abilita il Random Early Detection (RED) su questa coda.
  - *rio* - abilita RED con IN/OUT. In questo modo, RED conserverà molteplici valori di media per la lunghezza delle code e per i valori di soglia, uno per ogni livello di Quality of Service IP.
  - *ecn* - abilita Explicit Congestion Notification (ECN) su questa coda. Ecn implica red.
  - *borrow* - la coda può prendere in prestito banda dalla sua coda madre. Questo si può specificare solo quando si utilizza lo scheduler cbq.

# pf.conf - Queueing

- *queue\_list* - una lista di code figlie da creare al di sotto di questa coda. Una *queue\_list* può essere definita solo quando si usa lo scheduler cbq.

Assegnazione della coda ad un servizio: esempio

pass out on fxp0 from any to any port 22 queue(ssh\_bulk, ssh\_login)

# pf.conf - Translations

Con pf abbiamo 3 tipi di translations:

rdr, nat, bi-nat

# pf.conf – Translations rdr

rdr ha la funzione di ridirigere il traffico verso un'altra destinazione.

Il forwarding del traffico in ingresso può essere per porta o per indirizzo.

```
rdr on interface proto proto_type from src_addr [port port_no] \  
    to dst_addr port port_no -> redirected_addr port port_no
```

# pf.conf – Translations nat

nat permette di mascherare molteplici host con un singolo indirizzo (è più o meno quello che si fa con un comune router).

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

# pf.conf – Translations nat

Diamo un'occhiata alla pagina delle FAQ su nat

<http://www.openbsd.org/faq/pf/it/nat.html#config>

# pf.conf – Translations binat

bi-nat: è un caso particolare in cui possiamo mappare un indirizzo, in maniera statica, ad un altro.

Per esempio vogliamo esporre un server privato su internet. Oppure dobbiamo mettere in VPN due uffici che hanno lo stesso piano di indirizzi, e molti altri casi.

*binat on interface from internal\_addr to any -> external\_addr*



# pf.conf – Filtering Rules

Passiamo ora alla sezione più ampia di pf.conf: le regole vere e proprie.

Le regole sono processate dalla prima all'ultima. Nel caso di pf, vale la regola: last-match wins.

Sarà quindi buona regola definire come prima la regola di base (o di default).

Vediamo ora la sintassi di base di una rule:

# pf.conf – rules

```
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]
```

# pf.conf – Filtering Rules

Analizzeremo, in questo contesto, solo alcune delle opzioni della sintassi.

Per una spiegazione più dettagliata ed esaustiva si può ricorrere alla man page di pf.conf e alle FAQ

(<http://www.openbsd.org/faq/pf/it/filter.html#syntax>).

# pf.conf – Filtering Rules

*action* questa è la regola principale che spiega a pf cosa fare con il pacchetto che stiamo analizzando (se risponde ai requisiti del filtro).

Può avere come opzioni o *pass* o *block*.

Nel caso di *block* potremmo anche variare la policy di default con lo specifico: *block drop* o *block reset*.

# pf.conf – Filtering Rules

La *direction* è un altro elemento essenziale: ci specifica se dobbiamo considerare il pacchetto in ingresso o in uscita dal firewall

**.in** pacchetto *VERSO* il firewall

**.out** pacchetto che *ESCE* dal firewall

# pf.conf – Filtering Rules

*quick* è una keyword molto utile (ma molti puristi la odiano).

pf processa dalla prima all'ultima riga delle regole, in quanto la sua policy è LAST-MATCH wins.

L'utilizzo della keyword *quick*, istruisce pf di non procedere oltre nella ricerca delle regole.

Questo rende la ricerca più rapida ma si deve essere sicuri di quel che si sta facendo.

# pf.conf – Filtering Rules

*proto* ci permette di definire quale dei protocolli IP andremo ad utilizzare (tcp, udp, icmp, o quant'altro presente nel file /etc/protocols)

*src\_addr* e *dst\_addr* sono concetti-idee simili. Si tratta dell'indirizzo da cui parte o a cui è destinato un pacchetto.

# pf.conf – Filtering Rules

*src\_port* e *dst\_port* sono come gli indirizzi. Il riferimento in tal caso è la porta (che spesso identifica il servizio)

*tcp\_flags* sono i flag inerenti la connessione

(F)IN, (S)YN, (R)ST, (P)USH, (A)CK, (U)RG, (E)CE, e C(W)R.



# pf.conf – Filtering Rules

*state* si riferisce allo stato della connessione. Di default è imposta su *keep state (S/SA)* (ovvero viene mantenuto lo stato per la connessione e la regola viene mantenuta fino a che una delle due controparti non cessa la comunicazione o questa non decade per timeout).

# pf.conf – Filtering Rules

Gli altri possibili valori sono:

*no state* previene che la regola crei una stateful session.

*modulate* (funziona solo con TCP) essa fa sì che venga generato un numero di sequenza iniziale poco prevedibile per tutti i pacchetti che hanno a che fare con questa regola (il sistema, quindi cercherà di proteggere la sessione se questa ha una sequenza prevedibile).

# pf.conf – Filtering Rules

*synproxy* è una unione tra *keep* e *modulate*. Ha la funzione di cercare di prevenire attacchi SYN Flood su TCP.

Una ultima nota meritano le keyword *user* e *group*.

Queste si riferiscono a uid e gid di unix, e permettono, per esempio ad un proxy, di avere un controllo diverso dalle regole base.

# pf.conf – Filtering Rules

Un'altra feature interessante è la gestione del *connection tracking*.

La particolarità di questa feature risulta molto utile per creare regole dinamiche su host che tendono ad abusare di risorse.

Tipici esempi sono quelli di un pc infettato da un virus che manda molteplici mail, o di chi cerca di accedere, troppo insistentemente, ad un servizio.

# pf.conf – Filtering Rules

La struttura per gestire il connection tracking si basa su:

- 1 tabella

- 1 regola inerente la tabella

- 1 regola che valuti il numero di connessioni.

# pf.conf – Filtering Rules

Vediamo un esempio concreto:

```
table <ssh_attacker> persist
```

```
block in quick on egress inet proto tcp from <ssh_attacker> to (egress) port ssh  
pass in on egress inet proto tcp to (egress) port ssh flags S/SA keep state \  
(max-src-conn 10, max-src-conn-rate 3/15 , overload <ssh_attacker> flush)
```

Ovvero quando un host supererà le 10 connessioni o quando supererà un rate di connessioni superiore a 3 in 15 secondi, il suo IP verrà aggiunto alla tabella e la connessione verrà scartata.

Nelle FAQ di pf troverete tutte le possibili opzioni di tracking.

# pf.conf – anchors

Le *anchors* (o ancore) sono una funzione che permette di utilizzare dei placeholder per regole che possono essere create in un secondo momento.

Sono divise per le sezioni di nat/rdr e per le regole nat-anchor, bi-nat anchor, rdr-anchor e anchor

Le anchor possono essere chiamate da applicativi esterni o caricate in pf.

# pf.conf – anchors

Per caricarle in pf si ricorre alla seguente sintassi:  
*load anchor <name> from <file>.*

Per manipolare le anchor possiamo usare la seguente sintassi

*echo "<regola di pf corretta>" | pfctl -a <name> -f -*

Questo fa sì che la anchor permetta una modifica dinamica della regola.



# pf.conf – anchors

Note sulla keyword *quick* e le anchors:

Se *quick* è presente all'interno delle regole dell'ancora questo farà terminare la ricerca.

Se *quick* è presente sulla DEFINIZIONE (o placeholder) dell'ancora, al termine della valutazione positiva della regola la ricerca di pf termina.

# pf.conf – anchors

Alcuni tipici esempi per le anchors sono:

ftp-proxy, spamd, e authpf.

Questi tre applicativi sono nati per risolvere alcuni problemi e per aumentare le features.

# pf.conf – ftp-proxy

Tutti noi abbiamo almeno affrontato le insidie del protocollo ftp dietro un firewall/nat.

Il tutto si presenta nel caso di una negoziazione non concorde tra client e server.

Non sempre la porta che il client/server vorrebbe usare è disponibile, facendo sì che la sessione non prosegua.

Il team ha optato per risolvere il problema in maniera drastica: è stato scritto un proxy server ftp.

# pf.conf – ftp-proxy

Questo proxy ftp (che è di norma in ascolto su 127.0.0.1:8021) gestisce la traduzione tra il client ed il server, risolvendo i problemi della modalità in uso.

Le regole suggerite si basano su un redirect delle sessioni ftp in uscita, e su una anchor che permette al proxy di far entrare e uscire il traffico.

# pf.conf – spamd

Questo servizio nasce per limitare gli attacchi automatizzati ai server (utenti) di posta.

Si basa sul sistema di grey-listing: se si tratta di un vero server di posta, non si arrenderà al primo tentativo (fallito) ma riproverà. Raggiunta una certa soglia, la sessione smtp in ingresso verrà inviata al vero server di posta che gestirà i mail.

# pf.conf – authpf

Ultima anchor, presente nel sistema base, degna di nota è authpf.

Questo applicativo in realtà è una shell per l'utente che si collegherà (si spera) solo in ssh.

authpf richiede che sia presente, anche se vuoto, un file di regole generali (/etc/authpf/authpf.conf)

Nella cartella /etc/authpf/users potremmo trovare un successivo livello che dovrà coincidere con il nome utente.

# pf.conf – authpf

/etc/authpf/users/\$user/authpf.rules sarà il set di regole specifiche per l'utente. Quest'ultime saranno integrate con il set principale (/etc/pf.conf).

Questo file ha la stessa sintassi di pf.conf. Se utilizzerete delle macro o tabelle con lo stesso nome del set principale avrete una *name collision* che potrebbe portare a risultati non sperati.

# pf: una caso particolare

Andiamo ora a vedere una delle feature che è disponibile, di norma, solo su prodotti di fascia alta:

Transparent Firewall.

Questo è fattibile in OpenBSD grazie all'unione di *2 prodotti*:

*bridge*

*pf*



# pf.conf – transparent firewall

È obbligatoria una premessa:

il firewall trasparente ha numerosi vantaggi.

Si noti che questo è trasparente per il client e non è detto sia trasparente per un attacker.

# pf.conf – transparent firewall

Il primo passo da fare è creare una interfaccia di tipo bridge.

Possiamo ricorrere alla sintassi già vista per *hostname.if*. In tale caso il file sarà:

*/etc/bridgename.<nome>*

Questo file dovrà contenere almeno i comandi *add <nome interfaccia fisica> ed up*.

In esso si possono poi trovare molteplici opzioni.

# pf.conf – transparent firewall

Una interfaccia bridge può o meno avere indirizzo ip. Questo tipo di interfaccia può anche filtrare.

(vi rimando alla man page di *bridgename.if*)

Va anche detto che i comandi o presenti in tale file devono essere scritti uno per riga.

# pf.conf – transparent firewall

L'idea che sta dietro ad un trasparente firewall è semplice:

Il bridge permette il transito dei dati da una interfaccia all'altra (eventualmente filtrando protocolli o mac address).

pf, con le sue regole basate sulla singola interfaccia fisica permette di definire le regole.

# pf.conf – transparent firewall

Va notato che, se le regole che utilizzeremo hanno reazioni che non hanno nulla a che fare con il sistema che andremmo a proteggere, faciliteremo molto il nostro ‘nemico’ nell’individuare questo tipo di protezione.

Il caso più palese in cui un firewall trasparente si dimostra utile è quando abbiamo applicativi che non interagiscono molto bene in DMZ.

# pf.conf – keyword & tricks

Esistono delle keyword che vi possono semplificare la vita:

*interface:network* equivale alla connotazione aa.bb.cc.0/dd

*interface:broadcast* restituisce il broadcast di rete

*egress* è l'interfaccia associata alla default route

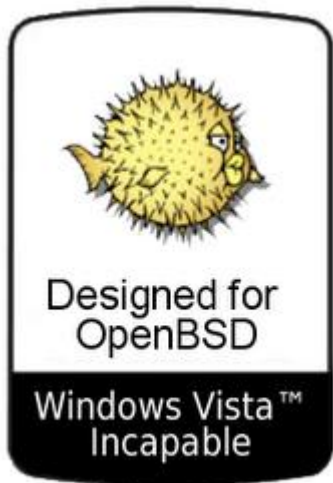
*(self)* restituisce tutti gli indirizzi assegnati alla macchina

*(interface)* restituisce l'indirizzo dell'interfaccia

*interface:peer* restituisce l'indirizzo del peer su un link ppp

*interface:0* restituisce l'indirizzo senza alias della scheda (1 il primo alias e così via)

# Credits



Tutto il materiale presentato è stato tratto da sito di [OpenBSD.org](http://OpenBSD.org) e dalle pagine di manuale di OpenBSD.