

# **Brannmur med OpenBSDs pakkefilter PF**

**Peter N. M. Hansteen**

**`peter@bsdly.net`**

Copyright © 2005 - 2007 av Peter N. M. Hansteen

This document is © Copyright 2005 - 2007, Peter N. M. Hansteen. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS DOCUMENTATION IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Dokumentet er 'work in progress', basert på manuskriptet til et foredrag jeg holdt for BLUG (se <http://www.blug.linux.no/>) 27. januar 2005.

Jeg interessert i kommentarer av alle slag, og om du ønsker det, må du gjerne oppgi web-referanse eller andre referanser til html- og pdf-utgaven av manuskriptet. Hvis du gjør det, vil jeg gjerne, uten å kunne kreve det, at du sender meg en epost-melding om at at du har laget en slik referanse. For kommunikasjon om dette dokumentet, vennligst bruk adressen <[peter@bsdly.net](mailto:peter@bsdly.net)> *For fullstendig revisjonshistorie, se HTML-versjonen eller SGML-kildeteksten.*

## Revisjonshistorie

Revisjon 0.094n 5 februar 2006

nytt om icmp, rdr med 'reflect'

Revisjon 0.0955n 21. august 2006

I sync med 0.0955e

Revisjon 0.0956n 27 august 2006

Eysteins skarpe øyne fant ting som måtte rettes

Revisjon 0.0958e 25 september 2006

feil url rettet - takk til Robby Cauwerts

# Innholdsfortegnelse

<b>Innledende kommentarer .....</b>	<b>1</b>
<b>PF? .....</b>	<b>3</b>
<b>Pakkefilter? Brannmur? .....</b>	<b>5</b>
<b>NAT? .....</b>	<b>6</b>
<b>PF i dag.....</b>	<b>8</b>
<b>BSD kontra Linux - konfigurering.....</b>	<b>9</b>
<b>Enkleste oppsett (OpenBSD).....</b>	<b>10</b>
<b>Enkleste oppsett (FreeBSD) .....</b>	<b>12</b>
<b>Enkleste oppsett (NetBSD) .....</b>	<b>13</b>
<b>Første regelsett - enslig maskin.....</b>	<b>14</b>
<b>Litt strengere .....</b>	<b>15</b>
<b>Statistikk med pfctl .....</b>	<b>17</b>
<b>Enkel gateway, NAT hvis du trenger det.....</b>	<b>19</b>
Gatewayer og fallgrubene in, out og on.....	19
Hva er egentlig lokalnettet ditt? .....	20
Sette opp gatewayen.....	21
<b>Problembarnet FTP.....</b>	<b>24</b>
FTP gjennom NAT: ftp-proxy.....	24
FTP, PF og rutbare adresser: ftpsesame, pftpx og ftp-proxy! .....	26
ftp-proxy, ny utgave .....	26
<b>Gjøre nettverket ditt feilsøkingssvennlig.....</b>	<b>28</b>
Skal vi så slippe alt gjennom? .....	28
Den enkle måten: Dette stopper her .....	29
Slippe ping gjennom.....	29
Hjelpe traceroute.....	30
Path MTU discovery.....	31
<b>Nettverkshygiene: block-policy, scrub og så videre.....</b>	<b>32</b>
<b>block-policy</b> .....	<b>32</b>
<b>scrub</b> .....	<b>32</b>
<b>antispoof</b> .....	<b>32</b>
Håndtere ikke-rutbare adresser fra andre steder .....	33
<b>En webserver og epostserver på innsiden .....</b>	<b>35</b>
Ta vare på sine egne - innsiden .....	36

<b>Gjøre livet lettere med tabeller .....</b>	<b>38</b>
<b>Logging .....</b>	<b>40</b>
Ta en titt med tcpdump.....	40
Men det finnes grenser (en liten historie fra virkeligheten).....	41
<b>Få oversikt med pftop .....</b>	<b>42</b>
<b>Usynlig gateway med bridge .....</b>	<b>43</b>
<b>Trafikkregulering med ALTQ .....</b>	<b>45</b>
ALTQ - prioritere etter trafikktype .....	46
Så hvorfor virker dette? .....	46
ALTQ - prosentvis fordeling.....	47
ALTQ - håndtere uønsket trafikk .....	48
<b>CARP og pfsync .....</b>	<b>50</b>
<b>Trådløse nettverk, ganske enkelt.....</b>	<b>51</b>
Litt bakgrunnsinformasjon om IEEE 802.11.....	51
WEP (Wired Equivalent Privacy).....	51
WPA (WiFi Protected Access) .....	52
Sette opp et enkelt trådløst nettverk .....	52
<b>Et åpent, men strengt bevoktet trådløst nettverk med authpf.....</b>	<b>55</b>
<b>Stoppe maset fra robotene.....</b>	<b>59</b>
expiretable rydder tabellene .....	61
<b>Mobbe spammere .....</b>	<b>63</b>
Du er ikke alene: Svartelister.....	63
Svartelister, gråelister og klebrige tjærehull .....	63
Sette opp spamd .....	64
Hovedtrekk i erfaringene med spamd.....	65
Konklusjoner fra erfaringene med spamd .....	67
<b>PF - Haiku .....</b>	<b>69</b>
<b>Referanser .....</b>	<b>70</b>
<b>Finne manuskriptet på web .....</b>	<b>72</b>
Hvis du hadde glede av dette: Kjøp OpenBSD-CDer og annet, doner!. 72	

# Innledende kommentarer

Dette foredraget<sup>1</sup> skal handle om brannmurer og beslektede funksjoner, med utgangspunkt i *litt* teori og en del eksempler på filtrering og dirigering av nettverkstrafikk. Som i mange andre sammenhenger finnes det i mange av tilfellene jeg kommer inn på, *mer enn en måte* å gjøre det vi beskriver. Jeg vil i alle fall kunne la meg avbryte underveis, i alle fall hvis jeg får lov til å bruke det jeg lærer av kommentarene senere, enten i reviderte utgaver av foredraget eller i praktisk bruk en eller annen gang. Men først



## Dette er ikke en HOWTO

Dette dokumentet er ikke ment som en ferdiglaget oppskrift til å klippe og lime fra.

Bare for å banke det enda litt inn, gjenta etter meg

```
Høytidelig forsikring for nettverksadministratorer
```

```
Dette er mitt nettverk.
```

```
Det er mitt  
eller teknisk sett min arbeidsgiver sitt,  
det er mitt ansvar  
og jeg våker over det som mitt eget barn.
```

```
det er mange andre nettverk ser nesten ut som mitt,
```

```
men det finnes ingen som er helt lik det.
```

```
Jeg forsikrer på ære og samvittighet
```

1. Manuskriptet er en videreutvikling av et foredragsmanuskript til et møte som ble annonsert med omtalen "Foredraget handler om brannmurer og beslektete funksjoner, med eksempler fra det virkelige liv med PF (Packet Filter) fra OpenBSD. PF tilbyr brannmur, NAT, trafikkstyring og båndbreddefordeling i samme fleksible og administratorvennlige system. Peter satser på at foredraget skal gi deg noen ideer om hvordan du kan styre nettverkstrafikken din slik du selv vil - holde noe utenfor og annet innenfor nettet ditt, dirigere trafikk til bestemte maskiner eller tjenester, og mobbe spammere, selvsagt."

Personer som har kommet med vesentlige og nyttige kommentarer om dette manuskriptet er blant annet Eystein Roll Aarseth, David Snyder, Peter Postma, Henrik Kramshøj, Vegard Engen, Greg Lehey, Ian Darwin, Daniel Hartmeier, Mark Uemura, Hallvor Engen og sannsynligvis en del som vil forbli tapt i epostarkivet mitt til jeg kan grep'e dem ut derfra igjen.

Jeg vil takke følgende organisasjoner for støtten jeg har mottatt: The NUUG Foundation for reisestøtte til deltakelse på AUUG2005, AUUG, UKUUG, SANE, BSDCan, EuroBSDCon og AsiaBSDCon for at de inviterte meg til konferansene, og endelig FreeBSD Foundation for reisestøtte slik at jeg kom meg til til BSDCan 2006 og EuroBSDCon 2006.

at jeg aldri vil klippe og lime blindt fra HOWTOer

Poenget her er at selv om reglene og konfigurasjonene jeg viser senere virker, jeg har testet dem og de har ett eller annet slektskap med slikt som faktisk er i produksjon, så er de antakelig svært forenklede og er nesten helt sikkert i alle fall litt feil og kanskje helt ubrukelige for akkurat *ditt* nettverk.

Jeg ber deg om å huske på at dette dokumentet er ment for å vise deg noen nyttige ting kan du gjøre og inspirere deg til å oppnå gode resultater.

Jeg ber deg om å legge vekt på å forstå nettverket ditt og hva du må gjøre for å forbedre det.

Vær så snill å ikke klippe og lime blindt fra dette dokumentet eller noe annet.

Nå som dette er unnagjort, kan vi gå videre til det vi skal snakke om i dag.

# PF?

Hva er så PF? La oss starte med en kort gjennomgang av prosjektets historie for å sette begrepene i sin rette sammenheng.

OpenBSDs pakkefilter-delsystem, som de fleste omtaler med forkortelsen 'PF' (Packet Filter), ble opprinnelig skrevet på uhyre kort tid i en intens og samordnet innsats i løpet av sommeren og høsten 2001 av Daniel Hartmeier og en rekke OpenBSD-utviklere, og ble lansert som standarddel av basesystemet i OpenBSD 3.0 i desember 2001.

Behovet for ny brannmur-programvare i OpenBSD oppstod da Darren Reed gjorde oppmerksom på at IPFilter, som da var en tett integrert del av OpenBSD, likevel ikke var utgitt under BSD-lisens. Tvert imot. Lisensen var nesten ordrett lik BSD-lisensen, men en selvsagt del av BSD-lisensen, nemlig retten til å gjøre endringer i koden var ikke med. OpenBSDs utgave av IPFilter inneholdt en god del endringer og tilpasninger, og det var jo rett og slett ikke lov i følge lisensen. IPFilter ble fjernet fra OpenBSD 29. mai 2001, og i noen få uker var OpenBSD-current uten brannmurprogramvare.

Heldigvis var en sveitser ved navn Daniel Hartmeier på det tidspunktet allerede i gang med noen små eksperimenter på kjerne hacking i nettverkskoden.

Han hadde jobbet ut fra å hekte en liten funksjon han selv hadde skrevet inn i nettverksstakken, fått pakker til å passere gjennom, og hadde så smått begynt å tenke filtrering. Så kom lisenskrisen.

29. mai ble IPFilter luket ut, og første commit av PF skjedde søndag 24. juni 2001 klokken 19:48:58 UTC.<sup>1</sup>

Så fulgte noen måneder med ganske høy aktivitet, og den utgaven av PF som ble sluppet med OpenBSD 3.0 hadde nokså komplett pakkefilter-funksjonalitet, inkludert nettverksadresseoversetting.

---

1. Det er verd å merke seg at opphavsrett-episoden med IPFilter førte til at OpenBSD-utviklerne satte i gang med en *lisensrevisjon* (license audit) av hele treet med egen kildekode og ports (tredjepartsprogramvare) for å oppdage mulige problemer og unngå tilsvarende ubehagelige situasjoner i fremtiden. Prosessen førte til at man fant og avklarte en rekke potensielle problemer i løpet av månedene som fulgte, og dermed fjernet en betydelig mengde potensielle lisensproblemer for alle som deltar i utvikling av fri programvare. Theo de Raadt oppsummerte status i en melding til listen `openbsd-misc` den 20. februar 2003, i en melding som blant annet er tilgjengelig fra listearchivet MARC (<http://marc.info/?l=openbsd-misc&m=104570938124454&w=2>).

Mye tyder på at Daniel Hartmeier og de andre som utviklet PF hadde lært av erfaringene fra IPFilter-koden, i alle fall presenterte Daniel en artikkel på USENIX 2002 med data fra ytelsestester som viser at PF i OpenBSD 3.1 hadde ytelse under stressforhold på høyde med eller bedre enn både IPFilter på samme plattform og iptables på Linux.

Noen tester ble også kjørt på den opprinnelige PF fra OpenBSD 3.0, og de viste stort sett bare at koden hadde blitt mer effektiv fra 3.0 til 3.1. Artikkelen som beskriver akkurat hva de gjorde, ligger på Daniel Hartmeier sin web, se <http://www.benzedrine.cx/pf-paper.html>.

Jeg har ikke funnet tilsvarende tester som er gjort siden, men mine erfaringer og andre jeg kjenner til tyder på at filtreringsoverheaden i PF er helt ubetydelig. For å ha en viss referanseramme, så er den maskinen som står mellom Datadok sitt nett og verden forøvrig en Pentium III 450MHz med 384MB minne, og den maskinen er som regel når jeg husker å se etter, minst 96 prosent 'idle' i følge top.

# Pakkefilter? Brannmur?

Nå har jeg allerede brukt noen begreper før jeg har forklart dem, og det må jeg rette på nå. PF er et *pakkefilter*, altså kode som vurderer pakker på protokoll- og portnivå og bestemmer hva som skal skje med dem. I PFs tilfelle opererer pakkefilteret i det vesentlige på kjernenivå, inne i nettverkskoden.

PF lever i en verden av pakker, protokoller, forbindelser og porter.

På grunnlag av hvor en pakke kommer fra eller skal til, hvilken protokoll eller forbindelse eller port den er tiltenkt, kan PF avgjøre hvilken vei pakken skal ledes, eller om den skal slippe gjennom i det hele tatt.

Det går an å styre nettverkstrafikk på grunnlag av *innholdet* i pakkene også, det man ofte kaller på applikasjonsnivå, men det er ikke det PF gjør. Vi skal komme tilbake til noen tilfeller der PF kan overlate den jobben til andre programmer, men først skal vi ta noen grunnleggende ting.

Og siden vi allerede snakker om brannmur: En av de viktige funksjonene til programvare som PF, noen vil kanskje si den viktigste funksjonen, er å identifisere og stenge for trafikk som man ikke ønsker å slippe inn i lokalnettet sitt eller ut i verden. En vittig sjel en gang fant på å kalle dette for 'a *firewall*', som vi på norsk vanligvis kaller '*brannmur*'.

# NAT?

En annen ting som vi kommer til å snakke mye om, er 'indre' og 'ytre' adresser, eller 'rutbare' og 'ikke-rutbare' adresser. Det har egentlig, i utgangspunktet, ikke så mye med brannmurer og pakkefiltrering å gjøre, men verden er nå engang blitt slik at vi må innom det. Alt dette handler om at man en gang tidlig på 1990-tallet begynte å regne på hvor mange maskiner som ville kunne komme til å bli koblet til Internet hvis kommersialiseringen fortsatte og hordene av verdens forbrukere skulle koble seg til samtidig.

Den gangen Internet-protokollene ble laget, var det jo vanlig at en maskin var ganske dyr, den hadde gjerne mange brukere som gjerne kunne være innlogget samtidig fra hver sin mer eller mindre dumme terminal, og uansett så var det jo bare universitetene pluss en del firmaer som hadde god nok kontakt med enten universitetene eller forsvaret i USA som skulle koble seg til. Da var det jo uendelig nok med adresser på 32 bit, fordelt på fire oktetter. Millioner av maskiner skulle det holde til.

Så kom kommersialiseringen av Internet, og da ble det faktisk ganske fort noen millioner små og billige maskiner som skulle koble seg til. Og utviklingen så ut til å fortsette og kanskje til og med aksellerere. Da begynte de kloke hodene å gjøre noe. De gjorde flere ting. For det første begynte de arbeidet med å lage løsningen med et større adresseområde, det som har blitt hetende IP versjon 6 - forkortet IPv6 - og som bruker 128 bits adresser. Det er utnevnt til løsningen på lang sikt, og bare for å ha sagt det, så er IPv6-støtte innebygget i OpenBSD, og PF har alltid så vidt jeg vet støttet IPv6.

Så måtte man jo ha noe i mellomtiden, for å skifte hele verden over til et nytt regime med fire ganger så lange adresser, det regnet man med ville ta tid. Den prosessen er vel fortsatt ikke helt ute av startfasen. Det de fant på, var to ting - lage mekanismer for å 'lyve litt' for omverdenen ved å la gatewayer endre på adressene i nettverkspakkene, og å ta noen av de adresseområdene som ingen hadde altfor sterk hevd på og sette av som reserverte for intern bruk i nettverk som ikke skal kommunisere direkte med Internet. Da var det plutselig mulig at flere maskiner på forskjellige steder kunne ha samme IP-adresse lokalt, men det ville ikke gjøre noe, fordi adressene ble oversatt til noe annet før trafikken ble sluppet ut på nettet.

Hvis trafikk med slike "ikke-rutbare" adresser slapp ut på nettet, så har ruterne som ser trafikken gyldig grunn til å ikke sende den videre.

Dette er det vi kaller "Network Address Translation" på engelsk, i noen sammenhenger har det blitt kalt "IP masquerade" og liknende, på norsk heter det vel nettverksadresseoversetting. De to RFCene som definerer hvordan dette skal skje, stammer altså fra henholdsvis 1994 og 1996 <sup>1</sup>.

Det kan være flere grunner til at man bruker de såkalte "RFC 1918-adressene", men tradisjonelt og historisk har hovedgrunnen vært at offisielle adresser er enten ikke tilgjengelig eller ikke hensiktsmessig.

---

1. De to dokumentene er RFC 1631, "The IP Network Address Translator (NAT)", fra mai 1994 og RFC 1918, "Address Allocation for Private Internets", fra februar 1996. Se kapittelet kalt *Referanser*

# PF i dag

Da har vi i alle fall litt av bakgrunnen. Det har gått noen år siden 2001, og PF i dag, i OpenBSD 4.1, er et pakkefilter som kan gjøre mange ting, hvis man har lyst til det.

For det første så klassifiserer PF pakker, på grunnlag av protokoll, port, pakketype, fra- og til-adresse. Faktisk kan PF også med brukbart stor sikkerhet klassifisere pakker på grunnlag av operativsystem hos avsender.

Og selv om NAT ikke er noen nødvendig del av et pakkefilter, så er det i praksis ganske greit at logikken for å håndtere omskriving av adresser befinner seg i nærheten, så PF inneholder den logikken også.

PF kan - på grunnlag av ulike sammensetninger av protokoll, port og annet - dirigere trafikken til andre steder enn avsenderen har angitt, for eksempel til en annen maskin eller til behandling av et program - en daemon som ligger og lytter på en port, lokalt eller på annen maskin.

I OpenBSD hadde man fra før av kode for lastbalansering og trafikkforming i *ALTQ*, og etterhvert ble *altq*-koden integrert i PF. Rett og slett fordi det var praktisk.

Resultatet er i dag at du har alt dette tilgjengelig via en enkelt, i alt vesentlig menneskeleselig konfigurasjonsfil, som normalt heter `pf.conf` og ligger i `/etc/`-katalogen.

Dette er nå tilgjengelig som en del av basesystemet på OpenBSD, FreeBSD der PF fra versjon 5.3 er ett av tre alternative brannmursystemer som kan lastes etter smak og behag, og NetBSD og DragonFlyBSD. De to siste har jeg ikke fått lekt noe med selv. Det er gjerne noe med å ha disponibel tid og disponibel maskin samtidig. Men alt jeg har sett tyder på at det er helt ubetydelige detaljer som er forskjellige når du skal bruke PF på andre systemer.

# BSD kontra Linux - konfigurering

Jeg regner med at det er noen i salen som er mer vant med Linux eller andre systemer enn de er med BSD, så jeg skal ta noen ord om konfigurering på BSD.

Nettverksgrensesnittene på BSD heter ikke `eth0` og så videre. Grensesnittene får heller navn lik drivernavn pluss sekvensnummer, slik at 3Com-kort som bruker driveren `x1` får navn som `x10`, `x11` og så videre, og tilsvarende for Intel-kort som gjerne blir hetende `em0`, `em1`, SMC-kort med `sn0`, og så videre.

For konfigurering ved oppstart er BSDene stort sett organisert slik at konfigurasjonen leses fra `/etc/rc.conf`, som blir lest av scriptet `/etc/rc` når maskinen starter. På OpenBSD er anbefalingen å bruke `/etc/rc.conf.local` til lokale tilpasninger siden `rc.conf` inneholder standardverdiene, mens på FreeBSD er det `/etc/defaults/rc.conf` som inneholder standardverdiene, og `/etc/rc.conf` er riktig sted å gjøre endringene.

PF selv blir i alle fall konfigurert via redigering av `/etc/pf.conf` og via kommandolinjeverktøyet `pfctl`. `pfctl` har *mange* muligheter og alternativer. Vi skal se på noen av dem i dag.

For de som lurere, så finnes det web-grensesnitt for administrering, men de er ikke med i basesystemet. PF-utviklerne er ikke fiendtlig innstilt til slikt, men har vel heller ikke sett noe grafisk grensesnitt mot PF som er klart overlegent bedre enn `pf.conf` i en teksteditor supplert med litt `pfctl`-besvergelser og vanlige UNIX-triks.

# Enkleste oppsett (OpenBSD)

Da har vi endelig kommet til noe praktisk og greit, det helt enkleste oppsettet med PF. På en enslig maskin som skal snakke med et nettverk som godt kan være Internet.

For å starte PF, som vi var inne på i sted, så må du si fra til `+rc`-systemet at du vil starte tjenesten. Det gjør du i på OpenBSD `/etc/rc.conf.local`, med den magiske linjen

```
pf=YES                # aktiver PF
```

nokså enkelt og greit. I tillegg kan du, hvis du vil, angi hvilken fil PF skal finne reglene sine i.

```
pf_rules=/etc/pf.conf # angi hvilken fil som inneholder regler
```

Ved neste omstart vil da PF bli aktivert. Det ser du av meldingen `PF enabled` på konsollet. I den `/etc/pf.conf` som kommer ut av en normal installering av OpenBSD, FreeBSD eller NetBSD inneholder en del gode forslag, men alt sammen er kommentert ut.

Men du trenger ikke å starte maskinen på nytt for å aktivere PF. Det bruker du like gjerne `pfctl` til. Siden vi ikke har lyst til å reboote i tide og utide, sier vi på kommandolinjen

```
peter@skapet:~$ sudo pfctl -e ; sudo pfctl -f /etc/pf.conf
```

og da er PF aktivert<sup>12</sup>. Foreløpig har vi ikke noe regelsett, så PF gjør ingenting.

Det er antakelig verd å merke seg at hvis du starter maskinen på nytt nå, så vil `rc-scriptet`, i alle fall på OpenBSD, aktivere et standard-regelsett, som faktisk blir lastet *før* nettverksgrensesnittene blir aktivert.

Dette regelsettet er utformet som et sikkerhetsnett i tilfelle gatewayen starter med en ugyldig konfigurasjon. Dette gir deg mulighet til å logge inn og rette på det som måtte være av syntaksfeil som gjorde at det egentlige

---

1. Som en liten fotnote, jeg bruker `sudo` når jeg trenger å gjøre noe som krever privilegier. `Sudo` er med i basesystemet i OpenBSD, og innenfor rekkevidde som port eller pakke de fleste andre steder. Hvis du ikke bruker `sudo` nå, så bør du begynne. Da slipper du å skyte deg i foten fordi du glemte at du var rot i det terminalvinduet.

2. For de som ønsker det - `pfctl` kan håndtere flere operasjoner i samme kommandolinje, så du kan for eksempel både aktivere PF og laste regelsettet med kommandoen `sudo pfctl -e -f /etc/pf.conf`

### *Enkleste oppsett (OpenBSD)*

regelsettet ditt ikke lot seg laste. Standardregelsettet tillater trafikk for noen få basistjenester: ssh fra hvor som helst, navneoppslag fra maskinen og montering av NFS-filsystemer.

Noen tidlige versjoner av PF på andre systemer ser ut til å ha oversett, og kan dermed mangle, standardregelsettet.

# Enkleste oppsett (FreeBSD)

På FreeBSD kan det ved første øyekast se ut som om trenger du litt mer magi i `/etc/rc.conf`, spesifikt i følge [FreeBSD Handbook](#)<sup>1</sup>

```
pf_enable="YES"                # Enable PF (load module if required)
pf_rules="/etc/pf.conf"        # rules definition file for PF
pf_flags=""                    # additional flags for pfctl startup
pflog_enable="YES"            # start pflogd(8)
pflog_logfile="/var/log/pflog" # where pflogd should store the logfile
pflog_flags=""                # additional flags for pflogd startup
```

Heldigvis er neste alle disse allerede tatt med som standardverdier i `/etc/defaults/rc.conf`, og bare

```
pf_enable="YES"                # Enable PF (load module if required)
pflog_enable="YES"            # start pflogd(8)
```

er faktisk nødvendig å legge inn i `/etc/rc.conf` for å aktivere PF.

På FreeBSD blir PF i utgangspunktet compilert som lastbar kjernemodul, så du burde kunne komme godt i gang med `$ sudo kldload pf`, fulgt av `$ sudo pfctl -e` for å aktivere. Gitt at du har lagt inn de nødvendige linjene i `/etc/rc.conf`, vil også rc-skriptet for PF fungere for å starte PF:

```
$ sudo /etc/rc.d/pf start
```

for å aktivere PF, eller

```
$ sudo /etc/rc.d/pf stop
```

for å deaktivere pakkefileret. rcNG-scriptet for `pf` tilbyr også noen andre muligheter. Samtidig er det fortsatt verd å merke seg at vi på dette punktet ikke har noe regelsett, noe som betyr at PF faktisk ikke gjøre noe.

---

1. Det er visse forskjeller mellom FreeBSD 4.n og 5.n og senere når det gjelder PF. Se etter i [FreeBSD Handbook](#), spesifikt kapittelet om PF ([http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/firewalls-pf.html](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-pf.html)) hva som gjelder for deg, før du går videre.

# Enkleste oppsett (NetBSD)

NetBSD fra og med 2.0 har PF som lastbar kjernemodul, som er tilgjengelig via pakkesystemet som `pkgsrc/security/pf1km` eller kompilert statisk inn i kjernen. Fra NetBSD 3.0 vil PF være med i basesystemet.

Hvis du ønsker å aktivere PF i kjernekonfigurasjonen (og altså ikke bruke lastbar kjernemodul), tar du med disse linjene i konfigurasjonsfilen:

```
pseudo-device pf # PF packet filter
pseudo-device pflog # PF log interface
```

I `/etc/rc.conf` trenger du linjene

```
lkm="YES" # do load kernel modules
pf=YES
pflogd=YES
```

for å aktivere henholdsvis lastbare kjernemoduler, PF og PFs loggingsgrensesnitt.

Hvis du har installert modulen, laster du den med `NetBSD$ sudo modload /usr/lkm/pf.o`, fulgt av `NetBSD$ sudo pfctl -e` for å aktivere. Alternativt kan du bruke rc-scriptene, `NetBSD$ sudo /etc/rc.d/pf start` for selve PF og `NetBSD$ sudo /etc/rc.d/pflogd start` for logging.

For å laste modulen automatisk ved omstart legger du denne linjen inn i `/etc/lkm.conf`:

```
/usr/lkm/pf.o - - - - AFTERMOUNT
```

Gitt at du har gjort dette riktig, er du klar til å lage ditt første PF-regelsett.

# Første regelsett - enslig maskin

Dette er det enkleste tenkbare oppsettet, for en enslig maskin som ikke skal kjøre noen tjenester, og som snakker med ett nettverk, som godt kan være Internet. Da holder det foreløpig med en `/etc/pf.conf` med dette innholdet:

```
block in all
pass out all keep state
```

altså nekte all innkommende trafikk, tillate trafikk fra oss selv, og ta vare på tilstandsinformasjon om forbindelsene, slik at returtrafikk for forbindelsene du har startet selv, får slippe gjennom. Det er verd å merke seg at fra og med OpenBSD 4.1 er standardverdien for pass-regler at tilstandsinformasjon blir bevart<sup>1</sup>, så det tilsvarende regelsettet i OpenBSD 4.1-stil er enda enklere,

```
# minimalt regelsett, OpenBSD 4.1 bevarer tilstandsinformasjon som standard
block in all
pass out all
```

Det er nokså selvsagt at å la all trafikk fra en bestemt maskin passere innebærer en ganske uvanlig grad av tillit til at den aktuelle maskinen faktisk er pålitelig. Dette gjør du hvis du vet du kan stole på maskinen. Hvis du er klar til å bruke regelsettet, laster du det med

```
$ sudo pfctl -e ; sudo pfctl -f /etc/pf.conf
```

---

1. Faktisk tilsvarer en nye standardverdien `keep state flags S/SA`, noe som sikrer at bare de innledende SYN-pakkene under etablering av forbindelsen skaper ny oppføring i tilstandstabellen. Dette eliminerer enkelte ugreie feilscenarier.

# Litt strengere

En litt mer komplett og oversiktlig situasjon får vi ved å nekte alt og bare åpne for det vi vet vi trenger<sup>1</sup>. Da får vi se to av de tingene som gjør det behagelig å jobbe med PF, nemlig lister og makroer.

Altså endrer vi litt på `/etc/pf.conf` og begynner med

```
block all
```

Så går vi litt opp i filen, for makroer må være definert før de kan brukes:

```
tcp_tjenester = "{ ssh, smtp, domain, www, pop3, auth, pop3s }"  
udp_tjenester = "{ domain }"
```

Da har vi vist flere ting på en gang - hvordan makroer ser ut, at makroer godt kan være lister, og at PF forstår regler med protokollnavn like godt som de med portnummer. Navnene er de som er angitt i `/etc/services`. Da har vi noe som vi kan sette inn i reglene våre, som vi redigerer litt så de ser slik ut:

```
block all  
pass out proto tcp to any port $tcp_tjenester  
pass proto udp to any port $udp_tjenester
```

Husk på å legge til **keep state** i disse reglene hvis systemet du bruker har en PF-versjon som er eldre enn den som kommer fra OpenBSD 4.1.

Her vil sikkert noen innvende at UDP er tilstandsløs, men PF klarer faktisk likevel å ta vare på tilstandsinformasjon. Når du spør en navneserver om et domenenavn, ønsker du nok også å få svaret tilbake.

Siden vi har gjort endringer i `pf.conf`, laster vi reglene på nytt:

```
peter@skapet:~$ sudo pfctl -f /etc/pf.conf
```

---

1. Nå spør du kanskje hvorfor jeg skriver regelsettet slik, med å nekte alt i utgangspunktet? Det korte svaret er at det gir deg bedre kontroll, i bytte mot minimalt med tenking. Hele poenget med pakkefiltrering er å ta kontrollen, ikke å jage etter de siste påfunnene som slemmingene kommer med. Marcus Ranum har skrevet en svært underholdende og informativ artikkel om dette, *The Six Dumbest Ideas in Computer Security* ([http://www.ranum.com/security/computer\\_security/editorials/dumb/index.html](http://www.ranum.com/security/computer_security/editorials/dumb/index.html)), som er vel verd å lese.

og de nye reglene gjelder. Hvis det ikke er syntaksfeil, gir ikke `pfctl` deg noen meldinger under lastingen. Flagget `-v` vil gi mer 'verbose' eller ordrik oppførsel.

Hvis du har gjort store endringer i regelsettet ditt, kan det være lurt å sjekke reglene uten å laste dem. Det gjør du i så fall med `pfctl -nf /etc/pf.conf`. Alternativet `-n` gjør at reglene bare blir tolket uten å bli lastet, og du kan rette feil hvis du trenger det. Uansett vil det siste gyldige regelsettet du lastet, fortsette å gjelde til du enten deaktiverer PF eller laster et nytt regelsett.

# Statistikk med pfctl

Kanskje det er greit å sjekke at PF faktisk ble startet, og samtidig få med noe statistikk. pfctl kan gi mange typer informasjon hvis du bruker **pfctl -s** og legger til hvilken type informasjon du vil vise.

Her er et eksempel fra hjemmegatewayen min mens jeg holdt på med forberedelsene til en tidligere versjon av dette foredraget:

```
peter@skapet:~$ sudo pfctl -s info
Status: Enabled for 17 days 00:24:58          Debug: Urgent

Interface Stats for xl0                    IPv4                    IPv6
Bytes In                                   9257508558              0
Bytes Out                                  551145119               352
Packets In
  Passed                                   7004355                 0
  Blocked                                   18975                   0
Packets Out
  Passed                                   5222502                 3
  Blocked                                   65                      2

State Table                                Total                   Rate
current entries                            15
searches                                   19620603                 13.3/s
inserts                                    173104                   0.1/s
removals                                   173089                   0.1/s

Counters
match                                       196723                   0.1/s
bad-offset                                  0                       0.0/s
fragment                                    22                      0.0/s
short                                        0                       0.0/s
normalize                                    0                       0.0/s
memory                                       0                       0.0/s
bad-timestamp                              0                       0.0/s
congestion                                  0                       0.0/s
ip-option                                   28                      0.0/s
proto-cksum                                 325                      0.0/s
state-mismatch                              983                     0.0/s
state-insert                                0                       0.0/s
state-limit                                 0                       0.0/s
src-limit                                   26                      0.0/s
synproxy                                    0                       0.0/s
```

Her ser vi i første linjen at PF er aktivert og har vært oppe i omtrent to uker, altså siden jeg oppgraderte maskinen til siste snapshot. `pfctl -s all` gir svært detaljert informasjon. Prøv det og se, og bruk gjerne litt tid på å studere de andre alternativene som `pfctl` tilbyr. `man 8 pfctl` gir full oversikt.

I alle fall har du nå en enslig maskin som skal kunne snakke relativt godt og sikkert med andre maskiner på Internet.

Et par ting mangler fortsatt. For eksempel ønsker du antakelig å slippe gjennom i alle fall noe ICMP- og UDP-trafikk, om ikke annet for din egen feilsøking sin del.

Og selv om moderne og sikrere alternativer for filoverføring er tilgjengelig, vil du sannsynligvis komme borti krav om å kunne bruke ftp.

Alle disse tingene kommer vi tilbake til litt senere.

# Enkel gateway, NAT hvis du trenger det

Så skal vi endelig bevege oss over i mer realistiske eller i alle fall mer utbredte oppsett, der maskinen med brannmurkonfigurasjon også er gateway for minst en annen maskin.

Selvfølgelig kan maskinene på innsiden også kjøre brannmurprogramvare, men om de gjør det eller ikke, påvirker i liten grad det vi skal snakke om her.

## Gatewayer og fallgrubene in, out og on

I oppsett med en enkelt maskin er livet relativt enkelt. Trafikken du skaper, skal enten slippe ut til resten av verden eller ikke, og du bestemmer hva du vil slippe inn fra omverdenen.

Når du setter opp en gateway, blir perspektivet annerledes. Du går fra "meg mot nettverket der ute" til "jeg er den som bestemmer hva som skal passere meg eller ikke, til eller fra alle nettverkene jeg er koblet til". Maskinen har flere, eller i det minste to, nettverks grensesnitt, og hvert av dem er koblet til et separat nett.

Nå er det helt naturlig å tenke at hvis du vil at trafikken skal flyte fra nettverket som er koblet til **x11** til maskiner i nettverket som er koblet til **x10**, trenger du en regel som

```
pass in on x11 from x11:network to x10:network \
    port $porter keep state
```

og dermed også holde på tilstandsinformasjon.<sup>1</sup>

Dessverre er det slik at en av de mest utbredte feilene i brannmurkonfigurerings, og det som fører til mest klager, er å ikke innse at nøkkelordet "to" ikke i seg selv garanterer passasje hele veien frem.

Regelen vi akkurat skrev, lar bare trafikken passere inn til gatewayen på det indre grensesnittet.

---

1. Selv om **keep state** er standard og egentlig er overflødig hvis du bruker OpenBSD 4.1 eller nyere, er det ikke alltid nødvendig å fjerne dette leddet fra reglene når du oppgraderer regelsettet fra tidligere versjoner. For å gjøre overgangen lettere vil eksemplene i dette dokumentet påpeke forskjellene når det er behov for det.

For å la pakkene komme litt videre, trenger du en tilsvarende regel som denne:

```
pass out on x10 from x11:network to x10:network \  
    port $porter keep state
```

Disse reglene vil virke, men virkningen er kanskje ikke det du hadde ønsket.

Hvis det finnes gode grunner til at du trenger å ha så spesifikke regler i regelsettet ditt, vet du at du trenger dem og hvorfor. For de enkle gateway-konfigurasjonene jeg behandler her, er det sannsynlig at det du vil ønske å bruke er en regel som denne

```
pass from x11:network to any port $porter keep state
```

som lar ditt lokalnett få adgang til Internet, og så overlate detektivarbeidet til koden som ligger bak *antispoof* og *scrub*. Disse funksjonene er svært gode og pålitelige, og vi kommer tilbake til dem om en liten stund. Foreløpig bare godtar vi det som en sannhet at for enkle oppsett gir regler som er bundet til grensesnitt med in og out først og fremst gjør regelsettet mindre oversiktlig enn nødvendig.

For en travel nettverksadministrator er et lesbart regelsett et sikrere regelsett.

I resten av dette foredraget vil vi med noen få unntak holde reglene så enkle og lesbare som mulig.

## Hva er egentlig lokalnettet ditt?

Nå har vi akkurat introdusert notasjonen **grensesnitt:network**. Det er en nyttig måte å forkorte på, men du kan gjøre regelsettet enda mer lesbart og vedlikeholdsvennlig ved å gå enda litt videre med makrobruken.

Du kunne for eksempel definere makroen **\$localnet**, i utgangspunktet definert som det nettverket som er koblet direkte til det indre grensesnittet (**x11:network** i eksemplene over).

Alternativt kan du endre definisjonen av **\$localnet** til *IP-adresse/nettmaske* for å angi et nettverk, for eksempel **192.168.100.1/24** for et subnett med private IPV4-adresser eller **fec0:dead:beef::/64** for et IPv6-adresseområde.

Hvis nettverket ditt krever det, kan du til og med definere ditt `$localnet` som en liste av nettverk. Uansett hvilke spesifikke behov du har, kan en fornuftig definisjon av `$localnet` sammen med en typisk pass-regel av type

```
pass from $localnet to any port $ports keep state
```

spare deg for mye hodebry. Vi vil holde oss til den konvensjonen fra nå av.

## Sette opp gatewayen

Vi forutsetter at du nå har skrudd inn et nettverkskort til, eller i alle fall sørget for nettverksforbindelse ut fra lokalnett, for eksempel via PPP. Vi kommer ikke inn på konfigurering av grensesnittene.

I det som følger vil det bare være selve grensesnittnavnet som vil variere mellom Ethernet og PPP, og de konkrete grensesnittnavnene kvitter vi oss med så fort det lar seg gjøre.

For det første må vi slå på gateway-funksjonen, slik at maskinen kan overføre trafikk den mottar på ett nettverksgrensesnitt videre til andre maskiner via et annet grensesnitt. Det kan vi gjøre fra kommandolinjen med `sysctl`, for tradisjonell *IP versjon fire* med

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

og hvis vi trenger å sende videre *IP versjon seks*-trafikk, er kommandoen

```
# sysctl net.inet6.ip6.forwarding=1
```

For at dette skal fortsette å virke når du en gang i fremtiden starter maskinen på nytt, må du legge disse verdiene inn i konfigurasjonsfilene.

På OpenBSD og NetBSD gjør du dette ved å redigere `/etc/sysctl.conf`, der du endrer de linjene du trenger, slik

```
net.inet.ip.forwarding=1
net.inet6.ip6.forwarding=1
```

På FreeBSD gjør du det tilsvarende med disse linjene i `/etc/rc.conf`

```
gateway_enable="YES" #for ipv4
ipv6_gateway_enable="YES" #for ipv6
```

Nettoeffekten er den samme, rc-scriptet på FreeBSD setter de to verdiene via sysctl-kommandoen, men på FreeBSD er altså større deler av konfigurasjonen sentralisert i `rc.conf`.

Er begge de grensesnittene du har tenkt å bruke, oppe og kjører? Bruk **`ifconfig -a`**, eventuelt **`ifconfig grensesnittnavn`** til å finne det ut.

Hvis du fortsatt tenker at du vil tillate all trafikk som maskinene på innsiden tar initiativ til, kan din `/etc/pf.conf` se slik ut<sup>2</sup>:

```
localnet = $int_if:network
ytre = "xl0" # makro for ytre gr.snitt - bruk tun0 for PPPoE
indre = "xl1" # makro for indre gr.snitt
# dynamisk ekstern IP-adresse, angis med ($grensesnitt)
nat on $ytre from $localnet to any -> ($ytre)
block all
pass from { lo0, $localnet } to any keep state
```

Legg merke til at vi bruker makroer for å gi nettverksgrensesnittene logiske navn. Her dreier det seg altså om 3Com-kort, men dette er siste gangen i dette foredraget at det er interessant overhodet. I så enkle oppsett som dette er nok ikke gevinsten av å bruke disse makroene så veldig stor, men når regelsettene blir litt større, vil du sette stor pris på lesbarheten.

Legg også merke til nat-regelen. Her sørger vi for nettverksadresseoversettingen fra de ikke-rutbare adressene på innsiden til den ene offisielle adressen vi forutsetter at du disponerer.

Parentesene rundt det siste leddet (**`$ytre`**) er der for å kompensere for at IP-adressen på det ytre grensesnittet kan være dynamisk tildelt, og sørger for at trafikken din vil gå som den skal selv om du plutselig blir tildelt ny ytre adresse.

En annen ting er at dette regelsettet sannsynligvis tillater mer trafikk ut enn det du egentlig har lyst til. Der jeg jobber, er den tilsvarende makroen

```
klient_ut = "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, http, \
             https, 446, cvspserver, 2628, 5999, 8000, 8080 }"
```

og regelen

```
pass inet proto tcp from $localnet to any port $klient_ut \
        flags S/SA keep state
```

---

2. For en del ADSL-brukere, spesifikt de som bruker PPP over Ethernet (PPPoE) og de fleste med oppringt samband vil ytre nettverksgrensesnitt være `tun0`, ikke Ethernet.

Det er mulig at dette er et sært utvalg, men det er det utvalget akkurat jeg og mine kolleger trenger. Noen av de odde portene er for systemer som jeg ikke får lov til å si mer om. Dine behov er sannsynligvis annerledes igjen, men en del av de mer nyttige tjenestene er nok dekket her.

Vi har noen andre pass-regler, og de fleste av de som kan være interessante kommer vi tilbake til senere. En regel som er relativt nyttig for oss som liker å kunne administrere maskiner fra andre steder i verden er

```
pass in inet proto tcp from any to any port ssh
```

eller forsåvidt

```
pass in inet proto tcp from any to $ytte port ssh
```

alt ettersom. Endelig trenger vi, for at navnetjenesten skal fungere også fra innsiden

```
udp_tjenester = "{ domain, ntp }"
```

pluss en regel som slipper det vi ønsker gjennom brannmuren:

```
pass quick inet proto { tcp, udp } to any port $udp_tjenester keep state
```

Legg merke til her at vi bruker nøkkelordet **quick**. Når vi lager regelsett som består av flere regler, er det på tide å se på hvordan forholdet mellom reglene i et regelsett fungerer. Regelsett blir tolket i rekkefølge 'ovenfra og ned'. Det betyr at reglene blir tolket i den rekkefølgen de er skrevet i konfigurasjonsfilen, og *siste regel som passer* for en gitt pakke eller forbindelse, er den som blir brukt. *quick* er veien ut av den vanlige sekvensen. Når en pakke passer til en quick-regel, vil pakken bli behandlet etter den regelen, uten at pakken blir vurdert mot resten av regelsettet. Fint når du har behov noen isolerte unntak fra generelle regler.

Da har vi tatt med ntp også, som bgrukes til tidssynkronisering. En av de tingene som disse protokollene har til felles, er at de under visse forutsetninger kan komme til å kommunisere over både TCP og UDP.

# Problembarnet FTP

I den lille listen over TCP-porter fra det virkelige livet dukket FTP opp. FTP er kort og godt et problembarn, spesielt når vi ønsker å kombinere den med brannmurer. FTP er en gammel og sær protokoll som det kan sies mye vondt om. De vanligste, men langt fra eneste, innvendingene går på at

- Passord overføres i klartekst
- Protokollen forutsetter bruk av minst to TCP-forbindelser (kontroll og data), på hver sin port
- Når forbindelse er opprettet, vil det bli kommunisert data på tilfeldig valgte porter

Alt dette er for å si det pent sikkerhetsmessige utfordringer, allerede før man kommer inn på at både klienter og servere kan ha feil og svakheter som i seg selv kan føre til sikkerhetsproblemer. Det har jo også skjedd noen ganger.

Uansett finnes det mer moderne og sikrere alternativer for filoverføring, som for eksempel sftp eller scp, der både autentisering og dataoverføring foregår på kryptert samband. Alle oppegående IT-mennesker bør ha noe annet som førstevalg for filoverføring.

Uansett hvor oppegående vi er, så vet vi at vi noen ganger er nødt til å håndtere det vi i utgangspunktet ikke liker. I tilfellet FTP gjennom brannmur håndterer vi det grøvste ved å omdirigere trafikken til et lite program som er spesialskrevet for å håndtere akkurat det.

## FTP gjennom NAT: ftp-proxy



### Kun OpenBSD 3.8 og tidligere

I november 2005 ble den gamle ftp-proxy (`/usr/libexec/ftp-proxy`) erstattet i OpenBSD-current med den nye ftp-proxy, som finnes i `/usr/sbin`. Det er denne programvaren som er med i OpenBSD 3.9 og nyere. Se nærmere beskrivelser i seksjonen kalt *ftp-proxy, ny utgave*.

ftp-proxy er en del av basesystemet på OpenBSD og andre systemer der PF inngår, og kalles vanligvis via "superserveren" inetd med en linje i `/etc/inetd.conf`.

Denne linjen spesifiserer at ftp-proxy skal kjøre i NAT-modus på loopback-grensesnittet `lo0`:

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

Denne linjen finner du normalt utkommentert med `#` i `inetd.conf`. Du aktiverer endringen ved å starte inetd på nytt.

På FreeBSD, NetBSD og andre BSDer med rcng gjør du det med

```
FreeBSD+NetBSD$ sudo /etc/rc.d/inetd restart
```

eller tilsvarende. Se man 8 inetd hvis du er i tvil.

På OpenBSD er rc-systemet mer tradisjonelt, så der blir kommandoen

```
OpenBSD$ sudo kill -HUP `cat /var/run/inetd.pid`
```

Da er inetd i gang med de nye innstillingene.

Så til selve omdirigeringen. Omdirigeringsregler og NAT-regler kommer i samme klasse. De kan bli direkte referert eller være blant forutsetningene for at filtreringsreglene skal fungere, og må derfor komme før filtreringsreglene i regelsettet.

Vi setter inn rdr-regelen rett etter nat-regelen i `/etc/pf.conf`

```
rdr on $indre proto tcp from any to any port ftp -> 127.0.0.1 port 8021
```

Den omdirigerte trafikken må også få lov til å passere, og det gjør vi med

```
pass in on $ytte inet proto tcp from port ftp-data to ($ytte) \
user proxy flags S/SA keep state
```

Lagre `pf.conf`, og last regelsettet på nytt med

```
$ sudo pfctl -f /etc/pf.conf
```

På dette tidspunktet vil du antakelig ha fornøyde brukere som oppdager at FTP virker før du sier fra at du er ferdig.

Dette eksempelet forutsetter NAT på gateway og ikke-rutbare adresser på innsiden.

## FTP, PF og rutbare adresser: ftpsesame, pftpx og ftp-proxy!

I tilfeller der lokalnettet bruker offisielle og rutbare adresser innenfor brannmuren har i alle fall jeg hatt problemer med å få ftp-proxy til å fungere. Etter en del fikling var det en lettelse å se at en hyggelig nederlender ved navn Camiel Dobbelaar hadde skrevet en liten daemon som heter ftpsesame som løsning på akkurat det problemet.

Lokalnett med offisielle adresser på innsiden er visst såpass sært at jeg ikke skal komme nærmere inn på det her. Hvis du trenger denne funksjonaliteten og kjører OpenBSD 3.8 eller tidligere eller forsåvidt andre operativsystemer som PF er portet til, så kan du ha godt utbytte av å hente ftpsesame fra Sentia på <http://www.sentia.org/projects/ftpsesame/>. ftpsesame knyttes til regelsettet via et anker, eller navngitt del-regelsett. Dokumentasjonen består av en man-side med eksempler som du med stor sannsynlighet kan klippe og lime fra direkte.

ftpsesame ble aldri en del av basesystemet, og Camiel har nå laget en nyere løsning over samme lest.

Det nye programmet, som i begynnelsen het pftpx, er tilgjengelig fra <http://www.sentia.org/downloads/pftpx-0.8.tar.gz>. En videre utviklet versjon, nå omdøpt for å fylle rollen som den nye ftp-proxy, er fra og med OpenBSD3.9 en del av basesystemet som `/usr/sbin/ftp-proxy`.

## ftp-proxy, ny utgave



### For OpenBSD 3.9 og nyere

Hvis du oppgraderer til OpenBSD 3.9 eller går ut fra en fersk installasjon av 3.9 eller nyere, er det denne versjonen av ftp-proxy du skal bruke.

Akkurat som med forgjengeren, er pftpx består konfigureringen av ftp-proxy for det meste av klipp og lim fra man-siden.

Hvis du oppgraderer til den nye generasjonen av ftp-proxy fra en tidligere versjon, må du fjerne linjen for ftp-proxy fra `inetd.conf` og starte `inetd` på nytt.

Når det er gjort, aktiverer du ftp-proxy ved å legge til denne linjen i `/etc/rc.conf.local` eller `/etc/rc.conf`

```
ftpproxy_flags=""
```

Du kan starte proxyen manuelt ved å kjøre `/usr/sbin/ftp-proxy` hvis du ønsker det.

I `pf.conf` trenger du to anker-definisjoner i NAT-delen:

```
nat-anchor "ftp-proxy/*"  
rdr-anchor "ftp-proxy/*"
```

Begge to er nødvendige, selv om din konfigurasjon ikke bruker NAT. Hvis du går fra en tidligere versjon, har du antakelig et regelsett som inneholder en passende omdirigering allerede. Hvis ikke, legger du den til:

```
rdr pass on $indre proto tcp from any to any port ftp -> 127.0.0.1 \  
    port 8021
```

Blant filtreringsreglene legger du til ankeret som proxyen skal fylle ut,

```
anchor "ftp-proxy/*"
```

og endelig en pass-regel som lar pakkene slippe gjennom fra proxyen til resten av verden

```
pass out proto tcp from $proxy to any port 21 keep state
```

der `$proxy` er en makro for adressen proxy-daemonen har bundet seg til.

Dette eksempelet dekker den enkle situasjonen med klienter som har behov for å kontakte FTP-servere andre steder i verden. Hvis du har behov for andre variasjoner og mer kompliserte konfigurasjoner, kan du se i man-siden for ftp-proxy.

Hvis du er ute etter en måte å kjøre en FTP-server som er beskyttet av PF og ftp-proxy, kan du se på muligheten for å kjøre en separat ftp-proxy i revers-modus (med alternativet `-R`).

# Gjøre nettverket ditt feilsøkingssvennlig

Hvordan du gjør nettverket ditt feilsøkingssvennlig, er potensielt et svært omfattende emne. Som regel henger feilsøkingssmulighetene i TCP/IP-nettverk tett sammen med hvordan du behandler den internett-protokollen som ble laget spesifikt for feilsøkingssformål, *Internet Control Message Protocol*, eller *ICMP* som er den vanligste forkortelsen.

ICMP er protokollen for å sende og motta *kontrollmeldinger* mellom maskiner og gatewayer, hovedsakelig for å melde tilbake til avsender om eventuelle uvanlige eller vanskelige forhold på veien frem til maskinen det er ønske om å kommunisere med.

Det er normalt at en hel del ICMP-aktivitet foregår i bakgrunnen mens du surfer på web, leser epost eller overfører filer. Ruterne (du er klar over at du holder på å lage en, ikke sant?) bruker ICMP til å forandle om pakkestørrelser og andre overføringsparametre i en prosess som ofte omtales som *path MTU discovery*.

Det kan være du har hørt administratorer snakke om ICMP som enten 'pur ondskap', eller hvis de har noe dypere forståelse, 'et nødvendig onde'. Bakgrunnen for disse holdningene er rent historiske. Grunnen finner vi for noen år tilbake, da det ble oppdaget av flere operativsystemer hadde kode i nettverksstakkene sine som kunne få en sårbar maskin til å stoppe opp eller begynne å gjøre merkelige ting etter at maskinen hadde mottatt en tilstrekkelig stor ICMP-forespørsel.

Et av firmaene som ble hardt rammet av dette var Microsoft, og du kan finne ganske mye materiale om disse feilene ved å søke på 'ping of death' i et passende søkesystem. Alt dette hendte i andre halvdel av 1990-årene, og alle moderne operativsystemer, i alle fall de vi kan lese, har fått nettverkskoden sin grundig renoverert siden den gang. Det er i alle fall det vi har blitt fortalt.

En av måtene man opprinnelig omgikk problemet på, var å blokkere for enten all ICMP-trafikk, eller i det minste ICMP echo, som er det kommandoen ping bruker. I dag har disse regelsettene eksistert i rundt ti år, og personene som laget dem kan fortsatt være skvetne.

## Skal vi så slippe alt gjennom?

Det naturlige spørsmålet blir da, hvis ICMP er en slik god og nyttig ting, burde vi ikke la den slippe uhindret igjennom hele tiden? Svaret blir, 'det kommer an på'.

Hvis diagnosetrafikk får passere uhindret, vil det helt klart føre til at eventuell feilsøking går lettere. Samtidig vil det også gjøre det relativt enkelt for andre å hente ut informasjon om nettverket ditt. Det betyr at en regel av typen

```
pass inet proto icmp from any to any
```

gjærne ikke er optimal hvis du ikke ønsker å la omverdenen få fullt innsyn i hvordan nettverket ditt er satt sammen internt. Det bør også sies at i noe ICMP-trafikk helt harmløst følger med som følge av 'keep state'-reglene.

## Den enkle måten: Dette stopper her

Den enkleste løsningen kan meget vel være å la all ICMP-trafikk fra lokalnettet passere gjennom, og la tilsvarende trafikk fra andre steder stoppe på gatewayen:

```
pass inet proto icmp icmp-type $icmp_types from \  
    $localnet to any keep state  
pass inet proto icmp icmp-type $icmp_types from any to  
    $ytre keep state
```

Det kan uansett være fristende å stoppe informasjonssøking utenfra på gatewayen, men la oss først se litt på noen andre alternativer som viser fleksibiliteten PF gir.

## Slippe ping gjennom

En klar ulempe med regelsettet som det står akkurat nå, er at feilsøkingsskuddene ping og traceroute ikke vil virke. Det er ikke sikkert at det betyr spesielt mye for brukerne dine, og siden det var ping som skremte folk til å blokkere for ICMP i utgangspunktet, finnes det tydeligvis mennesker som mener vi har det bedre uten. Du som er i

målgruppen her, vil nok trives bedre når disse feilsøkningsverktøyene er tilgjengelige. Et par små tillegg i regelsettet ditt vil gi deg dem. ping bruker ICMP, og for å gjøre det ryddig lager vi først en makro:

```
icmp_typer = "echoreq"
```

og en regel som bruker definisjonen

```
pass inet proto icmp all icmp-type $icmp_typer keep state
```

Det kan være at du får behov for å slippe gjennom flere eller andre typer ICMP-pakker. Da kan du utvide **icmp\_typer** til en liste som inneholder de typene du vil tillate.

## Hjelp traceroute

En annen kommando som er veldig nyttig når brukerne sier at Internet ikke virker, er traceroute. På Unix bruker den normalt UDP-forbindelser etter en fast formel for å spore hvor trafikken kan gå. Denne regelen fungerer for traceroute på de UNIXene jeg har hatt adgang til, inkludert GNU/Linux:

```
# allow out the default range for traceroute(8):  
# "base+nhops*nqueries-1" (33434+64*3-1)  
pass out on $ytte inet proto udp from any to any \  
    port 33433 >< 33626 keep state
```

All erfaring så langt tyder på at traceroute-implementasjoner på andre operativsystemer fungerer på samme vis, bortsett fra, ikke overraskende, Microsoft Windows. På den plattformen bruker TRACERT.EXE ICMP ECHO til dette formålet. Så hvis du vil la pakkesporing fra Windows-maskiner komme gjennom, trenger du bare den første regelen. traceroute på unix kan også bruke andre protokoller, og vil oppføre seg på samme måten som programmet fra Microsoft hvis du bruker kommandolinjeflagget **-I**. Du kan se i man-siden til traceroute (eller forsåvidt kildekoden) for å få alle detaljer.

I alle fall stammer denne løsningen fra en posting på openbsd-misc. Den listen, og da spesielt de søkbare listearkivene (som finnes for eksempel på <http://marc.theaimsgroup.com/>), er en veldig verdifull ressurs når du lurar på noe med OpenBSD eller PF.

## Path MTU discovery

Det siste jeg skal minne dere på i denne omgang når det gjelder feilsøking, er prosessen som omtales som 'path MTU discovery'. Internett-protokollene ble konstruert for å være utstyrsuavhengige, og en konsekvens av denne utstyrsuavhengigheten er at det ikke alltid er mulig å forutse hva som vil være optimal pakkestørrelse for en gitt forbindelse. Den viktigste begrensningen kalles *Maximum Transmission Unit* (største overføringsenhet) eller *MTU*, som angir den øvre grensen for pakkestørrelsen på et nettverksgrensesnitt. Kommandoen `ifconfig` vil blant annet vise deg gjeldende MTU for nettverksgrensesnittene dine.

Moderne TCP/IP-implementasjoner er konstruert slik at de forventer å være i stand til å finne frem til riktig pakkestørrelse for en forbindelse ved hjelp av en prosess som består i å sende pakker med varierende størrelser med 'ikke fragmenter'-flagget (Do not fragment) slått på, med forventning om at det vil komme en ICMP-returpakke med indikasjonen "type 3, kode 4" når øvre grense er nådd. Det er ikke nødvendig å kaste seg over RFCene med en gang. Type 3 betyr "destination unreachable" (mål kan ikke nås), mens kode 4 er kortformen av "fragmentering nødvendig, men ikke fragmenter-flagget er på". Så hvis forbindelsene dine til nettverk der MTUen kan være en annen enn den er i ditt nettverk kan virke suboptimale og du ikke har andre grunner til å være så spesifikk, kan du prøve å endre listen over tillatte ICMP-typer noe, slik at også 'Destination unreachable'-pakker kommer gjennom:

```
icmp_types = "{ echoreq, unreach }"
```

det betyr som vi ser at vi ikke trenger å røre selve pass-regelen:

```
pass inet proto icmp all icmp-type $icmp_typer keep state
```

PF gir deg mulighet til å filtrere på alle variasjoner av ICMP-typer og koder, og hvis du vil fordype deg i hva du bør og ikke bør slippe gjennom av ICMP-trafikk, finner du listen over alle mulige typer og koder i man-sidene `icmp(4)` and `icmp6(4)`. Bakgrunnsinformasjonen er tilgjengelig i RFCene<sup>1</sup>.

1. De viktigste internet-RFCene som handler om ICMP og noen beslektede emner er RFC792, RFC950, RFC1191, RFC1256, RFC2521, RFC2765, mens oppdateringene som er nødvendige for ICMP for IPv6 finnes i RFC1885, RFC2463, RFC2466. Disse dokumentene er tilgjengelige mange steder på nettet, for eksempel på webområdene til [ietf.org](http://www.ietf.org) (<http://www.ietf.org>) og [faqs.org](http://www.faqs.org) (<http://www.faqs.org>), og er i tillegg antakelig tilgjengelig via pakkesystemet du bruker.

Det er mulighet at jeg kommer tilbake til filtrering av ICMP i en fremtidig versjoner.

# Nettverkshygiene: block-policy, scrub og så videre

Jeg føler at gatewayen vår ikke er helt komplett uten noen tillegg i konfigurasjonen som skal gjøre at den oppfører seg bedre overfor maskiner både på utsiden og innsiden.

## block-policy

**block-policy** er en verdi som settes i **options**-delen av regelsettet, altså før omdirigerings- og filtreringsregler. Dette bestemmer hvilken reaksjon PF skal sende tilbake til maskiner som prøver å opprette forbindelser som blir blokkert. Innstillingen har to mulige verdier, **drop** som gjør at blokkerte pakker blir droppet uten tilbakemelding, og **return** som gir returkoder av type `Connection refused` eller liknende.

Det har vært diskutert mye om hva som er den beste strategien. Vi velger her å være etterrettelige og angir at brannmuren skal gi retur:

```
set block-policy return
```

## scrub

**scrub** er et nøkkelord som utløser en normalisering av nettverkspakkene som blant annet innebærer at fragmenterte pakker blir satt sammen igjen. **scrub** innebærer blant annet en viss beskyttelse mot enkelte typer angrep som baserer seg på fragmenterte eller ufullstendige nettverkspakker. Det finnes en del mulige tilleggsargumenter, men vi velger den enkle formen, som vil være tilstrekkelig og tjenlig i de aller fleste tilfeller.

```
scrub in all
```

Noen tjenester, for eksempel NFS, stiller spesielle krav til behandling av fragmenterte pakker. Dette er mer inngående beskrevet i PF-dokumentasjonen.

## antispoof

**antispoof** er et spesialtilfelle av filtrering og blokkering. Mekanismen beskytter mot angrep fra forfalskede IP-adresser, hovedsakelig ved å blokkere for pakker som kommer via grensesnitt og med retning som logisk ikke skulle være mulig. Vi angir at vi ønsker å luke vekk slike forfalskninger fra omverdenen og de som mot formodning måtte dukke opp fra vårt eget nett:

```
antispoof for $ytre
antispoof for $indre
```

## Håndtere ikke-rutbare adresser fra andre steder

Selv når du har en riktig konfigurert gateway som håndterer nettverksadresseoversetting for ditt eget nettverk, kan du komme ut for at du må kompensere for feilkonfigurasjoner andre har gjort andre steder.

En direkte deprimerende vanlig type feilkonfigurasjoner er de som lar trafikk med ikke-rutbare adresser slippe ut på Internett. Trafikk fra ikke-rutbare adresser har også spilt en rolle i flere teknikker som er brukt i DOS-angrep, så det kan være verd å overveie å blokkere eksplisitt for trafikk fra ikke-rutbare adresser inn til nettverket ditt.

En mulig løsning er skissert nedenfor, her i en utgave som for sikkerhets skyld blokkerer for forsøk på å kontakte ikke-rutbare adresser gjennom det ytre grensesnittet på gatewayen:

```
martians = "{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, \
             10.0.0.0/8, 169.254.0.0/16, 192.0.2.0/24, \
             0.0.0.0/8, 240.0.0.0/4 }"
```

```
block drop in quick on $ytre from $martians to any
block drop out quick on $ytre from any to $martians
```

Her står makroen **martians** adressene fra RFC 1918 og noen få andre adresseområder som forskjellige RFCer angir ikke skal være i åpen sirkulasjon på Internett. Trafikk til og fra slike adresser blir stille forkastet på det ytre grensesnittet på gatewayen.

*Nettverkshygiene: block-policy, scrub og så videre*

Nøyaktig hvordan denne typen beskyttelse bør implementeres vil variere etter en rekke forhold, hovedsakelig hvordan nettverket ellers er konfigurert. Det er for eksempel ikke utenkelig at nettverksdesignen du arbeider ut fra gjør at du må tillate eller blokkere andre adresseområder enn det som er angitt her.

Dermed er vår enkle brannmur med NAT for et lite lokalnett komplett.

# En webserver og epostserver på innsiden

Tiden går, og behovene endrer seg. Et behov som ofte oppstår, er å kunne kjøre tjenester som er tilgjengelige utenfra. Dette blir ofte vanskelig fordi ekstra eksternt synlige adresser enten ikke er tilgjengelig eller dyrt, og det er ofte ikke ønskelig å kjøre mange tjenester på samme maskin som fungerer som brannmur.

Omdirigeringsmekanismene i PF gjør det relativt enkelt å ha servere på innsiden. Hvis vi tenker oss at behovet er å ha en webserver som tilbyr tjenester i klartekst (http) og kryptert (https), og du ønsker å ha en epostserver som sender og mottar epost, samtidig som den lar klienter innenfor og utenfor eget nett bruke en del kjente protokoller for henting og sending, kan disse linjene være alt du trenger å føye til regelsettet fra tidligere:

```
webserver = "192.168.2.7"
webporter = "{ http, https }"
epostserver = "192.168.2.5"
epost = "{ smtp, pop3, imap, imap3, imaps, pop3s }"

rdr on $ytte proto tcp from any to $ytte port \
    $webporter -> $webserver
rdr on $ytte proto tcp from any to $ytte port
    $epost -> $epostserver

pass proto tcp from any to $webserver port $webporter \
    flags S/SA synproxy state
pass proto tcp from any to $epostserver port $epost \
    flags S/SA synproxy state
pass proto tcp from $epostserver to any port smtp \
    flags S/SA synproxy state
```

Her er flagget 'synproxy' tatt med. Det betyr at det er PF, ikke din server eller forsåvidt klient, som håndterer opprettingen av forbindelsen (treveis-håndtrykket) før applikasjonen overtar selv. Dette gir en viss beskyttelse mot visse typer angrep.

Regelsett for konfigurasjoner med DMZ-nett som er isolert bak eget nettverksgrensesnitt og eventuelt tjenester på alternative porter vil ikke nødvendigvis skille seg særlig fra dette.

## Ta vare på sine egne - innsiden

Alt jeg har sagt så langt stemmer veldig godt så lenge du bare tar hensyn til trafikk som kommer fra andre steder enn ditt eget nettverk.

Hvis du ønsker at maskiner på lokalnettet skal bruke tjenestene på de samme maskinene, oppdager du fort at trafikken fra lokalnettet sannsynligvis aldri kommer innom det ytre nettverksgrensesnittet, der all omdirigering og adresseoversetting foregår. Dette problemet er faktisk så vanlig at PF-dokumentasjonen nå inneholder fire separate løsninger på problemet.<sup>1</sup>

Alternativene som foreslås i PF-brukerhåndboken er

- DNS med 'delt horisont', altså at navnetjenesten gir forskjellige svar avhengig av om forespørselen kommer fra lokalnettet eller andre steder
- proxying med for eksempel nc (NetCat)
- spesialbehandling av lokalenettet ved hjelp av omdirigering og NAT.

Vi skal se nærmere på dette alternativet nedenfor.

- Eller helt enkelt flytte serverne til et separat nett, såkalt 'DMZ', med bare mindre endringer i PF-regler

Vi trenger å fange opp pakkene som kommer fra lokalnettet, og sørge for at forbindelsene blir håndtert riktig slik at returtrafikk kommer frem til den kommunikasjonspartneren som faktisk opprettet forbindelsen.

I vårt eksempel klarer vi dette ved å legge til disse reglene:

```
rdr on $indre proto tcp from $localnet to $ytre \  
    port $webporter -> $webserver  
rdr on $indre proto tcp from $localnet to $ytre \  
    port $epost -> $epostserver  
no nat on $inter proto tcp from $indre to $localnet
```

---

1. Se *Redirection and Reflection* (<http://openbsd.org/faq/pf/rdr.html#reflect>) i PF-brukerhåndboken.

## *En webserver og epostserver på innsiden*

```
nat on $indre proto tcp from $localnet to $webserver \  
    port $webporter -> $indre  
nat on $indre proto tcp from $localnet to $epostserver \  
    port $epost -> $indre
```

# Gjøre livet lettere med tabeller

Nå er det sikkert en del som sitter og synes alt dette blir veldig statisk. Det finnes da virkelig data som kan være viktige for filtrering og dirigering der og da, men som ikke fortjener plass i en konfigurasjonsfil! Ganske riktig, og PF har mekanismer som dekker dette også. Ett eksempel er tabeller, som først og fremst egner seg for lister som det skal være mulig å manipulere uten å laste hele regelsettet på nytt, og der det er viktig at oppslaget går raskt. Tabellnavn settes i < >, slik:

```
table <klienter> { 192.168.2.0/24, !192.168.2.5 }
```

her er nettverket **192.168.2.0/24** med i tabellen, med unntak av adressen **192.168.2.5**, som utelukkes med operatoren ! (logisk NOT). Det går også an å laste tabeller fra filer der elementene står enkeltvis på hver sin linje, for eksempel med filen `/etc/klienter`

```
192.168.2.0/24
!192.168.2.5
```

som så brukes til å initialisere tabellen i `/etc/pf.conf`:

```
table <klienter> persist file /etc/klienter
```

Så kan du for eksempel endre regelen fra tidligere

```
pass inet proto tcp from <klienter> to any port $klient_ut \
    flags S/SA keep state
```

for å regulere trafikken ut fra klientmaskinene dine. Når det er gjort, kan du manipulere innholdet i tabellen i fart, for eksempel

```
$ sudo pfctl -t klienter -T add 192.168.1/16
```

Husk på at dette bare endrer tabellen i minnet, så endringen vil for eksempel ikke overleve neste strømbrudd eller annen omstart hvis du ikke sørger for å ta vare på endringene dine.

Tabellen på disk kan for eksempel vedlikeholdes med en cron-jobb som med jevne mellomrom dumper tabellen til disk, for eksempel ved å legge inn en kommando ala `pfctl -t klienter -T show >/etc/klienter`. Alternativt kan du redigere filen `/etc/klienter` og erstatte innholdet av tabellen når du har gjort endringer:

```
$ sudo pfctl -t klienter -T replace -f /etc/klienter
```

For operasjoner du trenger å gjøre ofte, vil du før eller senere lage shellscript som gjør slike ting som å sette inn og fjerne oppføringer eller erstatte innholdet i tabeller. Dine behov og din kreativitet er de eneste reelle begrensningene.

Vi kommer innom andre praktiske anvendelser for tabeller senere, inkludert noen programmer som bruker tabeller til forskjellige nyttige formål.

# Logging

Hittil har vi ikke sagt så mye om logging. PF gir deg muligheten til å logge akkurat det du har behov for via nøkkelordet 'log' i de reglene du måtte ønske. Det kan være fornuftig å begrense datamengden noe ved å angi hvilket grensesnitt det skal logges på. Det gjør du i så fall med

```
set loginterface $ytre
```

og så i de reglene du ønsker å ha data om, noe slikt som

```
pass out log from <klient> to any port $epost \  
    label klient-epost keep state
```

Trafikken blir da logget i et format som er ment som inndata for tcpdump.

*label*-leddet fører til at det blir opprettet egne tellere for en del statistikk akkurat for denne regelen. Slikt kan være hendig for eksempel ved viderefakturering av brukt båndbredde.

## Ta en titt med tcpdump

Når du har aktivert logging i en eller flere regler, logger PF via grensesnittet `pflog0`, og lagrer binære loggdata i loggfilen `/var/log/pflog`. Denne loggfilen er nyttig som permanent registrering og for de tilfellene der du med visse mellomrom ønsker å konvertere dataene til andre formater. Hvis du på den annen side ønsker å se på trafikken i noe nær sanntid, kan du be `tcpdump` om å se på logg-grensesnittet `pflog0` i stedet.

Her er et eksempel på hva som kan komme ut fra et par regler med logging på en ellers fredelig torsdag ettermiddag:

```
peter@skapet:~$ sudo tcpdump -n -e -ttt -i pflog0  
tcpdump: WARNING: pflog0: no IPv4 address assigned  
tcpdump: listening on pflog0, link-type PFLOG  
Feb 16 16:43:20.152187 rule 0/(match) block in on x10: 194.54.59.189.2559 >  
194.54.107.19.139: [|tcp] (DF)  
Feb 16 16:48:26.073244 rule 27/(match) pass in on x10: 61.213.167.236 >  
194.54.107.19: icmp: echo request  
Feb 16 16:49:09.563448 rule 0/(match) block in on x10: 61.152.249.148.80 >  
194.54.107.19.55609: [|tcp]  
Feb 16 16:49:14.601022 rule 0/(match) block in on x10: 194.54.59.189.3056 >
```

```
194.54.107.19.139: [|tcp] (DF)
Feb 16 16:53:10.110110 rule 0/(match) block in on x10: 68.194.177.173 >
194.54.107.19: [|icmp]
Feb 16 16:55:54.818549 rule 27/(match) pass in on x10: 61.213.167.237 >
194.54.107.19: icmp: echo request
Feb 16 16:57:55.577782 rule 27/(match) pass in on x10: 202.43.202.16 >
194.54.107.19: icmp: echo request
```

Brukerhåndboken for PF har en egen del som handler om logging, med mange gode råd. Med denne informasjonen og blant annet man-sidene for tcpdump skulle du være i stand til å trekke ut de loggdataene du måtte ha behov for.

## Men det finnes grenser (en liten historie fra virkeligheten)

En ting som kan være fristende å prøve i begynnelsen, er å legge inn noe slikt som

```
block log all
```

- sånn bare for å få oversikten.

PF-bruikerhåndboken inneholder en detaljert oppskrift på hvordan du får PF til å logge i menneskelesbart tekstformat via syslog, og det høres jo ganske forlokkende ut. Jeg fulgte den oppskriften da jeg satte opp min første PF-konfigurasjon for jobben, og erfaringen er svært entydig: Logging er nyttig, men vær for all del selektiv. I løpet av en time var tekstloggfilen for PF blitt på over en gigabyte, på en maskin med mindre enn ti gigabyte diskplass totalt.

Forklaringen er rett og slett at selv i en uspenkende avkrok av Internet, på enden av en nokså kjedelig ADSL-linje, så finnes det ufattelige mengder løs Windows-trafikk av type fildeling og søk etter alt mulig rart.

Windows-maskinene på innsiden var nok ikke helt tause, de heller. I alle fall: begrensn hva du logger, hvis du ikke har satt av nok plass, ett eller annet sted.

# Få oversikt med pftop

Et nyttig verktøy når du skal holde litt følge med hva som passerer inn og ut av nettet ditt, er Can Erkin Acars pftop. Som navnet antyder, viser pftop øyeblikksbilder av trafikken i et format som er sterkt inspirert av top(1):

```
pfTop: Up State 1-21/67, View: default, Order: none, Cache: 10000      19:52:28
```

PR	DIR	SRC	DEST	STATE	AGE	EXP	PKTS	BYTES
tcp	Out	194.54.103.89:3847	216.193.211.2:25	9:9	28	67	29	3608
tcp	In	207.182.140.5:44870	127.0.0.1:8025	4:4	15	86400	30	1594
tcp	In	207.182.140.5:36469	127.0.0.1:8025	10:10	418	75	810	44675
tcp	In	194.54.107.19:51593	194.54.103.65:22	4:4	146	86395	158	37326
tcp	In	194.54.107.19:64926	194.54.103.65:22	4:4	193	86243	131	21186
tcp	In	194.54.103.76:3010	64.136.25.171:80	9:9	154	59	11	1570
tcp	In	194.54.103.76:3013	64.136.25.171:80	4:4	4	86397	6	1370
tcp	In	194.54.103.66:3847	216.193.211.2:25	9:9	28	67	29	3608
tcp	Out	194.54.103.76:3009	64.136.25.171:80	9:9	214	0	9	1490
tcp	Out	194.54.103.76:3010	64.136.25.171:80	4:4	64	86337	7	1410
udp	Out	194.54.107.18:41423	194.54.96.9:53	2:1	36	0	2	235
udp	In	194.54.107.19:58732	194.54.103.66:53	1:2	36	0	2	219
udp	In	194.54.107.19:54402	194.54.103.66:53	1:2	36	0	2	255
udp	In	194.54.107.19:54681	194.54.103.66:53	1:2	36	0	2	271

Det kan vise forbindelsene sortert etter en rekke kriterier, blant annet etter pf-regel, volum, alder og så videre.

Dette programmet er ikke med i basesystemet, men er med i ports på både OpenBSD og FreeBSD som `/usr/ports/sysutils/pftop`, på NetBSD via `pkgsrc` som `sysutils/pftop`.

# Usynlig gateway med bridge

En *bridge* er i vår sammenheng en maskin med to eller flere nettverkskort som sitter mellom Internet og ett eller flere indre nettverk, og der nettverksgrensesnittene ikke har IP-adresse. Hvis maskinen kjører OpenBSD eller et annet kapabelt operativsystem, kan den likevel filtrere og om dirigere trafikk. Fordelen med et slikt oppsett er at det er mye vanskeligere enn ellers å angripe selve brannmuren. Ulempen er at all administrasjon må foregå på konsollet til brannmuren, hvis man da ikke setter av et eget nettverksgrensesnitt som er kontaktbart via et sikret nett, eller for den del serielt konsoll.

Metoden for konfigurering av bridge skiller seg i noen detaljer mellom operativsystemene. Her er en kort oppskrift for OpenBSD, der man for oversiktens skyld blokkerer for all annen trafikk enn Internet-protokoller. Sette opp broen med to grensesnitt:

```
/etc/hostname.x10

up

/etc/hostname.x11

up

/etc/bridgename.bridge0

    add x10 add x11 blocknonip x10 blocknonip x11 up

/etc/pf.conf

ytre = x10
indre  = x11
interessant-trafikk = { ... }
block all
pass quick on $ytre all
pass log on $indre from $indre to any port $interessant-trafikk \
    keep state
```

Mer kompliserte oppsett er mulige, for å si det slik. Den klare anbefalingen fra de som vet slikt, er å holde filtrering og om dirigering på ett grensesnitt. Siden alle pakker passerer to ganger, kan reglene bli innviklete.

### *Usynlig gateway med bridge*

Endelig gir kommandoen `brconfig` på OpenBSD selvstendige muligheter for filtrering i tillegg til andre konfigureringsmuligheter. Man-sidene `bridge(4)` og `brconfig(8)` viser vei til deg som ønsker å vite mer.

Syntaks for `bridge` på FreeBSD skiller seg noe fra dette. På NetBSD støtter ikke PF `bridge`-funksjonene.

# Trafikkregulering med ALTQ

ALTQ - forkortelse for ALternate Queueing - er en svært fleksibel trafikkstyringsmekanisme som levde et eget liv før den ble integrert i PF. Dette var enda en ting som det var mest praktisk å integrere.

Altq bruker begrepet queue, altså "kø", om hovedmekanismen for trafikkstyringen. Man definerer køer med gitt båndbredde eller del av båndbredde, der hver kø kan utstyres med underordnede køer av ulike typer.

Bildet blir komplett når du lager filtreringsregler som henviser pakkene til en gitt kø eller forsåvidt et utvalg delkøer der pakkene kan slippe til etter nærmere definerte kriterier.

Køene kan følge flere *disipliner*. Normal køordning uten ALTQ er FIFO, altså første inn første ut.

Mer interessante er disiplinen som gjør køene klassebaserte (CBQ), noe som i praksis betyr at du angir båndbredden for køen som absolutt datamengde per sekund, enten prosent eller i kilobit, megabit og så videre, med mulighet for prioritetsangivelse i tillegg, eller prioritetsbasert (priq), der du kun angir en prioritet.

Prioritet kan angis fra 0 til 7 for cbq-køer, 0 til 15 for priq-køer, der høyere verdi gir høyere prioritet og bedre behandling. Endelig finnes det en hierarkisk kødisiplin, algoritmen er kjent som "Hierarchical Fair Service Curve", eller HFSC.

Syntaksen for definering av køer er kort

```
altq on grensesnitt type [alternativer ... ] hovedkø { delkø1, delkø2 ..}
    queue delkø1 [ alternativer ... ]
    queue delkø2 [ alternativer ... ]
[...]
pass [ ... ] queue delkø1
pass [ ... ] queue delkø2
```

Når du skal bruke dette i praksis, bør du uansett lese man-sidene for `pf.conf` og brukerhåndboken for PF. Der er syntaks og annet forklart svært

detaljert og ganske oversiktlig.<sup>1 2</sup>

## ALTQ - prioritere etter trafikktype

Det første praktiske eksempelet er hentet fra Daniel Hartmeier. Daniel har som mange av oss en asymmetrisk forbindelse, og ønsket å få utnyttet båndbredden bedre.

Symptomet på at noen kunne gjøres bedre, var at inn-trafikk (nedlasting) så ut til å medføre at utgående trafikk gikk tregere.

En analyse kunne tyde på at dette kom av at ACK-pakkene for hver overførte datapakke ble forsinket uforholdsmessig mye, antakelig fordi den utgående trafikken ble køet etter FIFO-prinsippet, altså først inn, først ut.

En mulig hypotese som det var verd å prøve, var at de små ACK-pakkene, som praktisk talt ikke inneholder data, ville kunne snike seg forbi de større datapakkene hvis to køer med forskjellig prioritet var tilgjengelig. De relevante delene av regelsettet følger:

```
ext_if="kue0"

altq on $ext_if priq bandwidth 100Kb queue { q_pri, q_def }
queue q_pri priority 7
queue q_def priority 1 priq(default)

pass out on $ext_if proto tcp from $ext_if to any flags S/SA \
    keep state queue (q_def, q_pri)

pass in  on $ext_if proto tcp from any to $ext_if flags S/SA \
    keep state queue (q_def, q_pri)
```

Resultatet ble bedre båndbreddeutnyttelse.

1. På FreeBSD krever ALTQ alternativene for ALTQ og de kødisiplinene du vil bruke er med i konfigurasjonen for kjernen som er i bruk. Se i kapittelet om PF ([http://www.freebsd.org/doc/en\\_US.ISO8859-1/books/handbook/firewalls-pf.html](http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-pf.html)) i FreeBSD Handbook for nærmere beskrivelse.

2. I skrivende stund er ALTQ ikke integrert i PF på NetBSD, men Peter Postma vedlikeholder en patch som gjør funksjonaliteten tilgjengelig. Oppdatert informasjon om dette er tilgjengelig fra Peter Postmas PF på NetBSD-sider, <http://nedbsd.nl/~ppostma/pf/>

## Så hvorfor virker dette?

Så hvorfor virker dette?<sup>3</sup> Forklaringen finner vi i måten ALTQ-koden behandler underordnede køer med forskjellig prioritet. Når en forbindelse blir tilordnet hovedkøen, sjekker ALTQ 'type or service'-feltet (ToS) i hver enkelt pakke. ACK-pakker har 'Delay'-biten i Tos-feltet satt til 'low', noe som betyr at avsenderen ønsker raskest mulig levering.

Når ALTQ ser en pakke med lav ønsket forsinkelse og køer med forskjellige prioriteter er tilgjengelige, tilordner den pakken til køen med høyest prioritet. Dette betyr at ACK-pakkene hopper forbi køen med lavere prioritet og blir levert fortere, noe som i sin tur betyr at datapakkene også blir behandlet raskere.

Daniels artikkel er tilgjengelig på nettstedet hans, <http://www.benzedrine.cx/ackpri.html>

## ALTQ - prosentvis fordeling

Så til et eksempel som jeg i alt vesentlig har rappet fra unix.se. Her ser vi at vi begynner med å deklare at køen skal etableres på eksternt grensesnitt. Dette er antakelig det vanligste fordi det er der det er størst begrensninger på båndbredden, men i prinsippet er det ikke noe i veien for å etablere køer og kjøre trafikkforming på hvilket som helst nettverksgrensesnitt. Her er det snakk om en cbq-kø med total båndbredde på 640KB og seks underkøer.

```
altq on $ytte cbq bandwidth 640Kb queue { def, ftp, udp, http, \
    ssh, icmp }
queue def bandwidth 18% cbq(default borrow red)
queue ftp bandwidth 10% cbq(borrow red)
queue udp bandwidth 30% cbq(borrow red)
queue http bandwidth 20% cbq(borrow red)
queue ssh bandwidth 20% cbq(borrow red) { ssh_interactive, ssh_bulk }
queue ssh_interactive priority 7
queue ssh_bulk priority 0
queue icmp bandwidth 2% cbq
```

Her ser vi at delkøen def med 18 prosent båndbredde er angitt som standardkø, dvs all trafikk som ikke er eksplisitt tilordnet en annen kø,

3. Tidligere versjoner av dette manuskriptet lot det stort sett være en oppgave for leseren å finne ut av.

havner her. Nøkkelordene borrow og red betyr at køen kan 'låne' båndbredde fra overordnet kø, og vil prøve å unngå at køen blir full etter algoritmen RED (Random Early Detection). De andre køene følger stort sett same mønster, helt til vi kommer til delkøen ssh, som igjen har to delkøer med forskjellig prioritet.

I ssh-køen ser vi igjen en variant av ACK-prioritering via underordnede køer: SSH-overføringer merket 'bulk', typisk filoverføring via SCP, blir overført med ToS som angir normal forsinkelse, mens interaktiv SSH-trafikk har 'low delay'-biten satt og blir overført raskere enn 'bulk'-overføringene.

Dette oppsettet har antakeligvis også positiv innvirkning på hastigheten for SCP-filoverføringer, siden ACK-pakkene for SCP vil bli tilordnet køen med høy prioritet.

Så kommer vi til pass-reglene som sørger for at trafikk blir fordelt til køene, og etter hvilke kriterier:

```
pass log quick on $ytte proto tcp from any to any port 22 flags S/SA \
    keep state queue (ssh_bulk, ssh_interactive)
pass in quick on $ytte proto tcp from any to any port 20 flags S/SA \
    keep state queue ftp
pass in quick on $ytte proto tcp from any to any port 80 flags S/SA \
    keep state queue http
pass out on $ytte proto udp all keep state queue udp
pass out on $ytte proto icmp all keep state queue icmp
```

Det er rimelig å anta at denne fordelingen svarer til behovet på stedet.

## **ALTQ - håndtere uønsket trafikk**

Et siste altq-eksempel er et som dukket opp i forbindelse med en av de mange spam- eller virusstormene vi har hatt de siste årene. Som kjent er det i alt vesentlig maskiner med Windows som er opphav til slik epost-trafikk. PF har en nokså pålitelig funksjon for å finne ut hvilket operativsystem som kjøres i den andre enden. En OpenBSD-bruker ble lei av all denne meningsløse trafikken, og postet disse bitene av sin `pf.conf` i bloggen sin:

```
altq on $ext_if cbq queue { q_default q_web q_mail }
```

## Trafikkregulering med ALTQ

```
queue q_default cbq(default)
queue q_web (...)

## all mail limited to 1Mb/sec
queue q_mail bandwidth 1Mb { q_mail_windows }
## windows mail limited to 56Kb/sec
queue q_mail_windows bandwidth 56Kb

pass in quick proto tcp from any os "Windows" to $ext_if port 25 \
    keep state queue q_mail_windows
pass in quick proto tcp from any to $ext_if port 25 label "smtp" \
    keep state queue q_mail

" I can't believe I didn't see this earlier. Oh, how sweet. ...
  Already a huge difference in my load. Bwa ha ha. "
```

Randal L. Schwartz, 29. januar 2004,  
<http://use.perl.org/~merlyn/journal/17094>

Her blir epost-trafikk tildelt totalt en megabit av båndbredden, mens all epost-trafikk som kommer fra Windows-maskiner må dele på totalt 56 kilobit. Ikke så veldig rart at belastningen gikk ned og at postingen avslutter med noe som antyder rå latter.

Dette er noe som i alle fall jeg har hatt veldig lyst til å gjøre, men jeg tør ikke. Litt for mange av kundene våre, som vi trenger å motta epost fra, kjører akkurat eposttjenesten sin på en eller annen windows. Om en liten stund skal vi se på en annen bruk av PF som kanskje ville oppnådd mye av den samme effekten.

# CARP og pfsync

De to store nyhetene i OpenBSD 3.5 var CARP og pfsync. CARP står for Common Address Redundancy Protocol. Den ble utviklet som et ikke patentbelemret alternativ til VRRP (Virtual Router Redundancy Protocol, RFC 2281, RFC 3768) som var på god vei til å bli godkjent som IETF-standard til tross for at mulige begrensninger på grunn av patentkrav ikke var avklart (patenter eid av Cisco, IBM, Nokia).

Begge protokollene er tiltenkt å sikre redundans for viktige funksjoner i nettverk, med automatisk overflytting ved feil.

CARP baserer seg på at en gruppe maskiner settes opp med en 'master' og en eller flere redundante 'slaver' som alle kan håndtere en felles IP-adresse. Om master går ned, vil en av slavene overta IP-adressen, og i den grad det er sørget for synkronisering, overta de aktive forbindelsene. Overleveringen vil være sikret med kryptonøkler.

Ett av formålene med CARP, er å sikre at nettverket vil fungere normalt selv om en brannmur eller andre tjenester blir tatt ned, enten på grunn av feil eller for eksempel for vedlikehold som maskinvare- eller programvareoppgradering.

Synkroniseringen kan i tilfellet PF-brannmurer håndteres av pfsync, som er en type virtuelt nettverksgrensesnitt som er konstruert for å synkronisere tilstandsinformasjon mellom PF-brannmurer. pfsync-grensesnitt tilordnes fysiske grensesnitt med ifconfig. På nettverk der kravene til oppetid er så strenge at automatisk feilhåndtering er nødvendig, vil antakelig antallet nettverksforbindelser være så høyt at det vil være fornuftig å la pfsync-trafikken gå på et eget fysisk nettverk.

Dette er en av de spennende og avanserte funksjonene jeg håper på å få komme tilbake til i en fremtidig avansert del av manuskriptet. Foreløpig er de beste referansene om pfsync og CARP OpenBSDs FAQ, man-sidene og Ryan McBrides oversiktsartikkel på <http://www.countersiege.com/doc/pfsync-carp/>

# Trådløse nettverk, ganske enkelt

Det er ganske fristende å si at på BSD, og spesielt på OpenBSD, er det ikke nødvendig å 'gjøre trådløse nettverk enkelt', fordi det allerede er det. Det å få opp et trådløst nettverk er ikke grunnleggende forskjellig fra å få i gang et kablet nett, men det er noen hensyn vi må ta rett og slett fordi vi bruker radiobølger og ikke kabler. Det fører blant annet til at det er enklere å 'sniffe' eller avlytte dataene dine. Vi skal se kort på noen av disse temaene før vi kommer tilbake til de praktiske konfigureringsstegene.

## Litt bakgrunnsinformasjon om IEEE 802.11

Når vi går over til trådløse nett, får vi en mulighet til å se på sikkerheten på forskjellige nivåer i nettverksstakken fra et nytt perspektiv. Vi ser kort på de to grunnleggende sikkerhetsmekanismene i IEEE 802.11 nedenfor.<sup>1</sup>

Det er noe nær selvsagt at du trenger flere sikkerhetstiltak, eksempelvis SSH eller SSL-kryptering, for å oppnå noe vesentlig konfidensialitet for datastrømmen din.

## WEP (Wired Equivalent Privacy)

En konsekvens av å bruke radiobølger i stedet for kabler er at det er relativt enkelt for utenforstående å fange inn dataene dine mens de er underveie. De som utformet 802.11-familien av standarder for trådløse nettverk ser ut til å ha vært klar over problemstillingen, og fant på en løsning som ble markedsført under navnet *Wired Equivalent Privacy*, vanligvis forkortet til *WEP*.

WEP transportnivå-kryptering som regnes for å være nokså primitiv og amatørmessig blandt profesjonelle kryptografer. Det var ingen stor overraskelse at WEP-krypteringen ble reverskonstruert og knekket i løet av få måneder etter at de første produktene ble lansert. Hvem som helst kan laste ned gratis verktøy som kan dekode WEP-kodet trafikk på få

---

1. En mer fullstendig oversikt over emner som gjelder sikkerhet i trådløse nettverk finner du i Professor Kjell Jørgen Holes artikler og presentasjoner på [www.kjhole.com](http://www.kjhole.com) (<http://www.kjhole.com/Standards/WiFi/WiFiDownloads.html>). Webstedene [wifinetnews.com](http://wifinetnews.com) ([http://wifinetnews.com/archives/cat\\_security.html](http://wifinetnews.com/archives/cat_security.html)) og The Unofficial 802.11 Security Web Page (<http://www.drizzle.com/~aboba/IEEE/>) er varmt anbefalt for ferske nyheter om utviklingen i WiFi-feltet.

minutter, men av forskjellige grunner er systemet fortsatt svært utbedt og støttes av mange produkter. Det er fornuftig å regne nettverkstrafikk som bare er beskyttet med WEP som bare marginalt mer sikker enn data som kringkastes i klartekst. På den annen side kan den helt symbolske innsatsen som skal til for å knekke seg inn på et WEP-nettverk være tilstrekkelig til å holde late og lite sofistikerte angripere vekke.

## **WPA (WiFi Protected Access)**

Det tok ikke lang tid før det gikk opp for de som hadde designet 802.11 at systemet de hadde kalt Like Sikkert Som Kabel (Wired Equivalent Privacy) ikke helt levde opp til forventningene. Etter en tid kom de med en revidert og noe mer omfattende løsning som ble lansert under navnet *WiFi Protected Access*, forkortet til *WPA*.

WPA ser i alle fall på papiret bedre ut enn WEP, men spesifikasjonen er så pass komplisert at den ikke har nådd det nivået av støtte og utbredelse som de som utformet systemet hadde håpet på. I tillegg har WPA vært gjenstand for en del kritikk over utforming og rene feil. Sammen med velkjente temaer som adgang til dokumentasjon og utstyr har dette ført til at støtten i fri programvare er høyst variabel. Hvis spesifikasjonen for ditt prosjekt angir WPA, må du være svært nøye med å lese dokumentasjonen for operativsystem og drivere.

## **Sette opp et enkelt trådløst nettverk**

Første steg er å sørge for at du har et brukbart kort og sjekke utdataene fra dmesg for å se at driveren blir lastet og at kortet blir initialisert som det skal<sup>2</sup> Med riktig konfigurering av kortet skal du se noe slikt som

---

2. Støtten for trådløse nettverk i OpenBSD og BSDene generelt blir stadig bedre, men det betyr ikke nødvendigvis at det er lett å få samlet alt du trenger. Historien om mitt hjemme-nettverk kan være ganske illustrerende, og er som følger: Jeg startet med å kjøpe to CNet CWP-854-kort som skulle være støttet i OpenBSD 3.7 via den nye ral-driveren. Det kortet jeg satte i en splitter ny Dell-maskin som kjører et ufritt operativsystem fungerte umiddelbart. Gatewayen min, som hadde kjørt uten episoder av noe slag siden OpenBSD 3.3, hadde på sin side litt flere problemer. Kortet ble gjenkjent og konfigurert, men i det øyeblikket Dell-maskinen prøvde å få tildelt en IP-adresse, gikk gatewayen ned med en kjernepanikk. En detaljert beskrivelse er tilgjengelig som OpenBSD PR 4217. Jeg har lovet å teste kortet på nytt med et nyere snapshot - så snart jeg klarer å finne kortet igjen. Fra Dell-maskinen kunne vi

```
ath0 at pci1 dev 4 function 0 "Atheros AR5212" rev 0x01: irq 11
ath0: AR5212 5.6 phy 4.1 rf51111 1.7 rf21111 2.3, ETSI1W, address
00:0d:88:c8:a7:c4
```

Det neste du må gjøre, er å konfigurere grensesnittet for TCP/IP. På OpenBSD betyr dette en `/etc/hostname.ath0`-fil som ser omtrent slik ut:

```
up media autoselect mediaopt hostap mode 11b chan 6 nwid unwiredbsd \
  nwkey 0x1deadbeef9
inet 10.168.103.1
```

Legg merke til at konfigurasjonen er delt over to linjer. Den første linjen genererer en `ifconfig`-kommando som setter opp grensesnittet med riktige parametre for det fysiske trådløse nettverket, mens den andre kommandoen, som blir utført først etter at den første er fullført, setter IP-adressen. Merk at vi angir kanalen eksplisitt, og at vi aktiverer en svak WEP-kryptering ved å angi parameteren **nwkey**.

På FreeBSD vil du legge inn disse linjene i `/etc/start_if.ath0`, og eventuelt sette inn det aktuelle grensesnittnavnet i stedet for **ath0**:

Sannsynligvis ønsker du å sette opp `dhcpcd` til å utstyre klientene med adresser og annen relevant nettverksinformasjon. Klientene trenger da en konfigurasjon med en slik `/etc/hostname.ath0`-fil:

```
up media autoselect mode 11b chan 6 nwid unwiredbsd nwkey 0x1deadbeef9
dhcp
```

---

se en utrolig mengde nettverk, nesten alle sammen usikret, men det er en helt annen historie. Jeg kom til at jeg ville prøve `ath`-kort, og kjøpte et D-Link DWL-G520, som jeg så klarte å forlegge under flytting. Like etter kjøpte jeg et DWL-G520+, etter tankegangen om at pluss-tegnet måtte bety at det var bedre. Dessverre betydde plusstegnet at et helt annet brikkesett var brukt, nemlig TI ACX111, som har lav pris, men ingen dokumentasjon som er tilgjengelig for utviklere av fri programvare. Heldigvis hadde butikken ingen innvendinger mot å la meg levere tilbake kortet og få pengene igjen. På dette tidspunktet hadde jeg rukket å bli ganske frustrert, og dro til den andre siden av byen til en butikk som hadde flere DWL-AG520-kort på lager. Dette kortet var en del dyrere enn de andre, men det fungerte med en gang. Et par uker senere dukket G520-kortet opp igjen, og det fungerte selvfølgelig også, i en annen maskin. Den bærbare maskinen min (som da kjørte FreeBSD) ble levert med et Realtek 8180 trådløskort i mini-PCI-format, men av en eller annen grunn fikk jeg det aldri til å fungere. Jeg endte med å kjøpe et DWL-AG650 cardbus-kort, som virker prikkfritt med `ath`-driveren. Mitt råd er generelt at hvis du handler i nettbutikk, bør du ha man-sidene tilgjengelig i en annen fane eller et annet vindu, og hvis du går til en fysisk butikk må du passe på å si til butikkpersonalet at du skal bruke kortet med en BSD. Hvis du ikke er sikker på at kortet er støttet, kan du se om du kan låne en maskin for å lese man-sidene på nettet. Hvis du sier fra til butikkpersonalet på forhånd hva du har tenkt å gjøre, kan det bli lettere å få pengene igjen hvis delen ikke virker, og hvis du passer på å fortelle dem om deler som fungerer, er det god PR for BSDen du bruker. Det er antakelig verd å nevne at reverskonstruert støtte for ACX1nn-baserte kort har kommet til BSDene med driveren `acx`, som ble introdusert i OpenBSD 4.0.

og igjen, på FreeBSD ville du legge disse linjene inn i og NetBSD, `/etc/start_if.ath0`, og eventuelt sette inn det aktuelle grensesnittnavnet i stedet for `ath0`.

Vi antar at gatewayen også utfører NAT for nettverket ditt, og du vil også ønske å sette opp NAT for det trådløse nettverket ved å gjøre noen små endringer i `/etc/pf.conf`:

```
air_if = "ath0"
```

og

```
nat on $ext_if from $air_if:network to any -> ($ext_if) static-port
```

Du vil også trenge en nesten-duplikatlinje for konfigurering av ftp-proxy, og å ta med `$air_if` i pass-reglene.

Det er alt som skal til. Denne konfigurasjonen gir deg et fungerende BSD-basert aksesspunkt, med en symbolsk sikkerhet via WEP-kryptering.

# Et åpent, men strengt bevoktet trådløst nettverk med authpf

Som så ofte ellers, ser vi at det er andre måter å konfigurere sikkerheten i det trådløse nettverket enn det vi akkurat har sett på. Det lille grann av beskyttelse som WEP-kryptering gir er de fleste med noe innsikt i IT-sikkerhet enige om stort sett bare er nyttig som et signal til angripere om at du ikke har tenkt å la alle og enhver bruke nettverksressursene dine.

En helt annen tilnærming dukket opp en dag i eposten min i en melding fra min venn Vegard Engen, som fortalte at han hadde satt opp authpf. authpf er et skall du tildeler brukere, som så gir deg mulighet til å laste PF-regler tilpasset den enkelte brukeren. Dermed får du potensielt ganske finkornet kontroll over hvem som får lov til å gjøre hva.

Hvis du vil bruke authpf, må du opprette brukere som har programmet authpf som skall. For å få adgang til nettverket logger så brukeren inn på gatewayen med ssh. Når brukeren er ferdig autentisert over ssh, laster authpf de reglene du har definert for den aktuelle brukeren eller den klassen av brukere som brukeren tilhører.

Disse reglene, som da gjelder for den IP-adressen som brukeren er logget inn fra, forblir aktive så lenge brukeren er innlogget via ssh-forbindelsen. Når forbindelsen brytes, blir reglene fjernet, og i de aller fleste scenarier blir alt annet enn ssh-trafikk fra brukerens IP-adresse sperret. Med et fornuftig oppsett vil bare trafikk som stammer fra autentiserte brukere bli sluppet gjennom.

Vegards konfigurasjon med kommentarer følger nedenfor. Hans trådløse nettverk er konfigurert uten WEP-kryptering, siden han foretrekker å håndtere sikkerheten via PF og authpf:

Start med en tom `/etc/authpf/authpf.conf`. Filen må eksistere for at authpf skal virke, men den trenger ikke å ha noe innhold.

De relevante delene av `/etc/pf.conf` følger. Først grensesnitt-makroene:

```
int_if="sis1"  
ext_if="sis0"  
wi_if = "wi0"
```

Hva denne adressen brukes til vil bli klart senere:

## *Et åpent, men strengt bevoktet trådløst nettverk med authpf*

```
auth_web="192.168.27.20"
```

### Den tradisjonelle authpf-tabellen

```
table <authpf_users> persist
```

Vi kunne lagt NAT-delen i `authpf.rules`, men det skader ikke å beholde den i selve `pf.conf`:

```
nat on $ext_if from $wi_if:network to any -> ($ext_if)
```

Omdirigeringer for å slippe gjennom trafikk til servere på det interne nettet. Disse kunne vi også lagt inn i `authpf.rules`, men siden de ikke faktisk gir noen tilgang uten pass-regler, skader det ikke å beholde dem her.

```
rdr on $wi_if proto tcp from any to $myaddr port $tcp_in -> $server
rdr on $wi_if proto udp from any to $myaddr port $udp_in -> $server
```

Den neste omdirigeringen sender all web-trafikk fra brukere som ikke er autentiserte, til port 80 på `$auth_web`. I Vegards oppsett er dette en web-server som viser kontaktinfo til de som måtte komme over det trådløse nettet ved en tilfeldighet. I mer kommersielle sammenhenger ville dette vært stedet du plasserte noe som kunne håndtere kredittkort og opprette brukere.

```
rdr on $wi_if proto tcp from ! <authpf_users> to any \
port 80 -> $auth_web
```

Dette for å aktivere `nat`, `binat` eller omdirigeringer i `authpf`

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

Videre til filtreringsreglene, og vi tar et fornuftig utgangspunkt

```
block all
```

Andre globale, brukeruavhengige regler kommer her. Så kommer `authpf`-ankeret, der vi sørger for at ikke autentiserte brukere som kobler seg til det trådløse grensesnittet blir omdirigert til `$auth_web`

```
anchor "authpf/*" in on wi0
```

## *Et åpent, men strengt bevoktet trådløst nettverk med authpf*

```
pass in on $wi_if inet proto tcp from any to $auth_web \  
  port 80 keep state
```

**Det er tre ting vi uansett ønsker oss på det trådløse grensesnittet:  
Navnetjeneste (DNS), DHCP og SSH inn til gatewayen. Disse tre reglene  
sørger for det**

```
pass in on $wi_if inet proto udp from any port 53 keep state
```

```
pass in on $wi_if inet proto udp from any to $wi_if port 67
```

```
pass in on $wi_if inet proto tcp from any to $wi_if \  
  port 22 keep state
```

**Det neste vi gjør, er å definere regler som blir lastet for alle brukere som  
logger inn med /usr/sbin/authpf som skall. Disse reglene går i  
/etc/authpf/authpf.rules,**

```
int_if = "sis1"  
ext_if = "sis0"  
wi_if = "wi0"  
server = "192.168.27.15"  
myaddr = "213.187.n.m"
```

```
# Tjenester som lever på det interne nettet  
# og trenger å være tilgjengelige  
tcp_services = "{ 22, 25, 53, 80, 110, 113, 995 }"  
udp_services = "{ 53 }"  
tcp_in = " { 22, 25, 53, 80, 993, 2317, pop3 }"  
udp_in = "{ 53 }"
```

```
# Slipp gjennom trafikk til andre steder, altså verden utenfor  
pass in on $wi_if inet from <authpf_users> to ! $int_if:network \  
  keep state
```

```
# La autentiserte brukere bruke tjenester på  
# det interne nettverket.
```

```
pass in on $wi_if inet proto tcp from <authpf_users> to $server \  
  port $tcp_in keep state  
pass in on $wi_if inet proto udp from <authpf_users> to $server \  
  port $udp_in keep state
```

```
# Slipp også gjennom til ekstern adresse. Dette betyr at du kan nå
```

## *Et åpent, men strengt bevoktet trådløst nettverk med authpf*

```
# interne tjenester på eksterne adresser.  
  
pass in on $wi_if inet proto tcp from <authpf_users> to $myaddr \  
    port $tcp_in keep state  
pass in on $wi_if inet proto udp from <authpf_users> to $myaddr \  
    port $udp_in keep state
```

Da har vi kommet så langt at vi har et åpent nett der alle kan koble seg til og få en IP-adresse fra DHCP. Alle HTTP-forespørsler fra brukere som ikke er autentisert blir omdirigert til port 80 på 192.168.27.20, som er en server på det interne nettet der alle forespørsler besvares med den samme siden, som viser kontaktinfo i tilfelle du ønsker å bli registrert og få tillatelse til å bruke nettet.

Du får lov til å bruke ssh inn til gatewayen. Brukere med gyldig bruker-ID og passord får lastet regelsett med passende pass-regler for den IP-adressen de har blitt tildelt.

Vi kan finjustere dette enda mer ved å lage brukerspesifikke regler i `/etc/authpf/users/$bruker/authpf.rules`. Brukerspesifikke regler kan bruke makroen `$user_ip` som står for brukerens IP-adresse. Hvis jeg for eksempel ønsker å gi meg selv ubegrenset tilgang, lager jeg en `/etc/authpf/users/vegard/authpf.rules` som ser slik ut:

```
wi_if="wi0"  
pass in on $wi_if from $user_ip to any keep state
```

# Stoppe maset fra robotene

Hvis du kjører en ssh-innloggingstjeneste på en maskin som er tilgjengelig fra Internett, er jeg ganske sikker på at du har sett noe som likner på dette i autentiseringsloggene dine:

```
Sep 26 03:12:34 skapet sshd[25771]: Failed password for root from
200.72.41.31 port 40992 ssh2
Sep 26 03:12:34 skapet sshd[5279]: Failed password for root from
200.72.41.31 port 40992 ssh2
Sep 26 03:12:35 skapet sshd[5279]: Received disconnect from
200.72.41.31: 11: Bye Bye
Sep 26 03:12:44 skapet sshd[29635]: Invalid user admin from
200.72.41.31
Sep 26 03:12:44 skapet sshd[24703]: input_userauth_request:
invalid user admin
Sep 26 03:12:44 skapet sshd[24703]: Failed password for invalid user
admin from 200.72.41.31 port 41484 ssh2
Sep 26 03:12:44 skapet sshd[29635]: Failed password for invalid user
admin from 200.72.41.31 port 41484 ssh2
Sep 26 03:12:45 skapet sshd[24703]: Connection closed by 200.72.41.31
Sep 26 03:13:10 skapet sshd[11459]: Failed password for root from
200.72.41.31 port 43344 ssh2
Sep 26 03:13:10 skapet sshd[7635]: Failed password for root from
200.72.41.31 port 43344 ssh2
Sep 26 03:13:10 skapet sshd[11459]: Received disconnect from
200.72.41.31: 11: Bye Bye
Sep 26 03:13:15 skapet sshd[31357]: Invalid user admin from 200.72.41.31
Sep 26 03:13:15 skapet sshd[10543]: input_userauth_request: invalid
user admin
Sep 26 03:13:15 skapet sshd[10543]: Failed password for invalid user
admin from 200.72.41.31 port 43811 ssh2
Sep 26 03:13:15 skapet sshd[31357]: Failed password for invalid user
admin from 200.72.41.31 port 43811 ssh2
Sep 26 03:13:15 skapet sshd[10543]: Received disconnect from
200.72.41.31: 11: Bye Bye
Sep 26 03:13:25 skapet sshd[6526]: Connection closed by 200.72.41.31
```

Så blir gjentakelsene ganske kjedelige å følge med på. Det er slik et "brute force"-angrep ser ut. Det betyr at noen, eller mer sannsynlig en datamaskin noen har brutt seg inn på, prøver en tilnærmet uendelig rekke kombinasjoner av brukernavn og passord for å finne frem til noe som gir dem adgang til systemet ditt.

Det enkleste ville være å skrive en regel i `pf.conf` som blokkerte all tilgang. Det ville i sin tur føre til en helt ny klasse med problemer, ikke minst hva du måtte finne på for å la personer med legitime ærender på maskinen din slippe til. Du kunne kanskje overveie å flytte tjenesten til en annen port, men det er sannsynligvis lite som hindrer de som nå krafser på port 22 i å skanne seg frem til port 22222 for å fortsette.

Siden OpenBSD 3.7 har PF tilbudt en litt mer elegant løsning. Du kan skrive pass-regler slik at de håndhever visse begrensninger på hva maskinene som kobler seg til kan gjøre. I tillegg kan du legge de som går over grensene over i en tabell med adresser som du sperrer for all eller noe tilgang. Hvis du vil, kan du til og med velge å droppe alle eksisterende forbindelser fra maskiner som går ut over grensene du setter. Det gjør du slik:

Først setter du opp tabellen. I tabelldelen legger du til

```
table <bruteforce> persist
```

Så setter du opp ganske tidlig i regelsettet ditt en regel for å blokkere trafikk fra adressene i tabellen:

```
block quick from <bruteforce>
```

Og til slutt skriver du pass-regelen.

```
pass inet proto tcp from any to $localnet port $tcp_services \
    flags S/SA keep state \
(max-src-conn 100, max-src-conn-rate 15/5, \
    overload <bruteforce> flush global)
```

Dette er ganske likt noe vi har sett før, eller hva? Faktisk er den første delen identisk med den generelle pass-regelen vi satte sammen tidligere. Delen i parentes er det nye materialet, som vil fjerne en del av belastningen på nettet ditt.

*max-src-conn* er det antallet samtidige forbindelser du tillater fra en enkelt maskin. I dette eksempelet har jeg satt det til 100. I din konfigurasjon passer det kanskje med en noe lavere eller noe høyere verdi.

*max-src-conn-rate* er et mål på hvor ofte en gitt maskin får lov til å opprette nye forbindelser. Her er dette angitt til 15 forbindelser i løpet av 5 sekunder. Igjen er det opp til deg selv å avgjøre hva som passer for ditt oppsett.

*overload* <bruteforce> betyr at hvis en maskin prøver å gå ut over disse begrensningene, blir adressen dens lagt til i tabellen *bruteforce*. Regelsettet vårt blokkerer som vi vet all trafikk fra adresser i tabellen *bruteforce*.

Til slutt angir *flush global* at når en maskin går over grensene, vil de aktive forbindelsene fra den aktuelle maskinen bli avsluttet. Vi har også med *global*, som betyr at for sikkerhets skyld avslutter vi også forbindelser som ville blitt sluppet gjennom av andre pass-regler.

Effekten er ganske dramatisk. De som prøver seg, ender som oftes opp med å generere meldinger av type "Fatal: timeout before authentication". Da kan de ikke plage oss mer, om ikke annet.

Husk igjen på at regelen i dette eksempelet først og fremst er ment som illustrasjon. Det er ikke usannsynlig at du på ditt nettverk er bedre tjent med andre regler eller kombinasjoner av regler.

Det er for eksempel mulig at du ønsker å tillate et rimelig stort antall forbindelser generelt, men gjerne vil stramme litt til når det gjelder ssh. Da kan du supplere regelen over med noe slikt som dette, litt tidligere i regelsettet:

```
pass quick proto { tcp, udp } from any to any port ssh \  
    flags S/SA keep state \  
    (max-src-conn 15, max-src-conn-rate 5/3, \  
    overload <bruteforce> flush global)
```

Du finner frem til den beste kombinasjonen for akkurat din situasjon ved å lese de aktuelle man-sidene og brukerhåndboken for PF (<http://www.openbsd.org/faq/pf/>), og kanskje litt eksperimentering.

## expiretable rydder tabellene

Nå er vi kommet så langt at vi har tabeller som vil bli fylt opp automatisk av **overload**-regler, og siden det er rimelig å forvente at gatewayene våre vil ha opptid som måles i måneder, vil tabellene gradvis vokse seg større og ta stadig mer plass i minnet.

Du kan også oppleve at en IP-adresse du blokkerte i forrige uke på grunn av for stor trafikk faktisk var en dynamisk tildelt adresse, som nå brukes av en annen kunde hos den aktuelle ISP'en, som faktisk har legitim grunn til å prøve å kommunisere med maskiner i ditt nettverk.

Det var slike situasjoner som førte til at Henrik Gustafsson skrev `expiretable`, som fjerner tabelloppføringer som ikke har blitt referert i et gitt tidsrom.

Et nyttig eksempel kan være å bruke `expiretable` som metode for å fjerne foreldete oppføringer i `<bruteforce>`-tabellen.

Du kan for eksempel la `expiretable` fjerne oppføringer i `<bruteforce>`-tabellen som er eldre enn 24 timer ved å legge til en linje i `/etc/rc.local` med dette innholdet:

```
/usr/local/sbin/expiretable -v -d -t 24h bruteforce
```

`expiretable` ble nylig tatt med i `ports-treet` på FreeBSD og OpenBSD<sup>1</sup>.

Hvis `expiretable` ikke er tilgjengelig via pakkesystemet du bruker, kan du laste ned programmet fra Henriks web på <http://expiretable.fnord.se/>

---

1. som henholdsvis `/usr/ports/security/expiretable` og `/usr/ports/sysutils/expiretable`

# Mobbe spammere

Når alt dette er tilbaketrukket, er det fint å kunne presentere noe som er virkelig nyttig, nemlig PF som verktøy til å mobbe spammere. Med det vi nå vet om PF, er det greit å forstå dette oppsettet i `/etc/pf.conf`:

```
table <spamd> persist
table <spamd-white> persist file "/var/mail/whitelist.txt"
rdr pass on $ytres inet proto tcp from <spamd> to \
    { $ytres, $localnet } port smtp -> 127.0.0.1 port 8025
rdr pass on $ytres inet proto tcp from !<spamd-white> to \
    { $ytres, $localnet } port smtp -> 127.0.0.1 port 8025
```

Vi har to tabeller, der den ene blir fylt opp fra en fil i filsystemet. SMTP-trafikk fra adressene i den første tabellen pluss de som ikke er med i den neste tabellen, blir omdirigert til en daemon som lytter på port 8025.

## Du er ikke alene: Svartelister

Det sentrale poenget som hele spamd er bygget opp rundt, er at spammere sender ut store mengder meldinger, og at sannsynligheten for at akkurat du er den første som får en gitt melding, er forsvinnende liten. Spam blir også i alt vesentlig sendt ut via noen få spamvennlige nettverk og en stor mengde kaprete maskiner. Både enkeltmeldinger og maskiner vil bli rapportert til svartelister ganske fort, og dermed har vi noe å fylle opp den første tabellen med.

## Svartelister, gråelister og klebrige tjærehull

Det spamd gjør når den kommer ut fra SMTP-tilkoblinger fra adresser i svartelisten, er å presentere banneret sitt, og deretter svare på smtp-trafikk med 1 byte om gangen. Denne teknikken, som har til hensikt å kaste bort så mye tid som mulig på avsendersiden og praktisk talt ikke bruke ressurser for mottakeren, kalles *tjærehull* (tar-pitting). Denne spesifikke implementasjonen med SMTP-svar på 1 byte kalles ofte *stuttering* eller stamming.

spamd støtter også *grålisting* (greylisting), som består i å avvise meldinger fra ukjente maskiner midlertidig med 45n-koder, for så å slippe gjennom meldinger fra de maskinene som prøver på nytt innen rimelig tid. Trafikk fra maskiner som oppfører seg ordentlig, altså avsendere som ser satt opp til å oppføre seg i samsvar med rammene i de relevante RFCene<sup>1</sup> vil bli sluppet gjennom.

Grålisting som teknikk ble presentert i 2003 i en artikkel av Evan Harris<sup>2</sup>. En rekke implementasjoner dukket opp i løpet av månedene som fulgte. I OpenBSD fikk spamd evnen til grålisting i OpenBSD versjon 3.5, som ble utgitt i mai 2004.

Det mest forbløffende med grålistingen, bortsett fra at teknikken er så enkel, er at den fortsatt fungerer. Spammere og skadevareprodusenter har vært svært sene med å tilpasse seg. Vi skal se på noen eksempler senere.

## Sette opp spamd

Konfigurering av spamd er nokså rett frem<sup>3</sup>. Du redigerer `/etc/spamd.conf` etter dine behov. En god del er forklart i filen, og mer informasjon finnes i man-siden. Standardforslaget til svartlister inneholder svartlisting av koreanske adresser. Siden jeg jobber i et firma som har noen koreanske kunder, måtte jeg fjerne akkurat det fra vårt oppsett. Det er du som bestemmer hvilke datakilder som skal brukes, og det er fullt mulig å ta i bruk andre lister.

Legg inn linjene for spamd og oppstartsparemetrene du ønsker i `/etc/rc.conf` eller `/etc/rc.conf.local`. Merk at du for eksempel kan finjustere flere av parametrene for grålisting via kommandolinjeparametre til spamd.

Når oppsettet er ferdig redigert, starter du spamd med de alternativene du ønsker og kjører spamd-setup. Endelig legger du inn en cron-jobb som oppdaterer tabellene med de intervallene du ønsker.

Når tabellene er fylt opp, kan du vise og manipulere innholdet med `pfctl` eller andre programmer, på samme måte som med andre tabeller.

1. De relevante RFCene er hovedsakelig RFC1123 og RFC2821.
2. Harris' artikkel og en rekke andre nyttige artikler og ressurser finnes på webområdet [greylisting.org](http://www.greylisting.org) (<http://www.greylisting.org/>).
3. Merk at spamd er en egen port på FreeBSD, `/usr/ports/mail/spamd/`. Hvis du bruker PF på FreeBSD 5.n eller nyere, må du følge anvisningene fra installeringsprosessen, og komme tilbake hit

Merk at i eksempelet over har rdr-reglene også med et 'pass'-ledd. Hvis du bruker rdr uten 'pass', må du ha pass-regler som sørger for at trafikken skal slipper gjennom til omdirigeringen. Du må også huske på å ha regler for å slippe gjennom legitim post. Hvis du allerede har en epost-tjeneste i nettverket, kan du antakelig la SMTP-reglene du har fra før bli stående uforandret.

## Hovedtrekk i erfaringene med spamd

Hvordan er så spamd i daglig bruk? Vi tok i bruk spamd tidlig i desember 2004, etter å ha kjørt spamassassin og clamav som en del av leveringsprosessen i exim en stund. exim er satt opp til å merke og levere alt med poengsum fra 5 til 9.99 poeng og kaste alt med 10 poeng eller mer og meldinger med virus. I løpet av høsten var spamassassin blitt stadig sløvere, og slapp gradvis flere uønskede meldinger gjennom.

Da vi satte spamd i drift gikk mengden meldinger totalt og mengden meldinger spamassassin måtte ta seg av, drastisk ned. Mengden som kommer umerket gjennom, har nå stabilisert seg på omtrent fem om dagen, fordelt på en håndfull rapporterende brukere.

Hvis du starter spamd med kommandolinjeflagget `-v` får du en del informasjon ut over bare hvilke IP-adresser som tar kontakt. Med `-v` kan et et typisk loggutsnitt se slik ut:

```
Oct  2 19:55:05 delilah spamd[26905]: (GREY) 83.23.213.115:
<gilbert@keyholes.net> -> <wkitp98zpu.fsf@datadok.no>
Oct  2 19:55:05 delilah spamd[26905]: 83.23.213.115: disconnected after
0 seconds.
Oct  2 19:55:05 delilah spamd[26905]: 83.23.213.115: connected (2/1)
Oct  2 19:55:06 delilah spamd[26905]: (GREY) 83.23.213.115:
<gilbert@keyholes.net> -> <wkitp98zpu.fsf@datadok.no>
Oct  2 19:55:06 delilah spamd[26905]: 83.23.213.115: disconnected after
1 seconds.
Oct  2 19:57:07 delilah spamd[26905]: (BLACK) 65.210.185.131:
<bounce-3C7E40A4B3@branch15.summer-bargainz.com> -> <adm@dataped.no>
Oct  2 19:58:50 delilah spamd[26905]: 65.210.185.131: From: Auto
Insurance Savings <noreply@branch15.summer-bargainz.com>
Oct  2 19:58:50 delilah spamd[26905]: 65.210.185.131: Subject: Start
SAVING MONEY on Auto Insurance
Oct  2 19:58:50 delilah spamd[26905]: 65.210.185.131: To: adm@dataped.no
```

## *Mobbe spammere*

```
Oct  2 20:00:05 delilah spamd[26905]: 65.210.185.131: disconnected after
404 seconds. lists: spews1
Oct  2 20:03:48 delilah spamd[26905]: 222.240.6.118: connected (1/0)
Oct  2 20:03:48 delilah spamd[26905]: 222.240.6.118: disconnected after
0 seconds.
Oct  2 20:06:51 delilah spamd[26905]: 24.71.110.10: connected (1/1),
lists: spews1
Oct  2 20:07:00 delilah spamd[26905]: 221.196.37.249: connected (2/1)
Oct  2 20:07:00 delilah spamd[26905]: 221.196.37.249: disconnected after
0 seconds.
Oct  2 20:07:12 delilah spamd[26905]: 24.71.110.10: disconnected after
21 seconds. lists: spews1
```

De tre første linjene viser at en maskin kobler seg til, som andre aktive forbindelse, med en forbindelse fra en svartlistet maskin. (GREY) og (BLACK) før adressene angir at adressene er henholdsvis grålistet eller svartlistet. Etter 404 sekunder (eller 6 minutter og 44 sekunder) gir den svartlistede maskinen opp uten å fullføre leveringen. Linjene som følger kan være første kontakt noensinne fra en maskin, som så blir grålistet.

Den foreløpige konklusjonen er at spamd stopper mye spam. Dessverre har det dukket opp noen falske positive også, som såvidt jeg vet skyldtes listen spews2 (spews level 2). Vi har foreløpig kuttet ut den listen, uten at spammengden har økt merkbart.

Så kommer det som lenge var høydepunktet i min erfaring med spamd. Min beste loggoppføring var lenge denne:

```
Dec 11 23:57:24 delilah spamd[32048]: 69.6.40.26: connected (1/1),
lists: spamhaus spews1 spews2
Dec 12 00:30:08 delilah spamd[32048]: 69.6.40.26: disconnected
after 1964 seconds. lists: spamhaus spews1 spews2
```

Dette dreier seg om en avsender hos wholesalebandwidth.com. Denne maskinen gjorde 13 forsøk på levering fra 9. til 12. desember 2004. Siste forsøk tok 32 minutter og 44 sekunder, uten levering.

Jeg oppdaterer dette foredraget med noen mellomrom, og nylig fant jeg enda noen oppføringer som overgikk dette:

```
peter@delilah:~$ grep disconnected /var/log/spamd | awk '{print $9}' \
| sort -rn | uniq -c | head
  1 36099
  1 14714
  1 10170
```

1 5495  
1 2193  
1 1964  
1 1872  
1 1718  
1 1677  
1 1617

Den første, på 36099 sekunder, altså mer enn ti timer,

```
Aug 26 10:10:17 delilah spamd[26905]: 146.151.48.74: connected (1/0)
Aug 26 20:11:56 delilah spamd[26905]: 146.151.48.74: disconnected
after 36099 seconds.
```

dreier seg om en maskin som tilsynelatende er på University of Wisconsin, Madison, og som antakelig var infisert av en ekstremt lite intelligent spamsender-orm som rett og slett tok seg svært lang tid på å vente på resten av SMTP-dialogen. De neste oppføringene, med henholdsvis 4 timer, 5 minutter og 14 sekunder og 2 timer, 49 minutter og 30 sekunder, ser ut til å ha oppført seg på samme måte.

## Konklusjoner fra erfaringene med spamd

Oppsummeringen må bli at selektiv bruk av svartelister og spamd er en nokså treffsikker og effektiv spambekjempelsesmetode. Belastningen på maskinen som kjører spamd er minimal. Likel er spamd aldri bedre enn dårligste datakilde, så en viss overvåking og aktiv bruk av hvitlisting kan være nødvendig.

Det er også fullt mulig å kjøre spamd i ren grålistingsmodus, uten svartelister overhodet. Faktisk tyder enkelte rapporter på at spamd i ren gråliste-konfigurasjon ikke ligger mye etter konfigurasjoner med svartelister i effektivitet som spambekjempelse.

I alle fall har vi har endt opp med å bruke Bob Becks 'traplist'<sup>4</sup>, som blir generert ved hjelp av det Bob kaller "the ghosts of USENET postings past", altså spamds 'greytrapping'-funksjon<sup>5</sup> og adresser som ikke forventes å motta legitim epost. En ting som gjør at denne listen skiller seg ut, er at

4. Listen finnes tilgjengelig for bruk på <http://www.openbsd.org/spamd/traplist.gz>.

5. Med tilleggsprogrammet spamdb er det mulig å angi mottakeradresser i domener vi normalt mottar post for, men som ikke forventes å motta legitim post. Maskiner som sender post til disse adressene, blir lagt til svartelisten uavhengig av grålistingsstatus.

Bob har satt opp systemet slik at adressene blir fjernet automatisk etter 24 timer. Det betyr et ekstremt lavt antall falske positive.

# PF - Haiku

Til slutt kan det passe å vise hvilke følelser PF kan inspirere hos brukerne. På epostlisten for PF dukket det i mai 2004 opp en melding med emne "Things pf can't do?" fra en som ikke hadde drevet så mye med brannmurer før, og som forståelig nok syntes at det var vanskelig å få alt til å virke slik det var meningen at det skulle.

Det førte selvfølgelig til litt diskusjon, og flere mente at om PF var vanskelig for en nybegynner, så var i alle fall alternativene ikke bedre. Tråden endte med at Jason Dixon brøt ut i prisnings-haiku, som vi likegodt gjengir slik det kom, med Jasons kommentarer:

Compared to working with iptables, PF is like this haiku:

```
A breath of fresh air,  
floating on white rose petals,  
eating strawberries.
```

Now I'm getting carried away:

```
Hartmeier codes now,  
Henning knows not why it fails,  
fails only for n00b.
```

```
Tables load my lists,  
tarpit for the asshole spammer,  
death to his mail store.
```

```
CARP due to Cisco,  
redundant blessed packets,  
licensed free for me.
```

Jason Dixon, på PF-epostlisten 20. mai 2004  
(<http://marc.theaimsgroup.com/?l=openbsd-pf&m=108507584013046&w=2>)

# Referanser

OpenBSDs web <http://www.openbsd.org/>

OpenBSDs FAQ, <http://www.openbsd.org/faq/index.html>

PF User Guide <http://www.openbsd.org/faq/pf/index.html>

Daniel Hartmeiers PF-sider, <http://www.benedrine.cx/pf.html>

Daniel Hartmeier: Design and Performance of the OpenBSD Stateful Packet Filter (pf), <http://www.benedrine.cx/pf-paper.html> (presentert på Usenix 2002)

Nate Underwood: HOWTO: Transparent Packet Filtering with OpenBSD, <http://ezine.daemonnews.org/200207/transpfobsd.html>

Randal L. Schwartz: Monitoring Net Traffic with OpenBSD's Packet Filter, <http://www.samag.com/documents/s=9053/sam0403j/0403j.htm>

Unix.se: Brandvägg med OpenBSD, [http://unix.se/Brandv%E4gg\\_med\\_OpenBSD](http://unix.se/Brandv%E4gg_med_OpenBSD)

Randal L. Schwartz: Blog for Thu, Jan 29, 2004, <http://use.perl.org/~merlyn/journal/17094>

RFC 1631, "The IP Network Address Translator (NAT)", mai 1994  
<http://www.ietf.org/rfc/rfc1631.txt?number=1631>

RFC 1918, "Address Allocation for Private Internets", februar 1996  
<http://www.ietf.org/rfc/rfc1918.txt?number=1918>

Hjemmeside for PF på FreeBSD, <http://pf4freebsd.love2party.net/>

Peter Postmas PF på NetBSD-sider, <http://nedbsd.nl/~ppostma/pf/>

Marcus Ranum: The Six Dumbest Ideas in Computer Security  
([http://www.ranum.com/security/computer\\_security/editorials/dumb/index.html](http://www.ranum.com/security/computer_security/editorials/dumb/index.html)),  
September 1, 2005

Kjell Jørgen Holes WiFi-kursmaterieell,  
<http://www.kjhole.com/Standards/WiFi/WiFiDownloads.html>, se også  
wifinetnews.com ([http://wifinetnews.com/archives/cat\\_security.html](http://wifinetnews.com/archives/cat_security.html)). I tillegg er The Unofficial 802.11 Security Web Page  
(<http://www.drizzle.com/~aboba/IEEE/>) varmt anbefalt.

Greylisting.org [greylisting.org](http://www.greylisting.org/) (<http://www.greylisting.org/>)

Evan Harris: The Next Step in the Spam Control War: Greylisting  
(<http://greylisting.org/articles/whitepaper.shtml>) (the original greylisting  
paper)

# Finne manuskriptet på web

Dette manuskriptet er *under utvikling*.

Du finner oppdaterte versjoner her:

Flere formater: <http://home.nuug.no/~peter/pf/>

Oppdateringer kommer normalt i forbindelse med arrangementer (konferanser, kurs og liknende) eller når det ellers er naturlig å gjøre endringer. Gi meg beskjed hvis du har lyst til å bli varslet om oppdateringer.

## Hvis du hadde glede av dette: Kjøp OpenBSD-CDer og annet, doner!

Hvis du har hatt glede eller nytte av dette manuskriptet, ber vi deg om at du går til bestillingssiden på OpenBSD.org (<http://www.openbsd.org/orders.html>) og kjøper CD-sett og andre gjenstander fra prosjektet. For den del kan du gjerne også støtte videre utvikling av OpenBSD-prosjektet via endonasjon (<http://www.openbsd.org/donations.html>).

Hvis du har fått med deg dette på en konferanse eller tilsvarende arrangement, er det god mulighet for at det er en OpenBSD-stand i nærheten hvor du kan kjøpe CD-sett, T-skjorter eller andre gjenstander.

Husk på at også fri programvare som du ikke er forpliktet til å betale noen for koster ekte penger å utvikle og vedlikeholde.