Brannmur med OpenBSDs pakkefilter PF

Dette foredraget 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 senere.

SLIDE: PF?

Først litt om programvaren vi skal snakke om, OpenBSDs PF.

PF ble skrevet 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 kjernehacking i nettverkskoden.

Han hadde jobbet ut fra å hekte en liten funksjon han selv hadde skrevet inn i netterksstakken, 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.

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.

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, ogde 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.

SLIDE: 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 i det vesentlige opererer 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’.

SLIDE: 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 fakisk 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 adresssene 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 internett. 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 adresser 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.

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.

SLIDE: 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 3.6, er et pakkefilter som kan gjøre mange ting, hvis man har lyst til det.

For det første så klassifiserer det pakker, på grunnlag av protokoll, port, pakketype, fra- og til-adresse og faktisk med brukbart stor sikkerhet på grunnlag av operativsystem.

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.

SLIDE: 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 xl får navn som `xl0`, `xl1` og så videre, og tilsvarende for Intel-kort som gjerne blir hetende `em0`, `em1`, SMC-kort med sn-ettellerannet og så videre.

BSDene er 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 standarderdiene, 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 lurer, 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.

SLIDE: Enkleste oppsett – enslig maskin (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 Internett.

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

```
pf=YES
```

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
```

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 eller FreeBSD ligger det 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
```

og da er PF aktivert. Foreløpig har vi ikke noe regelsett, så PF gjør ingenting.

SLIDE: Enkleste oppsett – enslig maskin (FreeBSD)

På FreeBSD trenger du litt mer magi i `rc.conf`, spesifikt i følge FreeBSD Handbook

```
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
```

På FreeBSD blir pf i utgangspunktet kompilert som lastbar kjernemodul,så du burde kunne komme godt i gang med `$ sudo kldload pf`, fulgt av `$ sudo pfctl -e` for å aktivere.

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 xterm-vinduet.

Vel, nok om det. Vi skal ha et regelsett. 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

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

SLIDE: Litt strengere

En litt mer komplett og oversiktlig situasjon ved å nekte alt og bare åpne for det vi vet vi trenger. 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 `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 foreløpig skal se slik ut:

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

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
```

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.

SLIDE: 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 dette foredraget:

```
peter@skapet:~$ sudo pfctl -s info
Status: Enabled for 6 days 01:30:14           Debug: Urgent

Hostid: 0x9c6b095b

Interface Stats for xl0                 Ipv4
Ipv6
Bytes In                          431807046                    0
Bytes Out                          40105602                  352
  Packets In
  Passed                             362534                    0
  Blocked                             45033                    0
  Packets Out
  Passed                             285888                    1
  Blocked                                 1                    4

State Table                            Total
Rate
current entries                            7
searches                             1026325              2.0/s
inserts                                26577              0.1/s
removals                               26570              0.1/s
Counters
match                                  48962              0.1/s
bad-offset                                 0              0.0/s
fragment                                  10              0.0/s
short                                     20              0.0/s
normalize                                  0              0.0/s
memory                                     0              0.0/s
bad-timestamp                              0              0.0/s
```

Her ser vi i første linjen at PF er aktivert og har vært oppe i noen dager, sannsynligvis siden siste strømbrudd. `pfctl -s all` gir svært detaljert informasjon. Prøv det og se. `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.

SLIDE: Enkel gateway med NAT

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 det påvirker i liten grad det vi skal snakke om her.

Vi forutsetter at du nå har skrudd inn et nettverkskort til, eller i alle fall sørget for nettverksforbindelse ut fra lokalnettet, 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 ganske fort.

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

```
peter@skapet:~$ sudo sysctl net.inet.ip.forwarding=1
```

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

```
peter@skapet:~$ sudo 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 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"
ipv6_gateway_enable="YES"
```

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.

SLIDE: Enkel gateway med NAT (forts.)

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 `pf.conf` se slik ut:

```
ekstern = "xl0" # makro for ytre gr.snitt — bruk tun0 for PPPoE
intern = "xl1"  # makro for indre gr.snitt

# ekstern IP-adresse tildeles dynamisk
nat on $ekstern from $intern:network to any -> ($ekstern)

block all
pass from { lo, $intern:network } 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 interssant 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 `($ekstern)` 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, \
      https, 446, cvspserver, 2628, cvsup, 8000, 8080 }»
```

og regelen

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

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 sannsyligvis 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 $ekstern 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_services \
     keep state
```

Da har vi tatt med ntp også. Nyttig for oss som vil synkronisere tid. En av de tingene som disse protokollene har til felles, er at de kan finne på å kommunisere over både tcp og udp

SLIDE: 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.

SLIDE: ftp: Må vi så må vi (omdirigering)

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-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 lo:

```
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 og andre BSDer med rcng gjør du det med

```
FreeBSD$ 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 $intern 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 $ext_if inet proto tcp from port ftp-data to ($ext_if) \
      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.

SLIDE: ftp + brannmur + rutbar = ftpsesame

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 deamon 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, så kan du ha godt utbytte av å hente ftpsesame fra Sentia på http://www.sentia.org/projects/ftpsesame/.

Dokumentasjonen består av en man-side med eksempler som du med stor sannsynlighet kan klippe og lime fra direkte.

SLIDE: Hjelp til feilsøking – ping og traceroute

En klar ulempe med regelsettet som det står akkurat nå, er at feilsøkingskommandoene `ping` og `traceroute` ikke vil virke. Det vil sannsynligvis ikke brukerne dine merke noe til, og enkelte windowsmennesker mener visst at `ping` er en farlig kommando, så i alle fall icmp må forbys. Du som er i målgruppen her, vil nok ønske deg feilsøkingsmulighetene, i alle fall. 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 in inet proto icmp all icmp-type $icmp_typer keep state
```

Hvis du trenger flere eller andre typer icmp-pakker, kan du for eksempel utvide `icmp_typer` til en liste som inneholder de typene du vil tillate.

En annen kommando som er veldig nyttig når brukerne sier at internett ikke virker, er `traceroute`. Den bruker 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 $ytre inet proto udp from any to any \
      port 33433 >< 33626 keep state
```

Om andre operativsystemer vil gjøre det annerledes, vil det vise seg i testing, i den grad det er interessant. Denne løsningen stammer i alle fall fra en posting på `openbsd-misc`. Den listen, og da spesielt de søkbare listearkivene, er en veldig verdifull ressurs når du lurer på noe med OpenBSD eller PF.

SLIDE: 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 = "192.168.0.7"
webporter = "{ http, https }"
epostserver = "192.168.0.5"
epost = "{ smtp, pop3, imap, imap3, imaps, pop3s }"

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

  pass in on $ekstern proto tcp from any to $webserver
    port 80 \ flags S/SA synproxy state
  pass in on $ekstern proto tcp from any to $epostserver
    port $epost \ flags S/SA synproxy state
  pass out on $ekstern 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.

SLIDE: 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 med

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

Så kan du for eksempel endre regelen fra tidligere

```
pass out 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
```

Tabellen på disk kan for eksempel vedlikeholdes med en cron-jobb som med jevne mellomrom dumper tabellen til disk. 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.

SLIDE: 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.

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-brukerhåndboken 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 uspennende avkrok av internett, 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: begrens hva du logger, hvis du ikke har satt av nok plass, ett eller annet sted.

SLIDE: 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
udp  In  194.54.107.19:61670  194.54.103.66:53      1:2      36     0     2   219
udp  Out 194.54.107.19:58732  194.54.103.66:53      2:1      36     0     2   219
udp  Out 194.54.107.19:54402  194.54.103.66:53      2:1      36     0     2   255
udp  Out 194.54.107.19:54681  194.54.103.66:53      2:1      36     0     2   271
udp  Out 194.54.107.19:61670  194.54.103.66:53      2:1      36     0     2   219
udp  In  194.54.103.66:50007  194.54.103.65:53      2:2     157    24    26  3216
udp  In  194.54.103.76:138    194.54.103.127:138    0:1      38    22     1   236
```

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`.

SLIDE: Usynlig brannmur med bridge

En bridge er i vår sammenheng en maskin med to eller flere nettverkskort som sitter mellom internett 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 omdirigere 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.

Metoden for konfigurering av bridge skiller seg i noen detaljer mellom operativsystemene. Her er en kort oppskrift for OpenBSD, der man for oversiktens skyld blikkerer for all annen trafikk enn internet-protokoller.

Sette opp broen med to grensesnitt:

/etc/hostname.xl0

```
up
```

/etc/hostname.xl1

```
up
```

/etc/bridgename.bridge0

```
add xl0
add xl1
blocknonip xl0
blocknonip xl1
up
```

/etc/pf.conf

```
ekstern = xl0
intern  = xl1
interessant-trafikk = { ... }

block all
pass quick on $extern all
pass log on $intern 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 omdirigering på ett grensesnitt. Siden alle pakker passerer to ganger, kan reglene bli innviklete.

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.

SLIDE: 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 være 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. Syntaksen er kort

```
altq on grensesnitt køtype [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.

SLIDE: 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 deklarere 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 $ekstern 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, 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.

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

```
pass log quick on $ekstern proto tcp from any to any \
      port 22 flags S/SA keep state \
     queue (ssh_bulk, ssh_interactive)
pass in quick on $ekstern proto tcp from any to any \
      port 20 flags S/SA keep state queue ftp

pass in quick on $ekstern proto tcp from any to any \
     port 80 flags S/SA keep state queue http

pass out on $ekstern proto udp all keep state queue udp
pass out on $ekstern proto icmp all keep state queue icmp
```

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

SLIDE: ALTQ – prioritere etter trafikktype

Vi fortsetter med et annet eksempel, som er rappet 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.

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

SLIDE: 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 }

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 Scwartz, 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.

SLIDE: 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 Ciscos VRRP (Virtual Router Redundancy Protocol) 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.

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 utforske mer over tid, foreløpig kan jeg bare henvise til OpenBSDs FAQ, man-sidene og Ryan McBrides oversiktsartikkel på http://www.countersiege.com/doc/pfsync-carp/

SLIDE: Mobbe spammere med spamd

Når alt dette er tilbakelagt, 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:

```
table <spamd> persist
table <spamd-white> persist file "/var/mail/whitelist.txt"


rdr pass on $ytre inet proto tcp from <spamd> to \
     { $ytre, $indre:network } port smtp -> 127.0.0.1 port 8025
rdr pass on $ytre inet proto tcp from !<spamd-white> to \
     { $ytre, $indre:network } 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.

SLIDE: Mobbe spammere (fortsatt)

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 svartlister ganske fort, og dermed har vi noe å fylle opp den første tabellen med.

Det spamd gjør overfor adresser i svartelisten, er å presentere banneret sitt, og deretter svare på smtp-trafikk med 1 byte om gangen. Den kan også gråliste, dvs midlertidig avvise meldinger og slippe gjennom slike som kommer nytt etter rimelig tid. De veloppdragne slipper gjennom.

Konfigurering av spamd er nokså rett frem. 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 de aktuelle linjene for `spamd` og oppstartsparametrene du ønsker i `/etc/rc.conf` eller `/etc/rc.conf.local`. 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` på samme måte som med andre tabeller.

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 reglene du har fra før bli stående uforandret.

SLIDE: Mobbe spammere (fortsatt)

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 alt med poengsum fra 5 til 9.99 poeng og avvise alt med 10 poeng eller mer og meldinger med virus. I løpet av høsten var `spamassassin` blitt stadig sløvere.

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.

Et typisk loggutsnitt ser slik ut:

```
Jan 26 18:52:32 delilah spamd[30816]: 163.125.69.60:
connected (4/4), lists: spamhaus
Jan 26 18:54:14 delilah spamd[30816]: 163.125.69.60:
disconnected after 102 seconds. Lists: spamhaus

Jan 26 18:55:28 delilah spamd[30816]: 207.182.142.209:
disconnected after 403 seconds. Lists: spamhaus spews1

Jan 26 18:56:03 delilah spamd[30816]: 192.112.102.8:
connected (3/2)

Jan 26 18:56:04 delilah spamd[30816]: 192.112.102.8:
disconnected after 1 seconds.
```

Den første linjen sier at en maskin kobler seg til, som fjerde aktive forbindelse av fire svartlistete. Denne adressen er svartlistet av spamhaus. De to neste linjene viser to som gir opp etter henholdsvis 1 minutt og 42 sekunder og 6 minutter og 43 sekunder, uten å ha fått avsluttet leveringene sine. De to siste linjene kan være første kontakt for en maskin som da blir grålistet. I alle fall viser den nest siste linjen at to svartelistede maskiner fortsatt prøver i det denne kobler seg til.

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.

SLIDE: Mobbe spammere (fortsatt)

Så kommer høydepunktet i min erfaring med `spamd`. Min beste loggoppføring ser slik ut:

```
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.

Ikke fullt så bra som Daniel Hartmeiers 47 minutter, men jeg er alt i alt ganske fornøyd.

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.

SLIDE: PF – Haiku

Til slutt kan det passe å vise hvilke følelser PF kan inspirere hos brukerne. På epostlisten for PF dukket det våren 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://www.benzedrine.cx/pf/msg04702.html

SLIDE: 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.benzedrine.cx/pf.html

Daniel Hartmeier: Design and Performance of the OpenBSD Stateful Packet Filter (pf), http://www.benzedrine.cx/pf-paper.html (presentert på Usenix 2000)

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

Randal L. Schwartz: Blog for Thursday, January 29, 2004,
http://use.perl.org/~merlyn/journal/17094