

# Ściany ogniowe oparte o IP Filter

**Brendan Conoboy**, [synk@swcp.com](mailto:synk@swcp.com)

**Erik Fichtner**, [emf@obfuscation.org](mailto:emf@obfuscation.org)

Wersja oryginalna: Fri Mar 1 22:29:33 EST 2002

Oryginał tego dokumentu znajduje się pod adresem: <http://www.obfuscation.org/ipf/>

**Tłumaczenie: Łukasz Bromirski**, [l.bromirski@mr0vka.eu.org](mailto:l.bromirski@mr0vka.eu.org)

Wersja tłumaczenia: 3.0, 2002/09/03 14:38:15

Oryginał tłumaczenia znajduje się pod adresem: <http://mr0vka.eu.org/docs/tlumaczenia/ipf/index.html>

---

*Dokument ten jest pomyślany jako wprowadzenie dla nowych użytkowników paczki IP Filter tworzącej ścianę ogniową. Jednocześnie, ma nauczyć użytkownika niektórych fundamentalnych zasad projektowania dobrych ścian ogniowych.*

---

## 1. Wprowadzenie

IP Filter to fajna, mała paczka ściany ogniowej. Robi prawie wszystko co inne, darmowe (`ipfwadm`, `ipchains`, `ipfw`), ale oprócz tego jest również przenośna między platformami i potrafi parę ciekawych rzeczy których inne nie robią. Dokument ma na celu zebranie wiedzy ze śladowej dokumentacji dostępnej do tej pory dla `ipfilter`. Co więcej, dokument ten może służyć jako dokumentacja do filtrów pakietów zgodnych na poziomie opcji (w szczególności `pf`) a jeśli to będzie potrzebne wskaże różnice; jest jednak adresowany do specyficznego języka opisu reguł i ich składni, a także nie pisano go pod kątem konkretnej platformy. Wskazana jest podstawowa znajomość zagadnień filtrowania pakietów, choć z drugiej strony jeśli zbyt dobrze się w tym orientujesz, czytanie tego dokumentu jest prawdopodobnie stratą czasu. By lepiej zrozumieć zagadnienia związane ze ścianami ogniowymi, autorzy polecają lekturę *'Building Internet Firewalls'*, autorstwa *Chapman & Zwicky*, wydawnictwa *O'Reilly and Associates*; oraz *'TCP/IP Illustrated, Volume I'*, *Stevens*, *Addison-Wesley*.

### 1.1 Oświadczenie

Autorzy tego dokumentu ( *ani tłumacz! - przyp. tłum.* ) nie są odpowiedzialni za żadne uszkodzenia wynikłe w wyniku podejmowania akcji bazujących na lekturze tego dokumentu. Dokument ten pomyślano jako wprowadzenie do budowy ścian ogniowych opartych o IP-Filter. Jeśli nie czujesz się dobrze biorąc odpowiedzialność za swoje czyny, powinieneś przestać czytać ten dokument i wynająć wykwalifikowany personel by zainstalował dla ciebie ścianę ogniową.

## 1.2 Prawa autorskie

Tam gdzie nie napisano inaczej, prawa autorskie dokumentów HOWTO należą do ich autorów. Dokumenty HOWTO mogą być reprodukowane i dystrybuowane w całości lub w części, na każdym nośniku fizycznym lub elektronicznym, tak długo jak informacje o prawach autorskich zostaną dołączone do każdej kopii. Komercyjna redystrybucja jest również zezwolona i jak najbardziej zachęcamy do niej; jednakże autorzy życzą sobie zostać o takim fakcie poinformowani.

Wszystkie tłumaczenia, prace oparte o ten dokument, lub prace zbiorowe zawierające dowolne HOWTO muszą opierać się na tej samej filozofii praw autorskich. To znaczy, nie możesz pisać prac opartych o HOWTO i narzucać jakieś dodatkowe ograniczenia na jego dystrybucję. Mogą się jednak zdarzyć wyjątki od tych reguł - proszę skontaktować się z koordynatorem HOWTO.

Krótko mówiąc, chcemy promować informacje przekazywane w tym dokumencie przez tyle dróg ile się da. Jednak życzymy sobie zachować prawa autorskie do tego HOWTO, i chcielibyśmy być informowani o jakichkolwiek planach redystrybuowania tego HOWTO.

## 1.3 Skąd uzyskać ważne rzeczy

Oficjalna strona IP Filter znajduje się pod adresem <http://coombs.anu.edu.au/~avalon/ip-filter.html>.

Kompatybilny filtr pakietów na licencji BSD znajduje się pod adresem <http://www.benzedrine.cx/pf.html>.

Najaktualniejszą wersję tego dokumentu można znaleźć pod adresem: <http://www.obfuscation.org/ipf/>

## 2. Podstawy ścian ogniowych

Tą sekcję napisaliśmy tak, by zapoznać się ze składnią poleceń ipfilter, oraz teorią ścian ogniowych w ogólności. Możliwości które tu opisano znajdziesz w każdej dobrej paczce ściany ogniowej. Ta sekcja da Ci solidne podstawy, tak by lektura i zrozumienie sekcji zaawansowanej było bardzo łatwe. Należy podkreślić, że przeczytanie tylko tej sekcji nie wystarczy do zbudowania dobrej ściany ogniowej, a zapoznanie się z sekcją zaawansowaną jest absolutnie wymagana dla każdego kto chce zbudować efektywny system bezpieczeństwa.

### 2.1 Dynamika pliku konfiguracyjnego i kolejność

IPF (Filtr IP) posiada plik konfiguracyjny ( w przeciwieństwie do trybu pracy w której uruchamia się cały czas komendy dla każdej nowej reguły ). Plik konfiguracyjny jest zgodny z filozofią Unixa: każda nowa linia to reguła, znak '#' oznacza komentarz, możesz również wpisać regułę i po niej komentarz w jednej linii. Oczywiście dozwolone są również nadmiarowe spacje, a nawet poleca się je by zestawy reguł był czytelniejszy.

### 2.2 Podstawy przetwarzania reguł

Reguły przetwarzane są z góry na dół, każda dodawana po poprzedniej. Oznacza to po prostu, że jeśli cały Twój plik konfiguracyjny wygląda tak:

```
block in all
pass  in all
```

...komputer widzi go jako:

```
block in all
pass  in all
```

Co oznacza, że po otrzymaniu pakietu, IPF najpierw stosuje regułę:

```
block in all
```

Jeśli IPF uzna, że należy przejść do następnej reguły, zinterpretuje ją:

```
pass  in all
```

W tym momencie możesz zadać sobie pytanie *"a uzna, że należy przejść do następnej reguły?"*. Jeśli znany jest ci `ipfwadm` czy `ipfw`, prawdopodobnie nie zadasz sobie tego pytania. Potem będziesz mocno zdziwiony, dlaczego pakiety są odrzucane lub przepuszczane, podczas gdy wskazałeś inaczej. Wiele filtrów pakietów przestaje porównywać pakiety w momencie, gdy znajdą pierwszą regułę która pasuje. IPF nie jest jednym z nich.

Odmienne, niż w przypadku innych filtrów pakietów, IPF utrzymuje flagę czy przepuścić pakiet czy nie. Dopóki nie przerwiesz porównywania, IPF sprawdzi cały zestaw reguł i podejmie decyzję czy przepuścić pakiet czy nie, na podstawie **ostatniej pasującej reguły**. Wygląda to tak: IPF pracuje. Dostał kawałek czasu procesora. Ma przed sobą listę do sprawdzenia, która wygląda tak:

```
block in all
pass  in all
```

Do interfejsu dociera pakiet i trzeba zabrać się do pracy. Pobiera pakiet i sprawdza pierwszą regułę:

```
block in all
```

IPF stwierdza "Jak na razie, zablokuję ten pakiet". Następnie ogląda drugą regułę:

```
pass  in all
```

"Jak na razie, wpuszczę ten pakiet", stwierdza IPF. Potem patrzy na trzecią regułę. Nie ma jej, więc sprawdza jaką decyzję podjął ostatnio - przepuścić pakiet.

W tym momencie nadszedł dobry moment by zauważyć, że nawet gdyby zestaw reguł wyglądał tak:

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

```
block in all
block in all
pass in all
```

To i tak pakiet zostałby przepuszczony. Nie istnieje tutaj coś takiego jak efekt kumulacyjny. Ostatnia pasująca reguła zawsze decyduje o losie pakietu.

## 2.3 Kontrolowanie przetwarzania reguł

Jeśli miałeś już do czynienia z innymi filtrami pakietów, możesz stwierdzić że ten sposób organizacji przetwarzania jest mylący, możesz również spekulować że istnieją problemy z przenoszalnością do innych filtrów oraz, że prędkość przetwarzania reguł może być mała. Wyobraź sobie że masz 100 reguł i wszystkie pasujące były pierwszymi 10. Dla każdego pakietu sprawdzenie pozostałych reguł byłoby wielką stratą czasu. Na szczęście, istnieje proste słowo kluczowe które możesz dodać do reguły by od razu spowodować reakcję. Tym słowem jest `quick`.

Poniżej przedstawiono zmodyfikowaną wersję oryginalnego zestawu, tym razem z nową komendą.

```
block in quick all
pass in      all
```

W tym przypadku, IPF sprawdza pierwszą regułę:

```
block in quick all
```

Pakiet pasuje i przeglądanie reguł na nim się kończy. Pakiet zostaje odrzucony bez żadnego piśnięcia. Nie ma żadnych komunikatów, logów, konduktu pogrzebowego. Ciasto nie zostanie podane. Co więc z następną regułą?

```
pass in      all
```

Do tej reguły IPF nigdy nie dociera. Mogłoby jej w ogóle nie być w pliku konfiguracyjnym. Działanie reguły `all` i słowo `quick` w poprzedniej regule powoduje, że nie są już sprawdzane żadne inne reguły.

Sytuacja w której połowa pliku konfiguracyjnego do niczego się nie przydaje, jest raczej stanem niepożądanym. Z drugiej strony, IPF ma za zadanie powstrzymywać pakiety tak jak został skonfigurowany, i robi bardzo dobrą robotę. Tak czy inaczej, IPF jest również po to by niektóre pakiety przepuszczać, więc wymagana jest pewna zmiana reguł by to zadanie zrealizować.

## 2.4 Podstawy filtrowania po adresie IP

IPF może sprawdzać pakiety pod kątem wielu kryteriów. Jednym z tych o których myślimy najczęściej jest adres IP. Istnieją pewne zakresy przestrzeni adresowej z których nigdy nie powinniśmy otrzymywać żadnych pakietów. Jednym z takich zakresów jest sieć nieroutowalna, `192.168.0.0/16` ( /16 to zapis maski w postaci CIDR. Możesz być bardziej przyzwyczajony do zapisu decymalnego, `255.255.0.0` - IPF akceptuje obydwa ). Jeśli chciałbyś zablokować `192.168.0.0/16`, jednym ze sposobów jest:

```
block in quick from 192.168.0.0/16 to any
pass  in      all
```

Tym razem mamy w końcu zestaw reguł który coś dla nas robi. Wyobraźmy sobie, że dociera do nas pakiet z adresu 1.2.3.4. Sprawdzana jest pierwsza reguła:

```
block in quick from 192.168.0.0/16 to any
```

Pakiet przyszedł z adresu 1.2.3.4 a nie z 192.168.\*.\*, więc reguła nie pasuje. Sprawdzana jest druga reguła:

```
pass  in      all
```

Pakiet przyszedł z adresu 1.2.3.4, który zdecydowanie należy do all ( czyli dowolnego adresu ), więc jest wysyłany tam gdzie chciałby dotrzeć.

Z drugiej strony, przypuśćmy że otrzymaliśmy pakiet z adresu 192.168.1.2. Sprawdzana jest pierwsza reguła:

```
block in quick from 192.168.0.0/16 to any
```

Pakiet pasuje, więc jest odrzucany i to koniec. Ponownie, IPF nie sprawdza drugiej reguły, ponieważ pierwsza reguła która pasowała, zawierała słowo quick.

W tym momencie możesz zbudować rozszerzalny zestaw adresów, z których niektóre należy zablokować a niektóre przepuścić. Ponieważ już zaczęliśmy blokować zakresy adresów prywatnych na naszej ścianie ogniowej, zadbajmy o resztę:

```
block in quick from 192.168.0.0/16 to any
block in quick from 172.16.0.0/12 to any
block in quick from 10.0.0.0/8 to any
pass  in      all
```

Pierwsze trzy reguły blokują niektóre z zakresów adresów prywatnych.

## 2.5 Kontrola interfejsów

Często zdarza się, że firmy mają najpierw sieć wewnętrzną, zanim zechcą podłączyć się do świata zewnętrznego. Tak naprawdę, sensowne wydaje się założenie, że to główny powód dla którego ludzie w ogóle rozważają ściany ogniowe. Maszyna, która pełni rolę mostu ( ang. *bridge* ) między siecią wewnętrzną a siecią zewnętrzną jest routerem. Router od każdej innej dowolnej maszyny różni jedna podstawowa cecha: ma więcej niż jeden interfejs.

Każdy pakiet który otrzymujesz, przychodzi którymś interfejsem sieciowym; każdy pakiet który wysyłasz wychodzi również interfejsem sieciowym. Powiedzmy że masz trzy interfejsy: pętlę zwrotną - lo0 ( ang. *loopback* ), x10 ( kartę ethernetową 3COM ) i tun0 ( podstawowy tunel we FreeBSD którego używa PPP ), ale nie chcesz otrzymywać pakietów przychodzących z interfejsu tun0?

```
block in quick on tun0 all
pass  in      all
```

W tym przypadku słowo `on` oznacza identyfikację danych przybywających wskazanym interfejsem. Jeśli pakiet przychodzi do interfejsu `tun0` (`'on tun0'`), pierwsza reguła go zablokuje. Jeśli pakiet przyjdzie do interfejsu `lo0` lub `x10`, pierwsza reguła nie będzie pasowała, a druga tak i pakiet zostanie przepuszczony.

## 2.6 Jednoczesne użycie adresu IP i nazwy interfejsu

To dziwny stan, w którym decydujesz, że chcesz mieć interfejs podniesiony ( w naszym przypadku `tun0` ), ale nie chcesz otrzymywać przez niego pakietów. Czym więcej jest kryteriów które sprawdza ściana ogniowa, tym jest bardziej szczelna ( lub przeciekająca ). Może chcesz otrzymywać dane przez `tun0`, ale nie od `192.168.0.0/16`? To początek potężnej ściany ogniowej.

```
block in quick on tun0 from 192.168.0.0/16 to any
pass  in                all
```

Porównaj to do naszego poprzedniego zestawu reguł:

```
block in quick from 192.168.0.0/16 to any
pass  in                all
```

Blokujemy w nim każdy ruch pochodzący z `192.168.0.0/16`, niezależnie od interfejsu. W nowym zestawie reguł, w którym używamy słów `'on tun0'` blokujemy tylko pakiety, które dotarły przez interfejs `tun0`. Gdyby pakiet przybył interfejsem `x10` zostałby wpuszczony.

W tym momencie możesz zbudować rozszerzalny zestaw adresów, z których niektóre należy zablokować a niektóre przepuścić. Ponieważ już zaczęliśmy blokować zakresy adresów prywatnych które docierają do interfejsu `tun0`, zajmijmy się resztą:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 0.0.0.0/8 to any
block in quick on tun0 from 169.254.0.0/16 to any
block in quick on tun0 from 192.0.2.0/24 to any
block in quick on tun0 from 204.152.64.0/23 to any
block in quick on tun0 from 224.0.0.0/3 to any
pass  in                all
```

Widziałeś już pierwsze trzy reguły, ale nie resztę. Czwarta wskazuje klasę A, w większości zmarnowaną, a używaną głównie na pętle zwrotne. Wiele oprogramowania komunikuje się ze sobą przez adres `127.0.0.1`, więc zablokowanie tego adresu przy połączeniach z zewnątrz to też dobry pomysł. Piąta linia, `0.0.0.0/8` nigdy nie powinna znaleźć się w Internecie. Większość stosów IP traktuje `0.0.0.0/32` jako domyślną bramę, a reszta sieci `0.*.*.*` jest traktowana na różne dziwne sposoby, co wynika ze sposobu w jaki podejmowane są decyzje o routingu. Powinieneś traktować `0.0.0.0/8` tak jak `127.0.0.0/8`. `169.254.0.0/16` zostało przydzielone przez IANA do użytku w procesie auto-konfiguracji, kiedy system nie otrzymał jeszcze adresu IP z serwera DHCP lub podobnego. Należy zwrócić uwagę, że w szczególności Microsoft Windows będą używać adresów z tego zasięgu gdy ustawione są na używanie DHCP a nie były w stanie znaleźć do tej pory serwera DHCP. `192.0.2.0/24` został również zarezerwowany dla użytku autorów dokumentacji jako przykład dzielenia na bloki. Celowo nie używamy tego zakresu, ponieważ mógłby on spowodować zamieszanie gdybyś je zablokował; wszystkie nasze przykłady używają adresów `20.20.20.0/24`, `204.152.64.0/23` to blok zarezerwowany przez Sun Microsystems

dla prywatnych połączeń klastrów, i zablokowanie go pozostawiamy tobie pod rozważę. Na koniec, 224.0.0.0/3 wycina klasy D i E sieci, która używana jest głównie do ruchu multicastowego ( rozgłaszania ). Dokładniejsze definicje `klasy E' możecie znaleźć w RFC 1166.

Istnieje bardzo ważna zasada w filtrowaniu pakietów, która była odraczana do momentu omówienia blokowania sieci i brzmi ona: w momencie gdy wiesz, że określony typ danych dociera z określonych miejsc, konfigurujesz system by zezwolić tylko na ruch tego typu danych z tych określonych źródeł. W przypadku klasy nieroutowalnej, wiesz że nic z 10.0.0.0/8 nie powinno docierać do ciebie przez tun0, ponieważ nie masz żadnego sposobu by na niego odpowiedzieć. Jest to pakiet nielegalny. Tak samo należy traktować inne nieroutowalne adresy, jak również 127.0.0.0/8.

Wiele oprogramowania, dokonuje uwierzytelniania na podstawie źródłowego adresu IP. Jeśli posiadasz sieć wewnętrzną, powiedzmy 20.20.20.0/24, wiesz że cały ruch dla tej sieci może wychodzić przez lokalny ethernet. Gdyby pakiet z 20.20.20.0/24 dotarł przez połączenie PPP, jest absolutnie sensownym rzucić go na podłogę, albo umieścić w ciemnym pokoju przesłuchań. Nie powinien w żaden sposób móc osiągnąć swojego celu. Możesz to osiągnąć stosując to co już wiesz o IPF. Nowy zestaw reguł wyglądać będzie tak:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 0.0.0.0/8 to any
block in quick on tun0 from 169.254.0.0/16 to any
block in quick on tun0 from 192.0.2.0/24 to any
block in quick on tun0 from 204.152.64.0/23 to any
block in quick on tun0 from 224.0.0.0/3 to any
block in quick on tun0 from 20.20.20.0/24 to any
pass in      all
```

## 2.7 Filtrowanie dwukierunkowe; Słowo kluczowe `out'

Do tej pory przepuszczaliśmy lub blokowaliśmy ruch przychodzący. By wyjaśnić, ruch przychodzący to cały ruch, który dociera do ściany ogniowej na dowolnym interfejsie. Analogicznie, ruch wychodzący to cały ruch, który ma zamiar opuścić interfejs ściany ogniowej ( obojętnie czy wygenerowany lokalnie czy tylko przekazywany ). Oznacza to, że wszystkie pakiety są filtrowane nie tylko gdy docierają do ściany ogniowej, ale również w momencie jej opuszczania. W związku z tym implikuje to komendę `pass out all', która może lub może nie być pożądana. Tak samo jak możesz przepuszczać lub blokować ruch wchodzący, możesz robić to samo z ruchem wychodzącym.

Teraz gdy wiemy że istnieje sposób by filtrować zarówno ruch wychodzący jak i wchodzący, sami musimy znaleźć sensowne zastosowanie dla czegoś takiego. Jednym z możliwych pomysłów, jest powstrzymywanie **sfalszowanych** ( ang. *spoofed* ) pakietów przed wchodzeniem do Twojej sieci. Zamiast wypuszczać na ruterze cały ruch, ograniczymy go tylko do pakietów pochodzących z 20.20.20.0/24. Możesz to zrobić w ten sposób:

```
pass out quick on tun0 from 20.20.20.0/24 to any
block out quick on tun0 from any to any
```

Jeśli pakiet przyjdzie z 20.20.20.1/32, zostanie przepuszczony przez pierwszą regułę. Jeśli pakiet przyjdzie z 1.2.3.4/32, zostanie zablokowany przez regułę drugą.

Możesz również wykonać podobne reguły dla adresów nieroutowalnych. Jeśli jakaś maszyna próbuje skierować pakiet przez IPF do 192.168.0.0/16, dlaczego by go nie odrzucić? Najgorsze co może się stać to to, że zaoszczędzisz trochę przepustowości:

```
block out quick on tun0 from any to 192.168.0.0/16
block out quick on tun0 from any to 172.16.0.0/12
block out quick on tun0 from any to 10.0.0.0/8
block out quick on tun0 from any to 0.0.0.0/8
block out quick on tun0 from any to 127.0.0.0/8
block out quick on tun0 from any to 169.254.0.0/16
block out quick on tun0 from any to 192.0.2.0/24
block out quick on tun0 from any to 204.152.64.0/23
block out quick on tun0 from any to 224.0.0.0/3
block out quick on tun0 from !20.20.20.0/24 to any
```

Z najbardziej ograniczonego punktu widzenia, zapis ten nie rozszerza Twojego bezpieczeństwa. Rozszerza natomiast bezpieczeństwo wszystkich innych i jest generalnie miłą rzeczą do zrobienia. Z drugiej strony, ktoś może stwierdzić, że skoro nie może rozsyłać sfałszowanych pakietów przez Twoją sieć, masz mniejsze znaczenie jako punkt przekaźnikowy dla cracker'ów i w związku z tym prawdopodobieństwo, że staniesz się celem ataku maleje.

Prawdopodobnie znajdziesz wiele sposobów użycia blokowania pakietów wychodzących. Jedną z rzeczy o których zawsze należy pamiętać, to fakt, że 'in' i 'out' są kierunkami odnoszącymi się do ściany ogniowej, nigdy w stosunku do dowolnej innej maszyny.

## 2.8 Logowanie tego co się dzieje; Słowo kluczowe 'log'

Do tego momentu, całe blokowanie i przepuszczanie pakietów odbywało się w całkowitej ciszy. Zwykle chcesz jednak wiedzieć, że jesteś atakowany, a nie zastanawiać się czy ta ściana ogniowa w ogóle Ci coś daje. Podczas gdy nie logowałbym każdego pakietu który został przepuszczony i w niektórych przypadkach wszystkich blokowanych pakietów, chciałbym wiedzieć parę rzeczy o blokowanych pakietach z 20.20.20.0/24. By to wykonać, dodajemy słowo kluczowe 'log':

```
block in      quick on tun0 from 192.168.0.0/16 to any
block in      quick on tun0 from 172.16.0.0/12 to any
block in      quick on tun0 from 10.0.0.0/8 to any
block in      quick on tun0 from 127.0.0.0/8 to any
block in      quick on tun0 from 0.0.0.0/8 to any
block in      quick on tun0 from 169.254.0.0/16 to any
block in      quick on tun0 from 192.0.2.0/24 to any
block in      quick on tun0 from 204.152.64.0/23 to any
block in      quick on tun0 from 224.0.0.0/3 to any
block in log   quick on tun0 from 20.20.20.0/24 to any
pass in       all
```

Do tej pory ściana ogniowa robi dobrą robotę blokując pakiety nadchodzące z podejrzanych miejsc, ale jest jeszcze trochę do zrobienia. Jedną z rzeczy o którą powinniśmy zadbać, jest by pakiety do 20.20.20.0/32 i 20.20.20.255/32 były zrzucane na podłogę. Jeśli tego nie zrobimy, otwieramy naszą sieć na atak typu smurf. Te dwie linie zabezpieczą naszą hipotetyczną sieć przed użyciem jako przekaźnik smurf:

```
block in log quick on tun0 from any to 20.20.20.0/32
```



```
block in log quick on tun0 from any to 20.20.20.255/32
```

Dodanie tych linijek, doprowadza nas do zestawu reguł wyglądającego mniej więcej tak:

```
block in      quick on tun0 from 192.168.0.0/16 to any
block in      quick on tun0 from 172.16.0.0/12 to any
block in      quick on tun0 from 10.0.0.0/8 to any
block in      quick on tun0 from 127.0.0.0/8 to any
block in      quick on tun0 from 0.0.0.0/8 to any
block in      quick on tun0 from 169.254.0.0/16 to any
block in      quick on tun0 from 192.0.2.0/24 to any
block in      quick on tun0 from 204.152.64.0/23 to any
block in      quick on tun0 from 224.0.0.0/3 to any
block in log   quick on tun0 from 20.20.20.0/24 to any
block in log   quick on tun0 from any to 20.20.20.0/32
block in log   quick on tun0 from any to 20.20.20.255/32
pass in       all
```

## 2.9 Kompletnie filtrowanie dwukierunkowe według interfejsu

Do tej pory przedstawialiśmy jedynie fragmenty kompletnego zestawu reguł. W momencie gdy tworzysz swój zestaw, powinieneś utworzyć reguły dla każdego kierunku i interfejsu. Domyślnie ipfilter przepuszcza wszystkie pakiety. Jest to sytuacja analogiczna do tej, w której istnieje niewidoczna reguła na początku która brzmi `pass in all` i `pass out all`. Zamiast polegać na domyślnym zachowaniu, zadbaj by wszystko było tak dokładne i konkretne jak to możliwe, interfejs po interfejsie, do momentu w którym każda ewentualność zostanie rozpatrzona.

Zacznijmy od interfejsu `lo0`, który będzie pracował bez ograniczeń. Ponieważ istnieją programy rozmawiające z innymi na systemach lokalnych, zezwalamy na to i utrzymujemy ten stan bez żadnych restrykcji:

```
pass out quick on lo0
pass in  quick on lo0
```

Następny jest interfejs `x10`. Później będziemy nakładać ograniczenia na interfejs `x10`, ale na początek zacznijmy tak jakby wszystko w naszej sieci lokalnej było warte zaufania i damy interfejsowi dokładnie to samo co w przypadku `lo0`:

```
pass out quick on x10
pass in  quick on x10
```

Na koniec, jest również interfejs `tun0`, który do tej pory filtrowaliśmy tylko połowicznie:

```
block out quick on tun0 from any to 192.168.0.0/16
block out quick on tun0 from any to 172.16.0.0/12
block out quick on tun0 from any to 127.0.0.0/8
block out quick on tun0 from any to 10.0.0.0/8
```

```

block out quick on tun0 from any to 0.0.0.0/8
block out quick on tun0 from any to 169.254.0.0/16
block out quick on tun0 from any to 192.0.2.0/24
block out quick on tun0 from any to 204.152.64.0/23
block out quick on tun0 from any to 224.0.0.0/3
pass out quick on tun0 from 20.20.20.0/24 to any
block out quick on tun0 from any to any

block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 0.0.0.0/8 to any
block in quick on tun0 from 169.254.0.0/16 to any
block in quick on tun0 from 192.0.2.0/24 to any
block in quick on tun0 from 204.152.64.0/23 to any
block in quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass in all

```

Mamy już dosyć dużo filtrowania, zabezpieczamy sieć 20.20.20.0/24 przed fałszowaniem pakietów i przed używaniem do fałszowania pakietów. Kolejne przykłady będą oparte na jednostronnym podejściu, ale miej na uwadze że to tylko dla jasności, a kiedy będziesz konfigurował swój własny zestaw reguł, musisz dodawać reguły dla każdego kierunku i interfejsu.

## 2.10 Kontrolowanie konkretnych protokołów; Słowo kluczowe `proto`

Ataki **Odmowy Usługi** ( ang. *Denial of Service* lub *DoS* ) są równie częste co exploity związane z **przepelnieniem bufora** ( ang. *buffer overflow* ). Wiele ataków DoS związanych jest z zawłościami stosu TCP/IP systemu operacyjnego. Często, sprowadzało się to do pakietów ICMP. Dlaczego nie zablokować ich w ogóle?

```
block in log quick on tun0 proto icmp from any to any
```

W tym momencie każdy pakiet ICMP nadchodzący przez tun0 będzie logowany i odrzucany.

## 2.11 Filtrowanie ICMP z użyciem słowa kluczowego `icmp-type`; Łączenie zestawów reguł

Oczywiście, odrzucanie całego ruchu ICMP nie jest idealną sytuacją. Dlaczego nie? Ponieważ lepiej i użyteczniej jest, gdy na ruch pakietów tego protokołu zezwalamy przynajmniej po części. Zapewne zatem będziesz chciał przepuszczać pewne rodzaje ruchu ICMP a odrzucać inne. Jeśli chcesz by działały traceroute i ping, musisz przepuszczać pakiety ICMP typu 0 i 11. Dokładnie rzecz biorąc, nie jest to dobry pomysł, ale jeśli potrzebujesz wyważyć bezpieczeństwo z jednej strony i wygodę z drugiej, IPF pozwoli ci to zrobić:

```
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
```

```
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
```

Pamiętaj, że kolejność w zestawie reguł jest ważna. Ponieważ każda z reguł ma słowo `quick`, musimy umieścić reguły przepuszczające ( **pass** ) przed blokującymi ( **block** ), więc tak naprawdę ostatnie trzy reguły powinny się znaleźć w pliku konfiguracyjnym w tej kolejności:

```
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
```

Dodanie tych trzech reguł do tych, które zabezpieczają przed fałszowaniem pakietów może być trochę kłopotliwe. Jednym z błędów może być włączenie nowych reguł ICMP na początku:

```
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 0.0.0.0/8 to any
block in quick on tun0 from 169.254.0.0/16 to any
block in quick on tun0 from 192.0.2.0/24 to any
block in quick on tun0 from 204.152.64.0/23 to any
block in quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass in all
```

Problem polega na tym, że pakiet ICMP typu 0 z 192.168.0.0/16 zostanie przepuszczony przez pierwszą regułę i nie zostanie zablokowany przez regułę czwartą. Również, ponieważ używamy ICMP ECHO\_REPLY (typ 0) by przepuścić pakiety do 20.20.20.0/24, z dołączonym słowem `quick`, otworzyliśmy się właśnie z powrotem na atak typu smurf, negując ostatnie dwie reguły blokujące. Ups. By temu zapobiec, ustawimy reguły dotyczące ICMP po regułach zabezpieczających przez fałszowaniem pakietów:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 0.0.0.0/8 to any
block in quick on tun0 from 169.254.0.0/16 to any
block in quick on tun0 from 192.0.2.0/24 to any
block in quick on tun0 from 204.152.64.0/23 to any
block in quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
```

```
pass in      quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in      quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
pass in      all
```

Ponieważ blokujemy ruch sfalszowany zanim zajmujemy się pakietami typu ICMP, pakiety sfalszowane nigdy nie docierają do zestawu reguł ICMP. Bardzo ważne jest pamiętanie o takich rzeczach podczas łączenia zestawów reguł.

## 2.12 Porty TCP i UDP; Słowo kluczowe `port`

Ponieważ zaczęliśmy już blokować pakiety na podstawie protokołu, możemy również zacząć blokować pakiety na podstawie specyficznych cech każdego z nich. Najczęściej używa się numeru portu. Usługi takie jak `rsh`, `rlogin` i `telnet` są bardzo przydatne, ale również bardzo niebezpieczne jeśli chodzi o **podsluchiwanie** ( ang. *sniffing* ) i fałszowanie. Można oczywiście pójść na kompromis i zezwolić na używanie tych usług w sieci wewnętrznej a zablokować przy wychodzeniu na zewnątrz. Można to osiągnąć w prosty sposób, ponieważ `rlogin`, `rsh` i `telnet` używają określonych portów TCP (odpowiednio 513, 514 i 23). W związku z tym stworzenie reguł, które te usługi zablokują jest proste:

```
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 513
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 514
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 23
```

Upewnij się, że wszystkie trzy znajdują się przed regułą `'pass in all'`, dzięki czemu zamkną sieć od zewnątrz ( część reguł odpowiadająca za ochronę przed fałszowaniem pomijam dla jasności ):

```
pass in      quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in      quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 513
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 514
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 23
pass in      all
```

Możesz również chcieć zablokować porty 514/udp ( `syslog` ), 111/tcp i 111/udp ( `portmap` ), 515/tcp ( `lpd` ), 2049/tcp i 2049/udp ( `NFS` ), 6000/tcp ( `X11` ) i tak dalej. Możesz uzyskać pełną listę portów na których aktualnie nasłuchujesz używając polecenia `netstat -a` ( lub `lsof -i`, jeśli masz go zainstalowanego ).

Blokowanie UDP zamiast TCP sprowadza się do zastąpienia `'proto tcp'` przez `'proto udp'`. Reguła dla `syslog'a` wyglądałaby następująco:

```
block in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 514
```

IPF ma również skrótowy sposób zapisu w przypadku gdy chodzi o `'proto tcp'` jak i `proto udp` jednocześnie, tak jak w przypadku `portmap` i `NFS`. Reguła dla `portmap'a` wyglądałaby tak:

```
block in log quick on tun0 proto tcp/udp from any to 20.20.20.0/24 port = 111
```

### 3. Wprowadzenie do zaawansowanych ścian ogniowych

Ta sekcja została napisana w ten sposób, by przeczytać ją bezpośrednio po poprzedniej części. Poniżej zawarto zarówno koncepcje projektowania zaawansowanych ścian ogniowych, jak i zaawansowane możliwości zawarte w programie ipfilter. W momencie gdy ta sekcja będzie ci doskonale znana, powinieneś być w stanie zbudować bardzo silną ścianę ogniową.

#### 3.1 Gwałtowna paranoja lub polityka Domyślnego Blokowania ( ang. *Default-Deny* )

Istnieje pewien poważny problem gdy blokujemy usługi na podstawie portów: czasami się one przesuwają . Programy które bazują na RPC są w tym naprawdę okropne - lockd, statd, nawet nfsd słucha na portach innych niż 2049. Jest bardzo trudno przewidzieć, a nawet gorzej zautomatyzować proces dostrajania się w kółko i na okrągło. A co jeśli zapomnisz o usłudze? Zamiast zmagać się z bałaganem, zacznijmy od stanu zupełnie czystego. Aktualny zestaw reguł wygląda tak:

Tak, naprawdę zaczynamy od nowa. Pierwszą regułą której użyjemy będzie:

```
block in all
```

Nie przechodzi żaden ruch sieciowy. Żaden. Nawet ty-ci-tyci. Jesteś w tym momencie raczej bezpieczny. Konfiguracja użyteczna, ale bezpieczna. Najlepsze w tym wszystkim to to, że niewiele musisz teraz zrobić by nadal pozostać bezpiecznym, ale stać się też troszkę użytecznym. Powiedzmy że maszyna pracuje jako serwer WWW, nic więcej, nic mniej. Nie wykonuje nawet zapytań DNS. Chce tylko odbierać połączenia na port 80/tcp i to wszystko. Możemy to zrobić. Wykonamy to dokładając drugą regułę, którą już znasz:

```
block in          on tun0 all
pass  in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 80
```

Maszyna przyjmie ruch na port 80 dla 20.20.20.1 i odrzuci wszystko inne. Dla podstawowych zastosowań ścian ogniowych to wszystko co potrzeba.

#### 3.2 Zezwolenie na ruch wynikające z innych reguł; reguła `keep state`

Zadaniem twojej ściany ogniowej jest zabezpieczenie przed niechcianym ruchem z punktu B do punktu A. Mamy generalne reguły które mówią *`jeśli tylko ten pakiet jest do portu 23, to go puszczamy`*. Mamy generalne reguły mówiące *`jeśli tylko ten pakiet ma flagę FIN ustawioną, to go puszczamy`*. Nasze ściany ogniowe nie znają początku, środka ani końca sesji TCP/UDP/ICMP. Mają tylko reguły które sprawdzają w stosunku do wszystkich pakietów. Musimy mieć nadzieję, że pakiet który ma flagę FIN ustawioną nie jest tak naprawdę skanem FIN, sprawdzającym nasze usługi. Mamy nadzieję że pakiet do portu 23 nie jest próbą przechwycenia naszej sesji telnetowej. A co jeśli byłaby szansa na zidentyfikowanie i zautoryzowanie poszczególnych sesji TCP/UDP/ICMP i rozróżnić te które są skanami portów czy też atakami DoS? Jest taki sposób, i nazywa się **utrzymywaniem stanu** ( ang. *keep state* ).

Chcemy wygody i bezpieczeństwa w jednym. Wielu ludzi również i dlatego Cisco ma klauzulę `established` ( nawiązane ) i pozwala przejść nawiązanym sesjom TCP. IPFW też ma również `established`, IPFWADM ma `setup/established` ( konfiguruje/nawiązuje ). Wszystkie mają tą opcję, ale nazwa jest bardzo myląca. Kiedy ją pierwszy raz zobaczyliśmy, myśleliśmy że nasz filtr pakietów śledzi każdą sesję i sprawdza co się w niej dzieje, że wie czy połączenie naprawdę jest nawiązane czy nie. Tak naprawdę, wszystkie wierzą pakietowi że jest tym czym twierdzi że jest, a każdy może przecież kłamać. Czytają sekcję flag nagłówka pakietu TCP i tu pojawia się problem bo nie mają opcji podobnego analizowania pakietów UDP/ICMP. Każdy kto potrafi spreparować nagłówki pakietów może pokonać taką ścianę ogniową.

No to co takiego szczególnego robi IPF, możesz zapytać? Cóż, inaczej niż w innych ścianach ogniowych, IPF naprawdę potrafi śledzić połączenia i stwierdzić czy połączenie jest nawiązane czy nie. I robi to zarówno dla pakietów TCP, UDP i ICMP, nie tylko TCP. IPF nazywa to właśnie **utrzymywaniem stanu**. Słowo kluczowe do zastosowania w regule brzmi `keep state`.

Do tej pory, mówiliśmy że pakiety przychodzą, zestaw reguł zostaje sprawdzony, pakiety wychodzą i znowu sprawdzany jest zestaw reguł. Dokładniej rzecz biorąc, to co się dzieje wygląda tak: pakiety przychodzą, sprawdzana jest tabela stanów, potem *być może* sprawdzany jest zestaw reguł dotyczących połączeń przychodzących, pakiety wychodzą, sprawdzana jest tabela stanów, i znów *być może* sprawdzany jest zestaw reguł dotyczących połączeń wychodzących. Tabela stanów to lista sesji TCP/UDMP/ICMP które są przepuszczane bez pytania przez ścianę ogniową, pomijając cały zestaw reguł. Brzmi jak poważna dziura w bezpieczeństwie? Poczekaj, to najwspanialsza rzecz która mogła przytrafić się twojej ścianie ogniowej.

Wszystkie sesje TCP/IP mają początek, środek i koniec ( aczkolwiek czasami jest nimi ten sam, jeden pakiet ). Nie możesz mieć końca bez środka, a środka bez początku. To oznacza, że wszystko co tak naprawdę potrzebujesz filtrować to początek sesji TCP/UDP/ICMP. Jeśli początek sesji ma prawo przejść przez ścianę ogniową, cała reszta ( środek i koniec ) również. Utrzymywanie stanu umożliwia Ci zignorowanie środka i końca, a skupienie się na blokowaniu/przepuszczaniu nowych sesji. Jeśli nowa sesja jest przepuszczana, wszystkie pakiety należące do niej również zostaną przepuszczone. Jeśli ma zostać zablokowana, żaden z pakietów który ma do niej należeć nie zostanie przepuszczony. Poniżej przykład dla pracy z serwerem ssh (i nic poza serwerem ssh):

```
block out quick on tun0 all
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 22 keep state
```

Pierwszą rzeczą którą możesz zauważyć, to brak komendy `'pass out'`. W rzeczywistości, jest tylko jedna, zawierająca wszystko reguła `block out`. Pomimo tego, zestaw reguł jest kompletny. Dzieje się tak, ponieważ poprzez utrzymywanie stanu tworzony jest cały zestaw reguł. W momencie, w którym pierwszy pakiet SYN dociera do serwera, tworzona jest pozycja w tabeli stanu i reszta sesji ssh jest również przepuszczana bez żadnej interferencji ze strony ściany ogniowej. Poniżej kolejny przykład:

```
block in quick on tun0 all
pass out quick on tun0 proto tcp from 20.20.20.1/32 to any keep state
```

W tym przypadku, serwer nie serwuje żadnych usług. Tak naprawdę, nie jest serwerem a klientem. I ten klient nie chce by żadne nieautoryzowane pakiety docierały do jego stosu IP. Jednakże, chce mieć pełen dostęp do internetu i naturalnie potrzebuje możliwości odpowiadania na pakiety które należą do połączeń przez niego inicjowanych. Ten prosty zestaw reguł tworzy listę stanów dla każdej nowej wychodzącej sesji TCP. I znowu, ponieważ tworzona jest nowa pozycja w liście stanów, nowe sesje TCP mają swobodę w komunikowaniu się tam i z powrotem tak jak chcą, bez niepotrzebnego zainteresowania ze strony ściany ogniowej. Wspomnieliśmy również, że działa to również dla UDP i ICMP:

```
block in quick on tun0 all
pass out quick on tun0 proto tcp from 20.20.20.1/32 to any keep state
pass out quick on tun0 proto udp from 20.20.20.1/32 to any keep state
pass out quick on tun0 proto icmp from 20.20.20.1/32 to any keep state
```

Tak Wirginio, możemy pingować. Teraz utrzymujemy stany połączeń TCP, UDP, ICMP. Możemy wykonywać połączenia wychodzące tak jakby nie było żadnej ściany ogniowej, a jednocześnie wszyscy hipotetyczni atakujący nie mogą wejść z powrotem. Jest to bardzo wygodne bo nie ma potrzeby śledzić na których portach słuchamy, a jedynie porty na które chcemy by można się było dostawać.

Utrzymywanie stanu jest bardzo wygodne, ale jednocześnie może być trochę zagmatwane. Możesz sobie strzelić w stopę w bardzo dziwne sposoby. Rozważmy następujący zestaw reguł:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23
pass out quick on tun0 proto tcp from any to any keep state
block in quick all
block out quick all
```

Na pierwszy rzut oka, wygląda na całkiem poprawną konfigurację. Umożliwiamy na nawiązywanie sesji przychodzących na port 23 i wychodzących wszędzie. Naturalnie, pakiety wychodzące na port 23 będą miały pakiety odpowiedzi, ale zestaw reguł jest ustawiony w ten sposób że reguła 'pass out' wygeneruje pozycję w liście stanów i wszystko będzie działało poprawnie. Przynajmniej tak ci się tylko wydaje.

Przykra prawda polega na tym, że po 60 sekundach bezczynności pozycja w tablicy stanów zostanie zamknięta ( w przeciwieństwie do normalnych 5 dni ). Dzieje się tak ponieważ mechanizm śledzący połączenia 'nie widział' oryginalnego pakietu SYN przeznaczonego do portu 23, a jedynie SYN ACK. IPF jest bardzo dobry w śledzeniu sesji TCP od początku do końca, ale nie jest zbyt dobry w odgadywaniu poprawności połączenia od połowy. Powinieneś przepisać reguły na takie:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 keep state
pass out quick on tun0 proto tcp from any to any keep state
block in quick all
block out quick all
```

Dodatkowe słowo w regułach spowoduje że pierwszy pakiet utworzy pozycję w tablicy stanów i wszystko będzie działało tak jak się tego spodziewałeś. W momencie gdy 3-stopniowy proces **nawiązywania połączenia** ( ang. *handshake* ) był widziany przez mechanizm utrzymywania stanu, połączenie oznaczane jest jako będące w trybie 4/4. Oznacza to, że jest ono skonfigurowane na długoterminowa wymianę danych, dopóki nie zostanie zamknięte ( kiedy to zmieniany jest również tryb ). Możesz sprawdzić aktualne tryby w tablicy stanów poleceniem 'ipfstat -s'.

### 3.3 UDP ze sprawdzaniem stanów

UDP nie ma stanów, więc naturalnie wykonanie dobrej roboty w śledzeniu stanu połączenia jest tutaj dużo trudniejsze. Mimo to ipf robi dobrą robotę. Kiedy maszyna A wysyła pakiet UDP do maszyny B z portu źródłowego X na port docelowy Y, ipf pozwoli na odpowiedź z maszyny B do A z portu źródłowego Y na port docelowy X. Jest to krótkoterminowa pozycja w tabeli stanów, stworzona tylko na 60 sekund.

Poniżej przykład tego co się dzieje gdy wykonujemy nslookup by pobrać adres IP maszyny <http://www.3com.com>:

```
$ nslookup www.3com.com
```

Generowany jest pakiet DNS:

```
17:54:25.499852 20.20.20.1.2111 > 198.41.0.5.53: 51979+
```

Pakiet pochodzi z 20.20.20.1 i z portu 2111, a skierowany jest do 198.41.0.5 na port 53. Tworzona jest 60-sekundowa pozycja w liście stanów. Jeśli w tym czasie nadejdzie pakiet z 198.41.0.5 portu 53 przeznaczony dla 20.20.20.1 na port 2111, zostanie przepuszczony. Jak możesz sprawdzić, milisekundy później:

```
17:54:25.501209 198.41.0.5.53 > 20.20.20.1.2111: 51979 q: www.3com.com
```

Pakiet pasuje do kryteriów opisywanych przez pozycję w liście stanów i jest przepuszczany. W tym samym momencie w którym pakiet wchodzi, pozycja znika z listy stanów i żaden nowy pakiet nie może zostać wpuszczony, nawet gdyby twierdził że jest z dokładnie tego samego źródła.

### 3.4 ICMP ze sprawdzaniem stanów

IPFilter traktuje stany ICMP dokładnie tak, jak możnaby się spodziewać rozumiejąc jak ICMP używany jest z TCP i UDP, i przy zrozumieniu jak działa komenda `'keep state'`. Są generalnie dwa typy wiadomości ICMP: zapytania i odpowiedzi. Gdy wpisujesz regułę taką jak na przykład:

```
pass out on tun0 proto icmp from any to any icmp-type 8 keep state
```

by zezwolić na wychodzące odpowiedzi na żądanie echa ( typowy ping ), wpuszczony zostanie pakiet `icmp-type 0`, który jest zwyczajową odpowiedzią. Pozycja w liście stanów ma domyślny czas wygaśnięcia niekompletnego stanu 0/0 wynoszący 60 sekund. W związku z tym, jeśli utrzymujesz stan każdej wychodzącej wiadomości icmp która wywołuje odpowiedź icmp, potrzebujesz reguły `'proto icmp [...] keep state'`.

Jednakże, większość wiadomości ICMP to wiadomości o statusie generowane przez błędy w UDP ( i czasami TCP ). W IPFilter w wersjach 3.4.x i wyższych każda wiadomość o błędzie ICMP ( powiedzmy `icmp-type 3 code 3` - port niedostępny, lub `icmp-type 11` - przekroczony czas ), która pasuje do aktywnego wpisu w liście stanów, powoduje że pakiet ICMP jest wpuszczany. Na przykład, w starszych wersjach IPFilter, jeśli chciałeś by działał traceroute musiałeś użyć:

```
pass out on tun0 proto udp from any to any port 33434><33690 keep state
pass in on tun0 proto icmp from any to any icmp-type timex
```

a teraz możesz uzyskać to samo poprzez wpis:

```
pass out on tun0 proto udp from any to any port 33434><33690 keep state
```

By zabezpieczyć się przed wślizgnięciem się nieproszonych pakietów ICMP przez Twoją ścianę ogniową, nawet gdy aktywny wpis w tabeli stanów istnieje, przychodzący pakiet ICMP jest sprawdzany nie tylko pod kątem poprawnego adresu źródłowego i przeznaczenia ( i portów, jeśli to go dotyczy ), ale jeszcze małej części danych w pakiecie określającej w wyniku którego pakietu ta wiadomość ICMP została wygenerowana.

### 3.5 Wykrywanie skanów FIN; słowa kluczowe `'flags'` i `'keep frags'`

Wróćmy do czterolinijkowego zestawu reguł z poprzedniej sekcji:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 keep state
pass out quick on tun0 proto tcp from any to any keep state
block in quick all
block out quick all
```

Jest on prawie, ale nie całkiem satysfakcjonujący. Problem polega na tym, że nie tylko pakiety SYN mogą dotrzeć do portu 23, ale również inne, stare pakiety. Możemy to zmienić przez zastosowanie słowa kluczowego `'flags'`:



```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 flags S keep state
pass out quick on tun0 proto tcp from any to any flags S keep state
block in quick all
block out quick all
```

Obecnie tylko pakiety TCP, skierowane do 20.20.20.1 na port 23 z ustawioną tylko flagą SYN mogą dotrzeć i utworzyć pozycję w liście stanów. Samotna flaga SYN ustawiona jest tylko w pierwszym pakiecie sesji TCP ( zwanej pakietem nawiązującym połączenie TCP ) i to jest to co chcieliśmy tak naprawdę uzyskać. Są przynajmniej dwie zalety takiego zapisu: nie dotrą do ciebie żadne inne pakiety które mogłyby namieszać w tabeli stanów. Po drugie, skany FIN i XMAS nie powiodą się ponieważ mają ustawione również inne flagi oprócz SYN. Aktualnie, wszystkie przychodzące pakiety muszą być albo nawiązującymi połączenie albo już do niego należeć. Jeśli nadejdzie cokolwiek innego, jest albo spreparowane albo jest skanem portów. Istnieje jednak jeden wyjątek - gdy dociera do nas pakiet sfragmentowany. IPF jest przygotowany na obsługę takich sytuacji, z pomocą słowa kluczowego `'keep frags'`. Przy jego zastosowaniu, IPF będzie potrafił śledzić również sesje z pakietami, które są sfragmentowane i pozwoli im przejść. Przepiszmy te trzy reguły by logować pakiety spreparowane i zezwolić na obsługę pakietów sfragmentowanych:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 flags S keep state keep frags
pass out quick on tun0 proto tcp from any to any keep state flags S keep frags
block in log quick all
block out log quick all
```

Taki zapis działa, ponieważ każdy pakiet który powinien być wpuszczony, zostanie najpierw wpisany do tabeli stanów, zanim reszta reguł zdąży go zablokować. Jedynym skanem którego ten zapis nie wykryje jest skan SYN. Jeśli naprawdę jesteś tym zaniepokojony, być może powinieneś logować również wszystkie pakiety SYN.

## 3.6 Odpowiadanie na zablokowane pakiety

Jak do tej pory, wszystkie nasze zablokowane pakiety były zrzucane na podłogę, i bez względu na to czy zostały zalogowane czy nie, nie odsyłaliśmy niczego do hosta który przysłał nam pakiet. Czasami nie jest to najbardziej pożądane rozwiązanie, ponieważ robiąc coś takiego, informujemy go że mamy działający filtr pakietów. Wydaje się dużo lepszym rozwiązaniem wprowadzenie w błąd atakującego, że nie działa żaden filtr pakietów i nie ma żadnych usług przy pomocy których możnaby się włamać. Tutaj dochodzimy do dużo bardziej wyrafinowanego blokowania.

Kiedy na systemie Unixowym nie działa usługa, zwykle wysyła on zdalnemu systemowi jakiś rodzaj odpowiedzi. W przypadku TCP, jest to pakiet RST (Reset). Kiedy blokujemy pakiet TCP, IPF może faktycznie odesłać pakiet RST do nadawcy jeśli użyjemy słowa kluczowego `'return-rst'`.

Podczas gdy do tej pory pisaliśmy:

```
block in log on tun0 proto tcp from any to 20.20.20.0/24 port = 23
pass in      all
```

Możemy teraz napisać:

```
block return-rst in log proto tcp from any to 20.20.20.0/24 port = 23
block in log quick on tun0
pass in      all
```

Potrzebujemy dwóch reguł `'block'`, ponieważ `'return-rst'` działa tylko dla TCP, a my nadal chcemy zablokować protokoły takie jak UDP, ICMP i inne. Kiedy użyjemy takich reguł, strona zdalna otrzyma komunikat `'connection refused'` (połączenie odrzucone), zamiast `'connection timed out'` (upłynął czas na nawiązanie połączenia).

Możliwe jest również odsyłanie wiadomości z błędami, gdy ktoś wysyła pakiet UDP do portu w twoim systemie. Do tej pory pisaliśmy:

```
block in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 111
```

Używając słowa kluczowego `return-icmp` możemy teraz napisać:

```
block return-icmp(port-unr) in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 111
```

Zgodnie z książką *TCP/IP Illustrated*, `port-unreachable` (port nieosiągalny) jest prawidłowym komunikatem ICMP odpowiedzi jeśli na danym porcie nie nasługuje żadna usługa. Możesz użyć dowolnego typu ICMP, ale `port-unreachable` jest prawdopodobnie najlepszą odpowiedzią. Jest również domyślna w przypadku użycia `'return-icmp'`.

Jednak, gdy zaczniesz używać `'return-icmp'`, zauważysz że nie jest to bardzo skryte, ponieważ zwraca pakiet ICMP z adresem ściany ogniowej, a nie adresem maszyny dla której pakiet był przeznaczony. Zostało to naprawione w ipfilter w wersji 3.3 i dodano nowe słowo kluczowe: `'return-icmp-as-dest'`. Nowy format wygląda tak:

```
block return-icmp-as-dest(port-unr) in log on tun0 proto udp from any to 20.20.20.0/24 port = 111
```

Powinieneś być bardzo uważny i generować pakiety odpowiedzi tylko w sytuacjach, w których dobrze wiesz na co odpowiadasz. Na przykład, odpowiadanie `return-icmp` na broadcast w sieci lokalnej może skończyć się zalaniem pakietami sieci.

## 3.7 Wymyślne techniki logowania

Bardzo ważne jest, by zauważyć, że sama obecność słowa kluczowego `'log'` zapewnia tylko dostępność pakietu dla urządzenia logującego ipfilter: `/dev/ipl`. Tak naprawdę aby zobaczyć tę informację, musisz mieć uruchomione narzędzie `ipmon` (lub inne, które czyta `/dev/ipl`). Typowe użycie słowa `'log'` jest skojarzone z poleceniem `ipmon -s`, które dopiero loguje informacje do `syslog`. Od ipfilter 3.3, można nawet kontrolować zachowanie logujące `syslog` poprzez użycie słowa `'log level'`, tak jak w regułach poniżej:

```
block in log level auth.info quick on tun0 from 20.20.20.0/24 to any
block in log level auth.alert quick on tun0 proto tcp from any to 20.20.20.0/24 port = 21
```

W dodatku, możesz ograniczać informacje które są logowane. Na przykład, możesz nie być zainteresowany faktem, że ktoś próbował twój port telnetu 500 razy, a jedynie, że w ogóle próbował to robić. Służy do tego opcja `'log first'` logująca tylko pierwszy pakiet. Oczywiście, pierwszeństwo dotyczy tylko jednej sesji i dla typowego zablokowania pakietu, trudno będzie ci uzyskać efekt by to robiło dokładnie to co chciałeś. Jednakże przy połączeniu tego z poleceniami `'pass'` i `'keep state'`, może być to bardzo pomocne narzędzie w śledzeniu ruchu.

Inna użyteczna opcja którą możesz wykorzystać przy logowaniu, to zachowanie interesujących fragmentów pakietu oprócz normalnej informacji logowanej z pakietem.

IPFilter zaloguje pierwsze 128 bajtów pakietu jeśli użyjesz słowa ``log body'`. Powinieneś starać się ograniczać takie logowanie, ponieważ czyni to twoje logi bardzo szczegółowymi. Dla niektórych zastosowań jest to jednak wygodne - można sprawdzić dokładniej pakiet, lub wysłać te informacje do jakiejś aplikacji do dalszej analizy.

### 3.8 Złożenie tego wszystkiego razem

Mamy teraz całkiem szczelną ścianę ogniową, ale nadal możemy ją jeszcze uszczelnić. Część oryginalnego zestawu reguł usuneliśmy, część została jako użyteczna. Sugerowałbym wrócenie do wszystkich dotyczących zabezpieczenia przed fałszowaniem. To sprawia, że zostajemy z:

```
block in          on tun0
block in    quick on tun0 from 192.168.0.0/16 to any
block in    quick on tun0 from 172.16.0.0/12 to any
block in    quick on tun0 from 10.0.0.0/8 to any
block in    quick on tun0 from 127.0.0.0/8 to any
block in    quick on tun0 from 0.0.0.0/8 to any
block in    quick on tun0 from 169.254.0.0/16 to any
block in    quick on tun0 from 192.0.2.0/24 to any
block in    quick on tun0 from 204.152.64.0/23 to any
block in    quick on tun0 from 224.0.0.0/3 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass  out quick on tun0 proto tcp/udp from 20.20.20.1/32 to any keep state
pass  out quick on tun0 proto icmp   from 20.20.20.1/32 to any keep state
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 80 flags S keep state
```

### 3.9 Zwiększanie wydajności przez tworzenie grup reguł

Rozszerzmy użycie naszej ściany ogniowej poprzez stworzenie bardziej skomplikowanej, i mamy nadzieję bardziej przystającej do świata rzeczywistego konfiguracji przykładowej. Na potrzeby tego przykładu, zmienimy nazwy interfejsów i numerację sieci. Założmy, że mamy trzy interfejsy na swojej ścianie ogniowej, `x10`, `x11` i `x12`.

- `x10` jest podłączony do sieci zewnętrznej 20.20.20.0/26
- `x11` jest podłączony do naszej sieci zdemilitaryzowanej 20.20.20.64/26
- `x12` jest podłączony do naszej chronionej sieci 20.20.20.128/25

Zdefiniujemy od razu cały zestaw reguł, ponieważ jak do tej pory powinieneś już umieć je czytać:

```
block in    quick on x10 from 192.168.0.0/16 to any
block in    quick on x10 from 172.16.0.0/12 to any
block in    quick on x10 from 10.0.0.0/8 to any
block in    quick on x10 from 127.0.0.0/8 to any
block in    quick on x10 from 0.0.0.0/8 to any
block in    quick on x10 from 169.254.0.0/16 to any
block in    quick on x10 from 192.0.2.0/24 to any
```

```

block in      quick on x10 from 204.152.64.0/23 to any
block in      quick on x10 from 224.0.0.0/3 to any
block in log   quick on x10 from 20.20.20.0/24 to any
block in log   quick on x10 from any to 20.20.20.0/32
block in log   quick on x10 from any to 20.20.20.63/32
block in log   quick on x10 from any to 20.20.20.64/32
block in log   quick on x10 from any to 20.20.20.127/32
block in log   quick on x10 from any to 20.20.20.128/32
block in log   quick on x10 from any to 20.20.20.255/32
pass out on x10 all

```

```

pass out quick on x11 proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state
pass out quick on x11 proto tcp from any to 20.20.20.64/26 port = 21 flags S keep state
pass out quick on x11 proto tcp from any to 20.20.20.64/26 port = 20 flags S keep state
pass out quick on x11 proto tcp from any to 20.20.20.65/32 port = 53 flags S keep state
pass out quick on x11 proto udp from any to 20.20.20.65/32 port = 53 keep state
pass out quick on x11 proto tcp from any to 20.20.20.66/32 port = 53 flags S keep state
pass out quick on x11 proto udp from any to 20.20.20.66/32 port = 53 keep state
block out on x11 all
pass in quick on x11 proto tcp/udp from 20.20.20.64/26 to any keep state

block out on x12 all
pass in quick on x12 proto tcp/udp from 20.20.20.128/25 to any keep state

```

Z tego arbitralnego przykładu, od razu można zauważyć że nasz zestaw reguł powoli staje się coraz mniej czytelny. By sprawy pogorszyć, w momencie gdy dodamy reguły dla naszej sieci zdemilitaryzowanej (DMZ), musimy dodać dodatkowe reguły testujące dla każdego pakietu, co pogorszy wydajność połączeń x10-x12. Jeśli zbudujesz ścianę ogniową z regułami takimi jak te, a masz dużą przepustowość łącza i średnio dużo mocy obliczeniowej procesora, każdy kto ma stację roboczą w sieci podłączonej do interfejsu x12 przyjdzie do ciebie by umieścić swoją głowę na talerzu. Zatem, by utrzymać połączenie głowy z torsem, możesz przyspieszyć znacznie porównywanie przez stworzenie grup reguł. Pozwalają one na zapisanie swoich reguł w formie drzewa, zamiast w postaci linearnej - więc jeśli pakiet nie ma nic wspólnego z pewnym zestawem testów ( powiedzmy, reguł dotyczących x11 ), nie będą one w ogóle brane pod uwagę. Jest to coś w rodzaju posiadania wielu ścian ogniowych na tej samej maszynie.

Poniżej przykład z którym zaczniemy:

```

block out quick on x11 all head 10
pass out quick proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state group 10
block out on x12 all

```

W tym bardzo prostym przykładzie, widzimy małą zapowiedź potęgi grup reguł. Jeśli pakiet nie jest przeznaczony dla interfejsu x11, nagłówek ( head ) dla grupy 10 ( group 10 ) w ogóle nie będzie pasował i nie zostanie uwzględniony przy porównywaniu. Jeśli pakiet pasuje do reguły x11, słowo kluczowe quick spowoduje ucięcie przetwarzania na poziomie podstawowym/korzenia ( grupa reguł 0 ) i skoncentruje się na testowaniu reguł które dotyczą grupy 10, w tym wypadku sprawdzenia flagi SYN w pakietach przeznaczonych dla 80/tcp. W ten sposób, możemy przepisać powyższe reguły tak by zmaksymalizować wydajność naszej ściany ogniowej.

```

block in quick on x10 all head 1
block in      quick on x10 from 192.168.0.0/16 to any   group 1
block in      quick on x10 from 172.16.0.0/12 to any   group 1

```

```

block in      quick on x10 from 10.0.0.0/8 to any      group 1
block in      quick on x10 from 127.0.0.0/8 to any     group 1
block in      quick on x10 from 0.0.0.0/8 to any       group 1
block in      quick on x10 from 169.254.0.0/16 to any   group 1
block in      quick on x10 from 192.0.2.0/24 to any    group 1
block in      quick on x10 from 204.152.64.0/23 to any  group 1
block in      quick on x10 from 224.0.0.0/3 to any     group 1
block in log   quick on x10 from 20.20.20.0/24 to any  group 1
block in log   quick on x10 from any to 20.20.20.0/32  group 1
block in log   quick on x10 from any to 20.20.20.63/32 group 1
block in log   quick on x10 from any to 20.20.20.64/32 group 1
block in log   quick on x10 from any to 20.20.20.127/32 group 1
block in log   quick on x10 from any to 20.20.20.128/32 group 1
block in log   quick on x10 from any to 20.20.20.255/32 group 1
pass in              on x10 all group 1

pass out on x10 all

block out quick on x11 all head 10
pass out quick on x11 proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state group 10
pass out quick on x11 proto tcp from any to 20.20.20.64/26 port = 21 flags S keep state group 10
pass out quick on x11 proto tcp from any to 20.20.20.64/26 port = 20 flags S keep state group 10
pass out quick on x11 proto tcp from any to 20.20.20.65/32 port = 53 flags S keep state group 10
pass out quick on x11 proto udp from any to 20.20.20.65/32 port = 53          keep state group 10
pass out quick on x11 proto tcp from any to 20.20.20.66/32 port = 53 flags S keep state
pass out quick on x11 proto udp from any to 20.20.20.66/32 port = 53          keep state group 10
pass in quick on x11 proto tcp/udp from 20.20.20.64/26 to any keep state

block out on x12 all
pass in quick on x12 proto tcp/udp from 20.20.20.128/25 to any keep state

```

Teraz możesz poobserwować grupy reguł w akcji. Dla komputera w sieci x12, kompletnie pomijamy wszystkie testy w grupie 10, kiedy nie komunikujemy się z żadnymi komputerami w tej sieci.

Zależnie od Twojej sytuacji, korzystne może być pogrupowanie reguł według protokołu, różnych maszyn, bloków sieciowych, lub czegokolwiek - tak by sprawić, że przepływ informacji będzie płynny.

### 3.10 słowo kluczowe - 'Fastroute' daje niewykrywalności ( ang. *stealth* )

Nawet mimo faktu, że przekazujemy część pakietów a inne blokujemy, zachowujemy się tak, jak dobrze zachowujący się router powinien się zachowywać - zmniejszamy TTL pakietu i niniejszym oznajmiamy całemu światu że tak, jest tutaj przejście ( ang. *hop* ). Możemy jednak ukryć swoją obecność przed zbyt dociekliwymi aplikacjami takimi jak unix'owy traceroute, który używa pakietów UDP z różnymi TTL by zmapować ilość przejść pomiędzy dwoma komputerami. Jeśli chcemy, by nadchodzące traceroute działało, ale nie chcemy oświadczać że mamy tu ścianę ogniową która spowoduje dodanie jednego przejścia, możemy zrobić to regułą taką jak ta:

```

block in quick on x10 fastroute proto udp from any to any port 33434 >> 33465

```

Obecność słowa `'fastroute'` zasygnalizuje ipfilter żeby nie przekazywać pakietu do stosu IP dla wykonania routingu, co spowodowałoby zmniejszenie TTL pakietu. Pakiet zostanie po prostu od razu umieszczony na interfejsie wyjściowym przez ipfilter i nic nie zostanie zmienione. ipfilter oczywiście sprawdzi systemową tablicę routingu by sprawdzić na jakim interfejsie powinien wystawić pakiet, ale wykona zadanie preroutowania pakietu sam.

Istnieje również powód, dla którego używamy tu słów `'block quick'`. Gdybyśmy użyli słowa `'pass'` i mielibyśmy włączone przekazywanie ( ang. *forwarding* ) pakietów IP w kernelu, skończyłoby się to na tym, że dostalibyśmy dwie ścieżki którymi pakiet mógłby wyjść, a to prawdopodobnie spowodowałoby błąd **kernel panic**.

Trzeba również dodać, że większość kerneli Uniksowych ( a w szczególności te, na których ipfilter zwykle pracuje ), mają dużo bardziej wydajny kod routingu niż ten w ipfilter, a w związku z tym nie powinieneś myśleć o słowie `'fastroute'` jako o sposobie na zwiększenie szybkości pracy swojej ściany ogniowej. Powinno być ono używane tylko w przypadku gdy ukrycie się jest najważniejsze.

## 4. NAT i proxy

Poza otoczeniem firmowym, jedną z największych zalet technologii ścian ogniowych jest możliwość połączenia wielu komputerów przez jeden interfejs zewnętrzny, często bez zgody, wiedzy czy nawet podejrzeń ze strony dostawcy internetowego. Ci którzy znają Linuksa nazywają to Maskaradą IP ( ang. *IP Masquerading* ), ale dla reszty świata jest ona znana pod nazwą Translacja Adresów Sieciowych, lub w skrócie NAT ( ang. *Network Address Translation* ).

### 4.1 Mapowanie wielu adresów w jeden

Najprostszym zastosowaniem NAT jest dokładnie to co robi jego odpowiednik - Linuksowa Maskarada IP. Zapisuje się to w jednej prostej regule:

```
map tun0 192.168.1.0/24 -> 20.20.20.1/32
```

Bardzo proste. Zawsze gdy pakiet wychodzi przez interfejs `tun0` ze źródłowym adresem pasującym do maski CDIR `192.168.1.0/24`, adres jest zamieniany jeszcze na stosie IP, tak by zawierał adres źródłowy `20.20.20.1` a dopiero wtedy zostaje wysłany do swojego celu przeznaczenia. System utrzymuje listę jakie zmapowane połączenia są w trakcie tak by mógł wykonać czynność odwrotną gdy nadejdzie odpowiedź ( skierowana do `20.20.20.1` ) i odesłać ją do oryginalnego nadawcy.

Jest jednak pewien minus reguły którą teraz wpisaliśmy. W wielu przypadkach, nie wiemy jaki mamy adres IP naszego zewnętrznego interfejsu ( jeśli używasz `tun0` czy `ppp0` i typowego ISP ), więc ustawienie tablicy NAT staje się raczej problematyczne. Na szczęście NAT jest na tyle mądry, że potrafi zaakceptować adres w postaci `0/32`. Sygnalizuje to, że musi sprawdzić sobie sam jaki właściwie adres ma interfejs zewnętrzny i prawidłowo zmodyfikować pakiet. Spójrz niżej:

```
map tun0 192.168.1.0/24 -> 0/32
```

Możemy teraz załadować nasze reguły do `ipnat` i połączyć się ze światem zewnętrznym bez konieczności edytowania czegokolwiek. Musisz jednak uruchomić `'ipf -y'` by odświeżyć adres przypisany połączeniu jeśli się rozłączysz i wdzwonisz ponownie, lub gdy wygaśnie Twoja dzierżawa na adres przydzielony przez DHCP.

Niektórzy z was mogą się zastanawiać, co dzieje się z portem źródłowym gdy wykonywane jest mapowanie. W przypadku naszej reguły, port źródłowy pakietu nie zostaje zmieniony w stosunku do oryginału. Są jednak przypadki gdy nie chcemy by tak się działo - może po drodze jest jeszcze jedna ściana ogniowa, lub wiele maszyn próbuje używać tego samego portu. Może to powodować kolizje i pakiet jest przepuszczany bez mapowania. `ipnat` pomaga w tym momencie, poprzez dostarczenie słowa kluczowego `'portmap'`:

```
map tun0 192.168.1.0/24 -> 0/32 portmap tcp/udp 20000:30000
```

Nasza reguła wrzuca wszystkie mapowane połączenia ( które mogą używać protokołu tcp, udp lub jednocześnie tcp/udp ) w zakres portów od numeru 20000 do 30000.

Możemy jeszcze ułatwić sobie życie, używając polecenia `auto` by poinformować ipnat by sam określił sobie jakich portów może użyć i zaalokował je proporcjonalnie wśród adresów, używanych w puli:

```
map tun0 192.168.1.0/24 -> 0/32 portmap tcp/udp auto
```

Pamiętaj, że reguły mapujące odnoszą się tylko do protokołów, które określiłeś ( np. tcp, udp lub tcp/udp ), i nie odnoszą się do innych, takich jak ICMP czy IPsec ESP/AH. Dla nich, musisz dodać dodatkowe mapowania odnoszące się do wszystkich innych protokołów:

```
map tun0 192.168.1.0/24 -> 0/32 portmap tcp/udp 20000:30000
map tun0 192.168.1.0/24 -> 0/32
```

## 4.2 Mapowanie wielu adresów w przydzieloną pulę adresów

Innym częstym zastosowaniem NAT jest wzięcie małej, statycznie przydzielonej puli adresów i zmapowanie wielu komputerów w tą mniejszą przestrzeń adresową. Jest to proste do uzyskania przy użyciu tego co już wiesz o słowach kluczowych `map` i `portmap`. Zapiszmy to jak poniżej:

```
map tun0 192.168.0.0/16 -> 20.20.20.0/24 portmap tcp/udp 20000:60000
```

Zdarza się również, że zdalna aplikacja wymaga, by wszystkie połączenia przychodziły z tego samego IP. Możemy pomóc, instruując NAT by statycznie zmapował sesje z danej maszyny na pulę adresów i wykonał trochę magii z wybraniem portu. Robi się to, stosując słowo kluczowe `map-block`, tak jak poniżej:

```
map-block tun0 192.168.1.0/24 -> 20.20.20.0/24
```

Tak jak słowo `map`, `map-block` ma również swoją wersję `portmap`. Używa się jej albo ze słowem kluczowym `auto`:

```
map-block tun0 192.168.1.0/24 -> 20.20.20.0/24 auto
```

lub ze słowem `ports`, jeśli chcesz wskazać konkretne numery portów należące do określonych adresów IP:

```
map-block tun0 192.168.1.0/24 -> 20.20.20.0/24 ports 64
```

## 4.3 Mapowania jeden do jednego

Czasami zdarza się, że chcemy by komputer za ścianą ogniową z danym IP pojawiał się z innym IP. Jednym z takich przykładów może być laboratorium komputerowe, dołączone do różnych sieci, a mające wziąć udział w testach. Nie chcemy rekonfigurować całego laboratorium, ponieważ możemy postawić przed nim komputer wykonujący NAT i zmienić adresy w jednym miejscu. Można to wykonać za pomocą słowa kluczowego `bimap`, od mapowania dwukierunkowego. Bimap posiada

pewne dodatkowe zabezpieczenia zapewniające, że wie jaki jest stan połączenia, podczas gdy słowo `map` używane jest tylko do zaalokowania adresu i portu źródłowego oraz odpowiedniej modyfikacji pakietu.

```
bimap tun0 192.168.1.1/32 -> 20.20.20.1/32
```

spowoduje zmapowanie dla jednej maszyny.

## 4.4 NAT według Zasad

Często zdarza się, że musimy zapewnić funkcjonowanie NATu zależne od adresów źródłowych i przeznaczenia. Na przykład, chcemy prowadzić NAT dla wszystkich, oprócz określonej podsieci:

```
map tun0 from 192.168.1.0/24 ! to 5.0.0.0/20 -> 20.20.20.1/32
```

Można również zastosować taką konstrukcję:

```
map tun0 from 192.168.1.5/32 port = 5555 to 1.2.3.4/32 -> 20.20.20.2/32
map tun0 from 192.168.1.0/24 to 5.0.0.0/20 -> 20.20.20.2/32 portmap auto
```

## 4.5 Usługi fałszujące

A co to ma do rzeczy? Wiele. Powiedzmy że mamy serwer WWW pracujący na 20.20.20.5, a ponieważ staliśmy się ostatnio bardzo podejrzliwi co do bezpieczeństwa naszej sieci, chcemy by serwer nie pracował na porcie 80, ponieważ wymaga to krótkiego momentu pracy z przywilejami roota. Ale jak uruchomić go na mniej uprzywilejowanym porcie 8000 w świecie 'wszystko kropka com'? Jak ktokolwiek znajdzie nasz serwer? Możemy użyć usług przekierowania NATu by rozwiązać ten problem, instruując go by przemapował wszystkie połączenia przeznaczone dla 20.20.20.5:80 na 20.20.20.5:8000. Używa się do tego słowa kluczowego `rdr`:

```
rdr tun0 20.20.20.5/32 port 80 -> 192.168.0.5 port 8000
```

Możemy również podać protokół, jeśli chcielibyśmy na przykład przekierować usługę UDP zamiast TCP ( która jest domyślna ). Na przykład, gdybyśmy chcieli zastawić zasadzkę na naszej ścianie ogniowej i udawać zainstalowanego Back Orifice dla Windows, możemy ochronić naszą sieć przez prostą regułą:

```
rdr tun0 20.20.20.0/24 port 31337 -> 127.0.0.1 port 31337 udp
```

Możliwe jest modyfikowanie zachowania słowa `rdr` w zależności od adresów źródłowego i docelowego:

```
rdr tun0 from 10.1.1.1/32 to 20.20.20.5/32 port = 80 -> 192.168.0.5 port 8001
rdr tun0 from 10.1.1.1/32 port = 12345 to 20.20.20.5/32 port = 80 -> 192.168.0.5 port 8002
```

Należy jednak zaznaczyć bardzo ważną sprawę. Nie możesz łatwo użyć tego jako lustra, tzn.:



```
rdr tun0 20.20.20.5/32 port 80 -> 20.20.20.6 port 80 tcp
```

Taka konstrukcja nie zadziała, jeśli .5 i .6 są w tym samym segmencie LAN. Funkcja `rdr` jest wykonywana na każdym pakiecie, który trafia do ściany ogniowej na określonym interfejsie. Gdy nadchodzi pakiet który pasuje do reguły, jego adres docelowy jest zmieniany, pakiet trafia do ipf w celu przefiltrowania i gdy tylko przetrwa boje z regułami filtra, jest wysyłany do kodu routującego Uniksa. Ponieważ pakiet pochodzi z tego samego interfejsu, którym będzie musiał opuścić system by dotrzeć do maszyny docelowej, system jest w kropce. Lustra po prostu nie działają. Nie działa również podawanie adresu interfejsu z którego pakiet właśnie nadszedł. Pamiętaj że adresy docelowe dla `rdr` muszą istnieć po drugiej stronie ściany ogniowej, w sieci do której prowadzi inny interfejs niż ten, którym wszedł pakiet pierwotny.

## 4.6 Transparentne proxy; W końcu przekierowanie do czegoś się przydaje.

Ponieważ właśnie instalujesz ścianę ogniową możesz zdecydować, że jest to dobry moment na zastosowanie proxy dla wielu wychodzących połączeń. Dzięki temu możesz jeszcze bardziej zacieśnić swoje reguły filtrowania chroniące sieć. Może się również zdarzyć tak, że natrafisz na sytuację, której proces mapowania NAT nie może aktualnie poprawnie obsłużyć. Taką sytuację można obejść następującą konstrukcją:

```
rdr xl0 0.0.0.0/0 port 21 -> 127.0.0.1 port 21
```

Ten wpis mówi, że każdy pakiet z interfejsu `xl0` przeznaczony dla dowolnego adresu ( `0.0.0.0` ) na port `ftp`, zostanie przepisany tak by połączyć się z proxy, które pracuje na systemie wykonującym NAT na porcie 21.

Ten specyficzny przykład proxy FTP prowadzi do pewnych komplikacji, kiedy chodzi o serwery WWW czy inne automatycznie logujące się klienty, które nie wiedzą o wymaganiach komunikacji z proxy. Istnieją łąty dla `ftp-gw` z TIS Firewall Toolkit które w połączeniu z procesem NAT pozwalają na zapewnienie funkcjonalności, dzięki której można sprawdzić gdzie chciałeś wejść i wysłać cię tam automatycznie. Wiele paczek proxy pracuje obecnie również w środowisku transparentnego proxy ( na przykład Squid, znajdujący się pod adresem <http://squid.nlanr.net>, pracuje w porządku ).

Takie zastosowanie słowa kluczowego `rdr` jest często użyteczne gdy chcesz zmusić użytkowników do autoryzowania się na proxy ( na przykład, gdy chcesz by twoi inżynierowie mogli przeglądać strony WWW, ale wolałbyś żeby raczej ludzie z call-center tego nie robili ).

## 4.7 Filtrowanie przekierowanych usług

Wielu użytkowników chce połączyć filtrowanie i translację adresów, by zapewnić usługi tylko określonym komputerom za ich ścianą ogniową. Na przykład, by zapewnić dostęp do serwera WWW za twoją maszyną `20.20.20.5` ( która tak naprawdę ma adres `192.168.0.5` w sieci wewnętrznej ) Twojemu przyjacielowi który ma adres `172.16.8.2`, możesz wpisać następujący wiersz do `ipnat.rules`:

```
rdr tun0 20.20.20.5/32 port 80 -> 192.168.0.5 port 8000
```

a następujący wiersz do `ipf.rules`:

```
pass in on tun0 proto tcp from 172.16.8.2/32 to 192.168.0.5/32 port = 8000 flags S keep state
```

Na pierwszy rzut oka może to wyglądać trochę dziwnie, ale dlatego że NAT odbywa się pierwszy, a w jego trakcie adres docelowy i port docelowy jest przepisywany

zanim zajmie się nim kod filtrujący pakietu.

## 4.8 Magia ukryta w NAT; proxy aplikacji

Ponieważ ipnat dostarcza metody na przepisywanie pakietów w trakcie ich podróży przez ścianę ogniową, staje się wygodnym miejscem do wbudowania proxy poziomu aplikacji, zbudowanej pomiędzy tą aplikacją a ścianą ogniową. Na przykład: FTP. Możemy kazać naszej ścianie ogniowej sprawdzać pakiety, które przez nią przechodzą i gdy zauważy, że ma do czynienia z aktywną sesją FTP, dopisze sobie pewne tymczasowe reguły. Dzieje się to trochę na wzór `keep state` i sprawia, że połączenie FTP będzie działało. Realizujemy to regułą podobną do tej:

```
map tun0 192.168.1.0/24 -> 20.20.20.1/32 proxy port ftp ftp/tcp
```

Musisz pamiętać by umieścić wyrażenie `'proxy'` przed jakimikolwiek regułami `'portmap'`, ponieważ w innym przypadku `'portmap'` wykona przepisanie pakietu zanim `'proxy'` będzie miało szansę na zajęcie się nim. Pamiętaj, że reguły ipnat działają na zasadzie pierwszy pasujący. Istnieją również proxy dla `rcmd` (które, jak sądzimy, są berkeley'owskimi komendami `r-*`, i i tak powinny być zabronione, więc nie przyglądaliśmy im się), oraz `raudio` dla strumieni Real Audio PNM. W każdym bądź razie obydwa proxy powinny zostać ustawione przed jakimikolwiek regułami `'portmap'`, jeśli zamierzasz robić NAT.

## 4.9 Jeszcze więcej magii; NAT jako równoważenie obciążenia

Jeśli ipnat i tak już przepisuje nam pakiety, możemy użyć go do obsługi prostszych właściwości drogich urządzeń równoważących obciążenie. Będzie przypisywał mapowaniom wiele adresów docelowych - uzyskamy to stosując słowo kluczowe `'round-robin'`:

```
rdr tun0 20.20.20.5/32 port 80 -> 192.168.0.5, 192.168.0.6, 192.168.0.7 port 8000
```

## 5. Ładowanie i manipulowanie regułami filtra; Narzędzie ipf

Reguły Filtra IP ładowane są przez narzędzie `ipf`. Reguły można przechować w dowolnym pliku, ale normalnie stosuje się pliki `/etc/ipf.rules`, `/usr/local/etc/ipf.rules` lub `/etc/opt/ipf/ipf.rules`.

Filtr IP ma dwa zestawy reguł, zestaw aktywny i nieaktywny. Domyślnie, wszystkie operacje przeprowadzane są na zestawie aktywnym. Możesz pracować na zestawie nieaktywnym przez dodanie `'-i'` do linii poleceń `ipf`. Zestawy zamienia się miejscami przez użycie opcji `'-s'`. Jest to bardzo przydatne podczas testowania nowych zestawów reguł, bez usuwania starego zestawu.

Reguły można również usuwać z listy, stosując opcję `'-r'`, ale generalnie bezpieczniejsze jest po prostu usunąć cały zestaw reguł nad którym pracujesz przy użyciu opcji `'-F'` i załadować go po wykonaniu zmian.

Podsumowując, najłatwiejszym sposobem by załadować zestaw reguł jest użycie polecenia `'ipf -Fa -f /etc/ipf.rules'`. Jeśli chodzi o bardziej skomplikowane manipulowanie zestawami reguł, sprawdź stronę podręcznika `ipf(1)`.

## 6. Ładowanie i manipulowanie regułami NAT; narzędzie ipnat

Reguły NAT ładowane są przez narzędzie `ipnat`. Reguły NAT można przechowywać w dowolnym pliku, ale normalnie stosuje się pliki `/etc/ipnat.rules`, `/usr/local/etc/ipnat.rules` lub `/etc/opt/ipf/ipnat.rules`.

Reguły można również usuwać z listy, stosując opcję `-r`, ale generalnie bezpieczniejsze jest po prostu usunąć cały zestaw reguł nad którym pracujesz przy użyciu opcji `-c` i załadować go po wykonaniu zmian. Żadne aktywne mapowania nie są usuwane przy użyciu `-c`, ale można je usunąć poleceniem `-F`.

Reguły NAT i aktualne mapowania można sprawdzić używając opcji `-l`.

Podsumowując, najłatwiejszym sposobem by załadować zestaw reguł jest użycie polecenia `ipnat -CF -f /etc/ipnat.rules`.

## 7. Monitoring i odpluskwanie

Przyjdzie kiedyś moment, że zainteresujesz się tym, co tak naprawdę robi Twoja ściana ogniowa, a `ipfilter` byłby niekompletny bez pełnego zestawu narzędzi monitorujących.

### 7.1 Narzędzie ipfstat

W swojej najprostszej formie, `ipfstat` wyświetla tabelę interesujących danych dotyczących tego, jak Twoja ściana ogniowa daje sobie radę, czyli między innymi ilość pakietów które zostały przepuszczone i zablokowane, czy zostały zalogowane czy nie, ile jest aktywnych pozycji w liście stanów i tak dalej. Poniżej przykład czegoś, co mógłbyś zobaczyć po uruchomieniu tego narzędzia:

```
# ipfstat
input packets:      blocked 99286 passed 1255609 nomatch 14686 counted 0
output packets:     blocked 4200 passed 1284345 nomatch 14687 counted 0
input packets logged: blocked 99286 passed 0
output packets logged: blocked 0 passed 0
packets logged:      input 0 output 0
log failures:        input 3898 output 0
fragment state(in):  kept 0  lost 0
fragment state(out): kept 0  lost 0
packet state(in):    kept 169364  lost 0
packet state(out):   kept 431395  lost 0
ICMP replies:        0      TCP RSTs sent: 0
Result cache hits(in): 1215208 (out): 1098963
IN Pullups succeeded: 2      failed: 0
OUT Pullups succeeded: 0      failed: 0
Fastroute successes: 0      failures: 0
TCP cksum fails(in): 0      (out): 0
Packet log flags set: (0)
                    none
```

Jest również w stanie pokazać aktualną listę reguł. Użycie flagi `-i` lub `-o` pokaże listę reguł dla odpowiednio pakietów wchodzących i wychodzących. Dodanie opcji `-h` doda trochę użytecznych informacji, podając również 'licznik trafień' dla każdej reguły. Na przykład:

```
# ipfstat -ho
2451423 pass out on xl0 from any to any
354727 block out on ppp0 from any to any
430918 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any keep state keep frags
```

Z przykładu powyżej widać, że prawdopodobnie dzieje się coś nienormalnego, ponieważ mamy bardzo dużo zablokowanych pakietów wychodzących, nawet przy użyciu reguły `pass out`. Coś może wymagać tutaj dalszego zainteresowania, albo po prostu pracuje bez zarzutu. `ipfstat` nie może powiedzieć Ci czy Twoje reguły są dobre czy złe, może tylko poinformować co dzieje się przy ich użyciu.

W dalszym procesie odpluskwania reguł, możesz zechcieć użyć opcji `-n` która pokaże numery reguł przy każdej z nich:

```
# ipfstat -on
@1 pass out on xl0 from any to any
@2 block out on ppp0 from any to any
@3 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any keep state keep frags
```

Ostatnią naprawdę interesującą informacją w `ipfstat`, jest zrzut tabeli stanów. Wykonuje się to dodając opcję `-s`:

```
# ipfstat -s
281458 TCP
319349 UDP
0 ICMP
19780145 hits
5723648 misses
0 maximum
0 no memory
1 active
319349 expired
281419 closed
100.100.100.1 -> 20.20.20.1 ttl 864000 pass 20490 pr 6 state 4/4
pkts 196 bytes 17394      987 -> 22 585538471:2213225493 16592:16500
pass in log quick keep state
pkt_flags & b = 2,          pkt_options & ffffffff = 0
pkt_security & ffff = 0, pkt_auth & ffff = 0
```

Widzimy, że mamy aktualnie jedno połączenie TCP. Dane wyjściowe będą się delikatnie różnić od wersji do wersji, ale generalnie informacje są te same. Widzimy dla tego połączenia, że jest ono w pełni nawiązane ( `state 4/4` na końcu linijki, inne stany są niekompletne i opiszemy je później ). Możemy również odczytać że wpis ten ma **czas życia** ( ang. *time to live* ) 240 godzin, co jest absurdalnie długim czasem, ale ustawianym domyślnie dla nawiązanej sesji TCP. Licznik ten jest zmniejszany z każdą sekundą bezczynności, aż w końcu połączenie zostanie zerwane jeśli zostanie pozostawione bezczynnie. Licznik jest również zerowany na wartość 864000 za każdym razem, gdy użyjemy wpisu, więc połączenie nie zostanie zamknięte w trakcie jego używania. Możemy również odczytać, że przepuściliśmy przez to połączenie 196 pakietów składających się na około 17kB danych. Oprócz tego można odczytać porty po obu stronach - 987 i 22, co oznacza że ten wpis symbolizuje połączeniu z adresem 100.100.100.1 portu 987 na adres 20.20.20.1 na port 22. Bardzo duże numery w drugiej linijce to numery sekwencyjne TCP wygenerowane dla tego

połączenia, pomagające zabezpieczyć je przed wpuszczeniem dodatkowych, spreparowanych pakietów. Pokazane jest również okno TCP. Trzecia linia stanowi wynik reguły która została wygenerowana przez regułę `keep state` i pokazuje ona że jest to połączenie przychodzące.

## 7.2 Narzędzie `ipmon`

`ipfstat` jest fajny jeśli chodzi o sprawdzenie aktualnego stanu systemu, ale zwykle chcemy mieć również jakiś log, by oglądać wydarzenia dziejące się w czasie. Służy do tego `ipmon`. Jest on zdolny do sprawdzania logów pakietów ( tworzonych przez słowo kluczowe `log` w regułach ), logu tabeli stanów i logu NAT, lub dowolnej kombinacji tych trzech. Narzędzie to może pracować zarówno na pierwszym planie, lub jako demon który loguje informacje do `syslog` lub pliku. Jeśli chcielibyśmy zobaczyć listę stanów w akcji, użyjemy polecenia `'ipmon -o S'`:

```
# ipmon -o S
01/08/1999 15:58:57.836053 STATE:NEW 100.100.100.1,53 -> 20.20.20.15,53 PR udp
01/08/1999 15:58:58.030815 STATE:NEW 20.20.20.15,123 -> 128.167.1.69,123 PR udp
01/08/1999 15:59:18.032174 STATE:NEW 20.20.20.15,123 -> 128.173.14.71,123 PR udp
01/08/1999 15:59:24.570107 STATE:EXPIRE 100.100.100.1,53 -> 20.20.20.15,53 PR udp Pkts 4 Bytes 356
01/08/1999 16:03:51.754867 STATE:NEW 20.20.20.13,1019 -> 100.100.100.10,22 PR tcp
01/08/1999 16:04:03.070127 STATE:EXPIRE 20.20.20.13,1019 -> 100.100.100.10,22 PR tcp Pkts 63 Bytes 4604
```

Widzimy zapytanie DNS z zewnętrznej maszyny do naszego serwera, dwa pingi `xntp` do dobrze znanych serwerów czasu i połączenie wychodzące `ssh` które trwało bardzo krótko.

`ipmon` jest również w stanie pokazać nam jakie pakiety są logowane. Na przykład, kiedy używamy stanów, często spotkasz pakiety takie jak ten:

```
# ipmon -o I
15:57:33.803147 ppp0 @0:2 b 100.100.100.103,443 -> 20.20.20.10,4923 PR tcp len 20 1488 -A
```

Co to oznacza? Pierwsze pole to oczywiście stempel czasu. Drugie jest również raczej oczywiste, to interfejs na którym wydarzyło się zdarzenie. Trzecie pole `@0:2` to coś, co ludzie zwykle pomijają. To oznaczenie reguły która spowodowała zdarzenie. Pamiętasz `'ipfstat -in'`? Jeśli chciałbyś wiedzieć co spowodowało zalogowanie pakietu, powinieneś obejrzeć regułę 2 w grupie 0. Czwarte pole, małe `'b'` mówi że pakiet został zablokowany, i generalnie będziesz to ignorował, chyba że logujesz również pakiety które przepuszczasz, co spowoduje pokazanie się literki `'p'`. Piąte i szóste pole nie wymagają chyba wyjaśnienia - mówią skąd pakiet przyszedł i gdzie miał dotrzeć. Siódme ( `PR` ) i ósme pole to protokół, a dziewiąte długość pakietu. Ostatnia część, `'-A'` to flagi które były ustawione; ten pakiet ma ustawioną flagę ACK. Dlaczego wspomniałem na początku stany? Ponieważ często w Internecie zdarzają się lagi, pakiety są regenerowane i czasami dostaniesz dwa takie same, co spowoduje stwierdzenie przez kod odpowiedzialny za śledzenie połączeń, że to pakiet należący do innego połączenia. Być może trafi on na regułę. Zwykle zobaczysz tutaj zalogowany ostatni pakiet sesji, ponieważ kod `'keep state'` wyrzuci połączenie zanim ostatni pakiet będzie miał szansę dotrzeć do Twojej ściany ogniowej. To normalne zachowanie, nie denerwuj się. Innym przykładem pakietu może być:

```
12:46:12.470951 x10 @0:1 S 20.20.20.254 -> 255.255.255.255 PR icmp len 20 9216 icmp 9/0
```

Jest to broadcast rozpoznawczy ICMP routera. Możemy to stwierdzić na podstawie typu ICMP: 9/0.

`ipmon` pozwala również na obejrzenie tabeli NAT:

```
# ipmon -o N
01/08/1999 05:30:02.466114 @2 NAT:RDR 20.20.20.253,113 <- -> 20.20.20.253,113 [100.100.100.13,45816]
01/08/1999 05:30:31.990037 @2 NAT:EXPIRE 20.20.20.253,113 <- -> 20.20.20.253,113 [100.100.100.13,45816] Pkts 10 Bytes 455
```

Widzimy tu przekierowanie do serwera identd, który oszukuje twierdząc, że udostępnia usługę ident dla maszyn za naszym NATem, ponieważ zwykle nie mogą one sobie same zapewnić tej usługi przy zwykłym NATcie.

## 8. Specyficzne zastosowania Filtra IP - rzeczy które nie pasowały wyżej, ale są warte wspomnienia

### 8.1 Utrzymywanie stanu oraz flagi i serwery

Utrzymywanie stanu jest fajną sprawą, ale bardzo łatwo jest popełnić błąd decydując w którą stronę będziemy to robić. Generalnie, będziesz chciał ustawić słowo kluczowe `'keep state'` przy pierwszej regule która wchodzi w interakcję z pakietami inicjującymi dany rodzaj połączenia. Jednym z najczęściej spotykanych błędów jest, w momencie łączenia śledzenia stanów z filtrowaniem według flag, jest tak jak poniżej:

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 flags S
pass out all keep state
```

Reguły wyraźnie umożliwiają tworzenie połączeń do serwera telnetu pod adresem `20.20.20.20` i odsyłanie odpowiedzi ze strony serwera. Jednak gdy spróbujesz użyć tych reguł, zadziałają ale na krótko. Ponieważ wpuszczamy tylko pakiety z ustawioną flagą SYN, pozycja w tabeli stanów nigdy nie zostanie dokończona, i po przekroczeniu domyślnego czasu na ustanowienie połączenia ( 60 sekund ) połączenie zostanie zamknięte.

Możemy rozwiązać ten problem, przepisując reguły na jeden z dwóch sposobów:

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 keep state
block out all
```

lub:

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 flags S keep state
pass out all keep state
```

Obie możliwości spowodują, że będzie możliwe nawiązanie pełnego połączenia oraz zapisanie go w tabeli stanów.

### 8.2 Radzenie sobie z FTP

FTP to jeden z protokołów, przy którym siadasz i zaczynasz się zastanawiać *'Co do cholery oni sobie myśleli?'*. FTP ma wiele problemów z którymi administrator musi

sobie poradzić. Co gorsza, problemy które administrator musi rozwiązać są różne dla klientów FTP i serwera FTP.

W obrębie protokołu FTP wyróżnia się dwa tryby transferu - **aktywny** i **pasywny**. **Aktywny** to ten, w którym serwer łączy się z otwartym portem na komputerze klienta i wysyła dane. Analogicznie, **pasywny** to taki, w którym to klient łączy się na otwarty port serwera i odbiera dane.

## Konfiguracja dla serwera FTP

Jeśli chodzi o obsłużenie aktywnych sesji FTP, zadanie jest proste. Jednocześnie skonfigurowanie obsługi pasywnych sesji FTP będzie dużym problemem. Najpierw zajmiemy się **aktywnym** FTP, później **pasywnym**. Generalnie, obsłużymy **aktywne** FTP tak jak przychodzące połączenia HTTP czy SMTP; po prostu otworzymy port ftp i pozwolimy regule z 'keep state' zrobić swoje:

```
pass in quick proto tcp from any to 20.20.20.20/32 port = 21 flags S keep state
pass out proto tcp all keep state
```

Powyższe reguły umożliwią nawiązywanie **aktywnych** połączeń FTP do Twojego serwera pod adresem 20.20.20.20.

Kolejnym wyzwaniem będzie obsłużenie **pasywnego** FTP. **Standardowo tak działają przeglądarki WWW, więc staje się to dosyć popularne i powinniśmy pomyśleć o tym poważnie. Problemem jest to, że dla każdego połączenia pasywnego, serwer musi zacząć nasłuchiwać na jakimś nowym porcie (zwykle powyżej portu numer 1023). Jest to coś w rodzaju tworzenia nowej usługi na serwerze. Zakładając że mamy dobrą ścianę ogniową, z domyślną zasadą blokowania, dostęp do tej nowej usługi (otwartego portu) zostanie zablokowany, a więc aktywne FTP nie będzie działało. Ale nie rozpaczaj. Jest nadzieja.**

Pierwszym co mogłaby zrobić osoba próbująca rozwiązać ten problem, to otworenie wszystkich portów powyżej 1023. Tak naprawdę, to zadziała:

```
pass in quick proto tcp from any to 20.20.20.20/32 port > 1023 flags S keep state
pass out proto tcp all keep state
```

Jest to jednak mało satysfakcjonujące. Przez przepuszczenie wszystkiego na porty powyżej 1023, tak naprawdę otwieramy się na wiele potencjalnych problemów. O ile porty 1-1023 zaprojektowano dla usług serwerowych, wiele programów używa portów wyższych niż 1023, na przykład nfsd czy Xy.

Dobre wiadomości są takie, że to twój serwer FTP decyduje, które porty zostaną otworzone by obsłużyć pasywne połączenie ftp. Oznacza to, że zamiast otwierać wszystkie porty powyżej 1023, możesz wybrać na przykład porty 15001-19999 na porty ftp i otwierać tylko ten zakres na swojej ścianie ogniowej. W przypadku serwera wu-ftp, wykonuje się to przez dodanie do pliku ftpaccess opcji passive ports. Proszę, sprawdź szczegóły przez wywołanie strony podręcznikowej do pliku ftpaccess. Ze strony ipfilter, wszystko co musimy zrobić to skonfigurować następujące reguły:

```
pass in quick proto tcp from any to 20.20.20.20/32 port 15000 >< 20000 flags S keep state
pass out proto tcp all keep state
```

Jeśli Cię to nie satysfakcjonuje, zawsze możesz dodać obsługę IPF w twoim serwerze FTP, lub odwrotnie.

## Konfiguracja dla klientów FTP

O ile obsługa serwerów FTP jest jeszcze w IPF daleka od doskonałości, obsługa klientów FTP działa dobrze od wersji 3.3.3. Tak samo jak w przypadku serwerów FTP, mamy dwa rodzaje połączeń - pasywne i aktywne.

Najprostszym trybem z punktu widzenia ściany ogniowej jest tryb pasywny. Zakładając, że stosujesz reguły z `keep state` dla wychodzących połączeń tcp, połączenia pasywne będą działały bez dalszych zabiegów. Jeśli jeszcze tego nie robisz, zastanów się nad poniższym:

```
pass out proto tcp all keep state
```

Drugi typ ruchu, aktywny, jest trochę bardziej problematyczny, ale również rozwiązany. Transfery aktywne otwierają na serwerze port dla przepływu danych do klienta.

Jest to normalnie problematyczne jeśli pośrodku istnieje ściana ogniowa, powstrzymująca połączenia wychodzące przed wracaniem. By to rozwiązać, ipfilter zawiera proxy ipnat, które tymczasowo otwiera dziurę w ścianie ogniowej po to by serwer FTP mógł przekazać dane klientowi. Nawet jeśli nie używasz ipnat, proxy będzie działało. Poniższe reguły to minimum tego co musisz dodać do konfiguracji ipnat ( ep0 powinieneś zamienić na nazwę interfejsu który podłączony jest do sieci zewnętrznej):

```
map ep0 0/0 -> 0/32 proxy port 21 ftp/tcp
```

Po więcej detali dotyczących proxy ipfilter, wróć do sekcji [Magia ukryta w NAT; proxy aplikacji](#). Dodatkowo, proxy ftp działa w `złą stronę` i może zostać użyte do obsługi serwera FTP za NATem, ale nie chcesz tego robić ze względów bezpieczeństwa. Naprawdę. To wielka dziura. Zajrzyj pod adres [http://www.false.net/ipfilter/2001\\_11/0273.html](http://www.false.net/ipfilter/2001_11/0273.html) by zobaczyć jak to naprawdę wygląda i dlaczego powinieneś o tym zapomnieć.

## 8.3 Zmienne kernela dotyczące tematu

Istnieje trochę rzeczy w kernelu, które albo muszą być ustawione by ipf działał, albo generalnie dobrze jest wiedzieć o ich istnieniu przy budowaniu ścian ogniowych. Pierwsza podstawowa rzecz to włączenie przekazywania IP, ponieważ w innym przypadku ipf będzie mógł zrobić niewiele, ponieważ stos IP nie będzie routować pakietów.

### IP Forwarding:

OpenBSD:

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

FreeBSD:

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

NetBSD:

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



Solaris:

```
ndd -set /dev/ip ip_forwarding 1
```

### **Zmiany dotyczące ustawień portów:**

OpenBSD:

```
net.inet.ip.portfirst = 25000
```

FreeBSD:

```
net.inet.ip.portrange.first = 25000  
net.inet.ip.portrange.last = 49151
```

NetBSD:

```
net.inet.ip.anonportmin = 25000  
net.inet.ip.anonportmax = 49151
```

Solaris:

```
ndd -set /dev/tcp tcp_smallest_anon_port 25000  
ndd -set /dev/tcp tcp_largest_anon_port 65535
```

Inne użyteczne wartości:

OpenBSD:

```
net.inet.ip.sourceroute = 0  
net.inet.ip.directed-broadcast = 0
```

FreeBSD:

```
net.inet.ip.sourceroute=0  
net.ip.accept_sourceroute=0
```

NetBSD:

```
net.inet.ip.allowsrct=0  
net.inet.ip.forwsrct=0
```

```
net.inet.ip.directed-broadcast=0
net.inet.ip.redirect=0
```

Solaris:

```
ndd -set /dev/ip ip_forward_directed_broadcasts 0
ndd -set /dev/ip ip_forward_src_routed 0
ndd -set /dev/ip ip_respond_to_echo_broadcast 0
```

Dodatkowo, FreeBSD ma pewne dodatkowe zmienne sysctl:

```
net.inet.ipf.fr_flags: 0
net.inet.ipf.fr_pass: 514
net.inet.ipf.fr_active: 0
net.inet.ipf.fr_tcpidletimeout: 864000
net.inet.ipf.fr_tcpclosewait: 60
net.inet.ipf.fr_tcpplastack: 20
net.inet.ipf.fr_tcptimeout: 120
net.inet.ipf.fr_tcpclosed: 1
net.inet.ipf.fr_udptimeout: 120
net.inet.ipf.fr_icmptimeout: 120
net.inet.ipf.fr_defnatage: 1200
net.inet.ipf.fr_ipfrttl: 120
net.inet.ipf.ipl_unreach: 13
net.inet.ipf.ipl_initied: 1
net.inet.ipf.fr_authsize: 32
net.inet.ipf.fr_authused: 0
net.inet.ipf.fr_defaultauthage: 600
```

## 9. Zabawa z ipf!

Ta sekcja być może nie nauczy Cię niczego nowego o ipf, ale może podjąć parę problemów do których sam jeszcze nie doszedłeś, lub skierować twój mózg w stronę wynalezienia czegoś interesującego o czym nie pomyśleliśmy.

### 9.1 Filtrowanie localhost

Dawno dawno temu, w bardzo dalekim uniwersytecie, Wietse Venema stworzył paczkę tcp-wrapper i odtąd, była ona dodawana jako dodatkowa warstwa ochronna do usług sieciowych na całym świecie. Bardzo fajna sprawa. Ale, tcp-wrappers mają problemy. Na początek, chronią tylko usługi TCP jak sugeruje nazwa. Po drugie, chronią tylko usługi uruchamiane z poziomu inetd lub po skompilowaniu programu z biblioteką libwrap. Powoduje to gigantyczne dziury w bezpieczeństwie twojej maszyny. Możemy je zakryć, przez użycie ipf w stosunku do lokalnej maszyny. Na przykład, mój laptop jest często podłączany bezpośrednio, lub wdzianiam się do sieci którym niezbyt ufam, a w związku z tym używam poniższego zestawu reguł:

```
pass in quick on lo0 all
```

```
pass out quick on lo0 all

block in log all
block out all

pass in quick proto tcp from any to any port = 113 flags S keep state
pass in quick proto tcp from any to any port = 22 flags S keep state
pass in quick proto tcp from any port = 20 to any port 39999 >< 45000 flags S keep state
pass out quick proto icmp from any to any keep state
pass out quick proto tcp/udp from any to any keep state keep frags
```

Wyglądały one tak już od dłuższego czasu i nie dotyczyły mnie żadne problemy w związku z tym, że używałem załadowanego na stałe ipf. Jeśli chciałem uszczelnić je jeszcze bardziej, mogłem zacząć stosować proxy FTP przez NAT i dodać trochę reguł by zabezpieczyć się przed preparowaniem pakietów. Ale tak skonfigurowany komputer jest dużo bardziej restrykcyjny w stosunku do sieci lokalnej niż zwykły komputer. Są one dobre w sytuacji, gdy masz maszynę która ma masę użytkowników, a chcesz być pewien, że żaden z nich nie uruchomi żadnej usługi której nie powinien. Nie zatrzyma to złośliwego hackera z dostępem do konta root'a przed poprawką w twoich regułach ipf i wystartowaniem usługi, ale powstrzyma większość ludzi, zapewni bezpieczeństwo twoim usługom nawet w podejrzanym sieci lokalnej. Duże zwycięstwo, moim zdaniem. Używanie filtrowania w odniesieniu do lokalnej maszyny, w połączeniu z mniej restryktywną 'główną ścianą ogniową' może rozwiązać wiele problemów wydajnościowych, jak również wiele politycznych koszmarów w stylu 'Dlaczego nie działa ICQ?', albo 'Dlaczego nie mogę uruchomić serwera WWW na mojej stacji roboczej? To przecież MOJA STACJA!!'. Kolejne zwycięstwo. Kto powiedział, że nie możemy zapewnić bezpieczeństwa i wygody jednocześnie?

## Jaka ściana ogniowa? Filtrowanie transparentne.

Jednym z podstawowych problemów przy stawianiu ściany ogniowej, jest integralność jej samej. Czy ktoś może włamać się na twoją ścianę ogniową i zmienić reguły? Jest to częsty problem przed którym stają administratorzy, szczególnie gdy używają ścian ogniowych opartych o maszyny Unix/NT. Niektórzy używają ich w formie rozwiązań sprzętowych, tzw. blackbox, sugerując się wrażeniem, że zamknięte systemy lepiej zabezpieczają sieć. Mamy lepszy sposób.

Wielu administratorów sieci zna zagadnienie **mostu ethernetowego** ( ang. *ethernet bridge* ). Jest to urządzenie które łączy dwa oddzielne segmenty ethernetowe by stworzyć z nich jeden. Most ethernetowy używany jest zwykle do połączenia dwóch budynków, zmieniania prędkości w sieci i przedłużenia maksymalnej długości okablowania. Ostatnie wersje Linuksa, OpenBSD, NetBSD i FreeBSD które zamieniają PeCety warte tysiące złotych w mosty warte setki! To co wszystkie mosty mają wspólnego, to fakt że znajdują się w połowie połączenia między dwoma maszynami, które nie wiedzą o jego istnieniu. Wchodzimy w świat ipfilter i OpenBSD.

Mostowanie ethernetu ma miejsce w warstwie drugiej modelu ISO. IP na warstwie trzeciej. IP Filter jest głównie zainteresowany warstwą trzecią, ale zajmuje się również warstwą drugą ponieważ ma dostęp do interfejsów. Poprzez połączenie IP Filter i urządzenia mostującego z OpenBSD, możemy stworzyć ścianę ogniową która jest zarówno niewidzialna jak i nieosiągalna. System nie potrzebuje adresu IP, nie musi nawet ujawniać swojego adresu ethernetowego. Jedynym znakiem że gdzieś jest filtr, mogą być trochę większe opóźnienia niż te generowane przez okablowanie kategorii piątej, a część pakietów po prostu nie dociera tam gdzie powinna.

Konfiguracja takiego rodzaju zestawu reguł jest zadziwiająco prosta. W OpenBSD, pierwsze urządzenie mostujące ma nazwę `bridge0`. Powiedzmy że mamy dwie karty sieciowe, `x10` i `x11`. By zamienić tą maszynę w most, wszystko co trzeba zrobić to wprowadzić następujące trzy komendy:

```
brconfig bridge0 add x10 add x11 up
ifconfig x10 up
ifconfig x11 up
```

W tym momencie, cały ruch przychodzący do `x10` jest wysyłany do `x11` i odwrotnie. Zauważ, że żadnemu z interfejsów nie przydzielono adresu IP, my również tego nie zrobiliśmy. Tak naprawdę, najlepiej tego nie robić.

Reguły zachowują się generalnie tak jak dotychczas. Mimo że istnieje urządzenie `bridge0`, nie filtrujemy pakietów w oparciu o nie. Reguły dalej oparte są o któryś z interfejsów, co sprawia że ważne jest która karta sieciowa jest podłączona do którego kabelka. Zaczniemy od podstawowego filtrowania by zilustrować to co się dzieje. Zauważmy, że twoja sieć wyglądała tak:

```
20.20.20.1 <-----> 20.20.20.0/24 koncentrator
```

To znaczy, że mamy ruter na `20.20.20.1` połączony do sieci `20.20.20.0/24`. Wszystkie pakiety z sieci `20.20.20.0/24` przechodzą przez `20.20.20.1` by wyjść do świata zewnętrznego i odwrotnie. Dodajemy teraz most ipf:

```
20.20.20.1 <-----/x10 IpfbBridge x11/-----> 20.20.20.0/24 koncentrator
```

Oraz następujący zestaw reguł na nim:

```
pass in quick all
pass out quick all
```

Z załadowanymi tymi regułami, funkcjonalność jest identyczna. Jeśli chodzi o ruter `20.20.20.1` i sieć `20.20.20.0/24` oba rysunki opisujące sieć są identyczne. Zmieńmy trochę reguły:

```
block in quick on x10 proto icmp
pass in quick all
pass out quick all
```

Nadal `20.20.20.1` i `20.20.20.0/24` myślą, że sieć jest identyczna, ale jeśli `20.20.20.1` spróbuje wykonać ping do `20.20.20.2` nie dostanie odpowiedzi. Co więcej, `20.20.20.2` nie dostanie w ogóle pakietu. ipfilter przechwyci pakiet zanim dotrze on do drugiego końca wirtualnego drutu. Filtr z mostem możemy postawić gdziekolwiek. Używając tej metody, możemy ograniczyć koło zaufania do pojedynczych maszyn, jeśli tylko starczy nam kart sieciowych :-).

Blokowanie icmp ze świata jest raczej śmieszne, szczególnie jeśli jesteś administratorem i lubisz pingować świat, wykonywać traceroute czy zmieniać swoje MTU. Skonstruujmy lepszy zestaw reguł by skorzystać z kluczowej zalety ipf: sprawdzania stanów.

```
pass in quick on x11 proto tcp keep state
pass in quick on x11 proto udp keep state
pass in quick on x11 proto icmp keep state
block in quick on x10
```

W tej sytuacji, sieć `20.20.20.0/24` ( może lepiej będzie ją nazywać siecią `x11` ) może teraz osiągnąć świat, ale świat nie może osiągnąć sieci i nie może nawet sprawdzić dłaczego. Router jest dostępny, maszyny są aktywne, ale świat nie może po prostu wejść. Nawet gdyby router został zaatakowany, ściana ogniowa nadal będzie aktywna i będzie działać poprawnie.

Na razie filtrowaliśmy tylko na podstawie interfejsu i protokołu. Mimo, że mostowanie wykonywane jest na warstwie drugiej, nadal możemy ograniczać dostęp na podstawie adresu IP. Zwykle mamy uruchomione parę usług, więc nasz zestaw reguł może wyglądać tak:

```
pass in quick on xl1 proto tcp keep state
pass in quick on xl1 proto udp keep state
pass in quick on xl1 proto icmp keep state
block in quick on xl1 # nie, wpuszczamy tylko tcp/udp/icmp proszę pana
pass in quick on xl0 proto udp from any to 20.20.20.2/32 port=53 keep state
pass in quick on xl0 proto tcp from any to 20.20.20.2/32 port=53 flags S keep state
pass in quick on xl0 proto tcp from any to 20.20.20.3/32 port=25 flags S keep state
pass in quick on xl0 proto tcp from any to 20.20.20.7/32 port=80 flags S keep state
block in quick on xl0
```

Mamy teraz sieć w której 20.20.20.2 jest serwerem nazw dla strefy, 20.20.20.3 obsługuje przychodzącą pocztę, a 20.20.20.7 jest serwerem WWW.

Mostowanie w wykonaniu IP Filter nie jest jednak doskonałe i musimy to przyznać.

Po pierwsze, zauważysz że wszystkie reguły używają kierunku `in`, zamiast kombinacji `in` i `out`. Dzieje się tak dlatego, że obecnie kierunek `out` nie jest zaimplementowany w OpenBSD. Oryginalnie chodziło o powstrzymanie spadków wydajności jeśli używało się wielu interfejsów. Prowadzi się prace nad przyspieszeniem pracy, ale nadal ten kierunek pozostaje niezaimplementowany. Jeśli naprawdę potrzebujesz tej funkcjonalności, może będziesz mógł pomóc pracując przy kodzie, lub spytać ludzi z OpenBSD jak mógłbyś pomóc.

Po drugie, używanie IP Filter z mostowaniem, sprawia że używanie funkcjonalności NAT nie jest zalecane, jeśli nie bezpośrednio niebezpieczne. Pierwszym problemem jest oznajmienie o obecności filtrującego mostu. Drugim problemem byłoby to, że most nie ma adresu IP który mógłby używać do maskarady, co spowoduje prawdopodobnie bałagan jeśli nie błąd kernel panic. Możesz, oczywiście, ustawić adres IP dla interfejsu wychodzącego by NAT działał, ale znikają wtedy zalety wynikające z mostowania.

## Używanie transparentnego filtrowania przy naprawie błędów w projektowaniu sieci

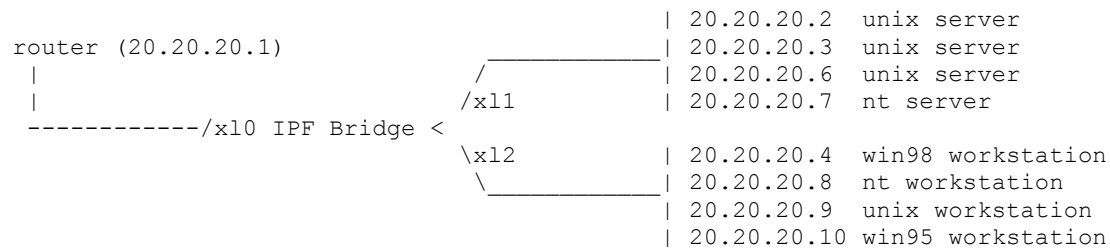
Wiele firm zaczęło używać IP zanim myśleli w ogóle o ścianach ogniowych czy podziale na podsieci. Mają teraz sieci wielkości klasy C czy nawet większe, które zawierają ich wszystkie serwery, stacje robocze, routery, maszyny do kawy i generalnie wszystko. Horror! Przenumerowanie, z poprawnym podziałem na podsieci, poziomy zaufania, filtry i tak dalej jest zarówno czasochłonne i kosztowne. Wydatek w postaci sprzętu i godzin roboczych ludzi już sam w sobie zwykle powstrzymuje większość firm przed rozwiązaniem tego problemu, nie mówiąc już nawet o okresie niedziałania sieci. Typowa problematyczna sieć wygląda jak poniżej:

20.20.20.1	router	20.20.20.6	unix server
20.20.20.2	unix server	20.20.20.7	nt workstation
20.20.20.3	unix server	20.20.20.8	nt server
20.20.20.4	win98 workstation	20.20.20.9	unix workstation
20.20.20.5	intelligent switch	20.20.20.10	win95 workstation

...tylko jest z 20 razy większa, zaśmiecona i w większości nieudokumentowana. W idealnej sytuacji, chciałbyś mieć wszystkie serwery w jednej podsieci, stacje robocze w drugiej a przełączniki ( ang. *switch* ) w trzeciej. Wtedy router filtrowałby pakiety pomiędzy podsieciami, dając stacjom roboczym ograniczony dostęp do serwerów, żadnego dostępu do przełączników, a tylko administrator miałby dostęp do maszynki do kawy. Nigdy nie widziałem sieci opartej o klasę C w takim porządku. IP Filter

može pomóc.

Na początek, rozdzielimy router, stacje robocze i serwery. Potrzebujemy dwóch koncentratorów ( lub dwa przełączniki ), które i tak prawdopodobnie mamy, oraz maszynę z zainstalowanym IP Filter i trzema kartami sieciowymi. Podłączymy wszystkie serwery do jednego huba, a stacje robocze do drugiego. Normalnie połączylibyśmy następnie oba koncentratory ze sobą, a potem do rutera. Zamiast tego, podłączymy router do interfejsu IPF x10, serwery do interfejsu x11, a stacje robocze do interfejsu x12. Diagram naszej sieci będzie wyglądał podobnie do tego:



Tam gdzie do tej pory nie było nic tylko kable połączeniowe, mamy most filtrujący który zapewnia nam, że nie trzeba modyfikować konfiguracji komputerów. Prawdopodobnie od razu włączyliśmy również mostowanie, więc sieć zachowuje się normalnie. Następnie, zaczynamy z zestawem reguł podobnym trochę do naszego ostatniego:

```

pass    in quick on xl0 proto udp from any to 20.20.20.2/32 port=53 keep state
pass    in quick on xl0 proto tcp from any to 20.20.20.2/32 port=53 flags S keep state
pass    in quick on xl0 proto tcp from any to 20.20.20.3/32 port=25 flags S keep state
pass    in quick on xl0 proto tcp from any to 20.20.20.7/32 port=80 flags S keep state
block   in quick on xl0
pass    in quick on xl1 proto tcp keep state
pass    in quick on xl1 proto udp keep state
pass    in quick on xl1 proto icmp keep state
block   in quick on xl1 # nie, wpuszczamy tylko tcp/udp/icmp proszę pana
pass    in quick on xl2 proto tcp keep state
pass    in quick on xl2 proto udp keep state
pass    in quick on xl2 proto icmp keep state
block   in quick on xl2 # nie, wpuszczamy tylko tcp/udp/icmp proszę pana

```

Ponownie, ruch nadchodzący ze strony rutera ograniczony jest do DNS'u, SMTP i HTTP. Na razie, serwery i stacje robocze nie mają ograniczeń w ruchu. Zależnie od rodzaju firmy, może być coś w dynamice sieci co Ci się nie podoba. Być może w ogóle nie chcesz by stacje robocze miały dostęp do serwerów? Wyrzucić reguły dla x12:

```
pass in quick on xl2 proto tcp keep state
pass in quick on xl2 proto udp keep state
pass in quick on xl2 proto icmp keep state
block in quick on xl2 # nie, wpuszczamy tylko tcp/udp/icmp proszę pana
```

i zamień je na:

```
block in quick on x12 from any to 20.20.20.0/24
pass in quick on x12 proto tcp keep state
pass in quick on x12 proto udp keep state
pass in quick on x12 proto icmp keep state
block in quick on x12 # nie, wpuszczamy tylko tcp/udp/icmp proszę pana
```

Być może chcesz by dostawały się tylko do serwerów by odebrać i wysłać swoją pocztę przez IMAP? Nic łatwiejszego:

```
pass in quick on x12 proto tcp from any to 20.20.20.3/32 port=25
pass in quick on x12 proto tcp from any to 20.20.20.3/32 port=143
block in quick on x12 from any to 20.20.20.0/24
pass in quick on x12 proto tcp keep state
pass in quick on x12 proto udp keep state
pass in quick on x12 proto icmp keep state
block in quick on x12 # nie, wpuszczamy tylko tcp/udp/icmp proszę pana
```

Teraz zarówno stacje robocze jak i serwery są chronione przed światem zewnętrznym, a serwery chronione są od stacji roboczych.

Być może prawdziwa jest sytuacja odwrotna, może chcesz by stacje robocze mogły mieć dostęp do serwerów, ale nie do świata zewnętrznego. W końcu, następna generacja exploit'ów działa na klientach a nie na serwerach. W tym przypadku, musisz zmienić swoje reguły dotyczące interfejsu x12 na:

```
pass in quick on x12 from any to 20.20.20.0/24
block in quick on x12
```

Teraz serwery mają wolną rękę, ale klienci nie mogą połączyć się do serwerów. Możemy obniżyć trochę obostrzenia dla serwerów:

```
pass in quick on x11 from any to 20.20.20.0/24
block in quick on x11
```

W połączeniu z tymi dwoma, klienci i serwery mogą wymieniać dane, ale żadne z nich nie może kontaktować się ze światem zewnętrznym ( pomimo tego, że świat zewnętrzny może dostać się do paru usług ). Cały zestaw reguł wyglądać będzie tak:

```
pass in quick on x10 proto udp from any to 20.20.20.2/32 port=53 keep state
pass in quick on x10 proto tcp from any to 20.20.20.2/32 port=53
pass in quick on x10 proto tcp from any to 20.20.20.3/32 port=25
pass in quick on x10 proto tcp from any to 20.20.20.7/32 port=80
block in quick on x10
pass in quick on x11 from any to 20.20.20.0/24
block in quick on x11
pass in quick on x12 from any to 20.20.20.0/24
block in quick on x12
```

Pamiętaj zatem, że jeśli twoja sieć to bałagan adresów IP i maszyn różnego przeznaczenia, most z transparentnym filtrowaniem może rozwiązać twój problem, z którym musiałbyś w innym przypadku żyć i być może, któregoś dnia zostałby on wykorzystany do włamania.

## Bezpieczne logowanie z komendami ``dup-to'` (zrzucić do) i ``to'` (do).

Do tej pory, używaliśmy filtrów do odrzucania pakietów. Zamiast je odrzucać, zastanówmy się nad przekazywaniem ich do innego systemu by móc zrobić z tymi informacjami coś bardziej użytecznego niż tylko logowanie za pomocą `ipmon`. Nasza ściana ogniowa, czy router czy most, może mieć tak dużo interfejsów jak dużo da się wsadzić do komputera. Możemy użyć tych informacji by stworzyć bezpieczne miejsce zbierania dla naszych pakietów. Dobrym przykładem byłaby implementacja sieci do wykrywania intruzów. Na początek, byłoby dobrze ukryć obecność systemów wykrywania intruzów przed światem, tak by nie mogły zostać wykryte.

Zanim zaczniemy, są pewne charakterystyki operacyjne o których musimy wiedzieć. Jeśli będziemy mieli do czynienia z pakietami, które zostały zablokowane, możemy używać zarówno słowa kluczowego `to` jak i `fastroute` (różnice omówimy później). Jeśli zamierzamy przepuszczać pakiety tak jak normalnie, musimy tworzyć kopię pakietów dla naszej sieci logującej przez użycie słowa kluczowego `dup-to`.

### Metoda ``dup-to'`

Jeśli, na przykład, chcemy wysłać kopię wszystkiego co wychodzi przez interfejs `x13` do naszej sieci podłączonej do interfejsu `ed0`, możemy wstawić taką regułę:

```
pass out on x13 dup-to ed0 from any to any
```

Możesz również mieć potrzebę wysłania tego pakietu do konkretnego adresu IP w Twojej sieci, zamiast tylko wysłać kopię i liczyć na to, że wszystko pójdzie dobrze. By to wykonać, zmodyfikujemy trochę regułę:

```
pass out on x13 dup-to ed0:192.168.254.2 from any to any
```

Zwróć uwagę, że ta reguła spowoduje zmianę adresu przeznaczenia kopiowanego pakietu, co może zanegować użyteczność logowania. Dlatego, zalecamy tą wersję reguły jeśli jesteś pewien co do przeznaczenia logowanych pakietów (np. nie używaj `192.168.254.2` dla logowania pakietów zarówno przeznaczonych dla serwera WWW jak i serwera poczty, ponieważ nie będziesz wiedział który pakiet miał dotrzeć do gdzie).

Ta technika może być użyta całkiem efektywnie jeśli będziesz używał adresu IP w swojej sieci bezpieczeństwa tak jak traktowałbyś grupy rozgłaszania w prawdziwym internecie (tzn. `192.168.254.2` może być kanałem dla analizy ruchu do HTTP, `23.23.23.23` kanałem dla sesji telnet i tak dalej). Nie musisz mieć nawet tych adresów czy aliasów faktycznie ustawionych dla któregośkolwiek adresu z sieci bezpieczeństwa. Normalnie, `ipfilter` musiałby używać ARP dla adresów nowego przeznaczenia (przy użyciu czegoś w stylu `ed0:192.168.254.2`), ale możemy temu zapobiec przez stworzenie statycznych wpisów w tablicy ARP dla każdego 'kanału' na naszym komputerze z `ipfilter`.

Generalnie, ``dup-to ed0'` to wszystko co jest wymagane by otrzymać nową kopię pakietu w naszej sieci bezpieczeństwa, dla celów logowania lub badań.

### Metoda ``to'`

Metoda ``dup-to'` ma jedną wadę. Ponieważ ma wykonać kopię pakietu i ewentualnie zmienić jego adres docelowy, musi potrwać chwila zanim będzie gotowa zająć się następnym pakietem.

Jeśli nie obchodzi nas wysyłanie pakietu do normalnego systemu a i tak mamy go zablokować, możemy używać słowa kluczowego ``to'` by przepchnąć ten pakiet przez



proces normalnego routingu i zmusić go by wyszedł innym interfejsem niż normalnie.

```
block in quick on xl0 to ed0 proto tcp from any to any port < 1024
```

Używamy 'block quick' dla routingu na interfejsie 'to', ponieważ podobnie jak 'fastroute', kod interfejsu 'to' wygeneruje dwie ścieżki dla pakietu jeśli użyjemy 'pass' a związku z tym prawdopodobnie spowoduje **kernel panic**.

## 10. Filtrowanie dziwnych sieci; najlepsze wyjście w aktualnych technologiach przeciwdziałających preparowaniu pakietów

Spędziliśmy trochę czasu śledząc szerokie zakresy adresów IP które zostały zarezerwowane przez IANA z różnych powodów, lub które nie były używane w momencie gdy pisano ten dokument. Ponieważ żadnego z tych adresów nie powinno się używać, nie powinien z niego wychodzić żaden ruch, jak również nie powinniśmy wysyłać tam niczego, prawda? Właśnie!

Więc, bez dodatkowych komentarzy, lista dziwnych sieci:

```
#
# s/OUTSIDE/interfejs-zewnetrzny (np: fxp0)
# s/MYNET/adres-sieci-w-formacie-CIDR (np: 1.2.3.0/24)
#
block in on OUTSIDE all
block in quick on OUTSIDE from 0.0.0.0/7 to any
block in quick on OUTSIDE from 2.0.0.0/8 to any
block in quick on OUTSIDE from 5.0.0.0/8 to any
block in quick on OUTSIDE from 10.0.0.0/8 to any
block in quick on OUTSIDE from 23.0.0.0/8 to any
block in quick on OUTSIDE from 27.0.0.0/8 to any
block in quick on OUTSIDE from 31.0.0.0/8 to any
block in quick on OUTSIDE from 69.0.0.0/8 to any
block in quick on OUTSIDE from 70.0.0.0/7 to any
block in quick on OUTSIDE from 72.0.0.0/5 to any
block in quick on OUTSIDE from 82.0.0.0/7 to any
block in quick on OUTSIDE from 84.0.0.0/6 to any
block in quick on OUTSIDE from 88.0.0.0/5 to any
block in quick on OUTSIDE from 96.0.0.0/3 to any
block in quick on OUTSIDE from 127.0.0.0/8 to any
block in quick on OUTSIDE from 128.0.0.0/16 to any
block in quick on OUTSIDE from 128.66.0.0/16 to any
block in quick on OUTSIDE from 169.254.0.0/16 to any
block in quick on OUTSIDE from 172.16.0.0/12 to any
block in quick on OUTSIDE from 191.255.0.0/16 to any
block in quick on OUTSIDE from 192.0.0.0/19 to any
block in quick on OUTSIDE from 192.0.48.0/20 to any
block in quick on OUTSIDE from 192.0.64.0/18 to any
```

```

block in quick on OUTSIDE from 192.0.128.0/17 to any
block in quick on OUTSIDE from 192.168.0.0/16 to any
block in quick on OUTSIDE from 197.0.0.0/8 to any
block in quick on OUTSIDE from 201.0.0.0/8 to any
block in quick on OUTSIDE from 204.152.64.0/23 to any
block in quick on OUTSIDE from 224.0.0.0/3 to any
block in quick on OUTSIDE from MYNET to any
# tutaj Twoje reguły przepuszczające...

block out on OUTSIDE all
block out quick on OUTSIDE from !MYNET to any
block out quick on OUTSIDE from MYNET to 0.0.0.0/7
block out quick on OUTSIDE from MYNET to 2.0.0.0/8
block out quick on OUTSIDE from MYNET to 5.0.0.0/8
block out quick on OUTSIDE from MYNET to 10.0.0.0/8
block out quick on OUTSIDE from MYNET to 23.0.0.0/8
block out quick on OUTSIDE from MYNET to 27.0.0.0/8
block out quick on OUTSIDE from MYNET to 31.0.0.0/8
block out quick on OUTSIDE from MYNET to 69.0.0.0/8
block out quick on OUTSIDE from MYNET to 70.0.0.0/7
block out quick on OUTSIDE from MYNET to 72.0.0.0/5
block out quick on OUTSIDE from MYNET to 82.0.0.0/7
block out quick on OUTSIDE from MYNET to 84.0.0.0/6
block out quick on OUTSIDE from MYNET to 88.0.0.0/5
block out quick on OUTSIDE from MYNET to 96.0.0.0/3
block out quick on OUTSIDE from MYNET to 127.0.0.0/8
block out quick on OUTSIDE from MYNET to 128.0.0.0/16
block out quick on OUTSIDE from MYNET to 128.66.0.0/16
block out quick on OUTSIDE from MYNET to 169.254.0.0/16
block out quick on OUTSIDE from MYNET to 172.16.0.0/12
block out quick on OUTSIDE from MYNET to 191.255.0.0/16
block out quick on OUTSIDE from MYNET to 192.0.0.0/19
block out quick on OUTSIDE from MYNET to 192.0.48.0/20
block out quick on OUTSIDE from MYNET to 192.0.64.0/18
block out quick on OUTSIDE from MYNET to 192.0.128.0/17
block out quick on OUTSIDE from MYNET to 192.168.0.0/16
block out quick on OUTSIDE from MYNET to 197.0.0.0/8
block out quick on OUTSIDE from MYNET to 201.0.0.0/8
block out quick on OUTSIDE from MYNET to 204.152.64.0/23
block out quick on OUTSIDE from MYNET to 224.0.0.0/3
# tutaj Twoje reguły przepuszczające...

```

Jeśli zamierzasz tego użyć, sugerujemy zapoznanie się z [whois.arin.net](http://whois.arin.net) i okresowe sprawdzanie tych adresów, ponieważ IANA nie powiadomi Cię jeśli przyzna któryś z nich dla nowych firm czy czegokolwiek innego. Zostałeś ostrzeżony.

## 11. Uwagi od tłumacza

Chciałbym podziękować następującym osobom za uwagi co do tłumaczenia, poprawki i zasugerowanie poprawek:

**dereck (at) box43.pl**

- literówki i dziwna zapaść w połowie zdania :)