

# Filtr Pakietów OpenBSD HOWTO

**Wouter Coene**

Wersja oryginału: version 20020405.2, generated April 5, 2002

Oryginał tego dokumentu znajduje się pod adresem: <http://www.inebriated.demon.nl/pf-howto/>

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

Wersja tłumaczenia: Wersja 1.3, 2002/09/03 15:04:13

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

## 1. Wprowadzenie

Filtr pakietów OpenBSD ( OpenBSD PF) jest paczką ściany ogniowej potrafiącą śledzić stany połączeń, włączoną jako część kernela OpenBSD od wersji OpenBSD 3.0. Ten dokument opisuje jak uruchomić i zarządzać zestawem reguł PF i mapowań NAT.

### 1.1 Do kogo adresowany jest ten dokument

Dokument ten adresowany jest do administratorów sieci i systemów, z przynajmniej podstawową znajomością sieci i protokołów sieciowych. Wiedza o innych systemach ścian ogniowych nie jest wymagana, ale może pomóc w opanowaniu bardziej skomplikowanych tematów.

### 1.2 Status

Dokument jest ciągle rozbudowywany, więc nie wszystko jeszcze jest opisane.

Na wykonanie czeka:

- Network Address Translation
- IPv6
- więcej informacji o poleceniu 'pfctl'
- AuthPF

### 1.3 Prawa autorskie i licencja

The OpenBSD Packet Filter HOWTO version 20020405.2  
Copyright (C) Wouter Coene, 2001, 2002.

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

- \* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- \* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- \* The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.
- \* This software may not be mass-distributed for sale without informing the author at least two weeks in advance.

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

## 1.4 Kontakt z autorem

Możesz skontaktować się z Wouter Coene przez e-mail pisząc na adres [wouter\[at\]inebriated.demon.nl](mailto:wouter[at]inebriated.demon.nl), lub zwykłą pocztą:

De Deel 17  
3471 EN Kamerik  
The Netherlands

## 2. Podstawy ścian ogniowych

Ściana ogniowa ma za zadanie chronić twoją sieć przed potencjalnymi atakami pochodzącymi z innej sieci. Wykonuje to zadanie poprzez sprawdzanie pakietów krążących między tymi dwoma sieciami i ograniczaniu ruchu na podstawie typów, celów a nawet zawartości tych pakietów.

Sekcja ta wyjaśnia jak rozpocząć pracę ze ścianą ogniową przy użyciu PF. Na użytek tej sekcji używać będę firmowej sieci i ściany ogniowej jako przykładu. Sieć jest

typową siecią TCP/IP z adresem 260.250.1.0/24. Pod adresem 260.250.1.3 pracuje firmowy serwer WWW, a ściana ogniowa ma dwa interfejsy, x10 i x11. Sieć firmowa podłączona jest do interfejsu x11 a połączenie do internetu do interfejsu x10.

## 2.1 Podstawy reguł

Komponent PF odpowiedzialny za ścianę ogniową używa zestawu reguł do opisanie akcji, które zostaną podjęte w stosunku do określonych pakietów. Reguły te czytane są z pliku i ładowane do kernela OpenBSD przy użyciu komendy `pfctl` (po więcej informacji dotyczących ładowania zestawu reguł, odsyłam do sekcji [Ładowanie zestawu reguł](#)).

Taki plik z regułami może wyglądać tak:

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

Popatrzmy co się tu dzieje. Pierwsza reguła mówi PF by zablokować wszystkie przychodzące pakiety. W przeciwieństwie do innych ścian ogniowych, PF nie przerywa przeglądania reguł gdy znajdzie pasującą. Zamiast tego, zaznacza to, że pakiet ma być zablokowany, ale przechodzi do następnej reguły.

Następna reguła mówi PF by wpuścić wszystkie przychodzące pakiety skierowane do dowolnego komputera. Ponownie, PF notuje że pakiet ma zostać przepuszczony i przechodzi do następnej reguły.

Ponieważ nie ma już następnej reguł, PF sprawdza co zamierzał zrobić. W tym przypadku, ostatnia pasująca reguła mówiła o przepuszczeniu pakietu, więc dokładnie to robi.

Cóż, nie wygląda to zbyt użytecznie, więc spróbujmy czegoś bardziej interesującego. Zezwólmy na dostęp do firmowego serwera WWW pracującego pod adresem 260.250.1.3. Możesz spróbować czegoś takiego:

```
block in all
pass  in from any to 260.250.1.3/32
```

Ponownie, pierwsza reguła mówi PF, że powinien zablokować ten pakiet.

Druga reguła mówi PF żeby przepuścić pakiety, które skierowane są do 260.250.1.3, gdzie `/32` oznacza że PF powinien dopasować adres w pełnym, 32 bitowym zakresie.

Ale co z ruchem powrotnym, mógłbyś spytać. Cóż, jest to raczej proste, po prostu pozwalamy na ruch z 260.250.1.3 wszędzie.

```
block in all
pass  in from any to 260.250.1.3/32
pass  in from 260.250.1.3/32 to any
```

## 2.2 Trochę bardziej zaawansowane zestawy reguł

Przypuśćmy że zdecydowano, że kolejny zestaw stron zostanie uruchomiony na firmowym serwerze WWW, ale zawiera delikatne informacje firmowe, więc nie powinien być dostępny z zewnątrz. Serwer ten, w przeciwieństwie do poprzedniego, uruchomimy na niestandardowym porcie 8000 TCP.

Teraz nie będziesz filtrował tylko na podstawie adresu docelowego, ale również na podstawie portu TCP, by upewnić się że nikt spoza firmowej sieci nie połączy się z portem 8000:

```
block in all
pass in proto tcp from any to 260.250.1.3/32 port = 80
pass in proto tcp from 260.250.1.3/32 port = 80 to any
```

Jak widzisz, nie filtrujemy jedynie na podstawie portu, ale wskazujemy również, że przejść mogą pakiety tylko należące do protokołu TCP.

## 2.3 Utrzymywanie stanu/śledzenie połączeń

Komponent odpowiedzialny za obsługę ściany ogniowej, potrafi zapamiętać jakie sesje TCP, UDP i ICMP zostały stworzone i potrafi filtrować pakiety na podstawie informacji z tej tabeli. Nazywamy to **utrzymywaniem stanu/śledzeniem połączeń** ( ang. *keeping state* ).

W momencie w którym PF zobaczy regułę mówiącą o utworzeniu stanu i śledzeniu tego połączenia, tworzy nowy wiersz w tabeli, bazując na informacjach z pakietu. Powoduje to, że następne pakiety będą przepuszczane bez przechodzenia przez fazę sprawdzania.

Śledzenie połączeń składa się z bardzo drobiazgowego sprawdzania numerów sekwencyjnych z informacjami zapisanymi w tabeli stanów i odrzucania tych, które nie pasują do niej, co zmniejsza prawdopodobieństwo że komputery za ścianą ogniową ze słabą implementacją TCP staną się ofiarą ataku TCP. W procesie śledzenia stanu dla sesji UDP, PF umożliwia pojedynczemu pakietowi odpowiedzi przejście przez ścianę ogniową z powrotem, w ramach pakietu który stworzył wiersz w tabeli stanów.

Przypuśćmy, że pracownicy naszej firmy, korzystający z komputerów w sieci firmowej, powinni móc oglądać strony WWW. Wymaga to przepuszczania zarówno połączeń TCP na port 80, jak również przepuszczania zapytań DNS na port 53.

Przy użyciu silnika śledzącego połączenia PF, możemy zapewnić bezpieczną pracę takiej konfiguracji. Przy okazji, możemy wykorzystać śledzenie połączeń, do kontrolowania połączeń do naszego firmowego serwera WWW, dzięki czemu zwiększamy bezpieczeństwo konfiguracji:

```
block in all
pass in proto udp from 260.250.1.0/24 to any port = 53 keep state
pass in proto tcp from 260.250.1.0/24 to any port = 80 keep state
pass in proto tcp from any to 260.250.1.3/32 port = 80 keep state
```

Trzecia reguła zezwala komputerom w sieci wewnętrznej na połączenia HTTP do zewnętrznych serwerów, przez poinstruowanie PF by tworzył nowe wpisy w tabeli stanów dla każdego z tych połączeń. Ostatnia reguła mówi to samo, ale dla połączeń z sieci zewnętrznej do naszego firmowego serwera WWW, co sprawia że reguła zezwalająca na ruch w odwrotną stronę nie jest już potrzebna.

Używanie mechanizmu utrzymywania stanów i śledzenia połączeń wydaje się obciążać ścianę ogniową dodatkową pracą i w wyniku tego zmniejszeniem przepustowości. Co ciekawe jednak, w wykonaniu PF sprawdzanie tabeli stanów jest szybsze niż sprawdzanie reguł. Typowy zestaw z 50 regułami to około 50 porównań, a tabela stanów z 50,000 wpisów to tylko około 16 porównań, co wynika z konstrukcji tabeli - w postaci binarnego drzewa. Ta informacja, w połączeniu ze zwiększonym bezpieczeństwem sprawia, że powinieneś używać śledzenia połączeń i utrzymywania stanów nawet dla prostych zadań, które łatwo można zrealizować bez tego mechanizmu.

## 2.4 Wylamywanie się z zestawu reguł: słowo kluczowe 'quick'

Czasami przydatne byłoby, gdyby PF przestawał w pewnym momencie przeglądać reguł i zrobił coś na podstawie tego czy pakiet pasuje do reguły czy nie. Można to zrealizować za pomocą słowa kluczowego 'quick'. Reguła z tym słowem, pasująca do pakietu, spowoduje przerwanie przeglądania reguł.

Jest to niebywale przydatne przy ochronie twojej sieci przed sfałszowanymi pakietami, tak jak pokazuje to poniższy przykład:

```
block in all
block in quick from 10.0.0.0/8 to any
block in quick from 172.16.0.0/12 to any
block in quick from 192.168.0.0/16 to any
block in quick from 255.255.255.255/32 to any
pass in all
```

Ten zestaw reguł mówi PF by natychmiast odrzucał pakiety z 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16 oraz z 255.255.255.255/32. Pozostałe pakiety zostaną przepuszczone.

Skutkuje to również przyspieszeniem przetwarzania reguł, które sprawdzane są z dużym ruchem, jeśli zostanie użyte poprawnie.

## 2.5 Wskazywanie interfejsów sieciowych

Możliwe jest również sprawdzanie interfejsów sieciowych, którym pakiet dotarł do systemu. Zaadaptujmy nasze poprzednie reguły sprawdzające pakiety, do nowej funkcjonalności, zachowując strukturę naszej sieci firmowej:

```
block in all
block in quick on x10 from 10.0.0.0/8 to any
block in quick on x10 from 172.16.0.0/12 to any
block in quick on x10 from 192.168.0.0/16 to any
block in quick on x10 from 255.255.255.255/32 to any
pass in all
```

## 2.6 Sprawdzanie flag TCP

By umożliwić blokowanie nieprawidłowych pakietów, możesz poinstruować PF by filtrował na podstawie flag TCP. Używa się do tego słowa kluczowego 'flags', po którym następuje lista flag które mają być ustawione i ewentualnie opcjonalny słasz po którym następuje maska flag.

PF dla każdego pakietu TCP, zamaskuje flagi które zostały wskazane, a następnie sprawdzi te które powinny być ustawione. W związku z tym `flags S/SA` mówi, że z flag `SYN` i `ACK`, tylko flaga `SYN` powinna być ustawiona.

PF rozpoznaje następujące flagi:

## **F**

FIN, dla zamykania połączeń

## **S**

SYN, dla otwierania połączeń

## **R**

RST, dla resetowania połączeń

## **P**

PSH, dla upewniania się że wszystkie dane dotarły

## **A**

ACK, dla potwierdzania pakietów

## **U**

URG, by zaznaczyć że pakiet jest pilny

Na przykład, pakiet żądający nowego połączenia, ma ustawioną tylko flagę `SYN`, a pakiet potwierdzający połączenie ma ustawione zarówno `SYN` jak i `ACK`. Z kolei pakiet oznaczający, że żądanie o połączenie zostało odrzucone ma ustawione flagi `ACK` i `RST`.

Użycie niepoprawnej kombinacji flag TCP jest bardzo popularną metodą skanowania komputerów by sprawdzić otwarte porty. Przez użycie słowa `flags` możesz obronić swój system przed takimi skanami i zmusić skanery portów do użycia technik, które są dużo prostsze do wykrycia.

Zmodyfikujmy nasz przykład używający śledzenia połączeń z powyższej sekcji. Chcemy wymusić, by tylko pakiety TCP, które spośród flag `SYN` i `ACK` mają ustawioną flagę `SYN`, przechodziły i tworzyły nowe wpisy w tabeli stanów:

```
block in all
pass in proto udp from 260.250.1.0/24 to any port = 53 keep state
```

```
pass in proto tcp from 260.250.1.0/24 to any port = 80 \
    flags S/SA keep state
pass in proto tcp from any to 260.250.1.3/32 port = 80 \
    flags S/SA keep state
```

Powinno to zapobiec technikom skanowania opisanym powyżej, przed przejściem przez naszą ścianę ogniową.

## 2.7 Zestawy

Oprócz podawania pojedynczych adresów źródłowych i docelowych, możemy również wskazywać zestawy komputerów. Wykonuje się to przez otoczenie ich w nawiasy klamrowe, a same adresy rozdziela się przecinkami.

Jeśli twój stary zestaw reguł wyglądał tak:

```
block in quick on x10 from 10.0.0.0/8 to any
block in quick on x10 from 172.16.0.0/12 to any
block in quick on x10 from 192.168.0.0/16 to any
block in quick on x10 from 255.255.255.255/32 to any
```

Możesz teraz napisać:

```
block in quick on x10 from { 10.0.0.0/8, 172.16.0.0/12, \
    192.168.0.0/16, 255.255.255.255/32 } to any
```

Możemy ten zapis wykorzystać również do wskazania interfejsów, protokołów i portów. Program `pfctl` rozdzieli takie reguły na pojedyncze dla każdej kombinacji, a dzięki temu PF może zoptymalizować Twój zestaw reguł używając techniki opisanej w sekcji [Optymalizacja zestawu reguł: pomijanie](#). Dodatkowo, zwiększa to czytelność reguł w przypadku wielu zestawów komputerów, interfejsów, protokołów czy portów.

## 2.8 Rozwijanie zmiennych

PF wspiera również rozwijanie zmiennych, na zasadach takich jak **powłoka** ( ang. *shell* ). Zmienne definiowane są przez przypisanie im wartości, a rozwija się je przez poprzeczenie ich nazwy znakiem dolara ( ``$'` ):

```
webserver="260.250.1.3/32"
pass in from any to $webserver port = 80 keep state
```

Wartość zmiennej musi znajdować się w cudzysłowie.

## 2.9 Optymalizacja zestawu reguł: pomijanie

W przeciwieństwie do IPFilter, PF z OpenBSD nie obsługuje słowa kluczowego `'group'`. Twórcy PF wybrali schemat zwany **pomijaniem** ( ang. *skip* ), w którym zestaw reguł optymalizowany jest automatycznie.

Załóżmy, że Twój zestaw reguł wygląda tak:

```
block in quick on x10 from 10.0.0.0/8 to any
block in quick on x10 from 172.16.0.0/12 to any
block in quick on x10 from 192.168.0.0/16 to any
block in quick on x10 from 255.255.255.255/32 to any
```

Dla każdego pakietu przychodzącego, zestaw reguł sprawdzany jest od góry do dołu. Wyobraź sobie pakiet, który dotarł interfejsem `x11`. Sprawdzana jest pierwsza reguła, która nie pasuje. Ponieważ wszystkie pozostałe reguły dotyczą pakietów z interfejsu `x10` PF może je pominąć.

Kiedy ładujesz zestaw reguł, następujące parametry porównywane pomiędzy kolejnymi regułami ( w tej kolejności ):

1. interfejs
2. protokół
3. adres źródłowy
4. port źródłowy
5. adres docelowy
6. port docelowy

Dla każdej reguły, PF automatycznie wylicza tak zwane **kroki do pominięcia** ( ang. *skip step* ), dla każdego z tych parametrów. Parametry te mówią PF ile reguł ma te same parametry.

Jeśli nadchodzący pakiet dociera do interfejsu `x11` i sprawdzany jest przez nasz zestaw reguł, PF stwierdza, że pakiet nie pasuje do pierwszej reguły, a ponieważ następne 3 również nie dotyczą tego interfejsu, pomija je.

Chcąc zmaksymalizować wydajność swojego zestawu reguł, powinieneś posortować go według: interfejsów, protokołu, adresu źródłowego, potem portu, w końcu adresu docelowego i również portu.

## 2.10 Składanie tego wszystkiego razem

Poskładajmy kawałki zestawu reguł, które do tej pory poznaliśmy. Poniżej wynik:

```
# ustawiamy trochę zmiennych
external="x10"
internal="x11"
corporate="260.250.1.0/24"
webserver="260.250.1.3/32"
spoofed="{ 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, \
          255.255.255.255/32 }"
```

```
# domyślnie blokujemy wszystko
block in all

# zezwalamy na oglądanie stron WWW z sieci firmowej
pass in quick on $internal proto udp from $corporate to any \
                                     port = 53 keep state
pass in quick on $internal proto tcp from $corporate to any \
                                     port = 80 flags S/SA keep state

# odrzucamy sfałszowane pakiety
block in quick on $external from $spoofed to any

# zezwalamy na dostęp do firmowego serwera WWW z Internetu
pass in quick on $external proto tcp from any to $webserver \
                                     port = 80 flags S/SA keep state
```

Masz - bezpieczna ściana ogniowa. Oczywiście, to nie wystarczy by zabezpieczyć sieć firmową. Ale to pierwsza linia obrony i zdaje się dobrze wykonywać swoją robotę.

## 2.11 Ładowanie zestawu reguł

Kiedy już jesteś szczęśliwy ze swoim zestawem reguł, możesz je zapisać jako `/etc/pf.conf` i zresetować maszynę, albo napisać:

```
pfctl -R /etc/pf.conf
```

Aby OpenBSD ładował automatycznie reguły, dodaj linijkę `pf=YES` w pliku `/etc/rc.conf`.

## 3. Filtrujące mosty

Most to urządzenie sieciowe, które łączy dwa lub więcej segmentów sieciowych razem. Zwykle to pojedynczy komputer, który replikuje przychodzące pakiety z jednego segmentu do drugiego. OpenBSD może również zostać użyty jako most, co sprawia że możliwe jest filtrowanie ruchu pomiędzy segmentami sieci.

Ta sekcja opisuje jak skonfigurować filtrujący most przy użyciu OpenBSD i PF. Sieci użyte w przykładzie to sieć uniwersytecka używająca IPv4: `10.0.0.0/8` składająca się z dużej ilości stacji studenckich, serwer WWW na `10.0.0.1/32` oraz serwer powłoki na `10.0.0.2/32`.

Ze względów bezpieczeństwa, pracownicy uniwersytetu chcieliby oddzielić dwa segmenty i filtrować ruch pomiędzy nimi, oczywiście z minimalnymi zmianami dla topologii sieci.

Wybrano serwer OpenBSD, z siecią studencką podpiętą do interfejsu `x10` i siecią serwerów podpiętą do `x11`.

### 3.1 Dwa kierunki

Pakiety przechodzące przez most przechodzą również dwukrotnie przez PF: najpierw wchodzi jednym interfejsem a potem wychodzą drugim. Nasz zestaw reguł musi zatem zezwolić na ruch wchodzący jak i wychodzący, co sprawia, że zaczniemy od następujących reguł:

```
pass in  on x10 any
pass out on x10 any
pass in  on x11 any
pass out on x11 any
```

Zezwoli to na wpuszczanie ruchu z interfejsów `x10` i `x11`, oraz na wypuszczanie go do właściwego segmentu.

## 3.2 Filtrowanie ze sprawdzaniem stanów

Filtr Pakietów OpenBSD ma bardzo fajną właściwość, nazywaną sprawdzaniem stanów, opisaną szerzej w sekcji [Utrzymywanie stanów/śledzenie połączeń](#). Użyjemy jej na naszym przykładzie by zwiększyć bezpieczeństwo segmentu serwerów sieciowych.

Jest jednak jedna rzecz, o której musisz pamiętać w przypadku filtrowania ze sprawdzaniem stanów: pozycje w tabeli stanów indeksowane są według klucza składającego się z adresu źródłowego, docelowego oraz portów TCP, w których kolejność tych par jest bez znaczenia. Jeśli pakiet wychodzący z A do B tworzy pozycję w tabeli stanów, PF przepuści pakiety wychodzące z A do B, oraz pakiety przychodzące z B do A. Zablokuje jednak pakiety wychodzące z B do A, oraz przychodzące z A do B, co w przypadku braku mostowania jest przypadkiem jasnym i zrozumiałym.

W przypadku mostu, pakiet przechodzi przez PF dwukrotnie; pakiet przychodzi jednym interfejsem i wychodzi drugim.

Są dwa rozwiązania. Pierwsze, to stworzenie dwóch wpisów w tabeli stanów dla każdego połączenia, przez użycie dwóch reguł z opcją `'keep state'`. Zwiększa to jednak obciążenie serwera mostującego i generalnie nie zaleca się robienia czegoś takiego, ponieważ jest inna opcja, dużo prostsza i elegancka.

Z perspektywy PF, pakiety przechodzą przez most dwukrotnie. Jeśli patrzysz na jeden interfejs, widzisz dokładnie taki sam ruch jak na drugim, zmieniają się jedynie kierunki. Możesz więc pominąć filtrowanie ruchu na jednym z interfejsów i filtrować tylko po jednej stronie.

Oczywiście chcemy śledzić połączenia zarówno do serwera WWW jak i serwera powłoki, a ponieważ najmniej ufamy sieci studentów, filtrować będziemy na interfejsie `x10`, przepuszczając cały ruch na interfejsie `x11`:

```
web="10.0.0.1/32"
shell="10.0.0.2/32"

pass  in  quick on x11 all
pass  out quick on x11 all

block in  on x10 all
block out on x10 all

pass  in  quick on x10 proto tcp from any to $web \
                                port = 80 keep state
pass  in  quick on x10 proto tcp from any to $shell \
```

```
port = { 22, 23 } keep state
```

## 4. Sztuczki z filtrowaniem pakietów

By zwiększyć bezpieczeństwo maszyn które ma ochraniać, PF OpenBSD udostępnia pewne unikalne funkcje poprawiające błędy w implementacjach stosu TCP/IP. Funkcje te opisano poniżej.

### 4.1 Modulowanie stanu ( ang. *state modulation* )

Aby zapewnić poprawne dostarczanie pakietów przez TCP, protokół ten używa mechanizmu numerów sekwencyjnych, w którym na początku połączenia wybierany jest losowy, początkowy numer sekwencyjny i zwiększany po każdym przesłanym bajcie. Wiele implementacji TCP używa bardzo słabego generatora losowego by generować takie numery, co pociąga za sobą ryzyko, że ktoś będzie mógł przechwycić takie połączenia.

Dlatego właśnie, programiści OpenBSD PF zdecydowali się na dodanie możliwości modulowania stanu ( ang. *state modulation* ). Polega ona na generowaniu bardziej losowego numeru sekwencyjnego dla połączeń pasujących do reguł i podmienianiu numerów sekwencyjnych pakietów przechodzących przez ścianę ogniową.

Można to zrobić przez dodanie słowa kluczowego `'modulate state'`, tak jak poniżej w regule chroniącej naszą sieć firmową z poprzedniego rozdziału:

```
pass in quick on x11 proto tcp from 260.250.1.0/24 to any \
      flags S/SA modulate state
```

Dodanie opcji `'modulate state'` implikuje opcję `'keep state'` opisaną w sekcji [Utrzymywanie stanów/śledzenie połączeń](#).

### 4.2 Normalizacja pakietów

Ponieważ niektóre stosy IP niepoprawnie implementują defragmentację pakietów IP, PF z OpenBSD obsługuje również opcję **normalizującą** ( ang. *scrub* - jest to bardzo swobodne tłumaczenie ;) ). Jeśli pasuje ona do pakietu, część PF odpowiedzialna za normalizację zapewni, że z pakietu usunięte zostaną wszelkie nieprawidłowości zanim zostanie on dalej przesłany.

Normalizacja całego ruchu przychodzącego, wymaga reguły podobnej do tej:

```
scrub in all
```

Opcja ta zużywa pewną ilość zasobów systemu, więc powinna być używana tam, gdzie istnieje realna potrzeba ochrony faktycznie słabych implementacji stosu TCP/IP.

Do opcji `'scrub'` można dołączyć dodatkowe:

**no-df**

wyczyść bit ( flagę ) fragmentu w pasującym pakiecie IP

#### **min-ttl numer**

wymuś minimalny czas życia ( ang. *time to live* ) na pasujących pakietach, odrzucając te, które nie spełniają tego warunku.

## **5. Migrowanie z IP Filter**

Model zestawu reguł w PF wzorowany na IPFilter. Są jednak pewne różnice, które ta sekcja stara się udokumentować.

### **5.1 Nie ma już opcji `head` i `group`**

Słowa kluczowe `'head'` i `'group'`, których używano w IPFilter do grupowania reguł, nie są już w PF potrzebne. Jeśli używałeś ich, będziesz musiał ręcznie uporządkować swój zestaw reguł by prawidłowo działał z OpenBSD PF.

PF OpenBSD używa automatycznego mechanizmu optymalizującego zestaw reguł, zwanego **pomijaniem**. Zajrzyj do rozdziału [Optymalizacja zestawu reguł: pomijanie](#) po więcej informacji.

## **6. Inna dokumentacja**

Dostępna jest pewna ilość informacji w Internecie dotyczących PF OpenBSD i ogólnie ścian ogniowych. Poniżej krótkie podsumowanie całego materiału, który może być interesujący:

<http://www.benzedrine.cx/pf.html>

Oryginalna strona tego, co teraz jest PF OpenBSD

<http://www.obfuscation.org/ipf/>

HOWTO IPFilter. Pomimo, że HOWTO które właśnie czytasz stara się być na tyle kompletne na ile to możliwe jeśli chodzi o PF, zajrzenie do tego dokumentu może być interesujące. W końcu są to korzenie PF OpenBSD.

<http://www.linuxdoc.org/HOWTO/Firewall-HOWTO.html>

The Linux Firewall and Proxy HOWTO, który opisuje konfigurację proxy działających w przestrzeni użytkownika, takich jak Squid czy SOCKS. Napisane dla Linuksa, ale może również być ciekawą lekturą.

<http://www.openbsd.org/cgi-bin/man.cgi?query=pfctl=8=html>

Strona podręcznikowa OpenBSD do programu `pfctl`

<http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf=5=html>

Strona podręcznikowa OpenBSD opisująca format reguł PF

<http://www.openbsd.org/cgi-bin/man.cgi?query=pf=4=html>

Strona podręcznikowa OpenBSD opisująca urządzenie `pf`. Raczej dla programistów.

## 7. Podziękowania

Autor chciałby podziękować następującym osobom za pomoc w niektórych skomplikowanych tematach, za wyjaśnienie pewnych wewnętrznych procesów PF OpenBSD, za wypunktowanie błędów w poprzednich wersjach tego dokumentu, w końcu ogólnie za sugestie (w kolejności alfabetycznej):

- Mike Frantzen [frantzen@w4g.org](mailto:frantzen@w4g.org)
- Markus Friedl [markus@openbsd.org](mailto:markus@openbsd.org)
- Artur Grabowski [art@blahonga.org](mailto:art@blahonga.org)
- Daniel Hartmeier [daniel@benzedrine.cx](mailto:daniel@benzedrine.cx)
- Erik Liden [erik@ipunplugged.com](mailto:erik@ipunplugged.com)
- Rod Whitworth [listener@witworx.com](mailto:listener@witworx.com)
- Jim Zajkowski [jim@jimz.net](mailto:jim@jimz.net)

Wouter Coene 2002-04-05

## 8. Przypisy

260.250.1.31

kompletnie fikcyjny przykład

ISN

więcej informacji o generowaniu ISN, jak również informacje o nich w kontekście popularnych systemów operacyjnych możesz znaleźć w dokumencie pod adresem <http://razor.bindview.com/publish/papers/tcpseq.html>

## 9. Uwagi od tłumacza

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

**Mariusz Drozdziel nova (at) tucznik.net**  
■ literówki