

Filtrowanie *stateful-inspection* w Linuksie i BSD

Paweł Krawczyk
kravietz@aba.krakow.pl

27 sierpnia 2001

Wersja 0.6

Spis treści

1	Wstęp	3
2	Filtry pakietowe	3
3	Filtry <i>stateful-inspection</i>	4
4	Filtry w systemach operacyjnych	4
4.1	Linux	5
4.2	BSD	5
4.3	Inne	5
5	Linux	5
5.1	Wymagania	5
5.2	Podstawy	6
5.3	Najprostsza konfiguracja dla stacji roboczej	6
5.3.1	O module <i>state</i>	7
5.3.2	Inne cechy modułu <i>state</i>	7
5.4	Moduł <i>state</i> i FTP	8
5.5	Stacja robocza a protokoły <i>IDENT</i> i <i>SOCKS</i>	9
5.6	Konfiguracja dla serwera	10

5.6.1	Ping	10
5.6.2	Dostęp do usług	10
5.6.3	Inne porty	11
5.6.4	Moduł <i>unclean</i>	11
5.7	Konfiguracja routerów	11
5.8	Filtrowanie ruchu wychodzącego	13
5.9	Jeszcze o ruchu wychodzącym	18
5.10	Inne ciekawe funkcje	18
5.11	Dokumentacja	19
6	BSD	19
6.1	Najprostszy przykład	20
7	Od autora	20
7.1	Uwagi końcowe	20
7.2	Zmiany	21

1 Wstęp

Filtry pakietowe są dostępne praktycznie we wszystkich systemach operacyjnych. Stanowią podstawę ograniczeń dostępu do lokalnych usług, co przekłada się bezpośrednio na możliwość rozpoznania i zablokowania prób ataku. Dobrze administrowana sieć nigdy nie udostępnia na zewnątrz usług, które nie są niezbędne, nawet jeśli w danej chwili nie są znane żadne dziury w obsługujących je demonach.

Celem tego artykułu jest przybliżenie użytkownikom Linuksa i systemów z rodziny BSD stosunkowo nowej i bardzo przydatnej techniki, jaką jest filtrowanie *stateful-inspection*.

Dwie uwagi odnośnie tego artykułu: po pierwsze, jest on w fazie intensywnego rozwoju i dość często jest on uzupełniany o nowe funkcje oraz przykłady. Na samym końcu jest informacja, gdzie można znaleźć aktualne wersje oraz jakie zmiany zostały wprowadzone do tej wersji. Po drugie, jeśli przymierzasz się do skonfigurowania filtra dla dużej sieci, zdecydowanie przeczytaj cały artykuł i wszystkie przykłady. Są one tak skomponowane, że każdy kolejny zawiera tylko kilka nowych funkcji by zbytnio nie zaciemnić towarzyszących im opisów. Są one również stopniowane względem stopnia skomplikowania oraz, co istotniejsze, skuteczności.

2 Filtry pakietowe

Klasyczny filtr pakietowy to zbiór reguł określających co system powinien zrobić z pakietem przychodzącym z sieci, lub do niej wychodzącym. Filtr jest sterowany zbiorem reguł, których podstawowymi elementami są wzorce oraz akcje, mówiące co zrobić z pakietem pasującym do danej reguły. W skład wzorca mogą wchodzić cechy charakterystyczne dla protokołu IP, takie jak adres źródłowy i docelowy pakietu, numery portów protokołów TCP i UDP, rozmaite flagi, typ komunikatu ICMP i inne, w zależności od zaawansowania i kompletności filtra.

Przetwarzanie tych reguł odbywa się dla każdego pakietu przychodzącego lub wychodzącego z danego węzła. Pakiet pasujący do określonego w danej regule wzorca jest traktowany zgodnie z przypisaną do niego akcją. Z reguły ogranicza się ona do przepuszczenia lub zablokowania pakietu, z ewentualnym odesłaniem odpowiedniego komunikatu ICMP.

Klasyczne filtry pakietowe mają jedną charakterystyczną cechę, a mianowicie ich reguły są całkowicie lub w większości statyczne, to jest raz skonfigurowane przez administratora działają bez zmian aż do kolejnej jego ingerencji. Pojęcie takie stwarza nieraz konieczność takiego tworzenia reguł filtra, które nie implementują wszystkich wynikających z polityki bezpieczeństwa reguł,

wymuszając pozostawienie w filtrze określonych furtek.

Niedoskonałości klasycznych filtrów uniemożliwiają także całkowite zabezpieczenie serwera przed skanowaniem portów i innymi atakami, wykorzystującymi cechy protokołów TCP/IP.

3 Filtry *stateful-inspection*

Filtry *stateful-inspection* stoją o stopień wyżej od tradycyjnych zapór i skutecznie eliminują ich niedogodności. Podstawą ich działania jest bieżące śledzenie i analiza przechodzących przez dany węzeł połączeń, co pozwala na znacznie skuteczniejsze kontrolowanie ich legalności. Filtr cały czas przechowuje w pamięci informacje na temat aktualnego stanu każdego połączenia, wiedząc przy tym jakie kolejne stany są dozwolone z punktu widzenia zarówno protokołu, jak i polityki bezpieczeństwa.

Filtry tego typu pozwalają na określenie możliwości dokonania danego połączenia bez konieczności operowania poszczególnymi stanami protokołu TCP. Do administratora należy tylko określenie kierunku oraz polityki względem rozpoczęcia danego połączenia, a filtr automatycznie weryfikuje kolejne etapy jego nawiązywania i późniejszy przebieg.

Ta ostatnia cecha pozwala również na odrzucanie pakietów, które do danej sesji nie należą, co w praktyce przekłada się na skuteczne blokowanie prób skanowania portów lub wprowadzania sfałszowanych pakietów (*spoofing*).

Przykładowo, klasyczny filtr pakietowy dla przepuszczenia pełnego połączenia TCP do danego serwera potrzebował co najmniej dwóch reguł: wpuszczania pakietów do danego adresu i ich wypuszczania na zewnątrz. Rozbudowywanie tej polityki na przykład o kierunek dozwolonych połączeń (czyli z której strony można je zaczynać) wymagało dalszego rozbudowania listy, na przykład o określenie że rozpoczynające połączenie pakiety z flagą SYN są wpuszczane tylko w danym kierunku.

Dla filtra *stateful-inspection* w tym wypadku wystarczająca jest wyłącznie jedna reguła, a mianowicie że pakiety z flagą SYN są wpuszczane do serwera na danym porcie. Pakiety będące częścią połączenia idące w obu kierunkach będą przepuszczane automatycznie. Równocześnie jednak analogiczne, ale nie będące częścią dozwolonego połączenia pakiety zostaną zablokowane.

4 Filtry w systemach operacyjnych

Filtry *stateful-inspection* są dostępne w chwili obecnej w większości systemów *open-source* oraz w części produktów komercyjnych.

4.1 Linux

Linux przeszedł do tej pory przez trzy wersje filtra pakietowego. W kernelach do 2.0 był to *ipfw*, następnie *ipchains* w 2.2 oraz *iptables* w kernelach 2.4. Tylko ostatni z nich jest filtrem *stateful-inspection*, działa jednak skutecznie i stabilnie. W chwili obecnej jest to jeden z najlepszych znanych mi filtrów pakietowych na świecie.

4.2 BSD

We FreeBSD są dostępne dwa filtry pakietowe: *ipfirewall/ipfw* oraz *ipfilter/ipf*, w OpenBSD i NetBSD tylko ten ostatni. W ostatnich wersjach oba te filtry posiadają funkcję śledzenia połączeń, przy czym *ipf* zdecydowanie wyróżnia się elastycznością konfiguracji i dojrzałością. Pod względem funkcjonalnym minimalnie ustępuje *iptables*, wbrew temu co na listach dyskusyjnych poświęconych BSD głoszą niekiedy zwolennicy tego systemu.

4.3 Inne

Filtry *stateful-inspection* są również spotykane w produktach komercyjnych. I tak, popularny *Cisco IOS* posiada tę funkcję pod nazwą *Context Based Access Control (CBAC)*. Filtrem tego typu jest również *CheckPoint FireWall-1*.

5 Linux

5.1 Wymagania

- **Kernel 2.4.x**, nie wiem które dystrybucje zawierają go domyślnie ¹ . Sam kompiluję ze źródeł.
- Program *iptables*, dostępny w źródłach na stronie *iptables* lub jako pakiet dla poszczególnych dystrybucji.

Jeśli konfigurujemy kernel samodzielnie, to istotne są następujące opcje w menu **Networking options**:

- **Network packet filtering (replaces ipchains)**
- Wchodzimy do menu **IP: Netfilter Configuration**

¹ Użytkownicy Debiana mogą doinstalować kernel wraz z koniecznymi narzędziami przez APT deb <http://people.debian.org/~bunk/debian-potato-main>.

- **IP tables support (required for filtering/masq/NAT)** należy wkompiłować na stałe²
- Wszystkie pozostałe opcje zaznaczamy jako moduły.

5.2 Podstawy

Konfiguracja *iptables* odbiega nieco od znanego z wcześniejszych wersji *ipchains*, głównie dlatego że nowy filtr jest w dużej mierze modularny. Opcje, które kiedyś były stałymi parametrami programu *ipchains* są obecnie realizowane przez poszczególne moduły. Do poprawnego działania potrzebne są odpowiednie moduły, wkompiłowane w kernel 2.4 oraz program *iptables*, służący do konfiguracji filtra.

Filozofia nowego filtra jest dość podobna — mamy tutaj także trzy domyślne zestawy reguł *INPUT*, *FORWARD* i *OUTPUT* oraz szereg celów (*targets*), które określają co należy zrobić z pakietem pasującym do danej regułki. Najczęściej używane to *ACCEPT*, *DROP* (zablokowanie pakietu bez powiadomienia nadawcy) oraz *REJECT* (zablokowanie ze zwróceniem komunikatu ICMP).

Warto jednak dodać, że w nowym filtrze zasadniczo zmieniły się reguły przepuszczania pakietów przez poszczególne zestawy reguł. Pakiety przesyłane (*forwarded*) z innych interfejsów przechodzą wyłącznie przez zestaw *FORWARD*. Dwa pozostałe zestawy — *INPUT* i *OUTPUT* obsługują wyłącznie pakiety kończące drogę na lokalnym systemie lub na nim generowane.

5.3 Najprostsza konfiguracja dla stacji roboczej

Funkcja śledzenia połączeń jest w *iptables* realizowana przez moduł *state*. Najprostsza konfiguracja, która realizować będzie takie filtrowanie i użyteczna na przykład na stacji roboczej, która nie udostępnia żadnych usług, wygląda tak:

```
iptables -F

iptables -A INPUT -i lo -j ACCEPT

iptables -A INPUT -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p udp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p icmp -j ACCEPT -m state --state ESTABLISHED

iptables -A INPUT -j LOG -m limit --limit 10/hour
iptables -A INPUT -j DROP
```

²Wynika to stąd, że kernel nie potrafi samodzielnie załadować tego jako modułu przy wywołaniu *iptables*.

Konfiguracja ta składa się z następujących fragmentów:

1. Usunięcie wszystkich reguł filtra.
2. Dodanie reguły wpuszczającej wszystko na interfejsie lokalnym *lo*.
3. Dodanie do tablicy *INPUT* reguł wpuszczających pakiety należące do już nawiązanych (*ESTABLISHED*) połączeń.
4. Dodanie, na końcu, dwóch reguł blokujących. Pierwsza tak na prawdę tylko loguje pakiety, które nie zostały wpuszczone przez poprzednie reguły. Druga faktycznie je blokuje.

5.3.1 O module *state*

Kluczowe w tym wypadku są regułki korzystające z modułu *state*. Ostateczny rezultat ich zastosowania jest taki, że wszystkie połączenia **wychodzące** będą działać bez ograniczeń (ponieważ nie skonfigurowaliśmy ograniczeń w tablicy *OUTPUT*), a wszystkie połączenia przychodzące będą blokowane. Każde takie połączenie będzie jednak zapamiętywane i należące do niego pakiety powracające będą automatycznie przepuszczane przez tablicę *INPUT*. Równocześnie filtr *stateful-inspection* wykryje także pakiety nie pasujące do zapamiętanych połączeń i zasygnalizuje ich wyblokowanie. Mogą to być na przykład pakiety przysłane podczas próby *spoofingu* lub skanowania zaawansowanymi technikami, takimi jak *ACK scanning*.

Zastosowanie celu *DROP* dla wyblokowanych pakietów będzie dodatkowym utrudnieniem dla skanującego wszystkie porty na danej maszynie, ponieważ brak typowej odpowiedzi (*ICMP Port unreachable* lub *ICMP Packet filtered*) spowoduje, że skaner będzie musiał czekać przez ustalony czas, zanim uzna taki port za nieaktywny.

5.3.2 Inne cechy modułu *state*

Dodajmy, że stan *ESTABLISHED* nie odnosi się tylko do TCP, który jest protokołem połączeniowym i łatwo stwierdzić czy połączenie jest nawiązane, czy nie. Dokumentacja filtra definiuje ten stan jako odnoszący się do „połączeń, które wymieniły pakiety w obu kierunkach”. Stąd możliwe jest zastosowanie tej flagi do protokołów takich jak UDP i ICMP.

Moduł *state* zna również inne stany połączeń. Są to:

- *NEW* — pakiety inicjujące nowe połączenie (lub przesyłane tylko w jednym kierunku)

- *RELATED* — pakiety nie należące bezpośrednio, ale związane w inny sposób ze znaną sesją; przykładem mogą być tutaj kanały danych w FTP lub błędy zwracane po ICMP ³
- *INVALID* — pakiety nie związane z żadną zapamiętaną sesją

5.4 Moduł *state* i FTP

Komentarza wymaga funkcja *RELATED*, która faktycznie rozszerza właściwości filtra warstwy trzeciej (*IP*, *ICMP*) i czwartej (*TCP*, *UDP*) o zdolność rozumienia i reagowania na stany protokołów wyższych warstw. Najprościej przedstawić to na przykładzie protokołu FTP, który od zawsze był zmorą osób projektujących filtry pakietowe.

Protokół ten wykorzystuje stałe połączenie na port 21 serwera do wydawania komend, natomiast samo przesyłanie plików działa w dwóch trybach:

- W trybie aktywnym (*PORT mode*) klient otwiera po swojej stronie jakiś wysoki port i podaje serwerowi jego numer. Serwer wykonuje na ten port połączenie i przesyła dane.
- W trybie pasywnym (*passive mode*) to serwer otwiera wysoki port, a klient nawiązuje na niego połączenie otrzymując dane.

Jak się łatwo domyślić, trybu aktywnego nie będziemy w stanie używać zza maskarady, ponieważ serwer FTP nie będzie w stanie połączyć się z naszym hostem, ukrytym za routerem realizującym NAT.⁴

Tryb pasywny również nie jest rozwiązaniem idealnym, ponieważ musimy zezwolić klientowi z sieci wewnętrznej na wykonywanie praktycznie dowolnych połączeń na zewnątrz⁵.

Z pomocą przychodzi nam tutaj funkcja *RELATED*, która wypuści na zewnątrz połączenia na określony wysoki port tylko wtedy, kiedy żądanie jego otwarcia zostało jawnie wydane podczas sesji FTP. W tym celu router musi śledzić każde przechodzące przez niego połączenia FTP i szukać w nich komend, otwierających porty pasywne. Jeśli takie znajdzie, zapamiętuje ten fakt na potrzeby regułki *RELATED*.

³Na przykład, komunikat *ICMP Time Exceeded* wysłany w odpowiedzi na wysłany przez nas pakiet UDP jest uznawany za *RELATED* w stosunku do niego.

⁴Chyba, że załadujemy moduł *ip_nat_ftp*, który będzie oszukiwał nasz router i przekazywał połączenia do środka.

⁵Większość serwerów FTP otwiera porty pasywne z pewnego określonego przedziału. Można więc obejść ten problem otwierając tylko ten zakres portów, ale nie jest to rozwiązanie eleganckie.

Dla każdego protokołu wyższej warstwy musimy załadować odpowiedni moduł interpretujący. W przypadku FTP jest to moduł *ip_conntrack_ftp*, dostępny w standardowej dystrybucji kernela (drugi to *ip_conntrack_irc*. W sieci można również znaleźć moduły do innych protokołów. Należy pamiętać, że moduły te nie są ładowane automatycznie — trzeba je załadować jawnie za pomocą poleceń *insmod* albo *modprobe*.

5.5 Stacja robocza a protokoły *IDENT* i *SOCKS*

Powyższa konfiguracja jest skuteczna, ma jednak jedną wadę, która wyjdzie na jaw prędzej czy później. Otóż łącząc się z tak skonfigurowanego hosta z niektórymi serwerami FTP zauważymy, że występuje kilkudziesięciosekundowe opóźnienie pomiędzy nawiązaniem połączenia, a zalogowaniem do serwera.

Wynika to stąd, że duża liczba serwerów FTP (i nie tylko) próbuje uzyskać od klienta informacje o użytkowniku po protokole *IDENT*, próbując nawiązać zwrotne połączenie na port 113, przypisany do tej usługi. Ponieważ nasz host nie akceptuje żadnych połączeń przychodzących i nie odrzuca ich w jawny sposób, serwer FTP wpuści nas dopiero po przekroczeniu czasu oczekiwania z usługą *IDENT*.

To samo dotyczy protokołu *SOCKS*, działającego domyślnie na porcie 1080. Wiele serwerów IRC wykonuje zwrotne połączenie na ten port aby sprawdzić, czy użytkownik nie loguje się zza proxy. Tradycyjne wyblokowanie tego portu również jest przyczyną opóźnień w logowaniu do serwera.

Aby uniknąć tej nieszkodliwej, ale irytującej niedogodności należy spowodować, by połączenia na porty 113 i 1080 były jawnie odrzucane, dając tym samym do zrozumienia że nie mamy serwera *IDENT* ani *SOCKS*. Poprawiona konfiguracja znajduje się poniżej:

```
iptables -F

iptables -A INPUT -i lo -j ACCEPT

iptables -A INPUT -p tcp --dport 113 \
-j REJECT --reject-with icmp-port-unreachable
iptables -A INPUT -p tcp --dport 1080 \
-j REJECT --reject-with icmp-port-unreachable

iptables -A INPUT -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p udp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p icmp -j ACCEPT -m state --state ESTABLISHED

iptables -A INPUT -j LOG -m limit --limit 10/hour
iptables -A INPUT -j DROP
```

Wykorzystaliśmy cel *REJECT*, ze wskazaniem że zwrócony ma być standar-

dowy w takim wypadku komunikat *ICMP Port unreachable*.

5.6 Konfiguracja dla serwera

O ile stacja robocza może jawić się z zewnątrz jako głuchy bastion, o tyle w przypadku serwera musimy dopuścić przynajmniej połączenia przychodzące na wybrane usługi. Konfiguracja komplikuje się jeszcze bardziej (pod względem ilości reguł), jeśli usługi te mają być dostępne tylko z niektórych adresów i tak dalej.

Tworząc konfigurację filtra możemy oprzeć się o przedstawione wyżej przykłady konfiguracji, dopisując nowe regułki przed ostatnimi dwoma, blokującymi wszystko.

5.6.1 Ping

Pierwszą modyfikacją może być dopuszczenie pakietów *ICMP Echo*, czyli popularnego *pinga*:

```
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
```

Tutaj należy się komentarz — o ile część administratorów blokuje tę usługę, o tyle ja sam jestem zdania, że jest to polityka przynosząca więcej szkody niż pożytku i nie przyczyniająca się w znaczący sposób do poprawienia bezpieczeństwa serwera. Tymczasem *ping* przydaje się po prostu do stwierdzania, czy serwer działa i jak długo wędrują do niego pakiety.

5.6.2 Dostęp do usług

Kolejnym krokiem jest wpuszczenie połączeń na te serwisy, które mają być dostępne. W poniższym przykładzie publicznie (zewsząd) dostępne są serwisy SMTP (port 25) oraz WWW (80). Serwer SSH (22) jest dostępny tylko z adresu 192.168.13.2. Przed tym wszystkim jednak blokujemy wszystkie połączenia z sieci 195.116.130.0/24, ponieważ podejrzewamy jej właścicieli o wysyłanie spamu.

```
iptables -A INPUT -p tcp -s 195.116.130.0/24 -j DROP
iptables -A INPUT -p tcp -d 0/0 --dport 25 -j ACCEPT
iptables -A INPUT -p tcp -d 0/0 --dport 80 -j ACCEPT
iptables -A INPUT -p tcp -s 192.168.13.2/32 -d 0/0 --dport 22 -j ACCEPT
```

Zamiast numerów portów można używać oczywiście nazw z pliku */etc/services*, czyli odpowiednio *smtp*, *www* i *ssh*. Przyczyni się to do poprawienia czytelności takiej konfiguracji, ale w przypadku jej przenoszenia na inną instalację może spowodować problemy, jeśli plik *services* nie będzie kompletny.

Warto używać ogólnej specyfikacji adresu docelowego w regułkach, czyli *0/0* (skrótowa postać *0.0.0.0/0*), choćby ze względu na to że serwer może posiadać dodatkowe adresy IP, na które również należy nałożyć ograniczenia by cały filtr był skuteczny. Wyjątkiem od tej reguły są systemy będące równocześnie routerami, ale o tym napiszemy dalej.

5.6.3 Inne porty

Stosowanie *stateful-inspection* na serwerze ma jeszcze jedną zaletę, przydatną szczególnie na serwerach z kontami *shell*. Otóż taka konfiguracja skutecznie uniemożliwi użytkownikom uruchamianie własnych demonów, nasłuchujących na wysokich portach i robiących różne rzeczy, nieraz niepożądane z punktu widzenia bezpieczeństwa (boty, prywatne *proxy* itp.).

Nie stanowi przy tym oczywiście żadnego problemu otwarcie wysokich portów dla użytkowników, którzy tego rzeczywiście potrzebują.

5.6.4 Moduł *unclean*

Moduł *unclean*⁶ ma za zadanie wylapywanie pakietów, które są niepoprawne w różny i trudny do sprecyzowania za pomocą standardowych reguł sposob. Moduł ten przeprowadza przetwarzanym pakiecie szereg testów, badając jego zgodność ze standardami, wewnętrzną spójność, poprawność flag i szereg innych cech. Moduł ten należy umieścić na początku listy, na przykład w postaci poniższych reguł. Pozwoli on wychwycić i zablokować szereg dotychczas znanych ataków związanych z błędną fragmentacją pakietów i fałszywymi flagami, a także nowe ataki lub pakiety uszkodzone na łączach.

```
iptables -A INPUT -j LOG -m limit --limit 10/hour -m unclean
iptables -A INPUT -j DROP -m unclean
```

5.7 Konfiguracja routerów

W tym przypadku jako *router* będziemy rozumieć każdy host, który posiada więcej niż jeden interfejs sieciowy i zajmuje się przesyłaniem pakietów z jednej sieci do drugiej. Host taki może równocześnie udostępniać rozmaite usługi ze swojego adresu. Sytuacja taka dodatkowo komplikuje konfigurację i stawia dodatkowe wymagania w kwestii poprawnego zrozumienia przepływu danych i zaprojektowania ograniczeń.

⁶**Uwaga:** autorzy modułu ostrzegają, że moduł ten ma charakter eksperymentalny i może nie działać do końca poprawnie. Wygląda na to, że do kernela 2.4.5 wszystko było dobrze, jednak od wersji 2.4.6 moduł zaczął u mnie błędnie ciąć poprawne pakiety, przez co ostatecznie musiałem go wyłączyć.

Główną różnicą w tym wypadku będzie konieczność operowania nie tylko na tablicy *INPUT*, ale i *FORWARD*, odnoszącej się do każdej sytuacji gdy podlegające filtrowaniu pakiety przechodzą pomiędzy różnymi interfejsami. Wymaga to z reguły zdublowania konfiguracji modułów *state* dla tablicy *FORWARD* oraz określenia dodatkowych reguł dla poszczególnych interfejsów.

Należy także pamiętać, że w przypadku routera tablica *INPUT* może się odnosić do każdego interfejsu i konieczne jest uwzględnianie jego nazwy (lub adresu) w konfiguracji filtra. Nie można już zakładać, że wszystkie dane przychodzą przez jeden interfejs, tak jak w poprzednio rozpatrywanych przypadkach. Z tego samego powodu należy rozważyć stosować maski globalne (typu *0/0*).

Poniżej znajduje się rozbudowana konfiguracja, oparta o jedną z moich rzeczywistych instalacji. Zawiera ona praktycznie wszystkie opisane powyżej elementy plus dodatkowe elementy, wymagane w tej konkretnej sytuacji (np. NAT). Plik ten ma postać skryptu *shella* i jest po prostu uruchamiany przy starcie systemu. Komentarze znajdują się w samym skrypcie ⁷.

```
#!/bin/sh

PATH="/sbin:/usr/local/sbin:$PATH"

# Ładujemy moduł niezbędne dla RELATED
modprobe ip_conntrack_ftp

iptables -F
iptables -F -t nat

# Przychodzące IDENT odrzucamy z komunikatem ICMP
iptables -A INPUT -p tcp --dport 113 \
-j REJECT --reject-with icmp-port-unreachable

# Interfejs lokalny
iptables -A INPUT -i lo -j ACCEPT

# Wpuszczamy wszystko z LAN
iptables -A INPUT -i eth1 -s 10.0.0.0/8 -j ACCEPT
iptables -A FORWARD -i eth1 -s 10.0.0.0/8 -j ACCEPT

# Połączenia już nawiązane
iptables -A INPUT -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p tcp -j ACCEPT -m state --state RELATED
iptables -A INPUT -p udp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p icmp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p icmp -j ACCEPT -m state --state RELATED
iptables -A FORWARD -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A FORWARD -p tcp -j ACCEPT -m state --state RELATED
```

⁷Skrypt ten jest również dostępny oddzielnie pod adresem <http://arch.ipsec.pl/iptables/router1>

```
iptables -A FORWARD -p udp -j ACCEPT -m state --state ESTABLISHED
iptables -A FORWARD -p udp -j ACCEPT -m state --state RELATED
iptables -A FORWARD -p icmp -j ACCEPT -m state --state ESTABLISHED
iptables -A FORWARD -p icmp -j ACCEPT -m state --state RELATED

# POP3 z serwera wewnętrznego wystawiony na zewnątrz za pomocą
# DNAT (odpowiednik dawnego port-forwarding)
# Adres 192.168.13.3 jest adresem zewnętrznym routera, serwer w LAN
# ma adres 10.1.1.241
iptables -t nat -A PREROUTING -p tcp -d 192.168.13.3/32 --dport 110 \
-j DNAT --to-destination 10.1.1.241

# Wpuszczamy ping
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

# Serwer DNS jest w publicznej sieci 192.168.14.0/24 podpiętej na drugiej
# karcie sieciowej (stanowiącej DMZ)
iptables -A FORWARD -p udp -d 192.168.14.5 --dport 53 -j ACCEPT
iptables -A FORWARD -p tcp -d 192.168.14.5 --dport 53 -j ACCEPT

iptables -A INPUT -p tcp -d 217.96.88.194 --dport 22 -j ACCEPT

# Na routerze też stoi serwer WWW
iptables -A INPUT -p tcp -d 192.168.13.3 --dport 80 -j ACCEPT

# Serwer WWW w DMZ
iptables -A FORWARD -p tcp -d 192.168.14.6 --dport 80 -j ACCEPT

# Serwer FTP w DMZ, otwarte porty dla połączeń pasywnych i aktywnych
iptables -A FORWARD -p tcp -d 192.168.14.4 --dport 21 -j ACCEPT
iptables -A FORWARD -p tcp -s 192.168.14.4 --sport 20 -j ACCEPT
iptables -A FORWARD -p tcp -d 192.168.14.4 --dport 40000:44999 -j ACCEPT

# Spammerów nie wpuszczamy
iptables -A INPUT -p tcp -s 195.116.130.0/24 -j DROP

# Dynamiczny NAT dla adresów z sieci wewnętrznej (maskarada)
iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -j MASQUERADE

iptables -A INPUT -j LOG -m limit --limit 10/hour
iptables -A INPUT -j DROP
iptables -A FORWARD -j LOG -m limit --limit 10/hour
iptables -A FORWARD -j DROP
```

5.8 Filtrowanie ruchu wychodzącego

W dotychczas omówionych przykładach stosowaliśmy wyłącznie filtrowanie ruchu przychodzącego z interfejsu zewnętrznego, zakładając milcząco że ruch wychodzący z naszej sieci jest godny zaufania. Nie zawsze jest to prawda. Ruch wychodzący należy filtrować z kilku powodów:

1. System *Windows* stosowany obecnie najczęściej na stacjach roboczych jest środowiskiem, w którym bujnie i chętnie kwitnie niewidzialne życie w postaci wirusów, koni trojańskich oraz *spyware* ⁸. Każde z tych zjawisk osobno powinno być motywacją do filtrowania ruchu wychodzącego. Łatwo się o tym przekonać uważnie obserwując ruch generowany przez komputery wyposażone w *Windows* oraz użytkowników, chętnie zaglądających na popularne portale, pornosajty i wymieniających się „śmiesznymi” programami.
2. W razie udanego wykorzystania dziury w jednym z naszych serwerów filtrowanie ruchu wychodzącego może uniemożliwić włamywaczowi faktyczne skorzystanie z otwartej furty. Sprytny złoczyńca może obejść nasze filtry przychodzące konstruując *exploit* tak, by to nasz serwer połączył się z wybraną przez niego maszyną i wystawił mu *shell*. Filtrowanie połączeń wychodzącym w najgorszym wypadku mu to utrudni, w najlepszym uniemożliwi.
3. Należy brać pod uwagę, że nasza sieć stanie się nie tylko celem, ale źródłem ataku na inne serwery. Dotyczy to w szczególności sieci dostawców internetowych i innych rozbudowanych sieci, z których korzystają nie-raz zupełnie obcy użytkownicy. Filtrowanie ruchu wychodzącego może utrudnić wiele tego rodzaju ataków, a w praktyce wyeliminować możliwość że nasza sieć stanie się źródłem ataków wykorzystujących *IP spoofing*.

Przykładowy, dość rozbudowany skrypt realizujący takie filtrowanie jest zamieszczony poniżej ⁹.

```
#!/bin/sh

# Przykładowa, restrykcyjna konfiguracja iptables dla Linuxa 2.4.
# Ten skrypt zawiera również funkcje zmniejszające szanse odcięcia
# sobie dostępu do danego hosta podczas zdalnej konfiguracji.
# Paweł Krawczyk <kravietz AT aba.krakow.pl> 2001

# Opcje skryptu:
# flush - czyszczy wszystkie regułki i ustawia otwartą politykę
# temp - konfiguruje tymczasowe regułki (patrz poniżej)
# bez opcji - konfiguruje iptables

# Kolejność operacji:
# 1. zerowanie wszystkich tablic
```

⁸Oprogramowanie w mniej lub bardziej zawoalowany sposób szpiegujące dany komputer i wysyłające rozmaite informacje do serwerów producenta, najczęściej na wysokich portach i własnymi protokołami.

⁹Jest on również dostępny jako samodzielny plik pod adresem <http://arch.ipsec.pl/iptables/shivling>

```
# 2. ustawienie restrykcyjnej polityki DROP
# 3. ustawienie specjalnego dostępu dla interfejsu lo
# 4. ustawienie specjalnego dostępu dla sieci LAN
# 5. konfiguracja modulu unclean
# 6. ustawienie odrzucania polaczen dla IDENT i SOCKS
# 7. zezwolenie dla pakietow ICMP Echo (ping)
# 8. konfiguracja stateful-inspection (modul state)
# 9. ustawienie dostępu dla konkretnych uslug
# 10. konfiguracja NAT i PAT
# 11. konfiguracja logowania zablokowanych pakietow

export PATH=""

iptables_flush()
{
/sbin/iptables -P INPUT ACCEPT
/sbin/iptables -P OUTPUT ACCEPT
/sbin/iptables -P FORWARD ACCEPT
/sbin/iptables -F
}
if [ "$1" = "flush" ]; then
iptables_flush
exit 0
fi

# Bezpieczne wywołanie iptables, gwarantujące że
# nasze regułki zostaną załadowane albo w całości
# albo w ogóle. Zapobiega to sytuacji, kiedy tracimy
# dostęp do danego hosta z powodu jednej, źle skonstruowanej
# regułki która miała nas do niego wpuszczać ;)
iptables() {
echo -n .
/sbin/iptables $@
if [ "$?" != "0" ]; then
iptables_flush
exit 1
fi
}

# Możemy włączyć nasze nowe regułki tylko na chwilę.
# Pozwala to sprawdzić, czy po ich uzupełnieniu nadal
# mamy dostęp do hosta, jeśli nie to automatycznie
# odzyskamy go po 30 sekundach. Skrypt należy tylko
# wywołać z opcją "temp".
if [ "$1" = "temp" ]; then
(sleep 30; /sbin/iptables -P INPUT ACCEPT ;\
/sbin/iptables -P OUTPUT ACCEPT ;\
/sbin/iptables -P FORWARD ACCEPT ;\
/sbin/iptables -F) &
fi

echo -n "Installing IPtables ruleset"

# Ładujemy modul stateful-inspection dla FTP
```

```
/sbin/modprobe ip_conntrack_ftp

# Czyszcimy wszystkie tablice
iptables -F
iptables -F -t nat

# Domyslnie nie przepuszczamy nic
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP

# Interfejs lokalny ma specjalne prawa
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
iptables -A FORWARD -o lo -j ACCEPT

# Wpuszczamy wszystko z sieci lokalnej i wypuszczamy
# wszystko na nia. Nie nalezy dodawac tutaj analogicznej
# regulki dla FORWARD, to zalatwi za nas modul state
iptables -A INPUT -i eth0 -j ACCEPT
iptables -A OUTPUT -o eth0 -j ACCEPT

# Logujemy pakiety wyblokowane przez modul unclean
# *** Tymczasowo wylaczone z powodu problemow z unclean w Linuxie 2.4.6
#iptables -A INPUT -j LOG -m limit --limit 10/hour -m unclean
#iptables -A INPUT -j DROP -m unclean
#iptables -A OUTPUT -j LOG -m limit --limit 10/hour -m unclean
#iptables -A OUTPUT -j DROP -m unclean
#iptables -A FORWARD -j LOG -m limit --limit 10/hour -m unclean
#iptables -A FORWARD -j DROP -m unclean

# Odrzucamy z komunikatem ICMP Port Unreachable polaczenia
# na IDENT oraz SOCKS (czesto sprawdzane przez serwery IRC)
iptables -A INPUT -p tcp --dport 113 \
-j REJECT --reject-with icmp-port-unreachable
iptables -A INPUT -p tcp --dport 1080 \
-j REJECT --reject-with icmp-port-unreachable

# Akceptujemy pakiety ICMP Echo (ping) wchodzace i wychodzace
# Akceptacja odpowiedzi jest realizowana przez modul state RELATED
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A FORWARD -p icmp --icmp-type echo-request -j ACCEPT

# Zezwalamy na wszystko co odbywa sie w ramach juz dozwolonych
# polaczen
iptables -A INPUT -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p udp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p icmp -j ACCEPT -m state --state ESTABLISHED
iptables -A INPUT -p icmp -j ACCEPT -m state --state RELATED
iptables -A FORWARD -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A FORWARD -p tcp -j ACCEPT -m state --state RELATED
iptables -A FORWARD -p udp -j ACCEPT -m state --state ESTABLISHED
iptables -A FORWARD -p icmp -j ACCEPT -m state --state ESTABLISHED
iptables -A FORWARD -p icmp -j ACCEPT -m state --state RELATED
```



```
iptables -A OUTPUT -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A OUTPUT -p tcp -j ACCEPT -m state --state RELATED
iptables -A OUTPUT -p udp -j ACCEPT -m state --state ESTABLISHED
iptables -A OUTPUT -p icmp -j ACCEPT -m state --state ESTABLISHED
iptables -A OUTPUT -p icmp -j ACCEPT -m state --state RELATED

# Indywidualne regułki akceptujące określone porty lub serwery
# Zalecana jest jak największa szczegółowość tych reguł,
# w razie problemów należy posilkiwać się regułkami LOG

# czat.onet.pl
iptables -A OUTPUT -o ppp0 -p tcp -j ACCEPT -m state --state NEW \
-d 213.180.130.190 -m multiport --destination-port 5012,5013
iptables -A FORWARD -o ppp0 -p tcp -j ACCEPT -m state --state NEW \
-d 213.180.130.190 -m multiport --destination-port 5011,5012,5013

# Usługi TCP, które wypuszczamy z naszej sieci:
# 80,8080 (WWW), 22 (SSH), 21 (FTP), 25 (SMTP), 119 (news), 53 (DNS)
# 8888 (Napster), 2064 (distributed.net), 706 (SILC)
TCP_OUT_ALLOW=80,8080,22,995,21,25,53,23,119,8888,2064,6667,706
# Usługi UDP: 123 (NTP), 53 (DNS)
UDP_OUT_ALLOW=123,53

iptables -A OUTPUT -o ppp0 -p tcp -j ACCEPT -m state --state NEW \
-m multiport --destination-port $TCP_OUT_ALLOW
iptables -A OUTPUT -o ppp0 -p udp -j ACCEPT -m state --state NEW \
-m multiport --destination-port $UDP_OUT_ALLOW
iptables -A FORWARD -o ppp0 -p tcp -j ACCEPT -m state --state NEW \
-m multiport --destination-port $TCP_OUT_ALLOW
iptables -A FORWARD -o ppp0 -p udp -j ACCEPT -m state --state NEW \
-m multiport --destination-port $UDP_OUT_ALLOW

# Przerzucamy port 6699 (Napster) z zewnętrznego interfejsu
# na host w sieci lokalnej (10.1.1.3). Znane jako port
# forwarding lub Port Address Translation
iptables -t nat -A PREROUTING -p tcp -d 251.61.252.110/32 --dport 6699 \
-j DNAT --to-destination 10.1.1.3

# Maskarada (NAT) dla wszystkich hostów z sieci lokalnej
iptables -t nat -A POSTROUTING -p all -s 10.1.1.0/24 -j MASQUERADE

# Logujemy pakiety które nie zostały zaakceptowane przez
# żadną z powyższych reguł. Zostaną one wyblokowane dzięki
# polityce DROP we wszystkich tablicach
iptables -A INPUT -j LOG -m limit --limit 10/hour
iptables -A OUTPUT -j LOG -m limit --limit 10/hour
iptables -A FORWARD -j LOG -m limit --limit 10/hour

echo .
```

5.9 Jeszcze o ruchu wychodzącym

Powyższa konfiguracja jest dostosowana tak na prawdę do niedużej sieci, ukrytej za maskaradą i w domyśle zaufanej. Stąd brak filtrowania ruchu przychodzącego z sieci lokalnej, które właściwie należałoby wprowadzić dla uniknięcia ataków *IP spoofing*. Po pierwsze, można stworzyć

Filtrować należy przede wszystkim ruch, co do którego jesteśmy pewni że nie może być legalny. Na przykład pakiet z adresem źródłowym spoza naszej sieci, który nagle próbuje się wydostać przez nasz router od strony LAN — coś takiego nie ma prawa zaistnieć w legalnym ruchu i najprawdopodobniej jest jakimś wariantem *spoofingu*. Analogicznie mają się sprawy, gdy od strony WAN przyjdzie do nas pakiet z adresem źródłowym należącym do naszej sieci lokalnej.

Filtrowanie takie można zrealizować co najmniej na dwa sposoby. Pierwszy z nich to stworzenie odpowiednich reguł. Założmy, że 200.117.223.0/24 to nasz LAN, a na świat wychodzimy przez interfejs *ppp0*:

```
iptables -A FORWARD -s 200.117.223.0/24 -i ppp0 -j DROP
iptables -A FORWARD -s ! 200.117.223.0/24 -o eth0 -j DROP
```

Druga, prostsza metoda to wykorzystanie wbudowanego w Linuxa mechanizmu filtrowania takich oczywistych fałszerstw. Jest to mechanizm wprost zalecany przez RFC 1812¹⁰ i włączany łatwo za pomocą interfejsu *sysctl*:

```
# echo 1 >/proc/sys/net/ipv4/conf/all/rp_filter
```

Wiele dystrybucji Linuxa posiada tę funkcję, ukrytą pod różnymi nazwami. Jest ona dostępna już od kerneli 2.2¹¹.

5.10 Inne ciekawe funkcje

Filtr *iptables* posiada również kilka dość interesujących i niespotykanych do tej pory funkcji, które mogą okazać się bardzo przydatne w mniej typowych konfiguracjach.

Nowością jest filtrowanie po adresach sprzętowych kart sieciowych dostępne w module *mac*. Klepania zaoszczędzi moduł *multiport* pozwalający na podanie w jednej regule kilku różnych portów (dotychczas można było podać albo pojedynczy port albo ciągły zakres).

¹⁰ F. Baker „*Requirements for IPv4 routers*” <http://www.ietf.org/rfc/rfc1812.txt>

¹¹W kernelach 2.2 należy użyć wartości 2, a nie 1.

Interesującą funkcją jest możliwość filtrowania pakietów na podstawie konkretnego użytkownika lub grupy za pomocą modułu **owner**. Jest to oczywiście możliwe tylko w przypadku pakietów generowanych w lokalnym systemie, ale daje ogromne możliwości jeśli chodzi o ograniczanie dostępu do sieci na serwerach, gdzie konta mają osoby niezaufane. Można również w ten sposób zmniejszyć ryzyko uruchomienia w systemie tylnej furtki w razie błędów w określonych aplikacjach serwerowych, na przykład zezwalając użytkownikowi *apache* wyłącznie na wysyłanie pakietów z portu 80. Możliwości jest wiele, zapraszam do nadsyłania własnych pomysłów i wdrożeń.

W tym artykule nie zostały także omówione inne opcje *iptables*, których nie miałem okazji używać i nie wydawały mi się wyjątkowo interesujące. Dobre opisy funkcjonalności filtra można znaleźć w dokumentacji, opisanej poniżej.

5.11 Dokumentacja

- <http://netfilter.samba.org/>
Główna strona projektu *netfilter/iptables*, wraz z kodem źródłowym oraz podręcznikami.
- <http://www.boingworld.com/workshops/linux/iptables-tutorial/> „*Linux IPtables tutorial*”
- **iptables(8)**
Strona manuala systemowego poświęcona programowi *iptables*.

6 BSD

Omawiany tutaj filtr *ipfilter* (*ipf*) jest obecny we wszystkich systemach z rodziny BSD (*FreeBSD*, *NetBSD* i *OpenBSD*). W dwóch ostatnich stanowi jedyny systemowy filtr pakietów, instalowany domyślnie. *Ipfilter* jest także dostępny dla innych systemów, np. dla *Solarisa*. Moim zdaniem jest to jeden z najlepszych i najelastyczniejszych filtrów pakietowych.

Niestety w maju 2001 pomiędzy autorem, którym jest Darren Reed z Australii, a twórcami *OpenBSD* doszło do konfliktu na tle licencji filtra i od kolejnych wersji tego systemu nie będzie on dostępny natywnie. W przypadku *FreeBSD* i *NetBSD* nic się jednak nie zmieniło, a w dwa miesiące później po tym jak *OpenBSD* usunęło filtr ze swojego systemu Darren ponownie zmienił licencję, tym razem na liberalną i usuwającą wszystkie wcześniejsze ograniczenia.

Zasadniczą cechą, która wyróżnia *ipf* od innych filtrów pakietowych jest kolejność przetwarzania reguł. W większości filtrów są one przeglądane kolejno, od góry do dołu listy i pierwsza, która pasuje jest wykonywana, kończąc tym

samym przetwarzanie listy. W przypadku *ipf* wygląda to nieco inaczej — zawsze przeglądana jest cała lista reguł i ostatnia z nich, która pasowała do aktualnego pakietu jest wykonywana.

Poniżej opisany przykład ma za zadanie uwypuklenie różnic pomiędzy *ipf*, a wcześniej opisywanym *iptables*. Podstawy teoretyczne działania obu filtrów są jednak zbliżone, dlatego zwolenników *ipf* również namawiam do przeczytania teoretycznych fragmentów rozdziałów poświęconych Linuksowi.

6.1 Najprostszy przykład

Interfejsy sieciowe w BSD biorą nazwy od konkretnych modeli kart — poniższe przykłady wykorzystują interfejs `fxp0`, czyli w rzeczywistości kartę *Intel EtherExpress/100*.

```
### FXP0 Internet
block in log on fxp0 from any to any

# Zwracamy RST w odpowiedzi na zadania IDENT
block return-rst in quick on fxp0 proto tcp from any \
to any port = 113 flags S/S

# Wypuszczamy wszystkie połączenia wychodzące z "keep state"
pass out quick on fxp0 proto tcp from any to any keep state
pass out quick on fxp0 proto udp from any to any keep state
pass out quick on fxp0 proto icmp from any to any keep state
```

Widać tutaj zasadniczą różnicę w działaniu filtra *ipf*, jaką jest brak wyraźnego rozróżnienia pomiędzy poszczególnymi kierunkami ruchu, tak jak to było w przypadku *iptables*. W powyższym przykładzie blokujemy **cały** ruch **przychodzący**, poza pakietami należącymi do połączeń zainicjowanych z tego hosta. Są one wpuszczane przez filtr *stateful-inspection*, włączony flagą `keep state`.

Przykłady dla *ipf* będą rozbudowywane w miarę uzupełniania tego artykułu.

7 Od autora

7.1 Uwagi końcowe

Konfiguracje przedstawione w przykładach nie są kompletne. Mam nadzieję, że pomogły Ci one zrozumieć działanie filtrów *stateful-inspection* i będą punktem wyjścia do tworzenia skutecznych zapór. Najskuteczniejszą w tym wypadku strategią jest tworzenie konfiguracji maksymalnie restrykcyjnej i uważne obserwowanie logów systemowych. Każdy zablokowany pakiet należy

przeanalizować i — jeśli okaże się że był on wysyłany legalnie — rozszerzyć filter o regułę przepuszczającą ten rodzaj ruchu.

Zdaję sobie sprawę, że mogłem popełnić w tym artykule błędy, rzeczowe i inne, w takim wypadku będę wdzięczny za ich wskazanie. Komentarze oraz uzupełnienia są również mile widziane, znajdują się one w kolejnych wersjach tego dokumentu.

7.2 Zmiany

- **Wersja 0.6** poprawione wiele literówek, informacje o nowym *FORWARD* (podziękowania dla *Wesa Bawora*)
- **Wersja 0.5** poprawki odnośnie ipf, przykładowy skrypt „shivling”, informacje o *unclean*, *sysctl*
- **Wersja 0.4** poprawki (*podziękowania dla Cezarego Jackiewicza*) oraz „inne ciekawe”
- **Wersja 0.3** rozdział o RELATED i FTP, zmiana adresu strony *netfilter* i nowe adresy, opis konfiguracji oprogramowania
- **Wersja 0.2** opis modułu *unclean*
- **Wersja 0.1** pierwsza wersja tego artykułu

Nowe wersje tego dokumentu dostępne są pod adresem
<http://ipsec.pl/>

Paweł Krawczyk kravietz@aba.krakow.pl	ABA sp. z o.o. ul. Bociana 6 31-231 Kraków
--	--