Linux IPCHAINS-HOWTO

Rusty Russell

Erik Fichtner, emf@obfuscation.org

Wersja oryginalna: v1.0.8, Tue Jul 4 14:20:53 EST 2000

Oryginał tego dokumentu znajduje się pod adresem: http://netfilter.samba.org/

Tłumaczenie: Łukasz Bromirski, l.bromirski@mr0vka.eu.org

Wersja tłumaczenia: 2.1, $Date: 2002/08/29 22:41:53 $

Oryginał tłumaczenia znajduje się pod adresem: http://mr0vka.eu.org/tlumaczenia/ipchains.html

---

Dokument ten ma na celu opisanie jak pozyskać, zainstalować i skonfigurować rozszerzone oprogramowanie ściany ogniowej IP dla Linuksa, oraz przekazanie paru pomysłów na to, jak mógłbyś jej użyć.

---

1. Wprowadzenie

Dokument ten to IPCHAINS-HOWTO; W sekcji Gdzie? znajdziesz adres głównej strony, z której możesz ściągnąć najświeższą wersję. Powinieneś przeczytać również dokument NET-3-HOWTO. Dodatkowo, dokumenty IP-Masquerading HOWTO, PPP-HOWTO, Ethernet-HOWTO i Firewall HOWTO będą również ciekawą lekturą (oraz, oczywiście, FAQ listy alt.fan.bigfoot).

Jeśli filtrowanie pakietów jest ci już znane, przeczytaj sekcję Dlaczego?, sekcję Jak? i przejrzyj nagłówki w sekcji Łańcuchy ściany ogniowej IP.

Jeśli przechodzisz z `ipfwadm`, przeczytaj sekcję Wprowadzenie, sekcję Jak?, i dodatki w sekcji Różnice pomiędzy ipchains i ipfwadm oraz sekcję Obsługa skryptu `ipfwadm-wrapper'.

1.1 Co?

ipchains to przepisany od nowa kod ściany ogniowej dla IPv4 Linuksa (który jest w głównej mierze ukradziony z BSD) oraz przepisany od nowa `ipfwadm`, który z kolei jest przepisanym `ipfw` z BSD (jak podejrzewam). Od wersji kernela 2.1.102, filtrowanie pakietów IP wymaga zarządzania.

1.2 Dlaczego?

Stary kod ściany ogniowej Linuksa nie radzi sobie z fragmentami, ma 32-bitowe liczniki (przynajmniej na platformie Intel), nie pozwala na specyfikację protokołów innych niż TCP, UDP i ICMP, nie może wykonywać dużych zmian na poziomie 'atomów'; nie można w nim podawać reguł z inwersją, ma trochę zawiłości i może być trudny w zarządzaniu (co czyni go podatnym na błędy użytkownika).

1.3 Jak?

Aktualny kod znajduje się już w standardowej paczce kernela od wersji 2.1.102. Dla serii kerneli 2.0, będziesz musiał ściągnąć patch do kernela ze stron WWW. Jeśli Twój kernel 2.0 jest nowszy niż dostarczany patch, mimo wszystko powinieneś go zastosować - te kernele są raczej stabilne (tzn. patch do kernela 2.0.34 działa w porządku z kernelem 2.0.35). Ponieważ patch do kerneli 2.0 jest niekompatybilny z `ipportfw` i patchami do `ipautofw`, nie zalecam używania ich dopóki naprawdę nie potrzebujesz funkcjonalności którą oferuje `ipchains`.

1.4 Gdzie?

Oficjalna strona znajduje się w trzech miejscach: Dzięki Penguin Computing Dzięki Zespołowi SAMBA Dzięki Jimowi Pickowi

Jest również lista e-mail'owa poświęcona raportom o błędach, dyskusjom, rozwijaniu programu i o sposobach jego używania. Można do niej dołączyć wysyłając wiadomość zawierającą słowo '`subscribe ipchains-list`' pod adres `subscribe` na serwerze `east.balius.com`. Poczta do listy powinna być adresowana na `ipchains-list` na serwerze `east.balius.com`.

2. Podstawy filtrowania pakietów

2.1 Co?

Cały ruch przechodzący przez sieć jest przesyłany w formie **pakietów**. Na przykład, ściągnięcie tego pliku (powiedzmy że ma wielkość 50kB) może spowodować otrzymanie około 36 pakietów o długości 1460 bajtów każdy (te liczby dobrałem raczej z głowy).

Początek każdego pakietu mówi gdzie podróżuje, skąd przybył, opisuje typ i inne detale administracyjne. Ten początek każdego pakietu nazywamy **nagłówkiem** (ang. *header*). Reszta pakietu, zawierająca faktyczne dane które są transmitowane, jest zwykle nazywana **ciałem** (ang. *body*) pakietu.

Niektóre protokoły, takie jak TCP, których używa się do obsługi ruchu w sieci, obsługi poczty i zdalnego logowania, używają koncepcji "połączenia" -- zanim wysłane zostaną jakiekolwiek pakiety danych, wymieniane są inne pakiety (ze specjalnymi nagłówkami), konfigurujące połączenie. Brzmi to mniej więcej tak: "Chciałbym się połączyć", "OK", "Dzięki". Dopiero wtedy zaczyna się wymiana normalnych pakietów.

Filtr pakietów to oprogramowanie które sprawdza nagłówki pakietów w trakcie jak przez niego przechodzą i decyduje o ich losie. Może zdecydować że **anuluje** (ang. *deny*) pakiet (tj. pominie pakiet jakby nigdy go nie otrzymał), **zaakceptuje go** (ang. *accept*) (tj. pozwoli mu przejść dalej) lub **odrzuci** (ang. *reject*) (podobnie jak anulowanie, ale zawiadomi źródło pakietu, że został on odrzucony).

Filtrowanie pakietów pod linuksem jest wbudowane w kernel łącznie z paroma jeszcze trochę innymi sprytnymi rzeczami które można zrobić z pakietami, ale generalnie zajęcie oglądania nagłówków i decydowania o ich losie jest w nim również.

2.2 Dlaczego?

Kontrola. Bezpieczeństwo. Czujność.

Kontrola:

> kiedy używasz Linuksa by połaczyć Twoją wewnętrzną sieć z inną siecią (powiedzmy z Internetem) masz okazję wpuścić trochę ruchu różnego typu a część odrzucić. Na przykład, nagłówek pakietu posiada adres docelowy pakietu, więc możesz odrzucać pakiety które podróżują do określonych części sieci zewnętrznej. Innym przykładem może być to: używam Netscape do oglądania archiwów Dilbert-a. Jest tam masa reklam pochodzących z adresu doubleclick.net, więc Netscape traci czas by je ładować. Pouczenie filtra pakietów by nie wpuszczał pakietów podróżujących do i z tego adresu rozwiązuje ten problem (jednakże jest parę innych sposobów by zrobić to lepiej).

Bezpieczeństwo:

> kiedy Twój Linuks jest jedynym komputerem pomiędzy chaosem Internetu i Twoją ładną, uporządkowaną siecią, miło jest wiedzieć że możesz obłożyć restrykcjami to co nadchodzi do Twych drzwi. Na przykład, możesz pozwolić by wszystko wychodziło z Twojej sieci, ale możesz być zaniepokojony znanym atakiem 'Ping of Death' przeprowadzanym przez rozmaitych złośliwych ludzi. Innym przykładem może być Twoje życzenie, by nie zezwalać na telnet'owanie się na Twój komputer, mimo że wszystkie konta mają hasła; prawdopodobnie chcesz być (jak większość ludzi) raczej obserwatorem w Internecie a nie serwerem - po prostu nie dawać się nikomu do Ciebie dołączać, poprzez filtrowanie nadchodzących pakietów służących do ustanawiania połączeń.

Czujność:

> czasami źle skonfigurowana maszyna w sieci lokalnej zadecyduje o skierowaniu paru pakietów do sieci zewnętrznej. Miło jest móc poinstruować filtr pakietów by dał Ci znać o takich anormalnych zachowaniach; może będziesz chciał coś z tym zrobić, albo jesteś po prostu ciekawy takich przypadków.

2.3 Jak?

Kernel z filtrowaniem pakietów

Potrzebujesz kernela który ma nowy kod ipchains ściany ogniowej zawarty w sobie. Możesz to sprawdzić zaglądając do pliku `/proc/net/ip_fwchains`. Jeśli w ogóle istnieje, masz taki kernel.

Jeśli nie, to potrzebujesz kernela który spełnia ten warunek. Po pierwsze, ściągnij źródła kernela którego potrzebujesz. Jeśli masz kernel w wersji 2.1.102 lub wyższej, nie będziesz potrzebował patcha (jest już w źródłach). W innym przypadku musisz zastosować patch dostępny spod adresu WWW podanego wcześniej i ustawić konfigurację jak to pokazano niżej. Jeśli nie wiesz jak to zrobić nie panikuj - przeczytaj Kernel-HOWTO.

Poniżej znajdują się opcje konfiguracji których będziesz potrzebował by ustawić **kernel w wersji 2.0.x**:

```
CONFIG_EXPERIMENTAL=y
CONFIG_FIREWALL=y
CONFIG_IP_FIREWALL=y
CONFIG_IP_FIREWALL_CHAINS=y
```

Dla kerneli w wersjach **2.1.x i 2.2.x** musisz ustawić:

```
CONFIG_FIREWALL=y
CONFIG_IP_FIREWALL=y
```

Narzędzie `ipchains` rozmawia z kernelem i mówi mu jak filtrować pakiety. Dopóki nie jesteś programistą, lub nadzwyczaj ciekawski, tak będziesz kontrolował filtrowanie pakietów.

ipchains

Narzędzie `ipchains` dodaje i usuwa reguły z sekcji filtrującej pakiety kernela. To oznacza, że cokolwiek byś nie ustawił, zostanie stracone po restarcie; zajrzyj do sekcji Zapisywanie reguł na stałe by dowiedzieć się jak upewnić się że reguły zostaną odzyskane po następnym uruchomieniu Linuksa.

`ipchains` zastępuje `ipfwadm`, który był używany ze starym kodem ściany ogniowej IP. Dostępny jest zestaw użytecznych skryptów z serwera http:

http://netfilter.filewatcher.org/ipchains/ipchains-scripts-1.1.2.tar.gz

Pakiet zawiera skrypt powłoki nazwany `ipfwadm-wrapper`, który pozwala Ci na filtrowanie pakietów w sposób podobny jak to robiłeś w starych wersjach kernela. Prawdopodobnie nie powinieneś jednak używać tego skryptu, chyba że chcesz naprawdę szybkiego sposobu upgradeu systemu który do tej pory używał `ipfwadm` (jest wolniejszy, nie sprawdza argumentów itp.). Jeśli tak zrobisz, nie musisz już praktycznie czytać tego HOWTO.

Zajrzyj do Załącznika Różnice pomiędzy ipchains i ipfwadm oraz do Używanie skryptu `ipfwadm-wrapper' po więcej informacji dotyczących `ipfwadm`.

Zapisywanie reguł na stałe

Twoje aktualne ustawienia ściany ogniowej zachowywane są w kernelu, więc zostaną stracone w przypadku resetu maszyny. Rekomenduję użycie skryptów 'ipchains-save' i 'ipchains-restore' by zachowywać reguły. By ich użyć, ustaw reguły a potem wykonaj (jako root):

```
# ipchains-save > /etc/ipchains.rules
#
```

Utwórz skrypt taki jak ten:

```
#! /bin/sh
# Skrypt kontrolujący filtrowanie pakietów

# Jeśli nie ma reguł, nie rób nic
[ -f /etc/ipchains.rules ] || exit 0

case "$1" in
    start)
        echo -n "Turning on packet filtering:"
        /sbin/ipchains-restore < /etc/ipchains.rules || exit 1
        echo 1 > /proc/sys/net/ipv4/ip_forward
        echo "."
        ;;
    stop)
        echo -n "Turning off packet filtering:"
        echo 0 > /proc/sys/net/ipv4/ip_forward
        /sbin/ipchains -F
        /sbin/ipchains -X
        /sbin/ipchains -P input ACCEPT
        /sbin/ipchains -P output ACCEPT
        /sbin/ipchains -P forward ACCEPT
        echo "."
        ;;
    *)
        echo "Usage: /etc/init.d/packetfilter {start|stop}"
        exit 1
        ;;
esac

exit 0
```

Upewnij się że zostanie on uruchomiony we wczesnej fazie procedury startowej. W moim przypadku (Debian 2.1) wykonałem symboliczny link 'S39packetfilter' w katalogu '/etc/rcS.d' (zostanie uruchomiony przed plikiem 'S40network').

3. Jestem skołowany! Ruting, masquerading, porforwarding, ipautofw...

Ten dokument HOWTO poświęcono filtrowaniu pakietów. To znaczy decydowaniu, czy pakiet powinien móc przejść przez nasz komputer czy nie. Jednakże Linuks, będąc tak naprawdę placem zabaw hackerów, jest na tyle narażony że prawdopodobnie chciałbyś wiedzieć trochę więcej o filtrowaniu.

Problemem jest, że to samo narzędzie (ipchains) używane jest do kontrolowania i maskarady i transparentnego proxy, mimo że są to różne techniki filtrowania pakietów (aktualna implementacja Linuksa zamazuje różnice między nimi w nienaturalny sposób, sprawiając wrażenie że obie techniki są naturalnie blisko ze sobą związane).

Maskarada i stosowanie proxy opisane są w osobnych HOWTO, a **automatyczne przekazywanie** (ang. *auto-forwarding*) i **przekazywanie portów** (ang. *port-forwarding*) są kontrolowane przez osobne narzędzia, ale ponieważ tak wielu ludzi ciągle pyta mnie o nie, włączę zestaw typowych scenariuszy i pokażę, które z narzędzi powinno zostać użyte. Zagadnienia bezpieczeństwa każdej konfiguracji nie będą przedmiotem dyskusji.

3.1 Trzy linijkowy Przewodnik Rusty'ego do Maskarady

Poniższy przykład zakłada, że twój zewnętrzny interfejs nazywa się 'ppp0'. Sprawdź to poleceniem `ifconfig` i ewentualnie zmień użyty niżej interfejs:

```
# ipchains -P forward DENY
# ipchains -A forward -i ppp0 -j MASQ
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

3.2 Bezpłatna promocja: WatchGuard rządzi

Można kupić ścianę ogniową prosto z półki. Doskonały jest WatchGuard FireBox. Jest taki dobry, ponieważ go lubię, jest bezpieczny, oparty na Linuksie i ponieważ funduje również utrzymanie ipchains jak również nowego kodu do ściany ogniowej (przeznaczonego do kerneli wersji 2.3.x i 2.4). W skrócie, WatchGuard płaci mi za jedzenie w czasie gdy pracuje dla was. Więc weźcie pod uwagę ich sprzęt. http://www.watchguard.com

3.3 Najpopularniejsze konfiguracje ze ścianami ogniowymi

Utrzymujesz serwer littlecorp.com. Masz sieć wewnętrzną i pojedyńczą linię (PPP) do Internetu (firewall.littlecorp.com który ma adres 1.2.3.4). Używasz Ethernetu w swojej sieci lokalnej a Twoja osobista maszyna nazywa się "myhost".

Ta sekcja pokaże różne konfiguracje które są dosyć powszechne. Czytaj uważnie, ponieważ każda z nich jest czasami tylko subtelnie różna od pozostałych.

Sieć prywatna: tradycyjne proxy

W tym scenariuszu, pakiety z sieci prywatnej nigdy nie podróżują do Internetu i vice versa. Adres IP sieci prywatnej powinien być przydzielony z jednej z trzech puli adresów zdefiniowanych w RFC1597 (10.*.*.*, 172.16.*.* lub 192.168.*.*).

Jedynym sposobem w jaki w ogóle można się połączyć do Internetu jest dołączenie się do ściany ogniowej, która jest jedyną maszyną podłączoną do obu sieci naraz. Uruchamiasz program (na ścianie ogniowej) nazywany proxy by to zrobić (są proxy do FTP, usług WWW, telnetu, RealAudio, newsów i innych usług). Zajrzyj po więcej informacji do Firewall HOWTO.

Każda usługa z której chcesz korzystać w Internecie, musi być uruchomiona na ścianie ogniowej (ale zajrzyj do Ograniczone usługi wewnętrzne poniżej).

Przykład: Umożliwienie dostępu do stron WWW z sieci prywatnej do Internetu:

1. Sieć prywatna ma przydzielone adresy 192.168.1.*, komputer 'myhost' ma adres 192.168.1.100 a interfejs Ethernetowy ściany ogniowej ma adres 192.168.1.1.
2. Proxy WWW (np. 'squid') jest zainstalowany i skonfigurowany na ścianie ogniowej, pracuje na porcie 8080.
3. Netscape w sieci prywatnej jest skonfigurowany i używa portu 8080 na ścianie ogniowej jako proxy.
4. DNS nie musi być skonfigurowany w sieci prywatnej.
5. DNS musi być skonfigurowany na ścianie ogniowej.
6. **Domyślna brama** (ang. *default gateway*) również nie musi być skonfigurowana w sieci prywatnej.

Netscape na komputerze myhost czyta http://slashdot.org.

1. Netscape łączy się ze ścianą ogniową z portem 8080, używając portu 1050 na komputerze myhost. Prosi o stronę http://slashdot.org.
2. Proxy sprawdza nazwę 'slashdot.org' i dostaje adres IP 207.218.152.131. Otwiera połączenie do tego adresu IP (używając portu 1024 na interfejsie zewnętrznym ściany ogniowej), i prosi serwer WWW (na porcie 80) o określoną stronę.
3. Gdy otrzymuje stronę, kopiuje te dane do połączenia z Netscape.
4. Netscape rysuje stronę

Na przykład z punktu widzenia slashdot.org, połączenie wykonuje maszyna 1.2.3.4 (interfejs PPP ściany ogniowej) z portu 1025, pod adres 207.218.152.131 (slashdot.org) na port 80. Z punktu widzenia maszyny myhost, połączenie wykonuje 192.168.1.100 (myhost) z portu 1025, adresem docelowym jest 192.168.1.1 (interfejs Ethernetowy ściany ogniowej) port 8080.

Sieć prywatna: transparentne proxy

W tym scenariuszu, pakiety z sieci prywatnej nigdy nie podróżują do Internetu i vice versa. Adres IP sieci prywatnej powinien być przydzielony z jednej z trzech puli adresów zdefiniowanych w RFC1597 (10.*.*.*, 172.16.*.* lub 192.168.*.*).

Jedynym sposobem w jaki w ogóle można się połączyć do Internetu jest dołączenie się do ściany ogniowej, która jest jedyną maszyną podłączoną do obu sieci naraz. Uruchamiasz program (na ścianie ogniowej) nazywany transparentnym proxy by to zrobić; kernel wysyła wychodzące pakiety do transparentnego proxy zamiast wysyłać je po prostu do drugiej sieci (tzn. pomija ruting).

Transparentne proxy oznacza, że klienci nie muszą w ogóle wiedzieć że działa jakieś proxy.

Każda usługa z której chcesz korzystać w Internecie, musi być uruchomiona na ścianie ogniowej (ale zajrzyj do Ograniczone usługi wewnętrzne poniżej).

Przykład: Umożliwienie dostępu do stron WWW z sieci prywatnej do Internetu:

1. Sieć prywatna ma przydzielone adresy 192.168.1.*, komputer myhost ma adres 192.168.1.100 a interfejs Ethernetowy ściany ogniowej ma adres 192.168.1.1.
2. Transparentne proxy WWW (jak podejrzewam są patche dla squida by pracował w tym trybie, możesz również spróbować 'transproxy') jest zainstalowany i skonfigurowany i używa portu 8080 na ścianie ogniowej.
3. Kernel ma zdefiniowane, by przekierowywać połączenia na port 80 do proxy, używając ipchains.
4. Netscape w sieci prywatnej jest skonfigurowany do połączeń bezpośrednich.
5. DNS musi być skonfigurowany w sieci prywatnej (tzn. musisz uruchomić serwer DNS jako proxy na ścianie ogniowej).
6. Domyślna brama musi być skonfigurowana dla sieci prywatnej, by można było wysyłać pakiety do ściany ogniowej.

Netscape na maszynie myhost odwołuje się do strony http://slashdot.org

1. Netscape szuka nazwy slashdot.org i dostaje adres 207.218.152.131. Otwiera połączenie do tego adresu IP przez port 1050, i prosi serwer WWW (na porcie 80) o określoną stronę.
2. W momencie, gdy pakiety z komputera myhost (port 1050) do slashdot.org (port 80) przechodzą przez ścianę ogniową, są przekierowywane do transparentnego proxy oczekującej na porcie 8080 ściany ogniowej. Transparentne proxy otwiera połaczenie (używając własnego portu 1025) do 207.218.152.131 na port 80 (który jest oryginalnym adresem, na który pakiety miały dojść).
3. Gdy proxy otrzymuje dane ze swojego połączenia z serwerem WWW, kopiuje dane do połączenia z Netscape.
4. Netscape rysuje stronę.

Z punktu widzenia slashdot.org, połączenie nadchodzi z 1.2.3.4 (interfejs PPP ściany ogniowej) i portu 1025 do 207.218.152.131 (slashdot.org) port 80. Z punktu widzenia komputera myhost połączenie wykonywane jest z adresu 192.168.1.100 (myhost) i portu 1050 do adresu 207.218.152.131 (slashdot.org) i na port 80, ale tak naprawdę rozmawia on przez transparentną proxy.

Sieć prywatna: Maskarada

W tym scenariuszu, pakiety z sieci prywatnej nigdy nie podróżują do Internetu i vice versa bez specjalnego potraktowania. Adres IP sieci prywatnej powinien być przydzielony z jednej z trzech puli adresów zdefiniowanych w RFC1597 (10.*.*.*, 172.16.*.* lub 192.168.*.*).

Zamiast używać proxy, używamy specjalnej właściwości kernela nazywanej **maskaradą** (ang. *masquerading*). Maskarada przepisuje pakiety kiedy przechodzą przez ścianę ogniową więc wydają się zawsze pochodzić właśnie z niej. Następnie przepisuje od nowa również odpowiedzi, więc wyglądają tak jakby nadchodziły od originalnego adresata.

Maskarada ma osobne moduły które obsługują 'udziwnione' protokoły, takie jak FTP, RealAudio, Quake itp. Dla bardzo trudnych do obsłużenia protokołów, możliwe jest włączenie 'auto-forwardingu', który może obsłużyć część z nich przez automatyczne ustawienie **przekazywania** (ang. *forwardingu*) dla określonych zestawów portów: szukaj `ipportfw` (w kernelach serii 2.0.x) lub `ipmasqadm` (w kernelach serii 2.1.x).

Każda usługa z której chcesz korzystać w Internecie, musi być uruchomiona na ścianie ogniowej (ale zajrzyj do

Ograniczone usługi wewnętrzne poniżej).

Przykład: Umożliwienie dostępu do stron WWW z sieci prywatnej do Internetu:

1. Sieć prywatna ma przydzielone adresy 192.168.1.*, komputer myhost ma adres 192.168.1.100 a interfejs Ethernetowy ściany ogniowej ma adres 192.168.1.1.
2. Ściana ogniowa ma skonfigurowaną maskaradę dla dowolnych pakietów które wychodzą z sieci prywatnej i podróżują w kierunku portu 80 hostów Internetowych.
3. Netscape w sieci prywatnej jest skonfigurowany do połączeń bezpośrednich.
4. DNS musi być skonfigurowany w sieci prywatnej.
5. Domyślna brama musi być skonfigurowana dla sieci prywatnej i ustawiona na ścianę ogniową.

Netscape na maszynie myhost odwołuje się do strony http://slashdot.org

1. Netscape szuka nazwy slashdot.org i dostaje adres 207.218.152.131. Otwiera połączenie do tego adresu IP przez port 1050, i prosi serwer WWW (na porcie 80) o określoną stronę.
2. W momencie, gdy pakiety z komputera myhost (port 1050) do slashdot.org (port 80) przechodzą przez ścianę ogniową, są przepisywane tak jakby nadchodziły z interfejsu PPP ściany ogniowej, z portu 65000. Ściana ogniowa ma prawidłowy adres Internetowy (1.2.3.4) więc pakiety odpowiedzi mogą wrócić bez przeszkód.
3. W momencie gdy pakiety z slashdot.org (port 80) docierają do ściany ogniowej (port 65000), są przepisywane i wysyłane do myhost na port 80. To prawdziwa magia maskarady, ponieważ pamięta kiedy przepisuje wychodzące pakiety i potrafi skoordynować je z pakietami odpowiedzi i je również prawidłowo przepisać.
4. Netscape rysuje stronę.

Z punktu widzenia slashdot.org, połączenie nadchodzi z 1.2.3.4 (interfejs PPP ściany ogniowej) i portu 65000 do 207.218.152.131 (slashdot.org) port 80. Z punktu widzenia komputera myhost połączenie wykonywane jest z adresu 192.168.1.100 (myhost) i portu 1050 do adresu 207.218.152.131 (slashdot.org) i na port 80.

Sieć publiczna

W tym scenariuszu, Twoja własna sieć jest częścią Internetu i pakiety przepływają bez zmian między sieciami. Pula adresów IP sieci prywatnej musi być przydzielona przez złożenie podania o blok adresów IP, tak by reszta sieci wiedziała jak skierować do Ciebie pakiety. Implikuje to połączenie stałe.

W tym przypadku, filtr pakietów jest używany do wymuszenia restrykcji, które pakiety mogą być przekazywane pomiędzy siecią a resztą Internetu, np. obostrzenie, że od strony Internetu można osiągnąć tylko serwery WWW Twojej sieci prywatnej.

Przykład: Umożliwienie dostępu do stron WWW z sieci prywatnej do Internetu:

1. Adresy w twojej sieci prywatnej są przydzielone zgodnie z zarejestrowanym blokiem numerów IP które otrzymałeś (powiedzmy 1.2.3.*).
2. Ściana ogniowa jest skonfigurowana tak, by przepuszczać cały ruch.
3. Netscape jest skonfigurowany do połączeń bezpośrednich.
4. DNS musi być skonfigurowany prawidłowo w Twojej sieci.
5. Ściana ogniowa powinna być domyślną bramą dla sieci prywatnej.

Netscape na maszynie myhost odwołuje się do strony http://slashdot.org

1. Netscape szuka nazwy slashdot.org i dostaje adres 207.218.152.131. Otwiera połączenie do tego adresu IP przez port 1050, i prosi serwer WWW (na porcie 80) o określoną stronę.
2. Pakiety przechodzą przez ścianę ogniową, tak jak przechodzą przez parę innych ruterów na drodze między tobą a slashdot.org.
3. Netscape rysuje stronę.

Tzn. jest tylko jedno połączenie: z 1.2.3.100 (myhost) port 1050, do 207.218.152.131 (slashdot.org) port 80.

Ograniczone usługi wewnętrzne

Jest parę sposobów, które możesz zastosować by udostępnić Internetowi dostęp do Twoich usług w sieci, zanim zaczniesz uruchamiać odpowiednie serwisy na ścianie ogniowej. Będą one pracowały na zasadach takich jak proxy lub

maskarada dla połączeń zewnętrznych.

Najprostszym podejściem jest zainstalowanie 'przekierowywacza', który jest odpowiednikiem proxy dla ubogich - czeka na połączenie na danym porcie, a następnie otwiera połączenie do zdefiniowanego hosta i portu w sieci wewnętrznej, oraz kopiuje dane między tymi dwoma połączeniami. Przykładem takiego programu może być 'redir'. Z punktu widzenia Internetu, połączenie jest realizowane do Twojej ściany ogniowej. Z punktu widzenia twojego serwera w sieci prywatnej, połączenie jest realizowane ze ściany ogniowej do twojego serwera.

Innym podejściem (które wymaga patcha dla kerneli serii 2.0.x dla `ipportfw` lub kernela w wersji 2.1.x i późniejszych) jest używanie przekazywania w samym kernelu. Robi on tą samą robotę co program 'redir' tylko trochę w inny sposób: kernel przepisuje pakiety w trakcie gdy one przechodzą przez ścianę ogniową, zmieniając ich adres docelowy i port na host i port w sieci prywatnej. Z punktu widzenia Internetu połączenie jest realizowane do Twojej ściany ogniowej. Z punktu widzenia serwera w sieci prywatnej, wykonywane jest bezpośrednie połączenie z Internetu do serwera.

3.4 Więcej informacji o maskaradzie

David Ranch napisał wspaniałe HOWTO poświęcone maskaradzie, które w większości dubluje się z tym HOWTO. Możesz je znaleźć pod adresem http://www.linuxdoc.org/HOWTO/IP-Masquerade-HOWTO.html.

Oficjalna strona maskarady znajduje się pod adresem http://ipmasq.cjb.net.

4. Łańcuchy IP ściany ogniowej

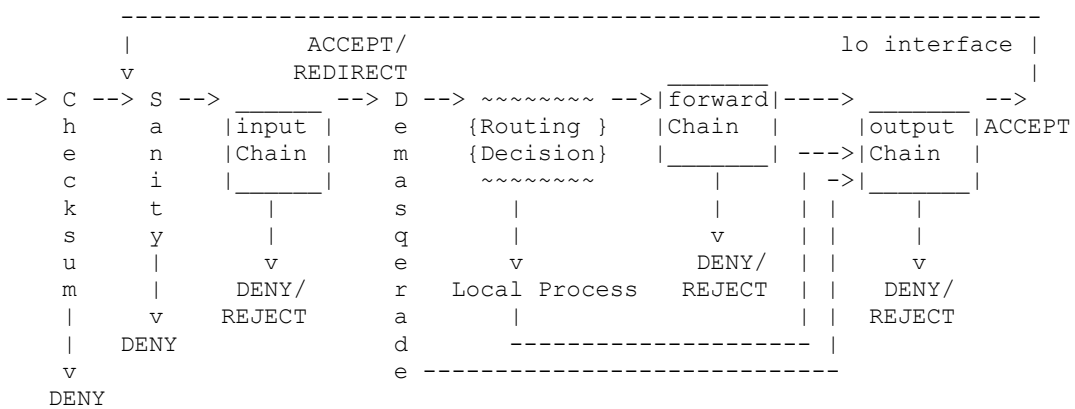
Ta sekcja opisuje co tak naprawdę powinieneś wiedzieć by zbudować sobie filtr pakietów który zaspokoi Twoje potrzeby.

4.1 Jak pakiety podróżują przez filtry

Kernel zaczyna z trzema listami reguł; listy te są nazywane **łańcuchami ściany ogniowej** lub po prostu **łańcuchami**. Nazywają sie one **input** (wejściowy), **output** (wyjściowy) i **forward** (przekazujący). Kiedy pakiet dociera do komputera (powiedzmy, przez kartę Ethernetową), kernel używa łańcucha `input` by zdecydować o jego losie. Jeśli pakiet przeżyje ten krok, kernel decyduje gdzie go wysłać (nazywa się to rutingiem). Jeśli jest skierowany do innej maszyny, sprawdza również `forward`. Na koniec, zanim pakiet opuści maszynę, kernel sprawdza jeszcze łańcuch `output`.

Łańcuch to zbiór reguł. Każda reguła mówi 'jeśli nagłówek pakietu wygląda tak, to zrobimy z nim właśnie to'. Jeśli reguła nie dotyczy pakietu, sprawdzana jest następna. Jeśli nie ma już żadnych reguł do sprawdzenia, kernel sprawdza jeszcze **politykę** (ang. *policy*) łańcucha by zdecydować co zrobić. W bardzo bezpiecznym otoczeniu, polityka mówi zwykle żeby odrzucić ten pakiet.

Dla fanów ASCII, poniżej rysunek drogi pakietu, który przechodzi przez maszynę:



A teraz opiszę punkt po punkcie każdy etap:

Checksum (suma kontrolna):

To test, czy pakiet nie jest uszkodzony w jakiś sposób. Jeśli jest, zostaje odrzucony (anulowany).

Sanity (poprawność):

To jeden z testów poprawności przed każdym łańcuchem ściany ogniowej, ale w przypadku łańcucha wejściowego jest najbardziej ważny. Niektóre zniekształcone pakiety mogłyby sprawić duży problem kodowi sprawdzającemu reguły i są one tutaj odrzucane (zostaje to odnotowane informacją w syslogu).

input chain (łańcuch wejściowy):

To pierwszy łańcuch ściany ogniowej, na którym pakiet będzie testowany. Jeśli werdykt nie zostanie orzeczony na `DENY` (anulować) lub `REJECT` (odrzucić), pakiet przechodzi dalej.

Demasquerade (demaskarada):

Jeśli pakiet jest odpowiedzią na poprzedni, zamaskaradowany pakiet, jest demaskaradowany i przechodzi bezpośrednio do łańcucha wyjściowego (output). Jeśli nie używasz maskarady IP, możesz ten etap wymazać z myśli.

Routing decision (decyzja o rutingu):

Kod odpowiedzialny za ruting analizuje pole przeznaczenia, by zdecydować czy pakiet ma trafić do lokalnego procesu (sprawdź 'proces lokalny' poniżej) lub przekazany do zdalnej maszyny (sprawdź 'łańcuch przekazujący', również poniżej).

Local process (proces lokalny):

Proces działający na maszynie może otrzymywać pakiety po decyzji z poprzedniego punktu, jak również wysyłać pakiety (które po decyzji o rutingu z punktu wyżej, trafiają do łańcucha wyjściowego (output)).

lo interface (interfejs lo):

Jeśli pakiety z lokalnego procesu mają trafić do lokalnego procesu, przejdą przez łańcuch wyjściowy (output) z interfejsem ustawiony na 'lo', a potem wrócą przez łańcuch wejściowy (input), również przez interfejs 'lo'. Interfejs ten jest zwykle nazywany **pętlą zwrotną** (ang. *loopback*).

local (lokalnie):

Jeśli pakiet nie został wykreowany przez proces lokalny, sprawdzany jest łańcuch przekazujący (forward), w innym wypadku pakiet przekazywany jest do łańcucha wyjściowego (output).

forward chain (łańcuch przekazujący):

Łańcuch ten jest sprawdzany kiedy pakiet stara się przejść przez tą maszynę do innej.

output chain (łańcuch wyjściowy):

Ten łańcuch jest z kolei sprawdzany dla wszystkich pakietów, zanim opuszczą one maszynę.

Używanie ipchains

Po pierwsze, sprawdź czy masz wersję ipchains na której opiera się ten dokument:

```
$ ipchains --version
ipchains 1.3.9, 17-Mar-1999
```

Proszę zauważyć, że zalecam 1.3.4 (które nie ma długich opcji, takich jak '--sport'), lub 1.3.8 i wyższe - są one bardzo stabilne.

ipchains posiadają całkiem szczegółowy podręcznik (`man ipchains`), i jeśli potrzebujesz szczegółów jakiejś konkretnej opcji, powinieneś sprawdzić interfejs programowy (`man 4 ipfw`), lub plik `net/ipv4/ip_fw.c` znajdujący się w źródłach kernela serii 2.1.x. Jest jak najbardziej autorytatywny.

Jest również ściąga autorstwa Scott'a Bronson'a w paczce źródłowej, zarówno w formatach A4 i US Letter w formacie PostScript(TM).

Istnieje parę różnych rzeczy które możesz zrobić z `ipchains`. Po pierwsze, możesz operować całymi łańcuchami. Zaczynasz z trzema wbudowanymi łańcuchami: `input` (wejściowym), `output` (wyjściowym) i `forward` (przekazującym), których nie możesz skasować.

1. Utworzenie nowego łańcucha (-N).
2. Skasowanie pustego łańcucha (-X).
3. Zmiana polityki dla wbudowanego łańcucha (-P).
4. Lista reguł w łańcuchu (-L).
5. Oczyszczenie łańcucha z reguł (-F).
6. Wyzerowanie liczników bajtów i pakietów we wszystkich regułach w danym łańcuchu (-Z).

Jest kilka sposobów na manipulacje regułami wewnątrz łańcucha:

1. Dodaj nową regułę do łańcucha (-A).
2. Dodaj nową regułę na jakiejś pozycji w łańcuchu (-I).
3. Zastąp regułę na jakiejś pozycji w łańcuchu (-R).
4. Skasuj regułę na jakiejś pozycji w łańcuchu (-D).
5. Skasuj pierwszą z pasujących reguł z łańcucha (-D).

Jest również parę operacji na maskaradzie, które wbudowane są w `ipchains` z chęci znalezienia dla nich dobrego miejsca:

1. Lista aktualnie maskaradowanych połączeń (-M -L).
2. Ustawienie timeoutów dla maskarady (-M -S). (i sprawdź Nie mogę ustawić timeoutów maskarady!).

Ostatnia (i najprawdopodobniej najużyteczniejsza) funkcja, pozwala Ci co by się stało gdyby dany pakiet dotarł do danego łańcucha.

Co widać gdy wystartujesz komputer

Zanim jakakolwiek komenda ipchains została wykonana (uwaga: niektóre dystrybucje uruchamiają ipchains w skryptach inicjujących), nie ma reguł we wbudowanych łańcuchach (`input', `forward' i `output'), a polityka każdego z nich ustawiona jest na ACCEPT. To najbardziej otwarte podejście jakie możesz mieć.

Operacje na pojedyńczej regule

To właśnie chleb powszedni ipchains: manipulacja regułami. Najczęściej, będziesz używał dodawania (`-A`) i kasowania (`-D`). Inne operacje (`-I` dla wstawiania i `-R` dla zastępowania) są prostymi rozszerzeniami tych koncepcji.

Każda reguła podaje zestaw wymagań, które pakiet musi spełniać, oraz co zrobić gdy je spełnia czyli **cel** (ang. *target*). Na przykład, mógłbyś chcieć odrzucać wszystkim pakiety ICMP nadchodzące z adresu IP 127.0.0.1. W tym przypadku protokołem będzie ICMP, adresem źródłowym będzie 127.0.0.1 a celem - `DENY` (anulowanie).

127.0.0.1 to interfejs loopback, który masz nawet wtedy, gdy nie ma połączenia z żadną siecią. Możesz użyć programu 'ping' by wygenerować takie pakiety (wysyła on pakiet ICMP typ 8 **żądnie echa** (ang. *echo request*) na który wszystkie współpracujące hostu powinny odpowiedzieć pakietem ICMP typ 0 **odpowiedź na echo** (ang. *echo reply*). Czyni go to użytecznym do testów.

```
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
```

```
round-trip min/avg/max = 0.2/0.2/0.2 ms
# ipchains -A input -s 127.0.0.1 -p icmp -j DENY
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
#
```

Widać tutaj że pierwszy ping powiódł się (parametr '-c 1' mówi pingowi by wygenerował tylko jeden pakiet).

Następnie dołączamy (`-A`) do łańcucha 'input' regułę mówiącą, że pakiety nadchodzące z adresu 127.0.0.1 ('`-s 127.0.0.1`') i używające protokołu ICMP ('`-p ICMP`') powinny trafić zostać anulowane ('`-j DENY`').

Następnie testujemy naszą regułę, wykonując drugi raz pinga. Nastąpi pauza, po której program podda się - po oczekiwaniu na pakiet który nigdy nie wróci.

Możemy skasować naszą regułę na dwa sposoby. Po pierwsze, ponieważ wiemy że jest to jedyna reguła w łańcuchu wejściowym, możemy użyć numerowania reguł, tak jak poniżej:

```
# ipchains -D input 1
#
```

Polecenie skasuje regułę numer 1 w łańcuchu wejściowym.

Drugi sposób to dokładne przepisanie poleceń po opcji `-A`, ale zamiast opcji `-A` podajemy opcję `-D`. Przydaje się to w przypadku gdy masz skomplikowany zestaw reguł i nie chce ci się liczyć ich wszystkich by ustalić w końcu że chcesz pozbyć się reguły numer 37. W tym wypadku użyjemy:

```
# ipchains -D input -s 127.0.0.1 -p icmp -j DENY
#
```

Składnia polecenia `-D` musi dokładnie odpowiadać opcjom które podałeś przy poleceniu `-A` (lub `-I` czy `-R`). Jeśli istnieje wiele identycznych reguł w tym samym łańcuchu, tylko pierwsza pasująca zostania skasowana.

Specyfikacja filtrowania

Widzieliśmy już jak używać opcji '`-p`' by wskazać protokół i '`-s`' by wskazać adres źródłowy, ale są jeszcze inne opcje których możemy użyć by scharakteryzować pakiet. Poniżej znajdziesz wyczerpujące kompendium.

Wskazanie adresów IP: źródłowego i docelowego

Adresy IP źródłowy ('`-s`') i docelowy ('`-d`') mogą być podane na cztery sposoby. Najczęściej robi się to przez podanie pełnej nazwy, takiej jak 'localhost' czy `www.linuxhq.com'. Drugim sposobem jest podanie adresu IP, tak jak np. '127.0.0.1'.

Trzeci i czwarty sposób pozwalają na wskazanie grupy adresów IP, tak jak na przykład '199.95.207.0/24' lub `199.95.207.0/255.255.255.0'. Oba wskazują na zakres adresów IP od 199.95.207.0 do 199.95.207.255 włącznie; cyfry po '/' mówią która część adresu IP ma znaczenie. '/32' czy inaczej `/255.255.255.255' jest domyślne (i mówi że wszystkie liczby w adresie IP są ważne). By wskazać dowolny adres, można użyć '/0' tak jak poniżej:

```
# ipchains -A input -s 0/0 -j DENY
#
```

Ale takich konstrukcji używa się rzadko, ponieważ efekt jest dokładnie taki sam jak w przypadku nie podania opcji '`-s`' w ogóle.

Inwersja

Wiele flag, włączając '`-s`' i '`-d`' można poprzedzić znakiem '`!`' (wymawianym 'nie') by wskazać adresy **nie** pasujące do tych podanych. Na przykład '`-s ! localhost`' będzie pasować do wszystkich pakietów nie pochodzących z localhost.

Nie zapomij o spacjach wokół '`!`': są naprawdę potrzebne.

Protokół

Protokół podaje się po parametrze '`-p`'. Protokół może być numerem (jeśli znasz wartości numeryczne protokołów IP) lub nazwą dla 'TCP', 'UDP' i 'ICMP'. Wielkość liter nie ma znaczenia, więc 'tcp' działa tak samo jak 'TCP'.

Nazwa protokołu może być poprzedzona przez znak '`!`' by wskazać na wszystkie oprócz wymienionego, tak jak na przykład '`-p ! TCP`' (warunek dotyczy wszystkich protokołów prócz TCP).

Porty dla UDP i TCP

W specjalnym przypadku gdy podajemy protokół TCP lub UDP, można podać dodatkowy parametr określający (lub ich grupę), który nas interesuje (sprawdź Obsługa fragmentów poniżej). Grupę podaje się używając znaku ':', tak jak na przykład '`6000:6010`' - ten przedział dotyczy 11 portów, od 6000 do 6010 włącznie. Jeśli ominiemy dolną granicę, jest ona przyjmowana domyślnie na 0. Jeśli ominiemy górną granicę, jest ona przyjmowana na 65535. Więc aby podać porty TCP poniżej 1024, możemy napisać '`-p TCP -s 0.0.0.0/0 :1023`'. Numery portów mogą być również podane jako nazwy, np. '`www`'.

Zauważ że porty również mogą być poprzedzane znakiem '`!`', który powoduje ich zanegowanie. Aby podać wszystkie pakiety TCP OPRÓCZ pakietów WWW, możesz napisać '`-p TCP -d 0.0.0.0/0 ! www`'.

Bardzo ważne, by zauważyć, że konstrukcja

```
-p TCP -d ! 192.168.1.1 www
```

jest bardzo różna od

```
-p TCP -d 192.168.1.1 ! www
```

Pierwsza mówi o pakietach TCP do portu WWW na każdej maszynie oprócz 192.168.1.1 a druga, o pakietach TCP na dowolne porty na maszynie 192.168.1.1 oprócz portu WWW.

Na koniec, przykład w którym chodzi nam o wszystko oprócz portu WWW i maszyny 192.168.1.1:

```
-p TCP -d ! 192.168.1.1 ! www
```

Kod i Typ dla ICMP

Protokół ICMP również umożliwia podanie dodatkowego argumentu, ale ponieważ nie posiada portów (ICMP ma **typ** i **kod**), mają one inne znaczenie.

Możesz podać je w formie nazw ICMP (użyj '`ipchains -h icmp`' by otrzymać listę nazw) po opcji '`-s`' lub jako typ i kod numeryczny ICMP, w którym typ występuje po opcji '`-s`' a kod po opcji '`-d`'.

Nazwy ICMP są raczej długie: musisz użyć tylko tylu liter, by wskazywała jednoznacznie na którąś ze zdefiniowanych.

Poniżej mała tabelka najbardziej popularnych pakietów ICMP:

```
Numer   Nazwa                   Wymagane przez

0       echo-reply              ping
3       destination-unreachable ruch TCP/UDP
5       redirect                ruting, jeśli nie działa demon rutingu
8       echo-request            ping
11      time-exceeded           traceroute
```

Zauważ że nazwy ICMP nie mogą być aktualnie poprzedzane parametrem '`!`'.

NIGDY NIGDY NIGDY nie blokuj wszystkich pakietów typu 3 ICMP! (sprawdź niżej Pakiety ICMP).

Interfejs

Opcja '`-i`' powoduje wskazanie **interfejsu**. Interfejs to fizyczne urządzenie do którego pakiet dociera lub z którego wychodzi. Możesz użyć polecenia '`ifconfig`' by wylistować interfejsy które są podniesione.

Interfejs dla pakietów przychodzących (tzn. pakietów przechodzących przez łańcuch wejściowy) jest uważany za interfejs z którego przyszły. Odpowiednio interfejs dla pakietów wychodzących to ten, przez który wyjdą pakiety po pokonaniu łańcucha wyjściowego. Pakiety które przechodzą przez łańcuch przekazujący, trafiają również do interfejsu wyjściowego.

Jest całkowicie poprawne podać interfejs, który aktualnie nie istnieje - reguła nie będzie dotyczyła niczego dopóki interfejs fizycznie nie zostanie podniesiony (nie zacznie działać). Jest to bardzo użyteczne dla połączeń PPP do wdzwaniania się (zwykle interfejs `ppp0`) i podobnych.

Jako specjalny przypadek, interfejs kończący się znakiem '+' będzie wskazywał na wszystkie interfejsy (czy istnieją czy nie), które zaczynają się od tego ciągu znaków. Na przykład, by zdefiniować regułę która dotyczy wszystkich interfejsów PPP, można napisać '`-i ppp+`'.

Nazwa interfejsu może również być poprzedzona znakiem '`!`' by oznaczyć wszystkie interfejsy oprócz wskazanego.

Podawanie tylko pakietów TCP SYN

Czasami przydatne jest pozostawienie możliwości połączeń TCP tylko w jedną stronę. Na przykład, mógłbyś chcieć zezwalać na połączenia do zewnętrznego serwera WWW, ale nie połączenia z tego serwera.

Najprostszym podejściem byłoby zablokowanie pakietów TCP nadchodzących z tego serwera. Niestety, połączenia TCP wymagają tego by pakiety mogły krążyć w jedną i drugą stronę.

Rozwiązaniem jest blokowanie tylko pakietów z prośbą o połączenie. Nazywa się je pakietami SYN (technicznie rzecz biorąc, są to pakiety z ustawioną flagą SYN i ze zgaszonymi flagami FIN i ACK, ale nazywamy je pakietami SYN). Uniemożliwiając ruch tylko tym pakietom, możemy powstrzymać próby połączeń.

Używa się do tego parametru '`-y`' - jest prawidłowy tylko dla reguł które dotyczą protokołu TCP. Na przykład, by wskazać próby nawiązania połączenia TCP z adresu 192.168.1.1:

```
-p TCP -s 192.168.1.1 -y
```

I ponownie, parametr może być odwrócony przez użycie '`!`', który oznacza: każdy pakiet oprócz pakietów inicjujących połączenie.

Obsługa fragmentów

Czasami pakiet jest zbyt duży by mógł zmieścić się naraz w maksymalnej jednostce transmisyjnej (MTU). Gdy coś takiego ma miejsce, dzielony jest na fragmenty i wysyłany jako parę pakietów. Adresat składa fragmenty by zrekonstruować cały pakiet.

Problem z fragmentami polega na tym, że część z opcji wymienionych wyżej (a szczególnie: port źródłowy i przeznaczenia, typ ICMP, kod ICMP i flaga TCP SYN) wymagają by kernel zajrzał na początek pakietu, który znajduje się tylko w pierwszym fragmencie całości.

Jeśli twoja maszyna jest jedynym połączeniem z siecią zewnętrzną, możesz polecić kernelowi Linuksa by składał wszystkie fragmenty które przechodzą przez niego - kompilując kernel z opcją 'IP: always defragment' ustawioną na 'Y'. To pozwala obejść problem.

W innym przypadku, ważne jest by zrozumieć jak pakiety traktowane są przez reguły filtrujące. Każda z reguł, która wymaga informacji których nie mamy po prostu **nie** będzie pasować. To oznacza, że pierwszy fragment jest traktowany jak każdy inny. Drugi i następne fragmenty **nie** będą tak potraktowane. Wobec tego reguła '`-p TCP -s 192.168.1.1 www`' (podająca port źródłowy 'www') nigdy nie będzie dotyczyć fragmentu (innego niż pierwszy fragment pakietu). Tak

samo reguła odwrotna: '`-p TCP -s 192.168.1.1 ! www`'.

Możesz jednak podać reguły dotyczące drugiego i każdego następnego pakietu, używając parametru '`-f`'. Oczywiście, nie jest w tym przypadku możliwe podawanie portów TCP czy UDP, typ czy kodu ICMP lub użycie parametru dotyczącego flagi TCP SYN.

Możliwe jest również podanie, że reguła nie dotyczy drugiego i każdego następnego pakietu przez podanie jednocześnie z parametrem '`f`' parametru '`!`'.

Zwykle uważa się, że bezpieczne jest przepuszczanie kolejnych fragmentów, ponieważ pierwszy powinien zostać odfiltrowany i dzięki temu niemożliwe będzie złożenie pakietu na maszynie docelowej, jednak znane były błędy, które powodowały zawieszanie się maszyn tylko przez wysyłanie odpowiednio spreparowanych fragmentów. Ty decydujesz.

Uwaga dla sieciowców: zniekształcone pakiety (pakiety TCP, UDP i ICMP zbyt krótkie by kod ściany ogniowej ustalił port, kod lub typ ICMP) są również traktowane jak fragmenty. Tylko fragmenty pakietów TCP zaczynające się na pozycji 8 są bezwarunkowo odrzucane przez kod ściany ogniowej (w pliku syslog powinna się pojawić o tym informacja).

Jako przykład, reguła która odrzuci fragmenty kierowane do 192.168.1.1:

```
# ipchains -A output -f -d 192.168.1.1 -j DENY
#
```

Strony uboczne filtrowania

Dobra, wiemy o wszystkich sposobach pozwalających na sprawdzenie pakietu pod kątem reguły. Jeśli pakiet pasuje do niej, dzieją się następujące rzeczy:

1. Licznik dla tej reguły zliczający bajty jest powiększany o rozmiar pakietu (nagłówka i reszty).
2. Licznik pakietów dla tej reguły jest zwiększany.
3. Jeśli reguła tego wymaga, pakiet jest logowany.
4. Jeśli reguła tego wymaga, pole **Typu Usługi** (ang. *Type Of Service*) jest zmieniane.
5. Jeśli reguła tego wymaga, pakiet jest oznaczany (nie w wersji kernela 2.0.x).
6. Sprawdzany jest adres docelowy, by zdecydować co zrobić z pakietem.

Z różnych powodów omówimy je według ważności.

Wskazywanie celu

Cel pakietu mówi kernelowi co robić z nim, jeśli pasuje do reguły. ipchains używają parametru '`-j`' by wskazać cel dla danego pakietu. Nazwa celu musi być krótsza niż 8 liter, rozróżniane są również małe i duże litery ('`POWROT`' i '`powrot`' to dwa różne cele).

Najprostszym przypadkiem jest gdy nie podano celu. Taka reguła (nazywana również 'zliczającą') jest użyteczna gdy po prostu chcemy tylko liczyć pakiety danego rodzaju. Czy taka reguła odpowiada pakietowi czy nie, kernel przechodzi do następnej. Na przykład, by zliczać ilość pakietów podróżujących do adresu 192.168.1.1 możemy napisać tak:

```
# ipchains -A input -s 192.168.1.1
#
```

(i używając komendy '`ipchains -L -v`' możemy sprawdzić liczniki bajtów i pakietów odpowiadających danej regule).

Jest sześć specjalnych celów dla pakietu. Pierwsze trzy: `ACCEPT` (akceptacja), `REJECT` (odrzucenie) i `DENY` (anulowanie) są raczej zrozumiałe. `ACCEPT` (akceptacja) pozwala pakietowi przejść. `DENY` (anulowanie) odrzuca pakiet tak jakby nigdy nie został odebrany. `REJECT` (odrzucenie) odrzuca pakiet, ale (jeśli to nie jest pakiet ICMP) generuje odpowiedź ICMP do źródła pakietu, by poinformować, że adres docelowy jest nieosiągalny.

Następny - `MASQ`, mówi pakietowi by zastosować maskaradę dla pakietu. By to działało, kernel musi być skompilowany z włączoną opcją 'IP Masquerading'. Co do szczegółów proszę zajrzeć do Masquearding-HOWTO i dodatku Różnice pomiędzy ipchains i ipfwadm. Cel ten można zastosować tylko dla pakietów które przechodzą przez łańcuch `forward`.

Innym specjalnym celem jest `REDIRECT` (przekierowanie), który mówi kernelowi by wysłał pakiet do lokalnego portu zamiast tam gdzie miał się dostać. Można go zastosować tylko w przypadku protokołów TCP i UDP. Dodatkowo można podać port (nazwę lub numer) po opcji '`-j REDIRECT`' co spowoduje że pakiet zostanie przekierowany do tego portu, nawet jeśli był zaadresowany do innego. Cel ten można zastosować tylko dla pakietów które przechodzą przez łańcuch `input`.

Ostatnim celem jest `RETURN` (zwrot) którego działanie jest identyczne ze spadkiem na koniec łańcucha (sprawdź Ustawianie polityki poniżej).

Każdy inny cel wskazuje na łańcuch zdefiniowany przez użytkownika (tak jak opisano to w Operacje na całych łańcuchach poniżej). Pakiet zacznie przechodzenie przez reguły w tamtym łańcuchu. Jeśli nie zdecyduje on o losie pakietu, wróci on z powrotem i zostanie sprawdzony w aktualnym łańcuchu reguł.

Czas na więcej rysunków ASCII. Rozważ dwa (proste) łańcuchy: input (wbudowany) i Test (zdefiniowany przez użytkownika):

```
 `input'                          `Test'
-----------------------------    -----------------------------
| Rule1: -p ICMP -j REJECT  |    | Rule1: -s 192.168.1.1     |
|---------------------------|    |---------------------------|
| Rule2: -p TCP -j Test     |    | Rule2: -d 192.168.1.1     |
|---------------------------|    -----------------------------
| Rule3: -p UDP -j DENY     |
-----------------------------
```

Teraz wyobraź sobie pakiet nadchodzący z 192.168.1.1 i adresowany do 1.2.3.4. Wchodzi do łańcucha `input` i zostaje przetestowany pod kątem reguły pierwszej - Rule1 - nie zgadza się. Zgadza się natomiast reguła druga - Rule2 - i pakiet trafia do łańcucha `Test`, więc następna reguła jest sprawdzana w nim. Pierwsza reguła w tym łańcuchu się zgadza, ale nie podaje przeznaczenia, więc badana jest druga reguła - Rule2. Ta się nie zgadza, więc docieramy do końca łańcucha. Wracamy zatem do poprzedniego łańcucha w miejscu gdzie go opuściliśmy i sprawdzamy następną regułę. Nie dotyczy ona jednak pakietu, więc zostaje pominięta.

Pakiet przechodzi więc taką drogę:

```
                         v    _____________________________
 `input'                 |   /    `Test'                   v
------------------------|--/    ------------------------|----
| Rule1                 | /|    | Rule1                  |   |
|-----------------------|/-|    |------------------------|---|
| Rule2                 /  |    | Rule2                  |   |
|-------------------------|     ------------------------v----
| Rule3                 /--+_____________________________/
------------------------|---
                         v
```

Sprawdź sekcję Jak zorganizować swoje reguły by sprawdzić jak efektywnie zdefiniować swoje łańcuchy.

Logowanie pakietów

To jeden z efektów ubocznych który może mieć reguła - może logować pasujące pakiety jeśli użyjesz parametru '`-l`'. Zwykle nie będziesz tego potrzebował, ale opcja ta jest użyteczna by zapisywać wyjątkowe zdarzenia.

Kernel loguje taką informację w ten sposób:

```
Packet log: input DENY eth0 PROTO=17 192.168.2.1:53 192.168.1.1:1025
  L=34 S=0x00 I=18 F=0x0000 T=254
```

Wiadomość tą zaprojektowano by była zwięzłą i jednocześnie zawierała informacje techniczne użyteczne tylko dla sieciowych guru, ale może być również użyteczna dla nas. Podzielmy ją:

1. `input' to łańcuch, w którym znajdowała się regułę, która zalogowała pakiet.
2. `DENY' to cel dla pakietu o którym zdecydowała reguła. Jeśli jest tutaj '-' to reguła w ogóle nie dotyczyła pakietu (była regułą zliczającą).

3. `eth0' to nazwa interfejsu. Ponieważ był to łańcuch wejściowy, oznacza to, że pakiet przybył do 'eth0'.
4. `PROTO=17' oznacza pakiet protokołu numer 17. Lista numerów protokołów i ich nazw znajduje się w pliku '/etc/protocols'. Najbardziej powszechne to 1 (ICMP), 6 (TCP) i 17 (UDP).
5. `192.168.2.1' oznacza adres źródłowy pakietu IP.
6. `:53' oznacza, że pakiet został wysłany z portu 53. Sprawdzenie w pliku '/etc/services' pozwala ustalić, że jest to port 'domain' (tzn. że jest to prawdopodobnie odpowiedź DNSu). Dla UDP i TCP, jest to numer portu źródłowego. Dla ICMP, jest to typ ICMP. Dla innych, będzie to 65535.
7. `192.168.1.1' to adres docelowy IP.
8. `:1025' to adres portu docelowego (1025) - dla UDP i TCP. Dla ICMP to kod ICMP. Dla innych, będzie to 65535.
9. `L=34' oznacza że pakiet miał długość 34 bajtów.
10. `S=0x00' oznacza Typ Usługi (podziel przez 4 by otrzymać odpowiednik ToS używany przez ipchains)
11. `I=18' to identyfikator IP.
12. `F=0x0000' to 16-bitowy offset fragmentu plus flagi. Wartość zaczynająca się od '0x4' lub '0x5' oznacza, że ustawiono bit **Nie fragmentować** (ang. *Don't Fragment*). Wartość '0x2' lub '0x3' oznacza ustawiony bit **Więcej fragmentów** (ang. *More Fragments*) - należy spodziewać się większej ilości fragmentów. Reszta tej wartości to offset fragmentu, podzielony przez 8.
13. `T=254' to **czas życia** (ang. *Time To Live*) pakietu. Za każdym przekazaniem pakietu dalej odejmuje się jedną jednostkę, a zwykle ustawia się na starcie wartość 15 lub 255.
14. `(#5)' to numer reguły która spowodowała zalogowanie pakietu do pliku.

W standardowych Linuksach, wyjście kernela przechwytywane jest przez klogd (demon logujący kernela) który przekazuje je następnie do syslogd (demona logującego systemu). Zachowanie syslogd kontroluje plik '/etc/syslog.conf', przez podanie gdzie każde **urządzenie** (ang. *facility*) (w naszym wypadku urządzeniem jest 'kernel') i na jakim **poziomie** (ang. *level*) ma zachowywać dane (dla ipchains, używanym poziomem jest 'info').

Dla przykładu na moim komputerze (Debian) plik '/etc/syslog.conf' zawiera dwie linie które zawierają 'kern.info':

```
kern.*                          -/var/log/kern.log
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
```

Oznaczają one, że do plików '/var/log/kernlog' i '/var/log/messages' wpisywane są te same informacje. By uzyskać więcej informacji użyj polecenia 'man syslog.conf'.

Manipulowanie Typem Usługi (ToS)

Są to cztery, rzadko używane bity w nagłówku IP. Powodują one zmianę sposobu w jaki pakiet jest traktowany; te cztery bity to 'Minimum Delay' (Minimalna zwłoka), 'Maximum Thgroughput' (Maksymalna przepustowość), 'Maximum Reliability' (Maksymalna niezawodność) i 'Minimum Cost' (Minimalny koszt). Zezwala się na ustawienie tylko jednego z tych bitów. Rob van Nieuwkerk, autor kodu TOS pisze o tym w ten sposób:

> Szczególnie bit 'Minimalna Zwłoka' jest dla mnie ważny. Ustawiam go dla 'interaktywnych' pakietów w ruterze wyprowadzającym ruch (Linux). Używam połączenia modemowego 33,6K. Linuks prioretyzuje pakiety w 3 kolejki. W ten sposób mam akceptowalną wydajność a w międzyczasie mogę ściągać wielkie pliki (mogłoby być nawet lepiej gdyby nie było tak dużej kolejki w sterowniku seriala, ale zwłoka jest utrzymywana na poziomie 1,5 sekundy).

Nota: oczywiście, nie masz kontroli nad pakietami przychodzacymi; możesz jedynie kontrolować priorytet pakietów które opuszczają Twój komputer. By negocjować priorytety z drugim końcem połączenia, muszą być użyte protokoły takie jak RSVP (o którym nic nie wiem, więc nie pytajcie).

Najczęstszym ustawieniem jest danie połączeniom telnetowym i ftp atrybutu 'Minimalna Zwłoka' a danym ftp 'Maksymalna Przepustowość'. Może to wyglądać tak:

```
ipchains -A output -p tcp -d 0.0.0.0/0 telnet -t 0x01 0x10
ipchains -A output -p tcp -d 0.0.0.0/0 ftp -t 0x01 0x10
ipchains -A output -p tcp -s 0.0.0.0/0 ftp-data -t 0x01 0x08
```

Parametr '-t' przyjmuje dwa dodatkowe parametry, oba w zapisie heksdecymalnym. Pozwalają one na złożone

manipulowanie bitami TOS jako maski: pierwsza służy do przeprowadzenia operacji logicznej AND na polu TOS aktualnego pakietu, a druga do operacji XOR na wyniku. Jeśli to zbyt trudne, spróbuj spojrzeć na poniższą tabelę:

```
Nazwa TOS               Wartość         Typowe zastosowanie

Minimum Delay           0x01 0x10       ftp, telnet
Maximum Throughput      0x01 0x08       ftp-data
Maximum Reliability     0x01 0x04       snmp
Minimum Cost            0x01 0x02       nntp
```

Andi Kleen zwraca uwagę na następującą rzecz (trochę zedytowane dla jasności):

> Być może użyteczne byłoby dodanie odwołanie do parametru txqueuelen w ifconfig'u przy dyskusji bitów TOS. Domyślna wielkość kolejki urządzenia jest dostosowana do kart ethernet'owych, a dla modemów jest zbyt duża i sprawia że 3-pasmowy scheduler (którego kolejki bazują na TOS) pracuje nieoptymalnie. Dobrym pomysłem byłoby ustawienie go na wartość z przedziału 4-10 dla modemów lub połączeń wykorzystujących jedno pasmo B ISDN'u. Na innych urządzeniach wymagana jest dłuższa kolejka. Problem ten występuje w kernelach serii 2.0.x i 2.1.x, ale w 2.1.x jest to flaga ifconfig'a (z nowszym nettools), a w 2.0.x wymaga to spatchowania źródłą sterowników urządzeń.

Tak więc, by sprawdzić maksymalny zysk z manipulacji TOS dla połączeń PPP, wstaw linijkę '`ifconfig $1 txqueuelen`' w swoim skrypcie '`/etc/ppp/ip-up`'. Numer który użyjesz zależy od prędkości modemu i ilości buforowania w modemie. Tutaj znowu oddam Andiemu głos:

> Najlepsza wartość jest kwestią eksperymentu. Jeśli kolejka jest za krótka dla rutera, pakiety będą odrzucane. Oczywiście są korzyści bez przepisywania TOS, ale zrobienie tego zwiększa zyski dla programów opornych przy współpracy (ale wszystkie standardowe programy Linuksowe są łatwe we współpracy).

Oznaczanie pakietów

Ta właściwość pozwala na złożone i potężne interakcje z nową implementacją **jakości usługi** (ang. *Quality of Service*) Aleksieja Kuzniecowa, jak również z przekazywaniem pakietów w kernelach serii 2.1.x opartym na tej właściwości. Więcej informacji podam, gdy zobaczę to u siebie. Opcja ta jest ignorowana w kernelach serii 2.0.x.

Operacje na całym łańcuchu

Bardzo użyteczną opcją w ipchains jest możliwość grupowania reguł w jakiś sposób powiązanych. Możesz nazwać te łańcuch jak chcesz, tak długo jak nie będą one wchodziły w konflikt z wbudowanymi łańcuchami (input, output i forward) oraz ze zdefiniowanymi celami (MASQ, REDIRECT, ACCEPT, DENY, REJECT i RETURN). Sugeruję używanie dużych liter, ponieważ być może zostaną one użyte w przyszłych rozszerzeniach. Nazwa łańcucha może mieć do 8 znaków.

Tworzenie nowego łańcucha

Stwórzmy nowy łańcuch. Ponieważ jestem kolesiem z fantazją, nazwijmy go '`test`':

```
# ipchains -N test
#
```

To takie proste. Możesz teraz dodawać do niego reguły jak to opisano wyżej.

Kasowanie łańcucha

Kasowanie łańcucha również jest proste:

```
# ipchains -X test
#
```

Dlaczego '`-X`'? Cóż, wszystkie dobre litery były już zajęte.

Istnieje parę ograniczeń w kasowaniu łańcuchów: muszą być puste (sprawdź Oczyszczanie łańcucha poniżej) i nie mogą być wskazywane przez jakąś regułę. Nie możesz oczywiście również skasować żadnego z trzech wbudowanych łańcuchów.

Oczyszczanie łańcucha

Jest prosty sposób by wykasować wszystkie reguły z łańcucha, robi się to używając komendy '`-F`':

```
# ipchains -F forward
#
```

Jeśli nie podasz łańcucha, oczyszczone zostaną **wszystkie**.

Listowanie zawartości łańcucha

Możesz wylistować wszystkie reguły w łańcuchu, używając komendy '`-L`':

```
# ipchains -L input
Chain input (refcnt = 1): (policy ACCEPT)
target     prot opt    source               destination          ports
ACCEPT     icmp -----  anywhere             anywhere             any
# ipchains -L test
Chain test (refcnt = 0):
target     prot opt    source               destination          ports
DENY       icmp -----  localnet/24          anywhere             any
#
```

Parametr '`refcnt`' podany dla łańcucha 'test' to ilość reguł, które dotyczą łańcucha. Musi być on równy zero (a łańcuch musi być pusty) by można go było skasować.

Jeśli pominięto nazwę łańcucha, listowane są wszystkie łańcuchy, nawet puste.

Są trzy opcje, które można dodać do '`-L`'. Opcja '`-n`' (numerycznie) jest bardzo użyteczna ponieważ zapobiega próbie sprawdzania adresów IP przez ipchains, co (jesli używasz DNSu jak większość ludzi) spowoduje duże późnienia jeśli Twój DNS nie jest prawidłowo ustawiony, lub odfiltrowałeś zapytania DNSowe. Opcja ta powoduje również drukowanie numerów portów zamiast ich nazw.

Opcja '`-v`' pokazuje wszystkie detale reguły, takie jak liczniki bajtów i pakietów, maski TOS, interfejs i oznaczanie pakietów. W innym przypadku te dane są pomijane. Na przykład:

```
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
 pkts bytes target     prot opt   tosa tosx  ifname    mark        source
   10   840 ACCEPT     icmp ----- 0xFF 0x00  lo                    anywhere
```

Zauważ, że liczniki bajtów i pakietów drukowane są z suffiksami 'K', 'M' i 'G' - odpowiadającymi wartościom 100, 1,000,000 i 1,000,000,000. Użycie opcji '`-x`' (pokaż pełne liczby) poda również pełne reprezentacje liczb, bez względu na ich wielkość.

Resetowanie (Zerowanie) Liczników

Użyteczna jest również możliwość resetowania liczników. Można to zrobić komendą '`-Z`'. Na przykład:

```
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
 pkts bytes target     prot opt   tosa tosx  ifname    mark        source
   10   840 ACCEPT     icmp ----- 0xFF 0x00  lo                    anywhere
# ipchains -Z input
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
 pkts bytes target     prot opt   tosa tosx  ifname    mark        source
    0     0 ACCEPT     icmp ----- 0xFF 0x00  lo                    anywhere
```

```
#
```

Problem z tym polega na tym, że czasami chciałbyś znać wartości liczników tuż przed tym zanim zostaną zresetowane. W powyższym przykładzie pomiędzy wydaniem komend '`-L`' i '`-Z`' mogło przejść jeszcze trochę pakietów. Możesz zatem użyć obu komend jednocześnie by wylistować i wyzerować liczniki. Niestety, możesz to zrobić tylko dla wszystkich łańcuchów:

```
# ipchains -L -v -Z
Chain input (policy ACCEPT):
 pkts bytes target     prot opt   tosa tosx  ifname    mark        source
   10   840 ACCEPT     icmp -----  0xFF 0x00  lo                    anywhere

Chain forward (refcnt = 1): (policy ACCEPT)
Chain output (refcnt = 1): (policy ACCEPT)
Chain test (refcnt = 0):
    0     0 DENY       icmp -----  0xFF 0x00  ppp0                  localnet/24
# ipchains -L -v
Chain input (policy ACCEPT):
 pkts bytes target     prot opt   tosa tosx  ifname    mark        source
   10   840 ACCEPT     icmp -----  0xFF 0x00  lo                    anywhere

Chain forward (refcnt = 1): (policy ACCEPT)
Chain output (refcnt = 1): (policy ACCEPT)
Chain test (refcnt = 0):
    0     0 DENY       icmp -----  0xFF 0x00  ppp0                  localnet/24
#
```

Ustawianie polityki

Patrzyliśmy już na to co się dzieje gdy pakiet dojdzie do końca wbudowanego łańcucha kiedy mówiliśmy o tym jak pakiet przechodzi przez łańcuch w Celu powyżej.. W tym przypadku, polityka dla łańcucha determinuje los pakietu. Tylko wbudowane łańcuchy (input, output i forward) mogą mieć politykę, ponieważ jeśli pakiet dotrze do końca łańcucha zdefiniowanego przez użytkownika wraca do poprzedniego łańcucha.

Polityką może być każda z pierwszych czterech akcji: `ACCEPT`, `DENY`, `REJECT` lub `MASQ`. `MASQ` jest prawidłowe tylko dla łańcucha 'forward'.

Ważne jest również by pamiętać o tym, że akcja `RETURN` w regule w jednym z wbudowanych łańcuchów jest użyteczna wtedy, gdy chcemy by pakiet od razu został potraktowany zgodnie z polityką łańcucha.

Operacje na Maskaradzie

Jest kilka parametrów którymi możesz zmieniać Maskaradę IP. Są one wbudowane w `ipchains` ponieważ nieopłacalne jest pisanie specjalnego narzędzia dla nich (aczkolwiek to się zmieni).

Komendą Maskarady IP jest '`-M`' i może być łączona z '`-L`' by wypisać aktualne zmaskaradowane połączenia, lub z '`-S`' by ustawić parametry maskaradowania.

Parametr '`-L`' może być użyty z '`-n`' (pokaż numery zamiast nazw hostów i nazw portów) lub '`-v`' (pokaż delty w numerach sekwencji dla połączeń maskaradowanych, jeśli Cię to interesuje).

Parametrowi '`-S`' powinny towarzyszyć trzy parametry timeout'ów, każdy w sekundach: dla sesji TCP, dla sesji TCP po pakiecie FIN i dla pakietów UDP. Jeśli nie chcesz zmieniać którejś z wartości podaj zamiast niej wartość '`0`'.

Domyślne wartości podane są w '`/usr/src/linux/include/net/ip_masq.h`', aktualnie wynoszą one 15 minut, 2 minuty i 5 minut.

Sprawdź również (Koszmary z FTP poniżej) i informację o problemach z ustawianiem timeoutów w Nie mogę ustawić timeoutów maskarady.

Sprawdzanie pakietu

Czasami chciałbyś sprawdzić, co się dzieje z określonym pakietem gdy trafia do Twojej maszyny, na przykład gdy debugujesz łańcuchy ściany ogniowej. `ipchains` posiadają komendę '`-C`' które pozwala na to, a używa dokładnie tych samych procedur których kernel używa do operacji na prawdziwych pakietach.

Po '`-C`' należy podać nazwę łańcucha który będziesz testował, wpisując jego nazwę. Podczas gdy kernel zawsze zaczyna od łańcucha `input`, potem `output` lub `forward`, Ty możesz zacząć od dowolnego łańcucha dla celów testowych.

Detale pakietu podawane są z użyciem takiej samej składni jak reguły ściany ogniowej. W szczególności, protokół ('`-p`'), adres źródłowy ('`-s`'), adres przeznaczenia ('`-d`') i interfejs ('`-i`') są obowiązkowe. Jeśli protokołem jest TCP lub UDP, trzeba podać jeden adres źródłowy i jeden przeznazenia, a jeśli jest nim ICMP, trzeba podać kod i typ (chyba, że użyto również parametru '`-f`', który wskazuje na fragmenty - w tym momencie kod i typ są nieprawidłowe).

Jeśli protokołem jest TCP (i nie podano parametru '`-f`'), można użyć parametru '`-y`', by zaznaczyć, że obiekt powinien mieć ustawioną flagę SYN.

Poniżej przykład testowania pakietu TCP SYN z 192.168.1.1 port 60000 na 192.168.1.2 port www, przychodzącego do interfejsu eth0 i wchodzącego do łańcucha input (klasyczna inicjacja połączenia WWW):

```
# ipchains -C input -p tcp -y -i eth0 -s 192.168.1.1 60000 -d 192.168.1.2 www
packet accepted
#
```

Wiele reguł na raz i patrzymy co się dzieje

Czasami pojedyńcza komenda daje w rezultacie testowanie przez wiele reguł. Można to zrobić na dwa sposoby. Po pierwsze, podać nazwę hosta dla której DNS zwraca wiele numerów IP - `ipchains` zachowa się tak, jakbyś wpisał wiele komend dla każdego numeru IP.

Na przykład jeśli nazwa hosta 'www.foo.com' odpowiada trzem adresom IP, a nazwa hosta 'www.bar.com' dwóm, to komenda '`ipchains -A input -j reject -s www.bar.com -d www.foo.com`' doda sześć reguł do łańcucha `input`.

Innym sposobem by `ipchains` wykonał wiele operacji, jest podanie flagi **dwukierunkowości** (ang. *bidirectional*) - '`-b`'. Flaga ta powoduje, że `ipchains` zachowuje się tak jakbyś wpisał komendę dwukrotnie, za drugim razem odwracając adresy w parametrach '`-s`' i '`-d`'. W związku z tym by zapobiec przekazywaniu do lub z 192.168.1.1, możesz napisać tak jak niżej:

```
# ipchains -b -A forward -j reject -s 192.168.1.1
#
```

Mi osobiście flaga '`-b`' nie odpowiada; jeśli chcesz wygód, sprawdź Użycie ipchains-save poniżej.

Opcja '`-b`' może być użyta z opcją wstawiania ('`-I`'), kasowania ('`-D`') (ale nie w przypadku gdy pobiera numer reguły), dodawania ('`-A`') i sprawdzania ('`-C`').

Innym użytecznym parametrem jest '`-v`' który drukuje dokładnie to co `ipchains` robi z Twoimi komendami. Jest to użyteczne gdy masz do czynienia z wieloma komendami które mogą mieć wpływ na wiele reguł. Na przykład, sprawdzamy zachowanie fragmentów pomiędzy 192.168.1.1 i 192.168.1.2:

```
# ipchains -v -b -C input -p tcp -f -s 192.168.1.1 -d 192.168.1.2 -i lo
  tcp opt   ---f- tos 0xFF 0x00  via lo    192.168.1.1  -> 192.168.1.2    * ->   *
packet accepted
  tcp opt   ---f- tos 0xFF 0x00  via lo    192.168.1.2  -> 192.168.1.1    * ->   *
packet accepted
#
```

4.2 Użyteczne przykłady

Mam połączenie dialup PPP ('`-i ppp0`'). Pobieram newsy ('`-p TCP -s news.virtual.net.au nntp`') i pocztę ('`-p TCP -s mail.virtual.net.au pop-3`') za każdym razem gdy się wdzwonię. Używam serwera FTP Debiana by

uaktualniać moją maszynę ('`-p TCP -y -s ftp.debian.org.au ftp-data`'). Przeglądam strony WWW przez proxy mojego dostawcy Internet'owego (ISP) ('`-p TCP -d proxy.virtual.net.au 8080`'), ale nienawidzę bannerów i reklam z doubleclick.net na archiwach Dilbert'a ('`-p TCP -y -d 199.95.207.0/24 i -p TCP -y -d 199.95.208.0/24`').

Nie mam nic przeciwko ludziom próbującym ftpować się na moją maszynę gdy jestem podłączony ('`-p TCP -d $LOCALIP ftp`'), ale nie chcę by ktokolwiek z zewnątrz udawał, że ma adres IP mojej sieci wewnętrznej ('`-s 192.168.1.0/24`'). Nazywane jest to zwykle **fałszowaniem** (ang. *spoofing*) IP i jest lepszy sposób na ochronienie się przed tym w kernelach serii 2.1.x i wyższych (sprawdź Jak ustawić ochronę przed fałszowaniem pakietów?).

Konfiguracja jest raczej prosta, ponieważ nie mam w mojej sieci innych maszyn.

Nie chcę aby żaden lokalny proces (tzn. Netscape, Lynx i inne) próbowały połączyć się z adresem doubleclick.net:

```
# ipchains -A output -d 199.95.207.0/24 -j REJECT
# ipchains -A output -d 199.95.208.0/24 -j REJECT
#
```

Teraz, chciałbym ustawić priorytetyzację dla wychodzących pakietów (ale nie ma po co robić tego dla przychodzących). Ponieważ reguł kontrolujących jest już raczej dużo, wsadzę je do jednego łańcucha który nazwę '`ppp-out`':

```
# ipchains -N ppp-out
# ipchains -A output -i ppp0 -j ppp-out
#
```

Minimalna zwłoka dla ruchu WWW i telnetu:

```
# ipchains -A ppp-out -p TCP -d proxy.virtual.net.au 8080 -t 0x01 0x10
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 telnet -t 0x01 0x10
#
```

Niski koszt dla ftp-data, nntp i pop-3:

```
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 ftp-data -t 0x01 0x02
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 nntp -t 0x01 0x02
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 pop-3 -t 0x01 0x02
#
```

Jest również parę restrykcji dla pakietów przychodzących do interfejsu `ppp0`: stwórzmy nowy łańcuch i nazwijmy go '`ppp-in`':

```
# ipchains -N ppp-in
# ipchains -A input -i ppp0 -j ppp-in
#
```

Żaden pakiet przychodzący do `ppp0` nie powinien podawać, że posiada adres źródłowy 192.168.*, więc odrzucamy je i logujemy:

```
# ipchains -A ppp-in -s 192.168.1.0/24 -l -j DENY
#
```

Pozwalam na ruch pakietów UDP dla działania DNSu (na mojej maszynie działa DNS przekazujący, który kontaktuje się z maszyną 209.29.16.1, więc spodziewam się odpowiedzi tylko od niego), ruch przychodzący ftp i ruch powrotny ftp-data (który powinien być kierowany na porty powyżej 1023 i nie w okolicy portów X11):

```
# ipchains -A ppp-in -p UDP -s 203.29.16.1 -d $LOCALIP dns -j ACCEPT
# ipchains -A ppp-in -p TCP -s 0.0.0.0/0 ftp-data -d $LOCALIP 1024:5999 -j ACCEPT
# ipchains -A ppp-in -p TCP -s 0.0.0.0/0 ftp-data -d $LOCALIP 6010: -j ACCEPT
# ipchains -A ppp-in -p TCP -d $LOCALIP ftp -j ACCEPT
#
```

Pozwalam również na wracanie odpowiedzi TCP

```
# ipchains -A ppp-in -p TCP ! -y -j ACCEPT
#
```

Na koniec, pakiety do mojej własnej maszyny z niej samej są OK:

```
# ipchains -A input -i lo -j ACCEPT
#
```

A teraz, ustawiam swoją domyślną zasadę na `DENY`, więc wszystko inne jest odrzucane:

```
# ipchains -P input DENY
#
```

UWAGA: Nie ustawiałbym swoich reguł w tej kolejności, ponieważ pakiety mogą dostać się do mojej maszyny w trakcie jej konfigurowania. Najbezpieczniejsze jest ustawienie domyślnej zasady na `DENY` najpierw, potem wstawienie reguł. Oczywiście, jeśli Twoje reguły wymagają sprawdzeń DNSowych możesz mieć problem.

Użycie ipchains-save

Ustawienie łańcuchów ściany ogniowej dokładnie tak jak chcesz je mieć ustawione, a potem jeszcze próba spamiętania poszczególnych komend i ich kolejności tak, byś mógł je wydać ponownie po restarcie, jest raczej bolesne.

Istnieje skrypt `ipchains-save`, który zczytuje aktualne ustawienia i zapisuje je do pliku. Na razie pozostawię Cię z odrobinę niepewności, co do działania skryptu `ipchains-restore`.

`ipchains-save` zapisuje pojedyńczy łańcuch, lub wszystkie łańcuchy (jeśli nie podano nazwy łańcucha). Jedyną opcją obecnie obsługiwaną jest '`-v`', która drukuje wszystkie reguły (do stderr) w momencie gdy są zapisywane. Polityka dla łańcucha jest również zapisywana dla łańcuchów `input`, `output` i `forward`.

```
# ipchains-save > my_firewall
Saving `input'.
Saving `output'.
Saving `forward'.
Saving `ppp-in'.
Saving `ppp-out'.
#
```

Użycie ipchains-restore

`ipchains-restore` odbudowywuje łańcuchy zapisane przez `ipchains-save`. Można go wywołać z dwoma opcjami: '`-v`' który powoduje opisanie każdej reguły w trakcie jej dodawania i '`-f`' który wymusza wyczyszczenie reguł które już są skonfigurowane.

Gdy znaleziony jest zdefiniowany przez użytkownika łańcuch w pliku który czyta `ipchains-restore`, sprawdzane jest czy taki już istnieje. Jeśli tak, zostaniesz spytany czy ten istniejący powinien zostać oczyszczony (usunięte wszystkie reguły) czy odzyskiwanie reguł powinno zostać ominięte. Jeśli podałeś '`-f`', nie zostaniesz spytany, a łańcuch zostanie oczyszczony.

Na przykład:

```
# ipchains-restore < my_firewall
Restoring `input'.
Restoring `output'.
Restoring `forward'.
Restoring `ppp-in'.
Chain `ppp-in' already exists. Skip or flush? [S/f]? s
Skipping `ppp-in'.
Restoring `ppp-out'.
Chain `ppp-out' already exists. Skip or flush? [S/f]? f
Flushing `ppp-out'.
#
```

5. Różne

Ta sekcja zawiera informacje i FAQ których nie mogłem zmieścić w strukturze powyżej.

5.1 Jak zorganizować reguły swojej ściany ogniowej

To pytanie wymaga przemyślenia. Możesz spróbować zorganizować je pod kątem prędkości (zminimalizować liczbę sprawdzeń reguł dla najczęściej spotykanych pakietów) lub pod kątem zarządzania.

Jeśli nie masz stałego łącza, powiedzmy łącze PPP, możesz chcieć by pierwsza reguła w łańcuchu `input` brzmiała '`-i ppp0 -j DENY`' w trakcie procesu uruchamiania, a potem coś podobnego w skrypcie podnoszącym interfejs - `ip-up`:

```
# Re-create the `ppp-in' chain.
ipchains-restore -f < ppp-in.firewall

# Replace DENY rule with jump to ppp-handling chain.
ipchains -R input 1 -i ppp0 -j ppp-in
```

Twój skrypt `ip-down` mógłby wyglądać tak:

```
ipchains -R input 1 -i ppp0 -j DENY
```

5.2 Czego nie filtrować

Jest parę rzeczy z których powinieneś zdawać sobie sprawę zanim zaczniesz filtrować wszystko czego nie chcesz.

Pakiety ICMP

Pakiety ICMP są używane (między innymi) do wskazywana na błędy innych protokołów (takich jak TCP czy UDP). Zwykle dotyczy to pakietów '`destination-unreachable`'. Zablokowanie takich pakietów oznacza że nie otrzymasz nigdy komunikatu zwrotnego '`Host unreachable`' lub '`No route to host`'; wszystkie połączenia będą po prostu czekały na odpowiedź która nigdy nie nadejdzie. Jest to trochę irytujące, ale rzadko fatalne.

Większym problemem jest rola pakietów ICMP w rozpoznawaniu MTU. Wszystkie dobre implementacje TCP (w tym Linuksa) używają rozpoznawania MTU by sprawdzić jaki największy pakiet da się wysłać pod dany adres unikając fragmentacji (która zmniejsza wydajność, zwłaszcza gdy zdarzy się że jakiś fragment zginął w ogóle). Rozpoznawanie MTU działa w ten sposób: wysyła pakiety z ustawionym bitem '`Don't Fragment`', a potem coraz mniejsze jeśli dostanie odpowiedź ICMP '`fragmentation-needed`'. Jest to pakiet z rodzaju '`destination-unreachable`' i jeśli taki komunikat nie zostanie odebrany, lokalna maszyna nie zmniejszy MTU a wydajność takiego łącza będzie koszmarna, lub w ogóle jej nie będzie.

Zwróć jednak uwagę, że powszechne jest blokowanie komunikatów ICMP o przekierowaniu (typ 5); mogą one być używane do manipulowania rutingiem (aczkolwiek dobre stosy IP mają zabezpieczenia), są więc odbierane jako raczej ryzykowne.

Połączenia TCP do serwerów DNS

Jeśli starasz się zablokować wychodzące pakiety TCP, pamiętaj o tym że DNS nie zawsze używa UDP; jeśli odpowiedź z serwera jest dłuższa niż 512 bajtów, klient używa połączenia TCP (do portu 53) by uzyskać dane.

Może to być pułapka, ponieważ DNS będzie nadal 'prawie zawsze' działał jeśli zabronisz takiego ruchu; możesz jednak napotkać dziwnie długie odpowiedzi lub inne problemy z DNSem.

Jeśli Twoje zapytania DNSowe są zawsze kierowane do tej samej zewnętrznej maszyny (albo bezpośrednio przez użycie serwera nazw w pliku '`/etc/resolv.conf`' lub przez użycie cache'u serwera nazw w trybie przekazywania), będziesz potrzebował tylko pozwolenia na połączenia TCP do tego serwera na port 'domain' z lokalnego portu 'domain' (jeśli używasz serwera nazw cache'ującego) lub na wyższy port (>1023) jeśli używasz '`/etc/resolv.conf`'.

Koszmary z FTP

Klasycznym problemem filtrowania pakietów jest FTP. FTP ma dwa tryby pracy: tradycyjny nazywany **trybem aktywnym** (ang. *active mode*) i trochę nowszy zwany **trybem pasywnym** (ang. *passive mode*). Przeglądarki WWW używają zwykle trybu pasywanego, ale programy FTP z kolei zwykle aktywnego.

W trybie aktywnym, kiedy zdalny koniec chce wysłać plik (lub nawet jeśli chce zwrócić rezultat poleceń '`ls`' czy '`dir`') próbuje otworzyć połączenie TCP do maszyny lokalnej. To oznacza, że nie możesz odfiltrować połączeń TCP bez wyłączania aktywnego FTP.

Jeśli masz możliwość używania trybu pasywnego, jest lepiej - dane w trybie pasywnym przesyłane są połączeniami klient-serwer, nawet jeśli chodzi o dane przychodzące od serwera. W innym przypadku, zalecane jest na zezwolenie na połączenia TCP tylko na portach powyżej 1024 i nie pomiędzy 6000 a 6010 (6000 używany jest przez X-Windows).

5.3 Odfiltrowanie Ping of Death

Komputery z Linuksem są aktualnie podatne na znany Ping of Death, który polega na wysłaniu niepoprawnie dużego pakietu ICMP, powodującego przepełnienie buforów i stosu TCP komputera, który go otrzyma, a w rezultacie poważne problemy.

Jeśli zajmujesz się ochroną komputerów, które mogą być narażone, możesz po prostu zablokować fragmenty ICMP. Normalne pakiety ICMP nie są na tyle duże by wymagać fragmentacji, więc nie zepsujesz niczego oprócz wielkich ping'ów. Słyszałem (niepotwierdzone) o tym, że niektóre systemy potrzebowały tylko ostatniego fragmentu wielkiego pakietu ICMP by zaburzyć ich działanie, więc blokowanie tylko pierwszego fragmentu jest raczej bezcelowe.

Podczas gdy exploity które widziałem wszystkie wykorzystują ICMP, nie ma powodów by nie użyć fragmentów pakietów TCP czy UDP (lub nieznanego protokołu) do takiego ataku, więc zablokowanie fragmentów ICMP to tylko rozwiązanie tymczasowe.

5.4 Odfiltrowanie Teardrop i Bonk

Teardrop i Bonk to dwa ataki (głównie przeciwko komputerom z Microsoft Windows NT), które polegają na nakładających się fragmentach. Rozwiązaniem jest posiadanie kompuera z Linuksem, który zajmuje się defragmentacją, lub zabronienie przechodzenia fragmentów do narażonych maszyn.

5.5 Odfiltrowanie Bomb Fragmentowych

Niektóre mniej stabilne implementacje stosu TCP mają problemy z obsługą dużej ilości fragmentów pakietów, jeśli nie otrzymają wszystkich fragmentów. Linux nie ma tego problemu. Możesz odfiltrowywać fragmenty (co może jednak spowodować problemy dla normalnych użytkowników), lub skompilować swój kernel z opcją 'IP: always defragment' ustawioną na '`Y`' (tylko jeśli twój Linux jest jedyną możliwą drogą dla tych pakietów).

5.6 Zmiana reguł ściany ogniowej

Są pewne czasowe zależności związane ze zmianą reguł ściany ogniowej. Jeśli nie jesteś ostrożny, możesz dać pakietom przejść w połowie twoich zmian. Najprostszym rozwiązaniem jest zrobienie czegoś takiego:

```
# ipchains -I input 1 -j DENY
# ipchains -I output 1 -j DENY
# ipchains -I forward 1 -j DENY

... wykonanie zmian ...

# ipchains -D input 1
# ipchains -D output 1
# ipchains -D forward 1
#
```

Odrzuca to wszystkie pakiety na czas zmian.

Jeśli twoje zmiany ograniczają się do jednego łańcucha, możesz chcieć stworzyć nowy łańcuch z nowymi regułami, a potem zastąpić ('-R') regułę, która wskazywała na stary łańcuch nową, wskazującą na nowy łańcuch, a potem skasować stary. To zastąpienie nastąpi niezauważalnie i nie będzie miało skutków ubocznych.

5.7 Jak skonfigurować ochronę przez fałszowaniem IP?

Fałszowanie IP to technika, w której host wysyła pakiety opisane jako pochodzące z innego hosta. Ponieważ filtrowanie pakietów podejmuje decyzje na podstawie adresu źródłowego, fałszowanie IP może być użyte do ogłupienia go. Jest również używane do ukrycia prawdziwego adresu osoby używającej ataków SYN, Teardrop, Ping of Death i podobnych (nie martw się jeśli nie wiesz na czym polegają).

Najlepszą ochroną przed tym jest **Weryfikacja Adresu Źródłowego** (ang. *Source Address Verification*) i jest dokonywana przez kod rutujący, a nie przez kod ściany ogniowej. Spójrz do pliku '/proc/sys/net/ipv4/conf/all/rp_filter'. Jeśli istnieje, to włączenie tej weryfikacji podczas każdego boot-u jest prawidłowym rozwiązaniem. By to zrobić, wstaw poniższe linie gdzieś do skryptów startowych, zanim jakikolwiek interfejs sieciowy zostanie zainicjowany:

```
# This is the best method: turn on Source Address Verification and get
# spoof protection on all current and future interfaces.
if [ -e /proc/sys/net/ipv4/conf/all/rp_filter ]; then
  echo -n "Setting up IP spoofing protection..."
  for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
      echo 1 > $f
  done
  echo "done."
else
  echo PROBLEMS SETTING UP IP SPOOFING PROTECTION.  BE WORRIED.
  echo "CONTROL-D will exit from this shell and continue system startup."
  echo
  # Start a single user shell on the console
  /sbin/sulogin $CONSOLE
fi
```

Jeśli nie możesz tego zrobić, możesz manualnie wstawić reguły dla ochrony każdego interfejsu. To wymaga niestety znajomości każdego interfejsu. W kernelach serii 2.1.x automatycznie odrzucane są pakiety nadchodzące z adresów 127.* (zarezerwowanych dla pętli zwrotnych - lo).

Na przykład, powiedzmy że mamy trzy interfejsy - eth0, eth1 i ppp0. Używamy ifconfig by sprawdzić jakie adresy i maski sieciowe mają interfejsy. Powiedzmy że eth0 jest podłączony do sieci 192.168.1.0 z maską 255.255.255.0, eth1 jest podłączony do sieci 10.0.0.0 z maską 255.0.0.0 a ppp0 jest podłączony do Internetu (w którym dowolny adres, poza prywatnymi IP, jest dozwolony) i używamy takich reguł:

```
# ipchains -A input -i eth0 -s ! 192.168.1.0/255.255.255.0 -j DENY
# ipchains -A input -i ! eth0 -s 192.168.1.0/255.255.255.0 -j DENY
# ipchains -A input -i eth1 -s ! 10.0.0.0/255.0.0.0 -j DENY
# ipchains -A input -i ! eth1 -s 10.0.0.0/255.0.0.0 -j DENY
#
```

To podejście nie jest tak dobre jak Weryfikacja adresu źródłowego, ponieważ jeśli twoja sieć zmieni się, będziesz musiał zmieniać również reguły ściany ogniowej.

Jeśli używasz kernela serii 2.0.x, możesz chcieć zabezpieczyć interfejs lo, używając reguły takiej jak ta:

```
# ipchains -A input -i ! lo -s 127.0.0.0/255.0.0.0 -j DENY
#
```

5.8 Zaawansowane projekty

Napisałem bibliotekę zawartą w dystrybucji źródłowej ipchains o nazwie 'libfw'. Używa ona możliwości łańcuchów IP w wersjach 1.3 i wyższych, by kopiować pakiety do **przestrzeni użytkownika** (ang. *userspace*) - aby to było możliwe należy dodać w konfiguracji jądra opcję 'IP_FIREWALL_NETLINK'.

Pole przeznaczone na oznaczenie pakietu, może być użyte do podawania parametru **jakości usługi** (ang. *Quality of Service*) dla pakietu, lub by podać jak pakiet ma zostać przekazany do zadanego portu. Nie używałem żadnej z tych opcji, ale jeśli ktoś chciałby o tym napisać, proszę skontaktować się ze mną.

Takie zagadnienia jak **filtrowanie ze sprawdzaniem stanów** (ang. *stateful inspection firewalling*) (preferuję termin **dynamiczna ściana ogniowa**) mogą zostać zaimplementowanie w przestrzeni użytkownika jako biblioteka. Innymi fajnymi pomysłami może być kontrola pakietów na poziomie użytkowników przez demona działającego w przestrzeni użytkownika. Powinno to być raczej łatwe.

Filtrowanie ze sprawdzaniem stanów (ang.*Stateful Packet Filtering*)

Pod adresem ftp://ftp.interlinx.bc.ca/pub/spf znajduje się strona Brian'a Murrell'a z projektem SPF. W ramach niego wykonuje się śledzenie połączeń w przestrzeni użytkownika. Dodaje to warstwę bezpieczeństwa dla serwerów z łączem o małej przepustowości.

Aktualnie dostępna jest tylko szczątkowa dokumentacja, ale cytuję tutaj post, na który Brian odpowiedział i wyjaśnił pewne pytania:

```
> Jak podejrzewam, działa to dokładnie tak jakbym chciał:
> zainstalowanie tymczasowej 'wstecznej' reguły by dać
> pakietom wejść jako odpowiedź na zapytanie wychodzące

Tak, dokładnie to robi. Im więcej protokołów rozumie, tym bardziej
'wsteczne' reguły jest w stanie poprawnie obsłużyć. Aktualnie jest w
stanie obsłużyć (z pamięci, proszę wybaczyć za błędy lub pominięcia)
FTP (aktywne i pasywne, wejście i wyjście), część RealAudio, traceroute,
ICMP i podstawowy ICQ (wejście z serwera ICQ i bezpośrednie połączenia
TCP, ale inne sprawy takie jak dodatkowe połączenia TCP dla przesyłania
plików itp. jeszcze nie).

> Czy to zamiennik dla ipchains czy suplement?

Suplement. Myśl o ipchains jako silniku który umożliwia i zapobiega
podróże pakietów przez system z Linuksem. SPF to sterownik, ciągle monitorujący ruch
i mówiący ipchains jak zarządzać zasadami by odzwierciedlać zmiany we wzorcach
ruchu.
```

Patch na ftp-data Michaela Hasenstein'a

Michael Hasanstein z SuSE napisał patch do kernela, który dodaje obsługę śledzenia połączeń ftp dla ipchains. Aktualnie patch można znaleźć pod adresem http://www.suse.de/~mha/patch.ftp-data-2.gz.

5.9 Przyszłe rozszerzenia

Ściany ogniowe i NAT są w trakcie przeprojektowywania dla wersji kerneli 2.4.x. Plany i dyskusje są dostępne w archiwum netfilter (zajrzyj pod http://lists.samba.org). Rozszerzenia te powinny oczyścić wiele wspaniałych zagadnień (ściany ogniowe i maskarada nie powinny być takie trudne) oraz pozwolić na rozwój o wiele elastyczniejszej ściany ogniowej.

6. Najczęstsze problemy

6.1 ipchains -L zawiesza się!

Prawdopodobnie blokujesz zapytania DNS; być może dostaniesz time-out i program ruszy dalej. Spróbuj używać opcji '`-n`', która zapobiega sprawdzaniu nazw.

6.2 Inwersja nie działa!

Musisz otoczyć znak '`!`' spacjami z obu stron. Klasycznym **błędem** jest (o czym ostrzegam w 1.3.10):

```
# ipchains -A input -i !eth0 -j DENY
#
```

Nie będzie nigdy interfejsu '`!eth0`', ale `ipchains` o tym nie wie.

6.3 Maskarada/Przekazywanie nie działa!

Upewnij się, że przekazywanie pakietów jest włączone (w nowych kernelach jest domyślnie **wyłączone**, tzn. pakiety nigdy nie przejdą do łańcucha '`forward`'). Możesz je ustawić (jako root) pisząc:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
#
```

Jeśli to działa, wstaw tą linię gdzieś w skrypcie bootującym żeby było włączone zawsze. Dobrze jest jednak ustawić najpierw reguły ściany ogniowej zanim uruchomisz tą komendę, żeby zapobiec przedostaniu się jakichś pakietów.

6.4 -j REDIR nie działa!

Musisz umożliwić przekazywanie pakietów (punkt wyżej) by działało przekierowanie; w innym przypadku kod rutujący będzie odrzucał pakiety. Jeśli używasz tylko przekierowania i w ogóle nie masz ustawionego przekazywania, powinieneś zwrócić na ten mechanizm uwagę.

Zauważ że REDIR (który jest w łańcuchu wejściowym) nie ma wpływu na połączenia z lokalnych procesów.

6.5 Maski w interfejsach nie działają!

W kernelach wersji 2.1.102 i 2.1.103 był błąd (i w jakiś starych patch'ach które wyprodukowałem), który powodował że interfejs wskazany ipchains przez maskę (tak jak np. '`-i ppp+`') był błędny.

Zostało to poprawione w nowszych kernelach, a w 2.0.34 przez patch na stronach WWW. Możesz to również poprawić odręcznie, zmieniając w źródłach kernela w pliku '`include/linux/ip_fw.h`' linię 63 z takiej postaci:

```
#define IP_FW_F_MASK    0x002F  /* All possible flag bits mask   */
```

Powinna zawierać '`0x003F`'. Popraw to i przekompiluj kernel.

6.6 TOS nie działa!

To był mój błąd: ustawienie pola TOS tak naprawdę nie zmieniało nic w kernelach wersji 2.1.102 do 2.1.111. Zostało to poprawione w 2.1.112 i późniejszych.

6.7 ipautofw i ipportfw nie działają!

Dla kerneli serii 2.0.x to prawda - nie mam czasu na tworzenie i utrzymywanie wielkich patch'y dla `ipchains` i `ipautofw/ipportfw`. Dla wersji 2.1.x ściągnij `ipmasqadm` od Juan Ciarlante: http://juanjox.linuxhq.com/ i używaj tego narzędzia dokładnie tak jak użyłbyś `ipautofw` czy `ipportfw`, tylko zamiast pisać `ipportfw` piszesz `ipmasqadm portfw`, a zamiast `ipautofw` piszesz `ipmasqadm autofw`.

6.8 xosview nie działa!

Wersja 1.6.0 lub wyższa, nie potrzebuje reguł ściany ogniowej dla wszystkich kerneli 2.1.x. Niestety, znowu nie działa w wersji 1.6.1 - proszę pomęczyć autora (to nie moja wina!).

6.9 Segmentation Fault przy `-j REDIRECT'!

To był błąd w ipchains w wersji 1.3.3. Proszę uaktualnić wersję.

6.10 Nie mogę ustawić timeout'ów Maskarady!

Faktycznie tak jest dla kerneli wersji 2.1.x do 2.1.123. W 2.1.124 próba ustawienia timeotów Maskarady kończy się zablokowaniem kernela (zmień `return` na `ret =` w lini 1328 pliku '`net/ipv4/ip_fw.c`'). W 2.1.125 działa już dobrze.

6.11 Chcę obsługiwać na ścianie ogniowej IPX!

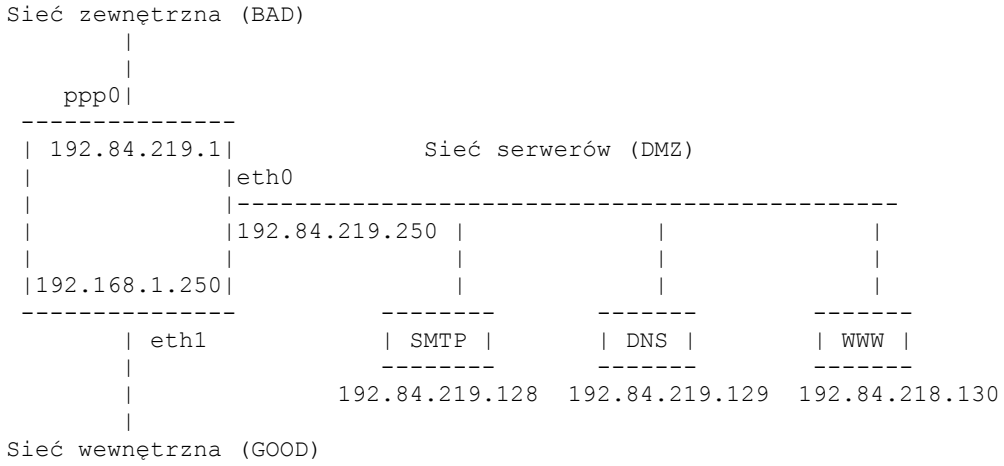
Tak samo jak inni. Niestety, mój kod zajmuje się tylko IP, . Z drugiej strony, przygotowano wszystko by to robić! Musisz tylko napisać kod. Będę szczęśliwy móc pomóc gdzie to możliwe.

7. Całkiem poważny przykład.

Ten przykład pochodzi z mojego i Michael'a Neulinga tutoriala z LinuxWorld z marca 1999. To nie jedyne rozwiązanie problemu, ale prawdopodobnie najprostsze. Mam nadzieję, że będzie dla ciebie cenne.

7.1 Sytuacja

- sieć wewnętrzna zmaskaradowana (różne systemy operacyjne), będziemy je nazywać 'Dobrymi' (`GOOD`);
- narażone serwery w oddzielnej sieci (które będziemy nazywać 'strefą DMZ' (zdemilitaryzowaną))
- połączenie PPP do Internetu (nazwane 'Złe' - `BAD`)



7.2 Nasze cele

Komputer filtrujący pakiety:

PING do każdej sieci

użyteczne by stwierdzić, czy maszyna nie działa

TRACEROUTE do każdej sieci

ponownie, do celół diagnostycznych

Dostęp do DNS

aby ping i DNS były użyteczne

W strefie DMZ:

Serwer pocztowy:

- SMTP na zewnątrz
- akceptowanie połączeń SMTP z zewnętrz i wewnątrz
- akceptowanie POP-3 z wewnątrz

Serwer nazw:

- wysyłanie DNSu na zewnątrz
- akceptowanie DNSu z wewnątrz, wewnątrz i z komputera filtrującego pakiety

Serwer WWW:

- akceptowanie HTTP z zewnątrz i wewnątrz
- dostęp rsync z wewnątrz

Sieć wewnętrzna:

Zezwalamy na WWW, ftp, traceroute, ssh do komputerów zewnętrznych

To raczej normalne rzeczy do "puszczenia" - można zacząć od umożliwienia maszynom wewnętrznym na dowolny ruch, ale tutaj jesteśmy bardzo restrykcyjni.

Zezwalamy na ruch SMTP do serwera pocztowego

Oczywiście, chcemy by można było wysyłać pocztę.

Zezwalamy na ruch POP-3 do serwera pocztowego

W ten sposób będzie można czytać pocztę.

Zezwalamy na ruch DNS do serwera nazw

Musi być możliwość używania nazw maszyn zewnętrznych dla WWW, ftp, traceroute i ssh.

Zezwalamy na ruch rsync do serwera WWW

Tak będziemy synchronizować serwer zewnętrzny z wewnętrznym.

Zezwalamy na ruch WWW do serwera WWW

Oczywiście powinna być możliwość połączenia się do naszego zewnętrznego serwera.

Zezwalamy na pingowanie komputera filtrującego pakiety

W ten sposób mogą pingować serwer i sprawdzać, czy naprawdę nie działa (a nie winić nas za problemy z jakąś maszyną w Internecie).

7.3 Przed filtrowaniem pakietów

- Ochrona przed fałszowaniem (ang.*anti-spoofing*)

  Ponieważ nie mamy asymetrycznego rutingu, możemy po prostu włączyć ochronę przed fałszowaniem dla wszystkich interfejsów:

```
# for f in /proc/sys/net/ipv4/conf/*/rp_filter; do echo 1 > $f; done
#
```

- Ustawienie zasad filtrujących na `DENY`

  Nadal jednak pozwalamy na ruch po interfejsie `lo`, ale zabraniamy wszystkiego innego:

  ```
  # ipchains -A input -i ! lo -j DENY
  # ipchains -A output -i ! lo -j DENY
  # ipchains -A forward -j DENY
  #
  ```

- Ustawienie interfejsów

  Ma to zwykle miejsce w skryptach startowych. Upewnij się, że powyższe punkty zostały wykonane przed ustawieniem interfejsów, by zapobiec przeciekowi pakietów zanim ustawisz reguły.

- Wstawienie modułów maskaradujących do poszczególnych protokołów

  Musimy dołączyć moduł FTP do naszej maskarady, by aktywne i pasywne sesje FTP działały z sieci wewnętrznej.

  ```
  # insmod ip_masq_ftp
  #
  ```

7.4 Filtrowanie pakietów dla pakietów które mogą będą tylko przechodzić

Jeśli chodzi o maskaradę, najlepiej jest filtrować w łańcuchu `forward`.

Podziel go na różne zestawy własnych łańcuchów, w zależności od adresów źródła/przeznaczenia interfejsów; pozwala to na podzielenie problemu na łatwiej zarządzalne kawałki.

```
ipchains -N good-dmz
ipchains -N bad-dmz
ipchains -N good-bad
ipchains -N dmz-good
ipchains -N dmz-bad
ipchains -N bad-good
```

Pozwalamy standardowym pakietom ICMP informującym o błędach na przejście, więc tworzymy dla nich oddzielny łańcuch:

```
ipchains -N icmp-acc
```

Ustawienie celów z łańcucha `forward`

Niestety, znamy w nim tylko interfejs wychodzący. Wobec tego, aby sprawdzić z którego interfejsu pakiet przyszedł, musimy sprawdzić adres źródłowy (ochrona przed fałszowaniem zapobiega podszywaniu się pod adresy).

Zauważ że logujemy wszystko co nie pasuje do tych reguł (oczywiście, nigdy nie powinno się zdarzyć).

```
ipchains -A forward -s 192.168.1.0/24 -i eth0 -j good-dmz
ipchains -A forward -s 192.168.1.0/24 -i ppp0 -j good-bad
ipchains -A forward -s 192.84.219.0/24 -i ppp0 -j dmz-bad
ipchains -A forward -s 192.84.219.0/24 -i eth1 -j dmz-good
ipchains -A forward -i eth0 -j bad-dmz
ipchains -A forward -i eth1 -j bad-good
ipchains -A forward -j DENY -l
```

Definiujemy łańcuch icmp-acc

Pakiety będące pakietami ICMP zawiadamiającymi o błędzie są akceptowane, a w innym wypadku kontrola jest

zwracana do łańcucha wywołującego.

```
ipchains -A icmp-acc -p icmp --icmp-type destination-unreachable -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type source-quench -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type time-exceeded -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type parameter-problem -j ACCEPT
```

Dobre (sieć wewnętrzna) do DMZ (serwery)

Restrykcje wewnętrze :

- Zezwalamy na WWW, ftp, traceroute, ssh to komputerów wewnętrznych
- Zezwalamy na SMTP do serwera poczty
- Zezwalamy na POP3 do serwera poczty
- Zezwalamy na DNS do serwera nazw
- Zezwalamy na rsync do serwera WWW
- Zezwalamy na WWW do serwera WWW
- Zezwalamy na ping do komputera filtrującego pakiety

Moglibyśmy zrobić maskaradę z sieci wewnętrznej do strefy DMZ, ale tutaj tego nie robimy. Ponieważ nikt z sieci wewnętrznej nie powinien próbować niczego złego, logujemy pakiety które są odrzucane/anulowane.

Zauważ, że stare wersje Debiana nazywały usługę '`pop3`' nazwą '`pop-3`' (w pliku '`/etc/services`') co jest niezgodne z RFC1700.

```
ipchains -A good-dmz -p tcp -d 192.84.219.128 smtp -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.219.128 pop3 -j ACCEPT
ipchains -A good-dmz -p udp -d 192.84.219.129 domain -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.219.129 domain -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.218.130 www -j ACCEPT
ipchains -A good-dmz -p tcp -d 192.84.218.130 rsync -j ACCEPT
ipchains -A good-dmz -p icmp -j icmp-acc
ipchains -A good-dmz -j DENY -l
```

Zła (sieć zewnętrzna) to DMZ (serwery).

- Restrykcje strefy DMZ:
  - Serwer poczty:
    - Serwer SMTP dla komputerów zewnętrznych
    - Zezwalamy na SMTP z wewnątrz i zewnątrz
    - Zezwalamy na POP3 z wewnątrz
  - Serwer nazw:
    - Wysyłamy DNS do komputerów zewnętrznych
    - Zezwalamy na DNS z sieci wewnętrznej, zewnętrznej i komputera filtrującego pakiety
  - Serwer WWW:
    - Zezwalamy na HTTP z sieci wewnętrznej i zewnętrznej
    - Zezwalamy na rsync z sieci wewnętrznej
- Rzeczy na które zezwalamy z sieci zewnętrznej do strefy DMZ:
  - Nie logujemy naruszeń reguł, ponieważ mogą się zdarzyć

```
ipchains -A bad-dmz -p tcp -d 192.84.219.128 smtp -j ACCEPT
ipchains -A bad-dmz -p udp -d 192.84.219.129 domain -j ACCEPT
ipchains -A bad-dmz -p tcp -d 192.84.219.129 domain -j ACCEPT
ipchains -A bad-dmz -p tcp -d 192.84.218.130 www -j ACCEPT
ipchains -A bad-dmz -p icmp -j icmp-acc
ipchains -A bad-dmz -j DENY
```

Dobra (sieć wewnętrzna) do Złej (zewnętrznej).

- Wewnętrzne restrykcje:
  - Zezwalamy na WWW, ftp, traceroute i ssh do sieci zewnętrznych
  - Zezwalamy na SMTP do serwera poczty
  - Zezwalamy na POP3 do serwera poczty

    - Zezwalamy na DNS do serwera nazw
    - Zezwalamy na rsync do serwera WWW
    - Zezwalamy na WWW do serwera WWW
    - Zezwalamy na ping do komputera filtrującego pakiety
- Wiele osób zezwala na dowolny ruch z sieci wewnętrznej do zewnętrznej, a potem dodaje restrykcje. My będziemy faszystami:
    - Logujemy naruszenia reguł
    - Pasywny FTP będzie obsługiwany przez moduł maskarady
    - Porty UDP 33434 i wyższe, używane są przez traceroute.

```
ipchains -A good-bad -p tcp --dport www -j MASQ
ipchains -A good-bad -p tcp --dport ssh -j MASQ
ipchains -A good-bad -p udp --dport 33434:33500 -j MASQ
ipchains -A good-bad -p tcp --dport ftp -j MASQ
ipchains -A good-bad -p icmp --icmp-type ping -j MASQ
ipchains -A good-bad -j REJECT -l
```

DMZ do Dobrej (wewnętrznej).

- Wewnętrzne restrykcje:
    - Zezwalamy na WWW, ftp, traceroute i ssh do sieci zewnętrznej
    - Zezwalamy na SMTP do serwera poczty
    - Zezwalamy na POP3 do serwera poczty
    - Zezwalamy na DNS do serwera nazw
    - Zezwalamy na rsync do serwera WWW
    - Zezwalamy na WWW do serwera WWW
    - Zezwalamy na ping do komputera filtrującego pakiety
- Jeśli użylibyśmy maskarady z sieci wewnętrznej do strefy DMZ, musielibyśmy odrzucać całą resztę pakietów. Ponieważ tak jest, zezwalamy na ruch pakietów które mogą być częścią nawiązanego połączenia.

```
ipchains -A dmz-good -p tcp ! -y -s 192.84.219.128 smtp -j ACCEPT
ipchains -A dmz-good -p udp -s 192.84.219.129 domain -j ACCEPT
ipchains -A dmz-good -p tcp ! -y -s 192.84.219.129 domain -j ACCEPT
ipchains -A dmz-good -p tcp ! -y -s 192.84.218.130 www -j ACCEPT
ipchains -A dmz-good -p tcp ! -y -s 192.84.218.130 rsync -j ACCEPT
ipchains -A dmz-good -p icmp -j icmp-acc
ipchains -A dmz-good -j DENY -l
```

DMZ do Złej (zewnętrznej).

- Restrykcje w strefie DMZ:
    - Serwer poczty:
        - SMTP do sieci zewnętrznej
        - Zezwalamy na SMTP z sieci wewnętrznej i zewnętrznej
        - Zezwalamy na POP3 z sieci wewnętrznej
    - Serwer nazw:
        - Wysyłamy DNS do sieci zewnętrznej
        - Zezwalamy na DNS z sieci wewnętrznej, zewnętrznej i komputera filtrującego pakiety
    - Serwer WWW:
        - Zezwalamy na WWW z sieci wewnętrznej i zewnętrznej
        - Zezwalamy na rsync z sieci wewnętrznej
-

```
ipchains -A dmz-bad -p tcp -s 192.84.219.128 smtp -j ACCEPT
ipchains -A dmz-bad -p udp -s 192.84.219.129 domain -j ACCEPT
ipchains -A dmz-bad -p tcp -s 192.84.219.129 domain -j ACCEPT
ipchains -A dmz-bad -p tcp ! -y -s 192.84.218.130 www -j ACCEPT
ipchains -A dmz-bad -p icmp -j icmp-acc
ipchains -A dmz-bad -j DENY -l
```

Zła (zewnętrzna) do Dobrej (wewnętrznej).

- Nie pozwalamy na nic (nie maskaradowanego) z sieci zewnętrznej do sieci wewnętrznej:

```
ipchains -A bad-good -j REJECT
```

Filtrowanie pakietów dla samego Linuksa

- Jeśli używamy filtrowania pakietów dla pakietów przychodzących do komputera filtrującego pakiety, musimy filtrować pakiety w łańcuchu `input`. Stworzymy jeden łańcuch dla każdego interfejsu przeznaczenia:

```
ipchains -N bad-if
ipchains -N dmz-if
ipchains -N good-if
```

- Następnie odpowiednie cele dla nich:

```
ipchains -A input -d 192.84.219.1 -j bad-if
ipchains -A input -d 192.84.219.250 -j dmz-if
ipchains -A input -d 192.168.1.250 -j good-if
```

Interfejs do Złej (zewnętrznej).

- Komputer filtrujący pakiety:
  - ping od każdej sieci
  - traceroute do każdej sieci
  - dostęp do DNS
- Interfejs zewnętrzny otrzymuje również odpowiedzi do pakietów zmaskaradowanych, pakiety ICMP z informacjami o błędach dla nich oraz odpowiedzi na ping:

```
ipchains -A bad-if -i ! ppp0 -j DENY -l
ipchains -A bad-if -p TCP --dport 61000:65095 -j ACCEPT
ipchains -A bad-if -p UDP --dport 61000:65095 -j ACCEPT
ipchains -A bad-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A bad-if -j icmp-acc
ipchains -A bad-if -j DENY
```

Interfejs do Strefy DMZ

- Restrykcje na komputerze filtrującym pakiety:
  - ping od każdej sieci
  - traceroute do każdej sieci
  - dostęp do DNS
- Interfejs strefy DMZ otrzymuje odpowiedzi DNS, odpowiedzi na ping oraz pakiety ICMP z informacjami o błędach:

```
ipchains -A dmz-if -i ! eth0 -j DENY
ipchains -A dmz-if -p TCP ! -y -s 192.84.219.129 53 -j ACCEPT
ipchains -A dmz-if -p UDP -s 192.84.219.129 53 -j ACCEPT
ipchains -A dmz-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A dmz-if -j icmp-acc
ipchains -A dmz-if -j DENY -l
```

Interfejs do Dobrej (wewnętrznej).

- Restrykcje na komputerze filtrującym pakiety:
  - ping od każdej sieci
  - traceroute do każdej sieci
  - dostęp do DNS
- Restrykcje wewnętrzne:
  - Zezwalamy na WWW, ftp, traceroute i ssh do sieci zewnętrznej
  - Zezwalamy na SMTP do serwera poczty
  - Zezwalamy na POP3 do serwera poczty
  - Zezwalamy na DNS do serwera nazw
  - Zezwalamy na rsync do serwera WWW
  - Zezwalamy na WWW do serwera WWW
  - Zezwalamy na ping do komputera filtrującego pakiety
- Interfejs wewnętrzny otrzymuje pingi, odpowiedzi na pingi oraz pakiety ICMP z informacjami o błędach.

```
ipchains -A good-if -i ! eth1 -j DENY
ipchains -A good-if -p ICMP --icmp-type ping -j ACCEPT
ipchains -A good-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A good-if -j icmp-acc
ipchains -A good-if -j DENY -l
```

7.5 Na koniec

- Kasujemy reguły blokujące:

```
ipchains -D input 1
ipchains -D forward 1
ipchains -D output 1
```

8. Dodatek: Różnice pomiędzy ipchains a ipfwadm.

Niektóre z tych zmian są rezultatem zmian w kernelu, a część wynika z tego, że `ipchains` są inaczej zbudowane od `ipfwadm`.

1. Wielu parametrom zmieniono znaczenie: duże litery wskazują na komendy, małe na opcje.
2. Dodano obsługę definiowalnych łańcuchów, nawet wbudowane łańcuchy mają swoje nazwy zamiast flag (tzn. '`input`' zamiast '`-I`')
3. Opcja `-k' zniknęła: używaj '`! -y`'.
4. Opcja '`-b`' dokłada/dodaje/kasuje dwie reguły, zamiast jedną 'dwukierunkową'.
5. Opcja '`-b`' może być przekazana do '`-C`' by wykonać dwa sprawdzenia (jedno dla każdego kierunku)
6. Opcja `-x' dla '`-l`' została zastąpiona przez '`-v`'.
7. Wskazywanie wielu portów źródłowych i przeznaczenia nie jest już obsługiwane. Mam nadzieję, że możliwość negowania zakresów portów jakoś to zastąpi.
8. Interfejsy mogą być wskazane tylko przez nazwę (nie przez adres). Stara semantyka i tak zmieniła się po cichu w serii kerneli 2.1.x.
9. Fragmenty są badane, a nie automatycznie przepuszczane.
10. Dedykowane łańcuchy zliczające zostały wyrzucone.
11. Można testować własne protokoły ponad IP.
12. Stare zachowanie dla testów SYN i ACK zmieniło się (było poprzednio ignorowane dla pakietów nie będących pakietami TCP); opcja SYN jest nieprawidłowa dla reguł nie dotyczących TCP
13. Liczniki są 64 bitowe na 32 bitowych maszynach, nie 32 bitowe.
14. Obsługiwana jest inwersja.
15. Obsługiwane są również kody ICMP.
16. Obsługiwane jest również wskazywanie interfejsów przez maskę.
17. Manipulacje na polach TOS są teraz sprawdzane pod kątem sensowności: stary kod kernela powstrzymywał bez żadnego komunikatu przed użyciem bitu 'Must Be Zero' w TOS; teraz `ipchains` zwracają błąd jeśli tego spróbujesz, tak jak w przypadku innych błędów.

8.1 Krótka ściąga

[W większości wypadków, argumenty komend są DUŻYMI LITERAMI a opcje małymi]

Jeszcze jedna sprawa - chęć użycia maskarady pisze się '`-j MASQ`'; różni się to kompletnie od '`-j ACCEPT`' i nie jest traktowane jako efekt uboczny, tak jak w `ipfwadm`.

```
================================================================
| ipfwadm      | ipchains              | Uwagi
----------------------------------------------------------------
| -A [both]    | -N acct               | Załóż łańcuch 'acct'
|              |& -I 1 input -j acct   | i niech pakiety wchodzące
|              |& -I 1 output -j acct  | oraz wychodzące go przechodzą
|              |& acct                 |
----------------------------------------------------------------
| -A in        | input                 | Reguła bez celu
----------------------------------------------------------------
| -A out       | output                | Reguła bez celu
----------------------------------------------------------------
```

-F	forward	Użyj jako [łańcuch].
-I	input	Użyj jako [łańcuch].
-O	output	Użyj jako [łańcuch].
-M -l	-M -L	
-M -s	-M -S	
-a policy	-A [chain] -j POLICY	(ale sprawdź -r i -m).
-d policy	-D [chain] -j POLICY	(ale sprawdź -r i -m).
-i policy	-I 1 [chain] -j POLICY	(ale sprawdź -r i -m).
-l	-L	
-z	-Z	
-f	-F	
-p	-P	
-c	-C	
-P	-p	
-S	-s	Pobiera jeden port lub zasięg, nie wiele portów.
-D	-d	Pobiera jeden port lub zasięg, nie wiele portów.
-V	<none>	Użyj -i [nazwa].
-W	-i	
-b	-b	Tworzy 2 reguły.
-e	-v	
-k	! -y	Nie działa chyba że podano z -p tcp.
-m	-j MASQ	
-n	-n	
-o	-l	
-r [redirpt]	-j REDIRECT [redirpt]	
-t	-t	
-v	-v	
-x	-x	
-y	-y	Nie działa chyba że podano z -p tcp.

8.2 Przykłady przetłumaczonych komend ipfwadm

Stara komenda: `ipfwadm -F -p deny`

Nowa komenda: `ipchains -P forward DENY`

Stara komenda: `ipfwadm -F -a m -S 192.168.0.0/24 -D 0.0.0.0/0`

Nowa komenda: `ipchains -A forward -j MASQ -s 192.168.0.0/24 -d 0.0.0.0/0`

Stara komenda: `ipfwadm -I -a accept -V 10.1.2.1 -S 10.0.0.0/8 -D 0.0.0.0/0`

Nowa komenda: `ipchains -A input -j ACCEPT -i eth0 -s 10.0.0.0/8 -d 0.0.0.0/0`

(zauważ że nie ma ekwiwalentu dla podania interfejsów przez adres: musisz użyć nazwy interfejsu. Na tej maszynie 10.1.2.1 odpowiada nazwa eth0).

9. Dodatek: użycie skryptu ipfwadm-wrapper.

Skrypt powłoki `ipfwadm-wrapper` powinien być używany jako zastąpienie dla wstecznej kompatybilności z `ipfwadm` w wersji 2.3a.

Jedyną rzeczą, której za jego pomocą nie da się obsłużyć jest '`-V`'. Jeśli użyje się tego parametru, zwracane jest ostrzeżenie. Jeśli jednocześnie użyje się opcji '`-W`', '`-V`' jest ignorowane. W innym przypadku, skrypt stara się znaleźć interfejs skojarzony z tym adresem, używając `ifconfig`. Jeśli to się nie powiedzie (na przykład dla interfejsu który jest położony) to skrypt zakończy działanie z komunikatem o błędzie.

Ostrzeżenie to może być powstrzymane przez albo zmianę '`-V`' na '`-W`' lub przekierowanie standardowego wyjścia na `/dev/null`.

Jeśli znajdziesz jakieś pomyłki w tym skrypcie, lub różnice między prawdziwym `ipfwadm` a tym skryptem, **proszę** o zgłoszenie tego do mnie: wyślij e-mail pod adres rusty@linuxcare.com z "BUG-REPORT" w temacie. Proszę podać swoją wersję `ipfwadm` (`ipfwadm -h`), wersję `ipchains` (`ipchains --version`) i wersję wrappera (`ipwadm-wrapper --version`). Proszę również dołączyć wyjście komendy `ipchains-save`. Dziękuje z góry.

Baw się w łączenie `ipchains` z tym skryptem na własne ryzyko.

10. Appendix: Thanks.

Wielkie podziękowania dla Michael'a Neulinga, który napisał pierwszy sensowny skrót kodu łańcuchów IP w czasie pracy dla mnie. Publicznie przepraszam za negowanie jego pomysłu cache'owania rezultatów, który potem zaproponował Alan Cox a ja w końcu zacząłem go implementować, widząc błąd mojego rozumowania.

Dziękuje Alanowi Coxowi za jego 24-godzinne e-mailowe wsparcie techniczne i zachęty.

Dziękuje wszystkim autorom kodu ipfw i ipfwadm, specjalnie Jos Vos. Stanie na barkach gigantów i tak dalej....to dotyczy również Linusa Torvalds'a i wszystkich ludzi zajmujących się kernelem i przestrzenią użytkownika.

Dziękuje również beta testerom i łowcom błędów, a specjalnie Jordan Mendelson, Shaw Carruthers, Kevin Moule, Dr. Liviu Daia, Helmut Adams, Franck Sicard, Kevin Littlejohn, Matt Kemner, John D. Hardin, Alexey Kuznetsov, Leos Bitto, Jim Kunzman, Gerard Gerritsen, Serge Sivkov, Andrew Burgess, Steve Schmidtke, Richard Offer, Bernhard Weisshuhn, Larry Auton, Ambrose Li, Pavel Krauz, Steve Chadsey, Francesco Potorti`, Alain Knaff, Casper Boden-Cummins i Henry Hollenberg.

10.1 Tłumaczenia

Ludzie, którzy wykonali tłumaczenia powinni umieścić się na *początku* tej sekcji, na przykład: "Podziękowania dla XXX, za dokładne przetłumaczenie wszystkiego z mojego Angielskiego.'. A potem poinformuj mnie o swoim tłumaczeniu, tak bym mógł je tutaj umieścić.

Arnaud Launay, asl@launay.org: http://www.freenix.fr/unix/linux/HOWTO/IPCHAINS-HOWTO.html

Giovanni Bortolozzo, borto@pluto.linux.it: http://www.pluto.linux.it/ildp/HOWTO/IPCHAINS-HOWTO.html

Herman Rodríguez, herman@maristas.dhis.org: http://netfilter.kernelnotes.org/ipchains/spanish/HOWTO.html

Łukasz Bromirski, l.bromirski@mr0vka.eu.org: http://mr0vka.eu.org/docs/tlumaczenia/ipchains/index.html