AKADEMIA GÓRNICZO-HUTNICZA

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

KATEDRA INFORMATYKI

Firewall

Metody filtracji

Wersja **1.0** z dnia **21.06.2002**

Kierunek, rok studiów: *Grupa (projekt):*

Informatyka, IV rok

Przedmiot:

Administrowanie systemami komputerowymi

Prowadzący zajęcia: *Rok akad:* **2001/2002**

mgr inż. Bogusław Juza *Semestr:* **letni**

Zespół autorski:

Piotr Nowak	ppnowak@poczta.onet.pl
Artur Majka	majka@student.uci.agh.edu.pl
Dariusz Kościelniak	kosciel@icslab.agh.edu.pl

Kraków, czerwiec 2002

Spis treści

1. Wstęp

Bezpieczeństwo to zagadnienie istotne zarówno dla firm jak i dla indywidualnych użytkowników. Internet to znakomite narzędzie zarówno do dystrybucji informacji o sobie jak i otrzymywaniu ich od innych, jednak z wszelkimi dobrodziejstwami globalnej sieci związana jest również pokaźna liczba zagrożeń. Przestępstwa komputerowe takie jak kradzież informacji czy celowe niszczenie danych to tylko niektóre z nich.

Najlepszym sposobem uniknięcia takiego biegu wydarzeń jest podjęcie działań prewencyjnych związanych z pozbawieniem możliwości uzyskania dostępu przez sieć do maszyny. To pole zastosowań dla firewalli. Konstrukcja bezpiecznych firewalli jest sztuką. Wymaga gruntownego zrozumienia technologii jak i filozofii projektowania firewalli.

Referat ma na celu przedstawienie metod filtracji stosowanych przez firewalle. Wśród opisanych przez nas metod znalazły się:

- Filtrowanie na poziomie pakietów IP (statyczne i dynamiczne access-listy) – Rozdziały 3, 4, 5, 6
- Maskowanie jako metoda filtracji – Rozdział 7
- Serwery proxy (kontrola strumienia danych na poziomach sesji i aplikacji) – Rozdział 8

Celem lepszego zrozumienia istoty problemu, w punkcie następnym przedstawiliśmy różne rodzaje zagrożeń na jakie jesteśmy narażeni ze strony sieci Internet. Przed częścią tych zagrożeń jesteśmy w stanie się zabezpieczyć właśnie poprzez zastosowanie firewalli.

1.1. Metody ataków

Ogólnie metody włamań możemy podzielić na:

- ataki z zewnątrz sieci lokalnej
- ataki z wnętrza sieci lokalnej
- ataki pośrednie

1.1.1. Ataki z zewnątrz

Są to ataki, których najczęstszą formą są zakłócenia stabilnej pracy. Przejmowanie kontroli nad systemami odbywa się z zewnątrz sieci lokalnej — na przykład z Internetu. Można w tym celu wykorzystać lukę w systemie zabezpieczeń, błąd serwisu sieciowego lub po prostu słaby poziom zabezpieczeń firmy. Do najczęstszych tego typu ataków należą:

- Ataki na poszczególne komputery bądź serwer główny (DoS,wirusy) - to jeden z najczęstszych typów ataków. Konsekwencjami są zwykle przerwy w pracy sieci lokalnej, uszkodzenie poszczególnych (bądź wszystkich) końcówek serwera, a co za tym idzie — całej sieci, co powoduje wielogodzinne przerwy w pracy.

- Ataki na serwer http - to ataki, których efektem jest utrata danych witryny internetowej lub uzupełnienie jej treściami kompromitującymi firmę.

Do ataków z zewnątrz sieci hakerzy często wykorzystują metodę zwaną DoS. jest to atak mający na celu zablokowanie konkretnego serwisu sieciowego (na przykład strony WWW) lub zawieszenie komputera. Możliwe jest przesterowanie ataków DoS do bardziej skomplikowanych metod, co może doprowadzić nawet do awarii całej sieci. Niejednokrotnie hakerzy, którzy włamują się do systemów za pomocą tzw. techniki spoofingu lub redyrekcji, ukrywają swój prawdziwy adres internetowy, więc ich zlokalizowanie często staje się niemożliwe. Anonimowość ułatwia więc zdecydowanie atakowanie systemów metodą Denial of Service, co powoduje uniemożliwienie wykonania przez serwer jakiejkolwiek usługi.

Spoofing (maskarada) — metoda ta stosowana jest zwykle przez doświadczonych i wyrafinowanych włamywaczy. Polega na podszyciu się włamywacza pod jednego z użytkowników systemu, który posiada własny numer IP (numer identyfikujący użytkownika). Technika ta ma na celu omijanie wszelkich zabezpieczeń, jakie zastosował administrator w sieci wewnętrznej. Jest bardzo skuteczna nawet, gdy bywa wykorzystywana przeciwko markowym firewallom, switchom i ruterom. Dzięki niej możliwe jest „udawanie" dowolnego użytkownika, a co za tym idzie — „podszywanie" się i wysyłanie sfałszowanych informacji. Ze swego komputera haker może dokonać przekierowania źródłowego adresu IP i „podszyć się" pod komputer sieciowy. Robi to po to, by określić jego bezpośrednią drogę do miejsca przeznaczenia oraz trasę powrotną. W ten sposób może przechwytywać lub modyfikować transmisje bez zliczania pakietów przeznaczonych dla komputera głównego. W przeciwieństwie do ataków polegających na rozsynchronizowaniu, „podszywanie się" pod adresy IP jest trudne do wykrycia. Jeśli serwer internetowy ma możliwość monitorowania ruchu w sieci w zewnętrznym ruterze internetowym, to należy kontrolować przechodzące przez niego dane. Do sieci nie powinny być wpuszczane pakiety zawierające adresy komputera źródłowego i docelowego, które mieszczą się w obrębie lokalnej domeny. Najlepszą obroną przed

podszywaniem się jest filtrowanie pakietów wchodzących przez ruter z Internetu i blokowanie tych, których dane wskazują na to, że powstały w obrębie lokalnej domeny.

1.1.2. Ataki z wnętrza sieci

Ataki z wnętrza sieci należą do jednych z groźniejszych. Włamanie następuje z wnętrza sieci lokalnej poprzez wykorzystanie konta jakiegoś użytkownika czy też luki w zabezpieczeniach systemu autoryzacji użytkowników. Najczęściej włamania tego typu są udziałem pracowników firmy, a nie użytkowników komputerów spoza jej obrębu, gdyż dostęp do końcówki Sieci takiej osoby rzadko pozostaje zauważony. Nadal istnieje możliwość dokonania wszystkich wyżej wymienionych ataków.

1.1.3. Ataki pośrednie.

Hakerzy stosują tu dość wyrafinowane metody, czego najlepszym przykładem są konie trojańskie. To grupa ataków najtrudniejszych do wykrycia. „Podłożenie" konia trojańskiego otwierającego dostęp do całej sieci może odbyć się za pośrednictwem poczty elektronicznej czy też podczas ściągania programu, który pochodzi z niepewnego źródła. Użytkownik prawie nigdy nie jest świadom tego, że ściągając na przykład najnowszą wersję odtwarzacza plików mp3 faktycznie otwiera dostęp do swojego komputera, a potem — całej Sieci osobom niepowołanym.

Packet sniffing (podsłuchiwanie pakietów) — jest to metoda zdobywania systemu polegająca na przechowywaniu przesyłanych przez sieć niezaszyfrowanych informacji. Można w ten sposób zdobyć hasło użytkownika i uzyskać dostęp do danego konta. Ataki polegające na „biernym węszeniu" stały się powszechne w Internecie. Stanowią one zazwyczaj wstęp do aktywnego przechwytywania cudzych plików. Aby rozpocząć tego rodzaju atak, haker musi zdobyć identyfikator i hasło legalnego użytkownika, a następnie zalogować się do Sieci. Kiedy wykona już te posunięcia, może bezkarnie podglądać i kopiować transmisje pakietów, zdobywając jednocześnie informacje o funkcjonowaniu danej sieci lokalnej

Ataki korzystające z autoryzowanego dostępu — są to ataki często stosowane przez osoby próbujące się włamać do sieci opartych na systemie operacyjnym (takim jak UNIX, VMS i Windows NT), korzystającym z mechanizmu autoryzowanego dostępu. Duże niebezpieczeństwo niesie ze sobą tworzenie plików zawierających nazwy serwerów, do których można uzyskać dostęp bez podawania hasła.

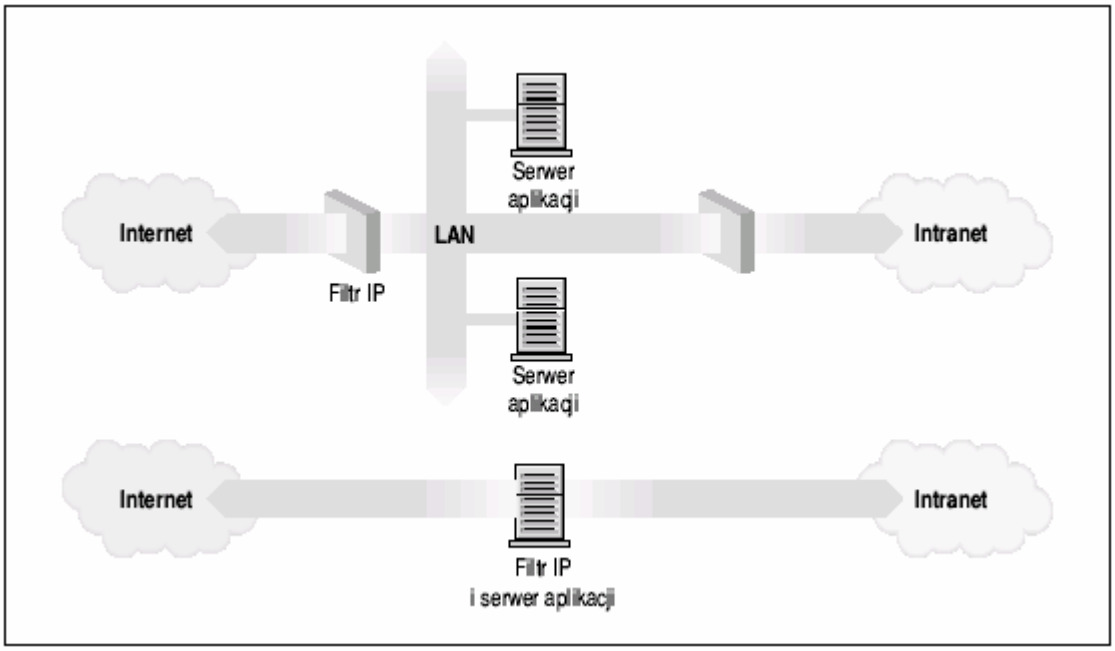
Firewalling IP jest bardzo pożyteczny przy zapobieganiu nieautoryzowanemu dostępowi, atakom typu DOS, i spoofingowi IP. Nie przydaje się przy wykorzystywaniu wszelkiego rodzaju exploit'ów w usługach sieciowych i programach oraz przy podsłuchiwaniu.

1.2. Czym jest firewall?

Firewall jest bezpieczną i zaufaną maszyną, która znajduje się pomiędzy siecią prywatną i publiczną. Maszyna ta jest skonfigurowana poprzez zestaw reguł, które determinują, który ruch sieciowy zostanie przepuszczony, a który zablokowany lub zawrócony. W niektórych dużych organizacjach firewalle stosowane są wewnątrz sieci korporacyjnej w celu odseparowania wrażliwych i czułych obszarów organizacji od reszty pracowników. W wielu przypadkach przestępstwa komputerowe zdarzają się z wewnątrz organizacji, nie z zewnątrz.

Firewalle mogą być konstruowane na wiele sposobów. Najbardziej wyrafinowany sposób to włączenie w projekt kilku niezależnych maszyn i jest znany pod nazwą sieci ograniczonej. Dwie maszyny odgrywają rolę filtrów i pozwalają przedostać się tylko pewnym typom ruchu sieciowego, pomiędzy tymi filtrami znajdują się serwery sieciowe takie jak gateway mailowy czy serwer proxy usługi www. Taka konfiguracja jest bardzo bezpieczna i pozwala na szeroki zakres kontroli nad połączeniami zarówno z wewnątrz na zewnątrz jak i odwrotnie.

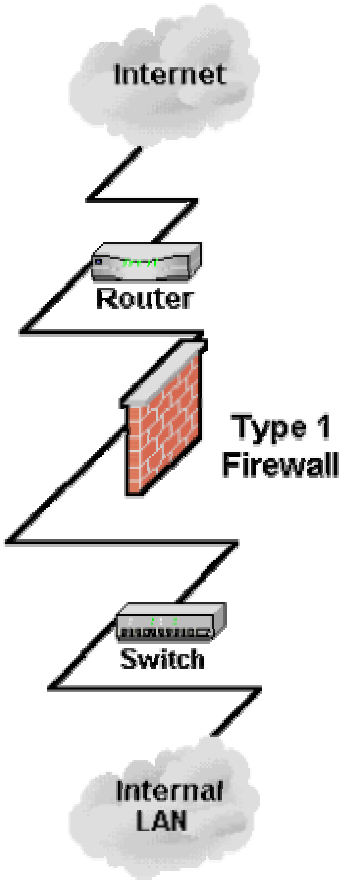
Typowe firewalle to pojedyncze maszyny, które spełniają wszystkie powyższe funkcje. Jest to rozwiązanie troszkę mniej bezpieczne, ponieważ jeśli istnieje jakaś dziura w samej maszynie firewallowej, którą ludzie mogą wykorzystać do uzyskania dostępu do niej, całe bezpieczeństwo sieci może runąć. Mimo tego, te typy firewalli są tańsze i łatwiejsze w zarządzaniu.
Poniższy Rysunek 1 przedstawia dwie najpowszechniej wykorzystywane konfiguracje firewalli.



Rysunek 1. Typy konfiguracji firewalli.

Podziału firewalle można jeszcze dokonać w inny sposób (na dwa fizyczne typy):

Typ pierwszy:



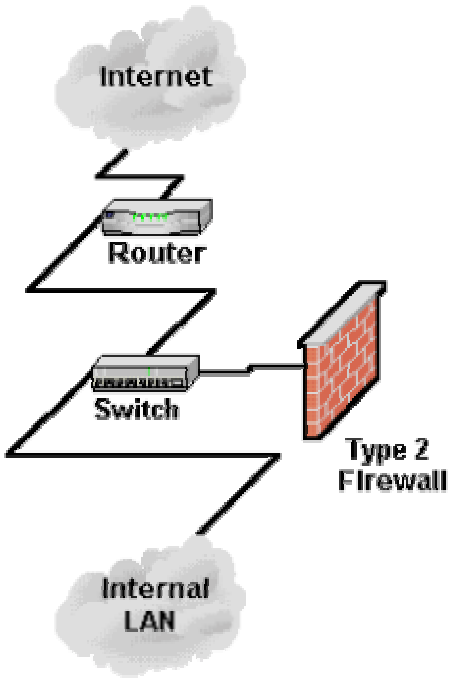
Komputer, który posiada dwie Network Interface Cards (NICs). Jedna karta służy do połączenia z wewnętrzną, lub „zaufaną" siecią, druga spina połączenie z Internetem lub siecią „niezaufaną". Specjalizowany software uruchomiony na firewallu analizuje cały ruch sieciowy aby wyznaczyć pakiety, które uzyskają możliwość przejścia i te, które powinny zostać zablokowane. Odbywa się to przy pomocy kilku metod:

- tablicy zaufanych hostów: kiedy firewall otrzymuje pakiet na interfejs podłączony do sieci zaufanej, software firewalla sprawdza, czy adres IP źródła pakietu znajduje się w tablicy zaufanych hostów. Jeśli tak, pakiet jest przepuszczany przez firewall lub jest promowany. Jeśli IP źródła nie znajduje się w tablicy zaufanych hostów, pakiet jest odrzucany i dodawany wpis do logów.

- cały ruch pakietowy przychodzący z niepewnej sieci jest sprawdzany pod kątem, czy nadejście danego pakietu jest wynikiem żądania, które wyszło z sieci zaufanej. Jeśli tak, pakiet jest wpuszczany do zaufanej sieci. W przeciwnym razie, pakiet jest odrzucany (ignorowany) a dane o nim (IP źródła i port, IP docelowe i port, etc.) są logowane.

- firewall może również badać pakiety, jeśli chodzi o Universal Resorce Lokator (URL) z którego pochodzą lub do którego zmierzają. Można zablokować dostęp do sieci pewnej grupie użytkowników, podczas gdy inna grupa będzie mogła surfować po Internecie (pakiety będą przesyłane na port 80)

Typ drugi:



Komputer posiadający tylko jedną kartę NIC i specjalizowany software. Posiada wszystkie możliwości takie jak i firewall typu pierwszego, ale ze względu na fakt posiadania tylko jednej karty sieciowej, nie jest uważany za bezpieczny.

2. Przygotowywanie Linuxa dla obsługi firewalli

Aby zbudować firewall oparty na Linuxie, niezbędne jest posiadanie kernela z wkompilowaną opcją wsparcia firewallingu IP i odpowiednim narzędziem konfiguracyjnym. We wszystkich wersjach kernela wcześniejszych niż seria 2.2, dostępne było narzędzie ipfwadm. Jądra 2.2.x posiadają trzecią wersję firewalla IP dla Linuxa zwaną IPChains. Mechanizm ten używa programu podobnego do ipfwadm zwanego ipchains. Jądra linuxa w wersjach 2.3.15 i późniejszych wspierają czwartą generację linuksowych firewalli IP zwaną netfilter. Kod tego narzędzia jest wynikiem dużego przeprojektowania w linuxie przepływu obsługi pakietów. Netfilter dostarcza bezpośredni, zgodny wstecz, support dla zarówno ipfwadm i ipchains jak i nowej alternatywnej komendy zwanej iptables.

2.1. Jądro z wkompilowaną opcją firewalla IP.

Jądro linuxa musi zostać skonfigurowane dla wsparcia firewallingu IP. Przed kompilacją jądra 2.2 należy wybrać odpowiednie opcje:

```
Networking options  --->
        [*] Network firewalls
        [*] TCP/IP networking
        [*] IP: firewalling
        [*] IP: firewall packet logging
```

W jądrach 2.4 z kolei taką:

```
Networking options  --->
        [*] Network packet filtering (replaces ipchains)
        IP: Netfilter Configuration  --->
          .
         <M> Userspace queueing via NETLINK (EXPERIMENTAL)
         <M> IP tables support (required for filtering/masq/NAT)
         <M>   limit match support
         <M>   MAC address match support
         <M>   netfilter MARK match support
         <M>   Multiple port match support
         <M>   TOS match support
         <M>   Connection state match support
         <M>   Unclean match support (EXPERIMENTAL)
         <M>   Owner match support (EXPERIMENTAL)
         <M>   Packet filtering
         <M>     REJECT target support
         <M>     MIRROR target support (EXPERIMENTAL)
          .
```

```
<M>   Packet mangling
<M>     TOS target support
<M>     MARK target support
<M>   LOG target support
<M> ipchains (2.2-style) support
<M> ipfwadm (2.0-style) support
```

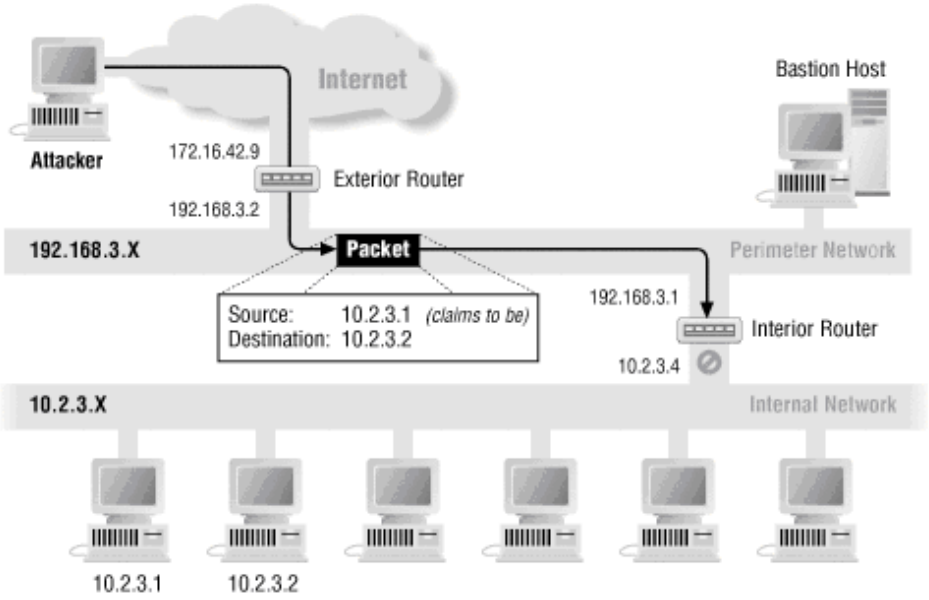
3. Filtrowanie pakietów

Filtrowanie IP jest prostym mechanizmem, który decyduje, które typy datagramów IP zostaną przetworzone, a które zostaną odrzucone. Może to następować ze względu na różnorakie kryteria wyboru:

- typ protokołu: TCP, UDP, ICMP, itp.
- numer socketu (dla TCP/UDP)
- typ datagramu: SYN/ACK, dane, odpowiedź ICMP na echo, itp.
- adres źródłowy datagramu
- adres docelowy datagramu

Filtrowanie IP odbywa się na poziomie warstwy sieciowej. Oznacza to, że filtr nie wie niczego o aplikacji używającej połączeń sieciowych, tylko o samych połączeniach. Na przykład, można odciąć użytkownikom dostęp do wewnętrznej sieci na domyślny port telneta, ale polegając tylko na filtrowaniu IP, nie można zapobiec użyciu programu telnet z portem, na którym można przedostać się przez firewalla. Takim sytuacjom można zapobiegać stosując serwery proxy dla każdej usługi, która jest udostępniana poza firewall. Serwery proxy rozumieją aplikacje, które zostały danemu proxy przypisane i dlatego są w stanie nie dopuszczać do nadużyć, takich jak np. użycie programu telnet do przedostania się przez firewalla poprzez port www. W takim wypadku jeśli firewall wspiera proxy www, to proxy będzie odpowiadał na połączenie telnetowe przez port www i pozwoli przechodzić tylko requestom HTTP. Istnieje ogromna masa programów spełniających rolę proxy-serwera. Niektóre z nich są dostępne wyłącznie komercyjnie. Więcej o serwerach proxy znajduje się w dalszej części referatu (Patrz rozdział „Serwer Proxy”).

Pewne środki bezpieczeństwa mogą być dostarczone tylko przez routery filtrujące i tylko wtedy, jeśli są uruchomione w odpowiedniej lokalizacji w sieci. Dla przykładu, dobrym pomysłem jest odrzucanie wszelkich pakietów, które mają wewnętrzny adres źródła, a przychodzą z zewnątrz – takie pakiety są zwykle częścią ataku typu spoofing. Atakujący podszywa się pod host z sieci wewnętrznej. Podjęcie decyzji o wstrzymaniu bądź przepuszczeniu pakietu może się dokonać tylko tylko w routerze filtrującym na granicy sieci.



Rysunek 2. Sfałszowanie adresu źródłowego.

3.1. Zalety filtrowania pakietów

Jedną z podstawowych zalet jest to, że pojedynczy, strategicznie umiejscowiony firewall filtrujący może pomóc strzec całą sieć.

Filtrowanie pakietów nie wymaga żadnego klienckiego oprogramowania ani konfiguracji maszyny klienckiej, nie wymaga także żadnego doszkalania użytkowników ani tworzenia specjalnych procedur postępowania dla nich. Kiedy router filtrujący decyduje o przepuszczeniu pakietu, jest on nieodróżnialny od innych routerów. Idealnie, jeśli użytkownicy nie zdają sobie sprawy z jego istnienia, chyba że próbują zrobić coś, co jest zakazane poprzez politykę filtrowania routera.

Możliwości filtrowania pakietów są dostępne w wielu hardware'owych i software'owych routerach, tak komercyjnych jak dostępnych za darmo.

3.2. Wady filtrowania pakietów.

Pomimo powszechności stosowania w różnych rozwiązaniach hardware'owych i software'owych filtrowania pakietów, nie jest ono narzędziem doskonałym. Filtrowanie pakietów, w mniejszym bądź większym stopniu posiada ograniczenia:

- reguły związane z filtrowaniem są trudne do skonfigurowania
- reguły po konfiguracji są raczej trudne do przetestowania
- zdolności filtrowania pakietów zaimplementowane w wielu produktach są niekompletne

- jak wszystkie rozwiązania informatyczne, także i produkty związane z filtrowaniem pakietów mogą posiadać bugi same w sobie.

Niektóre protokoły nie są dobrze dostosowane do bezpieczeństwa poprzez filtrowanie pakietów. Takie protokoły zawierają komendy z początkowym Berkeleyowskim „r" (rcp, rlogin, rdist, rsh, itp.). Inne takie protokoły są oparte na RPC, takie jak NFS i NIS/YP.

Kryteria według których następuje filtrowanie pakietów są ograniczone. Dla przykładu, pakiety zawierają informację, z jakiego hosta przyszły, ale nie od jakiego użytkownika. Dlatego nie można nałożyć restrykcji na poszczególnych użytkowników. Podobnie też, w pakietach zawarta jest informacja o porcie, na który pakiety są skierowane, ale nie do jakiej aplikacji; kiedy nakłada się restrykcje na protokoły wyższego poziomu, robi się to poprzez numer portu, mając nadzieję, że żadna inna aplikacja nie korzysta z portu przypisanego do tegoż protokołu.

3.3. Metody filtrowania

3.3.1. Statyczne access-listy

Access-listy to zbiory reguł, wg których następuje podejmowanie decyzji przez firewall o przepuszczeniu bądź zatrzymaniu danego pakietu, tak więc access listy mogą powstrzymać pewien ruch sieciowy przed wejściem do sieci lub też jej opuszczeniem. Kryteria, wg których access-listy dokonują filtracji pakietów to adres źródłowy lub przeznaczenia pakietu, protokół wyższej warstwy lub też inna informacja.

Istnieje wiele powodów dla konfigurowania access-list, na przykład można je wykorzystać dla ograniczania zawartości update'ów związanych ze ścieżkami routowania lub do sprawowania kontroli nad przepływem w sieci. Ale jedną z najważniejszych kwestii jest konfiguracja access-listy w celu zapewnienia sieci bezpieczeństwa. Można np. pewnym hostom zapewnić dostęp do naszej sieci, a zablokować innym. Dzięki access-listom możemy również decydować, które typy pakietów mają być przekazywane dalej, a które blokowane. Można np. pozwolić na przekazywanie e-maili, i jednocześnie blokować wszelkie połączenia telnetowe. Aby skorzystać z dobrodziejstw access-list w dziedzinie bezpieczeństwa, powinno się skonfigurować je przynajmniej na granicznych routerach, umiejscowionych na skrajach sieci, która ma być chroniona. To dostarcza podstawowego bufora chroniącego przed siecią zewnętrzną, lub przed w mniejszym stopniu kontrolowaną strefą naszej sieci. Na każdym z takich routerów, powinno się skonfigurować access-listy dla dla każdego protokołu sieciowego skonfigurowanego na każdym z interfejsów. Access-listy muszą zostać zdefiniowane przede wszystkim pod względem protokołu do którego się odnoszą, tzn. że należy je

zdefiniować dla każdego protokołu udostępnionego na danym interfejsie, jeśli tylko chce się mieć dla tego protokołu kontrolę przepływu.

Dla pojedynczej access-listy można zdefiniować wiele kryteriów w wielu, oddzielnych poleceniach access-listy. Można mieć dowolną ilość poleceń zarządzających kryteriami, jest to ograniczone tylko ilością dostępnej pamięci. Oczywiście, im więcej poleceń zawiera access-lista, tym trudniej jest nią zarządzać. Na końcu każdej access-listy powinno znaleźć się polecenie blokujące cały ruch. Jeśli pakiet nie podpada pod żadną regułę filtrującą, powinien zostać zablokowany.

Pomocne informacje przy konstruowaniu access-lists.

- Reguły filtrowania należy przygotowywać offline
  Narzędzia stosowane do edycji regułek na wielu systemach są zwykle bardzo minimalne. Nie zawsze jest jasne jak nowe reguły będą się mieć do już istniejącego zbioru reguł. W szczególności, trudno jest często skasować reguły lub dodać nowe w środku istniejącego zbioru reguł.
  Wygodnie jest przechowywać reguły filtrujące zapisane w jakimś pliku tekstowym, tak aby można było w każdej chwili edytować je i załadować później plik, tak jakby zawierał komendy wpisane prosto z konsoli. Różne systemy wspierają różne sposoby obsługi tego zagadnienia. Dla przykładu, w produktach Cisco, można używać TFTP dla otrzymania pliku komend z serwera.
  Dodatkową zaletą przetrzymywania reguł filtrujących w pliku tekstowym jest możliwość zapisania w nim komentarzy. Większość systemów filtrujących odrzuca wszelkie komentarze w komendach, które otrzymują.

- Należy ładować ponownie reguły za każdym razem.
  Pierwszą rzeczą, jaką powinien robić plik, jest czyszczenie wszystkich starych reguł, tak aby przy każdym załadowaniu pliku odbudowywać zbiór reguł od zera; w ten sposób nie trzeba przejmować się jak nowe reguły będą współdziałać ze starymi.

- Zawsze należy używać adresów IP, nigdy nazw symbolicznych hostów czy sieci
  Jeśli wyspecyfikuje się reguły filtrowania według nazw hostów, filtrowanie można obejść, gdy ktoś przypadkowo lub intencjonalnie zaingeruje w proces translacji nazw na adresy liczbowe (np. podmieniając dane w serwerze DNS).

Poniżej znajduje się przykład na zilustrowanie różnic pomiędzy różnymi systemami filtrującymi.
Aby zapewnić przepływ pakietów między komputerem z zewnętrznej sieci o przykładowym adresie IP 172.16.51.50 a hostami w wewnętrznej sieci (klasa C 192.168.10.0)

Symbolicznie wyglądałoby tak:

Rule	Direction	Source Address	Destination Address	ACK Set	Action
A	Inbound	Trusted external host	Internal	Any	Permit
B	Outbound	Internal	Trusted external host	Any	Permit
C	Either	Any	Any	Any	Deny

Używając screend, można by wyspecyfikować:

```
between host 172.16.51.50 and net 192.168.10 accept ;
between host any and host any reject ;
```

Korzystając z Telebit NetBlazer, trzeba także wyspecyfikować, którego interfejsu dana regułka się tyczy oraz czy regułki zastosowane do przychodzących czy wychodzących pakietów. Dla zewnętrznego interfejsu zwanego ”syn0”, regułki wyglądały by tak:

```
permit 172.16.51.50/32 192.168.10/24 syn0 in
deny 0.0.0.0/0 0.0.0.0/0 syn0 in

permit 192.168.10/24 172.16.51.50/32 syn0 out
deny 0.0.0.0/0 0.0.0.0/0 syn0 out
```

Na Livingston Portmaster lub IRXie, regułu specyfikuje się jako zbiór i odpowiedni zbiór stosuje się do właściwego kierunku na właściwym interfejsie. Dla zewnętrznego interfejsu nazwanego ”s1”, regułki wyglądałyby następująco:

```
add filter s1.in
set filter s1.in 1 permit 172.16.51.50/32 192.168.10.0/24
set filter s1.in 2 deny 0.0.0.0/0 0.0.0.0/0
set s1 ifilter s1.in

add filter s1.out
set filter s1.out 1 permit 192.168.10.0/24 172.16.51.50/32
set filter s1.out 2 deny 0.0.0.0/0 0.0.0.0/0
set s1 ofilter s1.out
```

Na routerach Cisco, regułki specyfikuje się także jako zbiory i stosuje się odpowiednie zbiory do odpowiedniego kierunku na odpowiednim interfejsie. Jeśli zewnętrzny interfejs nazywa się „serial1”, regułki mogą przyjąć postać:

```
access-list 101 permit ip 172.16.51.50 0.0.0.0 192.168.10.0 0.0.0.255
access-list 101 deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
interface serial 1
access-group 101 in
```

```
access-list 102 permit ip 192.168.10.0 0.0.0.255 172.16.51.50 0.0.0.0
access-list 102 deny ip 0.0.0.0 255.255.255.255 0.0.0.0 255.255.255.255
interface serial 1
access-group 102 out
```

Różne systemy filtrujące podejmują różne domyślne akcje, jeśli pakiet nie pasuje do żadnej z wyspecyfikowanych regułek. Niektóre systemy odrzucają wszystkie takie pakiety. Inne stosują się do zaprzeczenia ostatniej z reguł. W każdym przypadku dobrym zwyczajem jest umieszczenie na końcu zestawu reguł reguły domyślnej.

Odmianą statycznych access list są listy bazujące na porze dnia. Należy stworzyć przedział czasowy, który definiuje określone pory czasowe dnia i tygodnia. Przedział czasowy jest identyfikowany przez nazwę.
Obecnie, jedynymi funkcjami, które mogą używać przedziałów czasowych są rozszerzone access listy IP i IPX. Przedziały czasowe pozwalają na zdefiniowanie przez administratora sieci, kiedy mają odnosić skutek polecenia **permit** i **deny.** Dzięki temu administrator posiada większą kontrolę nad zezwalaniem i blokowaniem dostępu użytkowników do określonych zasobów. Te zasoby to mogą być aplikacje (identyfikowane przez parę adres IP/maska oraz numer portu), routowanie wg jakiejś polityki lub zestawianie połączenia na żądanie. Takie access listy zmniejszają także ilość logowanych wiadomości, gdyż logowanie ruchu sieciowego może odbywać się w tylko ściśle określonych porach dnia.
Poniżej znajduje się składnia komend dla tworzenia takich access list w rozwiązaniach Cisco.
Aby stworzyć rozszerzoną nazwaną access listę IP, należy używać:

Krok	**Polecenie**	**Cel**
1.	**ip access-list extended** *name*	Definiuje rozszerzoną access listę IP
2.	{**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**precedence** *precedence*] [**tos** *tos*] [**established**] [**log**] [**time-range** *time-range-name*] {**deny** | **permit**} *protocol* **any any** [**log**] [**time-range** *time-range-name*]	Zezwolenie lub zablokowanie dostępu. Użycie słowa kluczowego **log** w celu logowania informacji o pakietach. Time-range specyfikuje przedział czasowy w jakim **permit** lub **deny** ma działać Definicja rozszerzonej access listy IP przy użyciu skrótu dla adresu źródła i maski źródła 0.0.0.0 i 255.255.255.255, oraz skrótu dla adresu i maski przeznaczenia -również 0.0.0.0 i 255.255.255.255. Definicja dynamicznej access-listy opisana w

	dynamic *dynamic-name* [**timeout** *minutes*] {**deny** | **permit**} *protocol source source-wildcard destination destination-wildcard* [**precedence** *precedence*] [**tos** *tos*] [**established**] [**log**] [**time-range** *time-range-name*]	następnym punkcie.

Jak wiadomo statyczne access-listy mogą być tworzone w systemie Linux na wiele sposobów, z wykorzystaniem różnych narządzi tj. `ipfwadm`, `itchains`, `iptables`. Opis poszczególnych narzędzi do tworzenia statycznych access-list został zawarty w kolejnych rozdziałach: 4, 5, 6.

3.3.2. Dynamiczne access-listy

Aktualnie access-listy są raczej statyczne. Ma to kilka mankamentów, ponieważ oznacza, że istnieje w jednym miejscu przez cały czas globalna, nieelastyczna polityka bezpieczeństwa. Ten fakt uniemożliwia zaspokajanie wszystkich potrzeb różnych klas użytkowników, jak i różnych rodzajów aplikacji, które mogą potrzebować różniących się od siebie typów połączeń, aby poprawnie działać. Jeśli polityka bezpieczeństwa jest zbyt rygorystyczna, pewne aplikacje mogą nie być zdolne do uruchamiania się przy pełnych funkcjonalnościach, z drugiej strony zaś polityka zbyt liberalna może otworzyć niektóre dziury w bezpieczeństwie.

Jako prosty przykład można wyobrazić sobie pewne typy przywilejów otrzymywane przy przechodzeniu przez firewall. Otrzymanie ich na stałe otworzy pewne luki w bezpieczeństwie, nieotrzymanie ich w ogóle, zaowocuje tym, że aplikacja nie będzie w stanie poprawnie się wykonać. Najprostsza dynamiczna polityka zakłada, że regułki firewallowe będą uaktualniane na czas wykonania się aplikacji. Zmniejsza to czas otwarcia dziury w bezpieczeństwie do okresu, gdy działa aplikacja.

Dynamiczne access-listy zapewniają, że regułki firewallowe mogą być efektywnie uaktualniane, przy zapewnieniu podstawowego poziomu bezpieczeństwa i zachowaniu ogólnej semantyki access-listy. Niektóre projekty zakładają podział reguł firewallowych na dwie klasy: obligatoryjne i preferowane. Te drugie mogą zostać w każdej chwili tymczasowo przysłonięte przez żądanie użytkownika, podczas gdy obligatoryjne nie mogą.

Dynamiczne access listy przyznają dostęp na użytkownika do określonych hostów źródła/przeznaczenia poprzez proces autentykacji użytkownika. Tego typu listy stanowią kluczowy element technologii firmy Cisco zwanej Lock-and-Key. Pozwala ona na telnetowanie się na główny router lub serwer dostępu (do

danego przedsiębiorstwa) i autentykację na nim. Lock-and-Key tworzy w firewallu tymczasową „dziurę", która pozwala na przedostanie się poza firewall.
Lock-and-Key zakłada, że logujący się dysponuje bezpiecznym sposobem autentykacji siebie.
Dynamiczne access listy posiadają przewagę nad statycznymi ze względu na to, że te ostatnie:

- pozostawiają stałe wejścia, które hackerzy mogą wykorzystać
- są trudne w zarządzaniu przy dużych sieciach
- nadmiernie rozbudowane mogą przeciążać router
- nie zapewniają mechanizmu autentykacji indywidualnych użytkowników.

Dynamiczne access-listy są pozbawione tych wad.
Generalnie, adres IP źródła, z którego użytkownik telnetuje się zastępuje w dynamicznych odwołaniach adres źródła lub przeznaczenia, zależnie od tego czy access lista jest dla pakietów przybywających czy wychodzących.

4. Oryginalny firewall IP(jądra 2.0)

Rozdział ten oraz dwa następne (Łańcuchy firewalla IP(jądra 2.2), Netfilter i tabele IP(jądra 2.4)) traktują o konkretnych implementacjach firewalli w systemie Linux.

Pierwsza generacja obsługi firewalla IP w Linuksie pojawiła się w serii jąder 1.1. Było to przeniesienie *ipfw* z BSD dokonane przez Alana Coxa. Obsługa firewalla w jądrach serii 2.0, określana mianem drugiej generacji, została rozszerzona przez Josa Vos, Pauline Middelink i innych.

4.1. Korzystanie z `ipfwadm`

Polecenie *ipfwadm* było narzędziem konfiguracyjnym dla drugiej generacji firewalla IP w Linuksie. Polecenie *ipfwadm* ma wiele różnych argumentów odnoszących się do konfiguracji firewalla IP.

4.1.1. Składnia polecenia `ipwadm`

ipfwadm *kategoria polecenie parametry [opcje]*

Kategorie
Musi być podana jedna i tylko jedna z poniższych kategorii. Kategoria mówi firewallowi, jakiego typu reguła jest konfigurowana:

-I
Reguła wejściowa.

-O
Reguła wyjściowa.

-F
Reguła przekazywania.

Polecenia
Przynajmniej jedno z poniższych poleceń musi być podane i musi się ono odnosić do określonej wcześniej kategorii. Polecenia mówią firewallowi, co ma robić.

-a [polityka]
Dodanie nowej reguły.

-i [polityka]
Wstawienie nowej reguły.

-d [polityka]
Usunięcie istniejącej reguły.

-p polityka
Ustawienie polityki domyślnej.

-l
Wylistowanie wszystkich istniejących reguł.

-f
Usunięcie wszystkich istniejących reguł.

Polityki istotne dla firewalla IP:

accept
Pozwala na odbiór, przekazywanie lub wysyłanie pasujących datagramów.

deny
Nie pozwala na odbiór, przekazywanie lub wysyłanie pasujących datagramów.

reject
Nie pozwala na odbiór, przekazywanie lub wysyłanie pasujących datagramów i wysyła komunikat błędu ICMP do hosta, który przesłał datagram.

Parametry
Musi być podany przynajmniej jeden z poniższych parametrów. Parametry używane są do określania datagramów, których dotyczą reguły:

-P protokół
Może mieć wartość TCP, UDP, ICMP lub all.

-S adres/maska/[port]
Źródłowy adres IP, do którego pasuje ta reguła. Domyślna maska to „/32". Opcjonalnie można określić, którego portu dotyczy reguła. Należy także podać protokół za pomocą opisanego wyżej argumentu *–P*, aby ta opcja zadziałała. Jeżeli nie zostanie podany port lub zakres portów, przyjmuje się, że wszystkie porty pasują. Porty mogą być podane w postaci nazwy zgodnej z wpisem w */etc/services*. W przypadku protokołu ICMP pole portu jest używane do oznaczenia typu datagramu ICMP. Możliwe jest podanie zakresu portów, a służy do tego następująca składnia: *pierwszyport:ostatniport*.

-D adres/maska/[port]
Określenie adresu docelowego IP, do którego pasuje ta reguła. Adres docelowy jest zapisywany na tej samej zasadzie co adres źródłowy opisany poprzednio.

-V adres
Określenie adresu interfejsu sieciowego, na którym pakiet jest odbierany (*-I*) lub z którego jest wysyłany (*-O*). Pozwala to na stworzenie reguł dotyczących tylko niektórych interfejsów sieciowych komputera.

-W nazwa
Określenie nazwy interfejsu sieciowego. Ten argument działa w ten sam sposób co *–V*, ale podaje się nazwę urządzenia zamiast adresu.

Argumenty opcjonalne
Te argumenty są czasem bardzo przydatne:

-b
Jest używany dla trybu dwukierunkowego. Do tej opcji pasuje ruch w obie strony pomiędzy zadanymi adresami źródłowym i docelowym. Zaoszczędza nam to tworzenia dwóch reguł: jednej do wysyłania i drugiej do odbierania.

-o
Pozwala na zapisywanie pasujących datagramów do logu jądra. Wszelkie datagramy pasujące do reguły będą zapisywane jako komunikaty jądra. Jest to użyteczna opcja do wykrywania nieautoryzowanego dostępu.

-y
Ta opcja jest używana do filtrowania połączeniowych datagramów TCP. Dzięki niej reguła filtruje tylko datagramy podejmujące próbę zestawienia połączeń TCP. Przepuszczane będą jedynie datagramy posiadające ustawiony bit SYN i wyzerowany bit ACK. Jest to użyteczna opcja do filtrowania prób połączeń TCP i ignorowania innych protokołów.

-k
Jest używana do filtrowania datagramów-potwierdzeń TCP (ang. *acknowledgement*). Ta opcja powoduje, że do reguły pasują tylko datagramy będące potwierdzeniem odbioru pakietów próbujących zestawić połączenie TCP. Będą pasować jedynie datagramy, które mają ustawiony bit ACK. Opcja ta jest użyteczna do filtrowania prób połączeń TCP i ignorowania wszystkich pozostałych protokołów.

5. Łańcuchy firewalla IP(jądra 2.2)

Tradycyjna implementacja firewalla IP jest dobra, ale może być niewydajna przy konfiguracji bardziej złożonych środowisk. Aby sprostać nowym wymaganiom, opracowano nową metodę konfigurowania firewalla IP i rozwinięto związane z nią właściwości. Ta nowa metoda otrzymała nazwę „Łańcuchy firewalla IP" (w skrócie łańcuchy IP) i pierwszy raz została wprowadzona do użytku w jądrze Linuksa 2.2.0.
Łańcuchy IP pozwalają na tworzenie klas reguł, do których można następnie dodawać hosty albo sieci lub je stamtąd usuwać. Łączenie reguł w łańcuchy jest o tyle lepsze, że poprawia wydajność firewalla w konfiguracjach, gdzie używa się wielu reguł.
Łańcuchy IP są obsługiwane przez jądra serii 2.2, ale są także dostępne w postaci łat do jąder 2.0

5.1. Używanie `ipchains`

Z narzędzia konfiguracyjnego *ipchains* można korzystać na dwa sposoby. Pierwszy polega na użyciu skryptu *ipfwadm-wrapper*, który w zasadzie udaje *ipfwadm*, bo wywołuje w tle program *ipchains*.
Można też używać *ipchains,* korzystając bezpośrednio ze składni. Składnia *ipchains* jest prostsza niż *ipfwadm*. Program *ipfwadm* do skonfigurowania firewalla musiał operować na trzech regułach. W przypadku łańcuchów IP można stworzyć dowolną liczbę zestawów reguł, gdzie każda będzie połączona z inną, ale wciąż są obecne trzy zestawy reguł związane z firewallem. Standardowe zestawy reguł są bezpośrednimi odpowiednikami tych używanych w *ipfwadm*, poza tym, że mają nazwy: input, forward i output.

5.1.1. Składnia polecenia `ipchains`

Poniżej przedstawiamy najważniejsze elementy składni *ipchains*. Ogólna składnia większości poleceń *ipchains* jest następująca:

ipchains *polecenie określenie-reguły opcje*

Polecenia
Istnieje szereg sposobów na operowanie na regułach i zestawach reguł za pomocą polecenia *ipchains*. Istotne dla firewalla IP są:

-A łańcuch
Dodanie jednej lub kilku reguł na koniec zadanego łańcucha. Jeżeli jest podana nazwa źródłowego lub docelowego hosta i tłumaczy się na więcej niż jeden adres IP, zostanie dodana reguła dla każdego z tych adresów.

-I łańcuch numerreguły
Wstawienie jednej lub kilku reguł na początek zadanego łańcucha. Znów, jeżeli w określeniu reguły zostanie podana nazwa hosta, będzie dodawana do każdego adresu.

-D łańcuch
Usunięcie jednej lub kilku reguł z zadanego łańcucha, który pasuje do reguły.

-D łańcuch numerreguły
Usunięcie reguły znajdującej się na pozycji *numerreguły* w zadanym łańcuchu. Numeracja reguł zaczyna się od pierwszej reguły w łańcuchu.

-R łańcuch numerreguły
Zastąpienie reguły na pozycji *numerreguły* w zadanym łańcuchu podaną regułą.

-C łańcuch
Sprawdzenie zadanym łańcuchem datagramu opisanego regułą. Polecenie to zwraca komunikat opisujący, jak datagram był przetwarzany przez łańcuch. Jest to bardzo użyteczna opcja do testowania konfiguracji firewalla.

-L [łańcuch]
Listowanie reguł zadanego łańcucha lub wszystkich łańcuchów, jeżeli żaden nie zostanie zadany.

-F [łańcuch]
Usunięcie reguł z zadanego łańcucha lub usunięcie wszystkich reguł, jeżeli żaden łańcuch nie zostanie zadany.

-Z [łańcuch]
Wyzerowanie liczników datagramów i bajtów dla wszystkich reguł zadanego łańcucha lub wszystkich łańcuchów, jeżeli żaden nie zostanie zadany.

-N łańcuch
Stworzenie nowego łańcucha o zadanej nazwie. Nie może istnieć drugi łańcuch o tej samej nazwie. W ten sposób tworzone są łańcuchy definiowane przez użytkownika.

-X [łańcuch]
Usunięcie zadanego łańcucha zdefiniowanego przez użytkownika lub wszystkich łańcuchów zdefiniowanych przez użytkownika, jeżeli żaden nie zostanie zadany. Aby to polecenie zadziałało, nie może być odwołań do zadanego łańcucha z żadnych innych reguł.

-P łańcuch polityka
Ustawienie domyślnej polityki dla zadanego łańcucha. Dopuszczalne polityki to ACCEPT, DENY, REJECT, REDIR i RETURN. Polityki ACCEPT, DENY i REJECT mają takie samo znaczenie jak w tradycyjnych implementacjach firewalla. REDIR oznacza, że datagram powinien być niewidocznie przekierowany na port firewalla. RETURN powoduje, że kod firewalla IP powraca do tego łańcucha firewalla, który wywołał łańcuch zawierający tę regułę, i kontynuuje dalsze działanie, począwszy od następnej reguły.

Parametry określające regułę
Na regułę *ipchains* składa się wiele parametrów, które określają, jakie typy pakietów mają do niej pasować. Jeżeli któryś z tych parametrów zostanie w regule pominięty, zakładana jest jego wartość domyślna.

-p [!] protokół
Określa protokół datagramu, który będzie pasował do tej reguły. Dopuszczalne nazwy protokołów to tcp, udp, icmp lub all. Możliwe jest także podawanie numer protokołu. Jeżeli podane zostanie !, reguła zostanie zanegowana i datagram będzie dopasowywany do wszystkich protokołów poza zadanymi. Jeżeli parametr nie zostanie podany, zostanie przyjęta wartość all.

-s [!]adres[/maska][!][port]
Określa adres źródłowy i port w datagramie, który ma pasować do tej reguły. Adres może być podany w postaci nazwy hosta, nazwy sieci lub adresu IP. Opcja mask pozwala na zadanie używanej maski sieci, która może być podana albo w tradycyjnej postaci (tj. /255.255.255.0), albo w postaci współczesnej (tj. /24). Opcjonalny port określa port TCP lub UDP albo typ dopasowywanego datagramu ICMP. Numer portu może zostać wskazany tylko wtedy, gdy został użyty wcześniej parametr *–p*, oraz podany jeden z protokołów: tcp, udp lub icmp. Można też podać zakres portów – jego dolną i górną granicę rozdzielone dwukropkiem. Na przykład 20:25 opisuje wszystkie porty od 20 do 25 włącznie. Znak ! może być wykorzystany do zanegowania wartości.

-d [!]adres[/maska][!][port]
Określa adres i port docelowy zawarte w datagramie, który ma pasować do tej reguły. Kodowanie tego parametru jest identyczne jak parametru *–s*.

-j cel
Określa działanie do wykonania, jeżeli reguła będzie pasowała. Parametr ten może być tłumaczony jako „skocz do” (ang. *jump to*). Dopuszczalne cele to ACCEPT, DENY, REJECT, REDIR i RETURN. Ich znaczenie opisaliśmy wcześniej. Jednak możliwe jest podanie także nazwy łańcucha zdefiniowanego przez użytkownika, w którym będzie wykonywane dalsze przetwarzanie. Jeżeli ten parametr zostanie pominięty, to nawet jeśli datagram pasuje do reguły, nie zostanie podjęte żadne inne działanie, oprócz uaktualnienia datagramu i liczników bajtów.

-i[!]nazwa-interfejsu
Określa interfejs, który przyjął datagram lub przez który zostanie on wysłany. Znów znak ! odwraca wynik dopasowania. Jeżeli nazwa interfejsu kończy się znakiem +, pasował będzie każdy interfejs, którego nazwa rozpoczyna się zadanym ciągiem. Na przykład, -i ppp+ będzie pasować do dowolnego urządzenia sieciowego PPP, a -i ! eth+ będzie pasować do wszystkich urządzeń poza Ethernetem.

[!]-f
Mówi, że reguła ta dotyczy wszystkiego poza pierwszym fragmentem datagramu podzielonego na fragmenty.

Opcje
Poniżej pokazane opcje *ipchains* są bardziej ogólne. Niektóre z nich sterują raczej wyszukanymi funkcjami oprogramowania łańcuchów IP:

-b
Powoduje, że polecenie generuje dwie reguły. Jedna reguła uwzględnia podane parametry, a druga uwzględnia je w odwrotnym kierunku.

-v
Powoduje, że *ipchains* wyświetla bogate wyniki, czyli podaje więcej informacji.

-n
Powoduje, że *ipchains* wyświetla adres IP i porty jako liczby bez próby ich zamiany na odpowiadające im nazwy.

-l
Włącza zapisywanie przez jądro pasujących datagramów. Wszystkie datagramy pasujące do reguły są zapisywane przez jądro za pomocą funkcji *printk()*. Zwykle jest to obsługiwane przez program *syslogd* zapisujący do pliku log. Funkcja ta przydaje się do oglądania niestandardowych datagramów.

-o[maxrozmiar]
Powoduje, że oprogramowanie łańcuchów IP kopiuje datagramy pasujące do reguły do urządzenia „netlink", które działa w przestrzeni użytkownika. Argument *maxrozmiar* ogranicza liczbę bajtów każdego datagramu, która zostanie przekazana do urządzenia netlink. Opcja ta jest najchętniej stosowana przez programistów, ale może być w przyszłości wykorzystana w pakietach oprogramowania.

-m wartośćznakowania
Powoduje, że pasujące datagramy są *oznaczane* zadaną wartością. Te wartości to 32-bitowe liczby bez znaku. W obecnych implementacjach opcja ta nie działa, ale w przyszłości może decydować o obsłudze datagramu przez inne oprogramowanie, takie jak na przykład kod rutujący. Jeżeli *wartośćznakowania* rozpoczyna się od znaku + albo -, jest ona dodawana lub odejmowana od aktualnej wartości znakowania.

-t maskaand maskaxor
Pozwala na operowanie na bitach „typ usługi" w nagłówku IP każdego datagramu pasującego do reguły. Bity typu usługi są używane przez inteligentne rutery do nadawania priorytetów datagramom, zanim zostaną dalej przekazane. Oprogramowanie rutujące Linuksa potrafi realizować takie nadawanie priorytetów. Wartości *maskaand* i *maskaxor* oznaczają maski bitowe, które będą poddawane odpowiednio logicznej operacji AND i OR z bitami typu usługi datagramu. Jest to zaawansowana funkcja, która szczegółowo została omówiona w *IPCHAINS-HOWTO*.

-y
Powoduje, że do reguły pasują wszystkie datagramy TCP z ustawionym bitem SYN i wyzerowanymi bitami ACK i FIN. Jest używana do filtrowania żądań nawiązania połączenia TCP.

5.2. Korzystanie z łańcuchów

5.2.1. Łańcuchy definiowane przez użytkownika

Trzy zestawy reguł dla tradycyjnego firewalla IP stanowią mechanizm tworzenia prostych konfiguracji firewalla, którymi łatwo jest zarządzać w małych sieciach o niewielkich wymaganiach wobec systemu bezpieczeństwa. Gdy wymagania konfiguracyjne wzrastają, pojawia się szereg problemów. Po pierwsze, duże sieci często wymagają dużo więcej reguł firewalla, niż tych kilka, z którymi się do tej pory spotkaliśmy. Nieuchronnie rosną potrzeby dodawania do firewalla reguł obsługujących przypadki szczególne. Gdy liczba reguł rośnie, wydajność firewalla pogarsza się, bo na każdym datagramie jest przeprowadzanych coraz więcej testów; problemem staje się też zarządzanie. Po drugie, nie jest możliwe

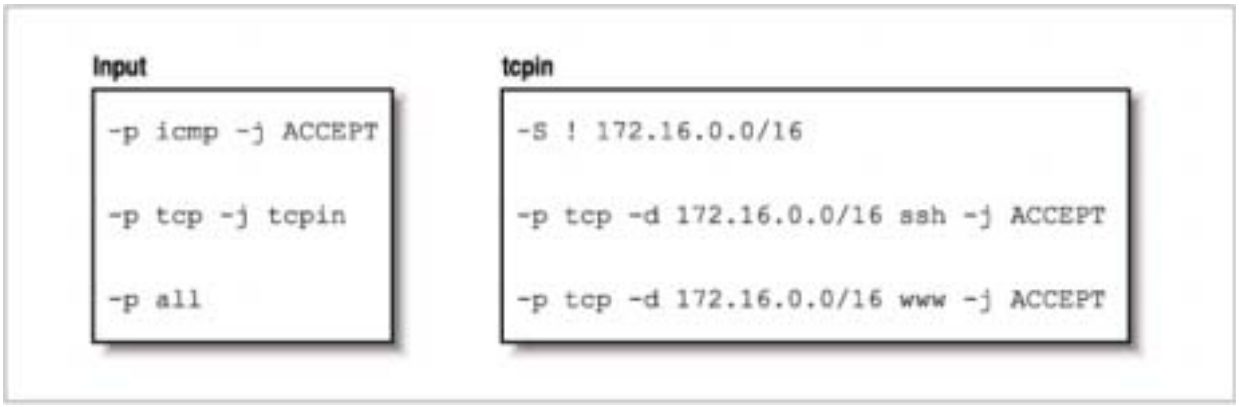
włączanie i wyłączanie zestawu reguł w sposób rozdzielny. Gdy jesteśmy w trakcie przebudowy zestawu reguł, narażamy sieć na ataki.

Zasady budowy łańcuchów IP pomagają złagodzić te problemy, gdyż umożliwiają administratorowi tworzenie dowolnych zestawów reguł firewalla, które można następnie dołączać do trzech wbudowanych zestawów reguł. Do utworzenia nowego łańcucha można użyć opcji *–N* programu *ipchains*. Trzeba podać jego nazwę, składającą się z 8 (lub mniejszej ilości) znaków. Opcja *–j* konfiguruje działanie podejmowane wtedy, gdy datagram pasuje do wymagań reguły. Mówi, że jeżeli datagram będzie pasował do reguły, dalsze testowanie powinno być realizowane w łańcuchu zdefiniowanym przez użytkownika.

Rozważmy następujące polecenia *ipchains*:

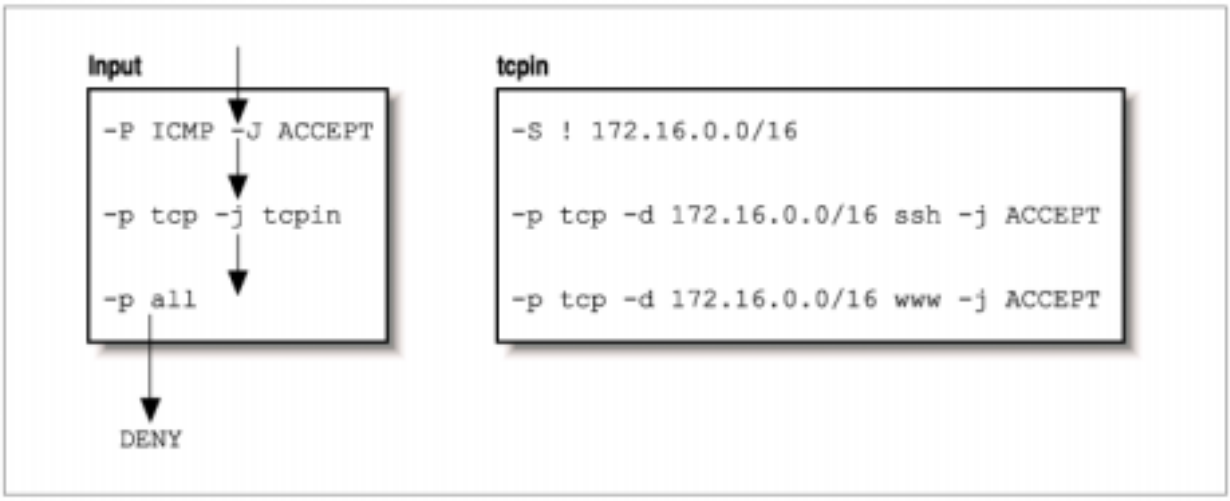
```
ipchains -P input DENY
ipchains -N tcpin
ipchains -A tcpin -s ! 172.16.0.0/16
ipchains -A tcpin -p tcp -d 172.16.0.0/16 ssh -j ACCEPT
ipchains -A tcpin -p tcp -d 172.16.0.0/16 www -j ACCEPT
ipchains -A input -p tcp -j tcpin
ipchains -A input -p all
```

Domyślną politykę łańcucha wejściowego ustawiamy na deny. Drugie polecenie tworzy definiowany przez użytkownika łańcuch o nazwie „tcpin”. Trzecie polecenie dodaje do łańcucha tcpin regułę, do której pasują wszystkie datagramy pochodzące spoza naszej sieci lokalnej. Nie jest podejmowane żadne dodatkowe działanie. Jest to reguła zliczająca. Do następnych dwóch reguł pasują datagramy przeznaczone dla naszej sieci lokalnej na porty ssh lub www. Pasujące datagramy są akceptowane. W następnej regule tkwi prawdziwa magia *ipchains*. Reguła ta powoduje, że oprogramowanie firewalla sprawdza każdy datagram TCP za pomocą łańcucha tcpin, zdefiniowanego przez użytkownika. Na koniec dodajemy regułę do naszego łańcucha input, do którego pasuje każdy datagram. Jest to kolejna reguła zliczająca. W ten sposób uzyskujemy łańcuchy firewalla pokazane na poniższym Rysunku 3.



Rysunek 3. Prosty zestaw reguł łańcucha IP

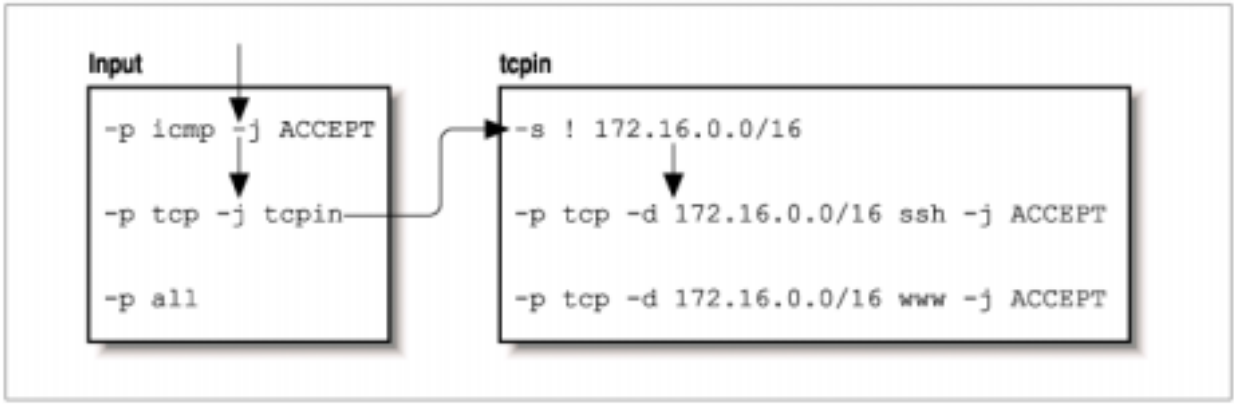
Nasze łańcuchy input i tcpin są zapełniane regułami. Przetwarzanie datagramu zawsze rozpoczyna się w jednym z łańcuchów wbudowanych. Zobaczymy, jak zdefiniowany przez nas łańcuch jest używany przy przetwarzaniu różnych typów datagramów. Najpierw przyjrzymy się, co się dzieje, gdy zostanie odebrany datagram UDP dla jednego z naszych hostów. Rysunek 4 pokazuje przepływ przez reguły.



Rysunek 4. Kolejność sprawdzanych reguł dla odebranego datagramu UDP

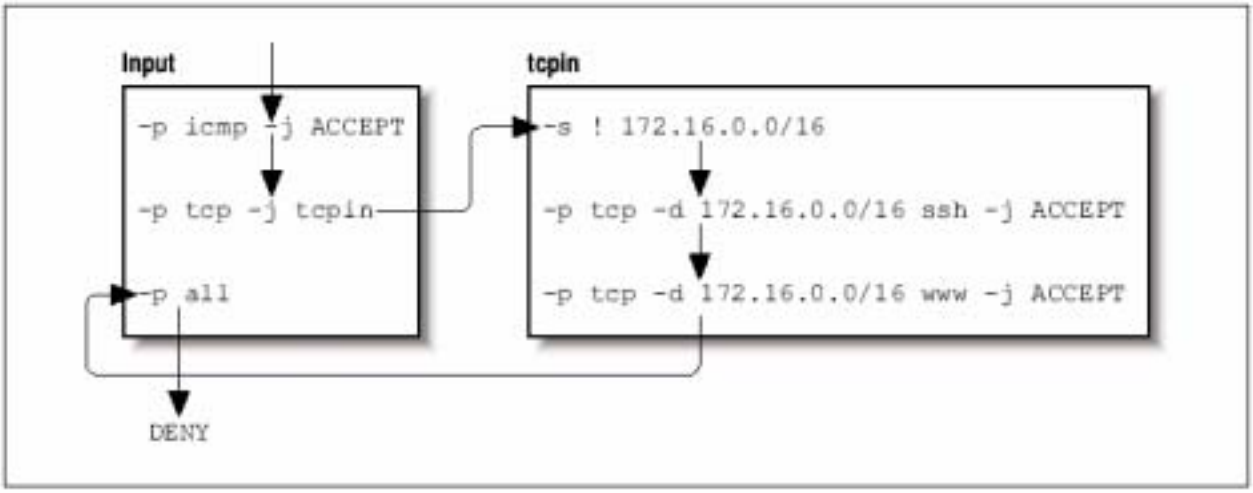
Datagram zostaje odebrany przez łańcuch input, ale nie pasuje do dwóch pierwszych reguł, ponieważ pasują do nich tylko datagramy protokołów ICMP i TCP. Zostaje dopasowany do trzeciej reguły łańcucha input, która nie zawiera celu. Są więc uaktualniane liczniki bajtowy i datagramów, ale nie jest podejmowane żadne inne działanie. Datagram dociera do końca łańcucha input, spełniając warunki jego domyślnej polityki i zostaje odrzucony.

Aby zobaczyć zdefiniowany przez nas łańcuch w działaniu, rozważmy, co się dzieje, gdy zostanie odebrany datagram TCP przeznaczony dla portu ssh jednego z naszych hostów. Kolejność pokazano na Rysunku 5.



Rysunek 5. Kolejność sprawdzania reguł dla odebranego pakietu TCP dla portu ssh.

Tym razem datagram pasuje do drugiej reguły w łańcuchu input, która kieruje go do celu tcpin – łańcucha zdefiniowanego przez użytkownika. Podanie łańcucha zdefiniowanego przez użytkownika jako celu powoduje, że datagram będzie sprawdzany przez zawarte w nim reguły, a więc następną sprawdzaną regułą będzie pierwsza reguła z łańcucha tcpin. Do tej reguły pasują datagramy, które mają adres źródłowy spoza sieci lokalnej i dowolny adres przeznaczenia, a więc jest to także reguła zliczająca i testowanie przechodzi do następnej reguły. Druga reguła w naszym łańcuchu tcpin pasuje do datagramu i określa cel ACCEPT. Dotarliśmy do celu, a więc nie będzie już żadnego przetwarzania przez firewall. Datagram zostaje przepuszczony. Na koniec zobaczmy, co się stanie, gdy dotrzemy do końca zdefiniowanego przez nas łańcucha. Aby to zobaczyć, musimy pokazać przepływ datagramu TCP przeznaczonego dla portu innego niż dwa przez nas obsługiwane. Pokazujemy to na rysunku 6.



Rysunek 6. Kolejność sprawdzania reguł dla odebranego pakietu TCP dla portu telnet

Łańcuchy zdefiniowane przez użytkownika nie mają polityki domyślnej. Gdy wszystkie zawarte w nich reguły zostaną sprawdzone i żadna nie pasuje, firewall działa tak, jakby istniała reguła RETURN, a więc jeżeli nie tego chcieliśmy, powinniśmy na końcu łańcucha użytkownika umieścić żądane działanie. W naszym przykładzie sprawdzanie powraca do reguły z zestawu input następnej po tej, przez którą przeszliśmy do łańcucha zdefiniowanego przez użytkownika. Ostatecznie docieramy do końca łańcucha input, który posiada politykę domyślną i datagram zostaje odrzucony. Ten przykład jest bardzo prosty, ale pokazuje to, o co nam chodziło. W praktyce działanie łańcuchów IP jest dużo bardziej skomplikowane.

5.2.2. Skrypty pomocnicze `ipchains`

Pakiet oprogramowania *ipchains* jest dostarczany wraz z trzema dodatkowymi skryptami. Pierwszy z nich został już krótko omówiony, natomiast pozostałe dwa zapewniają łatwe i wygodne sposoby zachowywania i odtwarzania konfiguracji firewalla. Skrypt *ipfwadm-wrapper* emuluje składnię wiersza poleceń *ipfwadm*, ale wymaga polecenia *ipchains* do tworzenia reguł. Jest to wygodny sposób na migrację istniejącej konfiguracji

firewalla do jądra lub alternatywa dla opanowania składni *ipchains*. Skrypt *ipfwadm-wrapper* zachowuje się inaczej niż polecenie *ipfwadm* pod dwoma względami. Po pierwsze, *ipchains* nie pozwala na określenie interfejsu przez adres, a *ipfwadm-wrapper* przyjmuje argument *–V*, ale próbuje zamienić go na właściwy dla *ipchains* odpowiednik *–W*, szukając nazwy interfejsu skonfigurowanej pod zadanym adresem. Skrypt *ipfwadm-wrapper* zawsze przypomina nam o tym, wypisując komunikat, gdy użyjemy opcji *–V*. Po drugie, reguły zliczania fragmentów nie są tłumaczone poprawnie.
Skrypty *ipchains-save* i *ipchains-restore* upraszczają tworzenie i modyfikowanie konfiguracji firewalla. Polecenie *ipchains-save* odczytuje aktualną konfigurację firewalla i zapisuje uproszczoną postać na standardowe wyjście. Polecenie *ipchains-restore* odczytuje dane w formacie wyprowadzanym przez *ipchains-save* i konfiguruje firewall IP zgodnie z odczytanymi regułami. Korzyścią z używania tych skryptów jest możliwość natychmiastowego dynamicznego tworzenia konfiguracji i jej zapisania, czego nie daje bezpośrednie modyfikowanie skryptu konfigurującego firewall i testowanie konfiguracji. Taką konfigurację można następnie odtworzyć, zmodyfikować i zapisać ponownie.
Aby użyć tych skryptów i zachować aktualną konfigurację firewalla, należy napisać coś takiego:

```
ipchains-save >/var/state/ipchains/firewall.state
```

Możliwe będzie późniejsze jej odtworzenie, zwykle w czasie uruchamiania systemu, w następujący sposób:

```
ipchains-restore </var/state/ipchains/firewall.state
```

Skrypt *ipchains-restore* sprawdza, czy istnieją już umieszczone w konfiguracji łańcuchy zdefiniowane przez użytkownika. Jeśli podana zostanie opcję -f, reguły z wcześniej skonfigurowanych łańcuchów użytkownika będą automatycznie usunięte przed wczytaniem nowych. Natomiast przy działaniu domyślnym zostaniemy odpytani, czy pozostawić, czy usunąć dany łańcuch.

6. Netfilter i tabele IP(jądra 2.4)

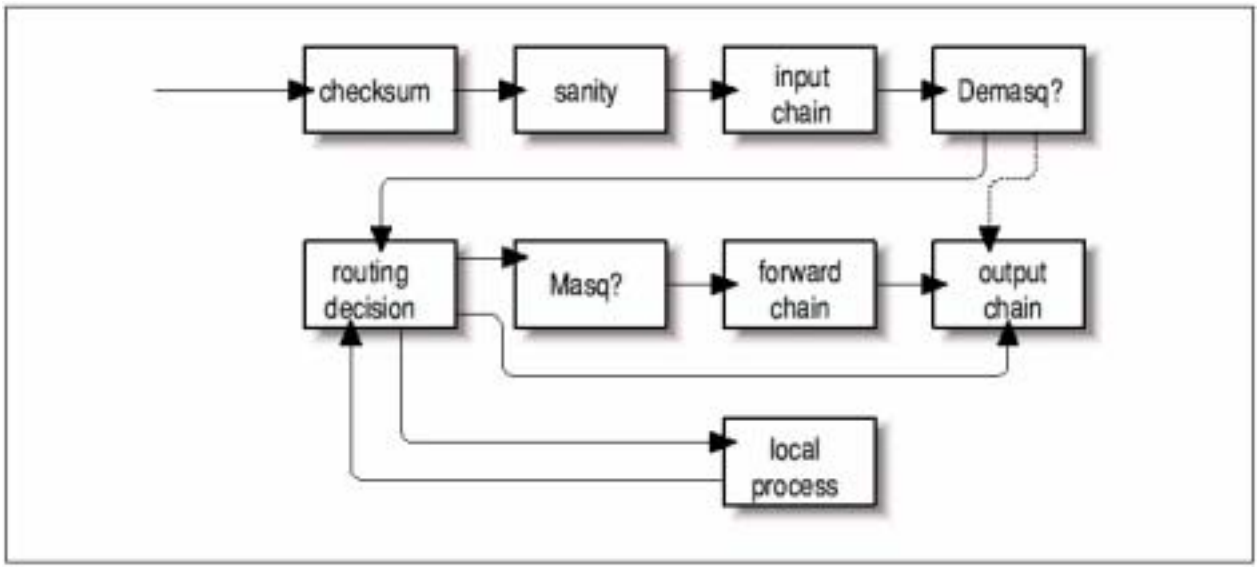
Podczas prac nad łańcuchami IP, twórcy doszli do wniosku że firewalle IP powinny być mniej skomplikowane. Zajęto się zatem upraszczaniem przetwarzania datagramów w kodzie firewalla zawartym w jądrze i stworzono strukturę filtrującą, która była dużo prostsza i dużo bardziej elastyczna. Ta nowa struktura została nazwana *netfilter*.

Cóż więc było nie tak z łańcuchami IP? Poprawiły one znacznie wydajność i zarządzanie regułami firewalla, ale sposób przetwarzania datagramów wciąż był skomplikowany, szczególnie w połączeniu z funkcjami związanymi z firewallem, takimi jak maskowanie IP i inne formy translacji adresów.
Częściowa złożoność wynikała z faktu, że maskowanie IP i translacja adresów sieciowych powstały niezależnie od kodu firewalla w jądrze i dopiero później zostały do niego dołączone. Niestety nie było to wszystko tworzone razem od początku i zintegrowane w kodzie firewalla. Gdyby twórcy chcieli dodać następne funkcje związane z przetwarzaniem datagramów, mieliby trudności ze znalezieniem miejsca na umieszczenie swojego kodu i musieliby dokonać zmian w jądrze, aby dopiąć swego.
Poza tym istniały jeszcze inne problemy. W szczególności łańcuch „input" opisywał wejście do całej warstwy sieci IP. Łańcuch wejściowy dotyczył zarówno datagramów *przeznaczonych dla* hosta, jak i datagramów *rutowanych przez* host. Było to nieco mylące, ponieważ mieszało funkcję łańcucha wejściowego z funkcją łańcucha przekazującego, dotyczącego tylko datagramów przekazywanych dalej, ale zawsze sprawdzanych po przejściu przez łańcuch wejściowy. Aby móc inaczej traktować datagramy przeznaczone dla hosta niż datagramy przekazywane dalej, należałoby stworzyć złożone reguły wykluczające jedne lub drugie. Ten sam problem dotyczył łańcucha wyjściowego. Oczywiście ta złożoność poniekąd dotknęła administratora systemu, ponieważ odbiła się na sposobie tworzenia reguł. Co więcej, wszelkie rozszerzenia filtrowania wymagały bezpośrednich modyfikacji jądra, ponieważ wszystkie polityki filtrowania były w nim zaimplementowane i nie było sposobu na stworzenie przezroczystego interfejsu. *netfilter* radzi sobie zarówno ze złożonością, jak i sztywnością starszych rozwiązań, implementując w jądrze ogólną strukturę określającą sposób przetwarzania datagramów i zapewniającą możliwość rozbudowy polityki filtrowania bez potrzeby modyfikacji jądra.

Przyjrzyjmy się dwóm kluczowym zmianom, które zostały dokonane. Rysunek 7 pokazuje, jak datagramy są przetwarzane w implementacji łańcuchów IP, natomiast rysunek 8 pokazuje, jak są one przetwarzane przez *netfilter*. Zasadnicze różnice to usunięcie z głównego kodu funkcji maskowania i zmiana umiejscowienia łańcuchów wejściowego i wyjściowego. Zmianom tym towarzyszy nowe, dające się rozbudować narzędzie o nazwie *iptables*.

W łańcuchach IP łańcuch wejściowy dotyczył wszystkich datagramów odebranych przez host bez względu na to, czy były one dla niego przeznaczone, czy rutowane do innego hosta. W *netfilter* łańcuch wejściowy dotyczy *tylko* datagramów przeznaczonych dla hosta lokalnego, a łańcuch przekazujący dotyczy tylko datagramów przeznaczonych dla *innego* hosta. Podobnie w łańcuchach IP, łańcuch wyjściowy dotyczy wszystkich datagramów wychodzących z hosta lokalnego bez względu na to, czy są to datagramy na nim stworzone, czy przez niego rutowane z innego hosta. W *netfilter* łańcuch wyjściowy dotyczy *tylko* datagramów wygenerowanych na danym hoście, a nie dotyczy datagramów przez niego rutowanych. Sama ta zmiana stanowi poważne uproszczenie wielu konfiguracji firewalla.

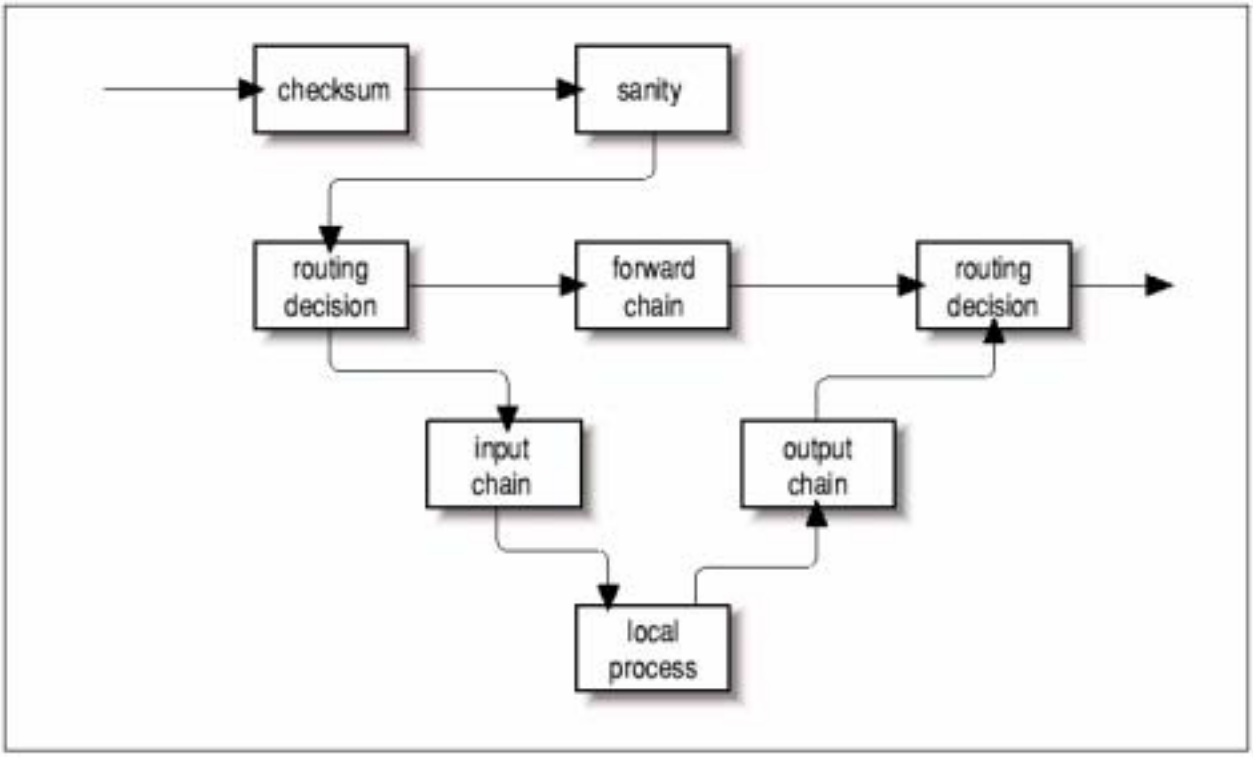
Na rysunku 7 elementy opisane jako „demasq" i „masq" są oddzielnymi elementami jądra odpowiedzialnymi za przetwarzanie przychodzących i wychodzących datagramów maskowanych. Zostały one ponownie zaimplementowane w *netfilter* jako moduły.



Rysunek 7. Łańcuch przetwarzania datagramów w łańcuchach IP

Przyjrzyjmy się konfiguracji, w której domyślna polityka dla każdego łańcucha: wejściowego, wyjściowego i przekazującego, jest ustawiona na deny. W łańcuchach IP potrzebne jest sześć reguł, aby przepuszczać jakąkolwiek sesję przez firewall: po dwie w każdym łańcuchu: wejściowym, wyjściowym i przekazującym (jedna obsługiwałaby przesyłanie w jedną stronę, a druga w drugą.). Można sobie wyobrazić, jak łatwo mogłoby to się stać nadzwyczaj skomplikowane i trudne do ogarnięcia, gdybyśmy chcieli mieszać sesje rutowane i sesje połączeń do hosta bez rutowania. Łańcuchy IP pozwalają na tworzenie łańcuchów nieco upraszczających to zadanie, ale ich budowa nie jest oczywista i wymaga pewnego doświadczenia. W implementacji *netfilter* ta złożoność znika zupełnie dzięki *iptables*. W przypadku usługi rutowanej przez firewalla, ale nie kończącej się na hoście lokalnym, potrzebne są tylko dwie reguły w łańcuchu przekazującym: jedna w jednym kierunku i druga w przeciwnym. Jest to oczywisty sposób tworzenia reguł

dla firewalla i znacznie upraszcza jego konfigurację. W dokumencie *PACKET-FILERING-HOWTO* można znaleźć szczegółową listę zmian, dokonywanych w czasie tworzenia oprogramowania, a więc tam należy szukać bardziej praktycznych zagadnień związanych z tym tematem.



Rysunek 8. Łańcuch przetwarzania datagramów w netfilter

6.1. Wsteczna zgodność z `ipfwadm` i `ipchains`

Niezwykła elastyczność *netfilter* w Linuksie uwidacznia się w możliwości emulowania interfejsów *ipfwadm* i *ipchains*. Dzięki emulacji przejście na oprogramowanie firewalla nowej generacji jest dużo prostsze. Dwa moduły jądra *netfilter*, zwane *ipfwadm.o* i *ipchains.o*, zapewniają kompatybilność wsteczną dla *ipfwadm* i *ipchains*. Możliwe jest załadowanie tylko jednego z nich na raz i używać tylko wtedy, gdy moduł *ip_tables.o* nie jest załadowany. Gdy odpowiedni moduł zostanie załadowany, *netfilter* działa dokładnie tak jak poprzednia implementacja firewalla. Netfilter naśladuje interfejs *ipchains* po wydaniu następujących poleceń:

```
rmmod ip_tables
modprobe ipchains
ipchains …
```

6.2. Używanie `iptables`

Program *iptables* jest używany do konfigurowania reguł filtrowania *netfilter*. Jego składnia ma wiele wspólnego z *ipchains*, ale różni się pod jednym bardzo szczególnym względem: jest *rozszerzalna*. Oznacza to, że jej funkcjonalność można rozszerzyć bez ponownej kompilacji. Służą do tego biblioteki dzielone. Istnieją standardowe rozszerzenia, które zostaną omówione w dalszej części opracowania. Zanim będzie możliwe używanie polecenia *iptables*, musi zostać załadowany moduł jądra *netfilter* niezbędny do jego obsługi. Najprościej można zrobić to za pomocą polecenia *modprobe*:

```
modprobe ip_tables
```

Polecenie *iptables* jest używane do konfigurowania zarówno filtrowania IP, jak i translacji adresów sieciowych (*Network Address Translation*). Aby temu sprostać, istnieją dwie tablice reguł: *filter* i *nat*. Tablica *filter* jest używana domyślnie, jeżeli nie zostanie podana opcji *–t* dokonująca zmianę zachowania. W *netfilter* udostępniono pięć wbudowanych łańcuchów. Łańcuchy INPUT i FORWARD są dostępne dla tablicy *filter*, a PREROUTING i POSTROUTING są dostępne dla tablicy *nat*, natomiast łańcuch OUTPUT jest dostępny dla obu tablic.

Ze względu na dosyć obszerną ilość poleceń, określeń reguł i rozszerzeń dostępnych w *iptables* w poniższy punkcie przedstawiamy tylko najważniejsze.
Ogólna składnia większości poleceń *iptables* jest następująca:

iptables *polecenie określenie-reguły rozszerzenia*

Przedstawiamy tylko kilka wybranych opcji.

Polecenia
Istnieje szereg sposobów operowania na regułach i ich zestawach za pomocą polecenia *iptables*. Istotne dla filtrowania IP są następujące:

-A łańcuch
Dodanie jednej lub kilku reguł na koniec zadanego łańcucha. Jeżeli zostanie podana nazwa hosta źródłowego lub docelowego, z którą związany jest więcej niż jeden adres IP, reguła zostanie dodana do każdego adresu.

-I łańcuch num_reguły
Wstawienie jednej lub kilku reguł na początku zadanego łańcucha. Znów, jeżeli w opisie reguły zostanie podana nazwa hosta, reguła będzie dodana dla każdego adresu IP odpowiadającego temu hostowi.

-D łańcuch
Usunięcie jednej lub kilku reguł z zadanego łańcucha, który takie reguły zawiera.

-D łańcuch num_reguły
Usunięcie reguły znajdującej się na pozycji *num_reguły* w zadanym łańcuchu. Liczenie reguł zaczyna się od 1 w przypadku pierwszej reguły w łańcuchu.

-R łańcuch num_reguły
Zastąpienie reguły na pozycji *num_reguły* w zadanym łańcuchu regułą o podanej charakterystyce.

-L[łańcuch]
Wylistowanie reguł z zadanego łańcucha lub ze wszystkich łańcuchów, jeżeli żaden nie zostanie wybrany.

-F [łańcuch]
Usunięcie reguł z zadanego łańcucha lub ze wszystkich łańcuchów, jeżeli żaden nie zostanie wybrany.

-N łańcuch
Utworzenie nowego łańcucha o zadanej nazwie. Nie mogą istnieć łańcuchy o tej samej nazwie. W ten sposób tworzy się łańcuch definiowany przez użytkownika.

-P łańcuch polityka
Ustawienie domyślnej polityki dla zadanego łańcucha. Dopuszczalne polityki to ACCEPT, DROP, QUEUE i RETURN. ACCEPT pozwala na przepuszczenie datagramu. DROP powoduje, że datagram jest odrzucany. QUEUE powoduje, że datagram jest przekazywany do przestrzeni użytkownika w celu dalszego przetwarzania. RETURN powoduje, że kod firewalla IP wraca do łańcucha, który go wywołał, i kontynuuje działanie od następnej reguły.

Parametry definicji reguły
Istnieje kilka parametrów *iptables* używanych do definiowania reguły. Gdy wymagane jest zdefiniowanie reguły, musi zostać podana wartość każdego z nich albo zostaną przyjęte wartości domyślne.

-p[!]protokół
Określa protokół datagramu, który ma pasować do tej reguły. Dopuszczalne nazwy protokołów to: tcp, udp, icmp lub numer, jeżeli znasz numery protokołu IP. Gdy ten parametr nie zostanie określony, domyślnie przyjęte będą wszystkie protokoły.

-s[!]adres[/maska]
Określa adres źródłowy datagramu, który będzie pasował do tej reguły. Adres może być podany w postaci nazwy hosta, nazwy sieci lub adresu IP. Opcjonalny parametr maska definiuje maskę sieci, która ma być zastosowana. Może być ona podana w postaci tradycyjnej (tj. /255.255.255.0) lub w postaci współczesnej (tj./24).

-d[!]adres[/maska]
Określa adres przeznaczenia i port datagramu, który będzie pasował do tej reguły. Kodowanie tego parametru jest takie samo jak parametru *–s*.

-j cel
Określa, jakie działanie ma zostać podjęte, gdy reguła zostanie dopasowana. Parametr ten może być tłumaczony jako „skocz do” (ang. *jump to*). Dopuszczalne cele to ACCEPT, DROP, QUEUE i RETURN. Ich znaczenie opisaliśmy wcześniej w sekcji „Polecenia”. Możliwe jest podanie także nazwy zdefiniowanego przez użytkownika łańcucha, w którym będzie wykonywane dalsze przetwarzanie. Można także podać cel obsługiwany przez rozszerzenie.

-i[!]nazwa-interfejsu
Określa interfejs, który przyjął datagram. Znów znak ! odwraca wynik dopasowania. Jeżeli nazwa interfejsu kończy się znakiem +, pasował będzie każdy interfejs, którego nazwa rozpoczyna się zadanym ciągiem

-o[!]nazwa-interfejsu
Określa interfejs, na który datagram będzie wysłany. Ten argument ma taką samą składnię jak *–i*.

[!]-f
Mówi, że reguła ta dotyczy tylko drugiego i dalszych fragmentów datagramu. Nie dotyczy pierwszego fragmentu.

Opcje
Poniżej pokazano bardziej ogólne opcje *iptable.*

-v
Powoduje, że *iptables* wyświetla pełne wyniki. Podawane będzie więcej informacji.

-n
Powoduje, że *iptables* wyświetla adres IP i porty tylko jako liczby, nie próbuje zamieniać ich na odpowiadające im nazwy.

-x
Powoduje, że wszelkie liczby w wyniku *iptables* są pokazywane dokładnie, bez zaokrąglania.

Rozszerzenia
Powiedzieliśmy wcześniej, że *iptables* jest narzędziem rozszerzalnym poprzez opcjonalne moduły bibliotek dzielonych. Istnieją standardowe rozszerzenia udostępniające funkcje *ipchains*. Aby z nich skorzystać, należy podać poleceniu *iptables* ich nazwę poprzez argument *-m nazwa*. Poniższa lista pokazuje opcje *–m* i *–p* określające kontekst rozszerzeń oraz opcje udostępniane przez rozszerzenie.

Rozszerzenia TCP: używane z –m tcp –p tcp

- -sport [!][port[:port]]
Określa port, z którego musi pochodzić datagram, aby pasował do reguły. Można wyznaczyć pewien zakres portów, wypisując górny i dolny limit (należy rozdzielić je dwukropkiem. Znak ! może być użyty do zanegowania wartości.

- -dport[!][port[:port]]
Określa port, do którego musi być skierowany datagram, aby pasował do reguły. Argument jest kodowany identycznie jak *--sport*.

[!] --syn
Powoduje, że reguła pasuje tylko do datagramów z ustawionym bitem SYN i wyzerowanymi bitami ACK i FIN. Datagramy z tymi bitami są używane do otwierania połączeń TCP i dlatego ta opcja jest używana do obsługi żądań połączeń.

Rozszerzenia UDP: używane z –m udp –p udp

- -sport[!][port[:port]]
Określa port, z którego musi pochodzić datagram, aby pasował do reguły. Porty mogą być podane jako zakres przez określenie górnego i dolnego limitu zakresu, które należy rozdzielić dwukropkiem.

- -dport[!][port[:port]]
Określa port, do którego musi być skierowany datagram, aby pasował do reguły. Argument jest kodowany tak samo jak *--sport*.

Rozszerzenia ICMP: używane z –m icmp –p icmp

- -icmp-type[!] nazwa_typu
Określa typ komunikatu ICMP pasującego do reguły. Typ może być określony przez nazwę lub numer. Niektóre dopuszczalne nazwy to: echo-request, echo-reply, source-quench, time-exceeded, destination-unreachable, network-unreachable, host-unreachable, protocol-unreachable i port-unreachable.

Rozszerzenia MAC: używane z –m mac

- -mac-source[!] adres
Określa adres hosta Ethernet, który wysłał datagram pasujący do tej reguły. Ma to sens jedynie w łańcuchach wejściowym i przekazującym reguły, ponieważ wszystkie datagramy, które przechodzą przez łańcuch wychodzący, są wysyłane przez nas.

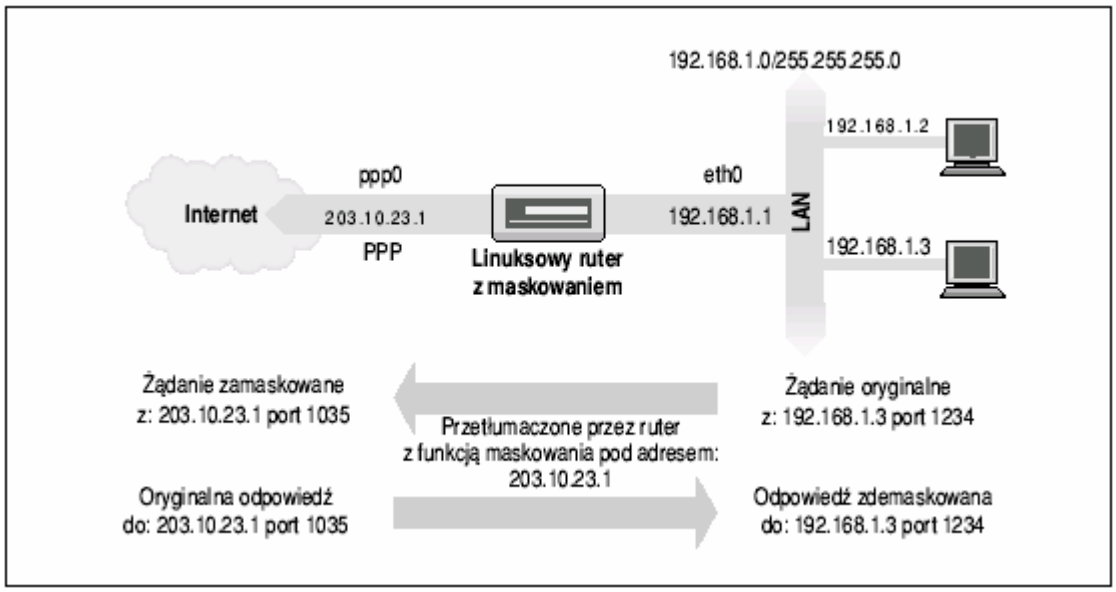
7. Maskowanie IP i translacja adresów sieciowych

Obecnie technologia sieciowa jest na tyle tania, że możemy zaobserwować dwa zjawiska. Po pierwsze, sieci LAN są teraz powszechne, nawet dostępne w gospodarstwach domowych. Wielu użytkowników Linuksa ma po kilka komputerów połączonych siecią Ethernet. Po drugie, zasoby sieciowe, szczególnie adresy IP, kończą się. Większość posiadaczy sieci LAN zwykle chce także mieć połączenie z Internetem, wykorzystywane przez każdy komputer w sieci. Tradycyjne rozwiązania tego problemu zakładają uzyskanie adresu IP dla sieci, być może klasy C w przypadku mniejszych ośrodków, i przypisanie adresu każdemu hostowi sieci LAN , a następnie podłączenie sieci LAN do Internetu przez ruter.

W skomercjalizowanych środowiskach internetowych jest to dosyć droga propozycja. Po pierwsze, musielibyśmy zapłacić za przypisanie twojej sieci adresów IP. Po drugie, musielibyśmy zapłacić dostawcy usług internetowych za przywilej posiadania odpowiedniego rutera dla twojej sieci, dzięki któremu reszta Internetu wiedziałaby, jak się do niej dostać. Może być to wciąż praktyczne rozwiązanie dla firm, ale właściciele domowych instalacji zwykle nie są w stanie udźwignąć jego kosztów.

Na szczęście Linux oferuje inne rozwiązanie tego problemu. Rozwiązanie to wymaga elementu z grupy zaawansowanych funkcji sieciowych, tak zwanej ***translacji adresów sieciowych*** (*Network Address Translation* – **NAT**). Jest to proces modyfikacji adresów sieciowych zawartych w nagłówkach datagramu, który zachodzi w czasie ich przesyłania. Jest to zatem idealne rozwiązanie opisywanego problemu i wiele osób z niego korzysta. Maskowanie IP (ang. *IP masquerading*) to nazwa nadana jednej z odmian translacji adresów sieciowych, która pozwala hostom z adresem sieci prywatnej przedstawić się w Internecie pod jednym publicznym adresem IP.

Maskowanie IP daje możliwość używania adresu IP z sieci prywatnej (zarezerwowanego) w twojej sieci LAN, a ruter oparty na Linuksie wykonuje w czasie rzeczywistym inteligentne tłumaczenie adresów IP i portów. Gdy ruter odbierze datagram od komputera z sieci LAN, sprawdza, czy jest to datagram typu „TCP", „UDP", „ICMP" itp. i modyfikuje go tak, że wygląda on jakby był wygenerowany przez sam ruter. Następnie ruter wysyła datagram do Internetu, nadając mu jeden adres IP. Gdy host docelowy odbierze datagram, sądzi, że przyszedł on z rutera i wyśle w odpowiedzi datagramy na jego adres. Gdy ruter linuksowy z włączonym maskowaniem odbierze datagram przez swoje połączenie sieciowe, zajrzy do swojej tablicy aktywnych, maskowanych połączeń, by zobaczyć, czy datagram rzeczywiście należy do komputera w sieci LAN. Jeżeli tak, odwróci dokonaną przez siebie wcześniej modyfikację oraz wyśle datagram temu komputerowi. Prosty przykład takiego działania pokazano na poniższym rysunku.



Rysunek 9. Typowa konfiguracja maskowania IP

Mamy małą sieć Ethernet wykorzystującą jeden z zarezerwowanych adresów sieci. W sieci jest linuksowy ruter z funkcją maskowania, dający dostęp do Internetu. Jedna ze stacji roboczych w sieci (192.168.1.3) chce połączyć się z hostem zdalnym o adresie 209.1.106.178 na porcie 8888. Stacja robocza kieruje swój datagram do rutera, który stwierdza, że żądanie połączenia wymaga maskowania. Przyjmuje datagram i alokuje port (1035), zastępuje adres i port hosta swoimi własnymi i przesyła datagram do hosta docelowego. Docelowy host myśli, że otrzymał żądanie połączenia od rutera linuksowego z włączonym maskowaniem i generuje datagram z odpowiedzią. Otrzymawszy datagram, ruter znajduje powiązanie w tablicy maskowania i odwraca operacje wykonane na wychodzącym datagramie. Następnie przesyła odpowiedź do hosta, od którego pierwotnie wyszedł datagram.

Lokalny host myśli, że komunikuje się bezpośrednio z hostem zdalnym. Zdalny host nic nie wie o hoście lokalnym, a myśli, że połączenie nawiązuje z ruterem. Ruter z maskowaniem wie, że te dwa hosty komunikują się ze sobą, jakich portów używają i dokonuje translacji adresów i portów wymaganej do uzyskania takiej komunikacji.

7.1. Skutki uboczne maskowania

Funkcji maskowania IP towarzyszą pewne skutki uboczne, z których niektóre są użyteczne, a inne mogą przeszkadzać.

Po pierwsze, żaden z hostów sieci, która jest obsługiwana przez ruter maskujący, nie jest widziany bezpośrednio, dzięki czemu potrzebujemy tylko jednego poprawnego i rutowalnego adresu IP, aby wszystkie hosty mogły łączyć się z Internetem. Ma to taką wadę, że żaden z hostów nie jest widoczny z Internetu i nie możesz się do niego podłączyć bezpośrednio. Jedynym widocznym hostem w maskowanej sieci jest sama

maszyna maskująca. Jest to istotne, gdy rozważamy usługi, takie jak poczta czy FTP. Pomaga ustalić, jakie usługi powinny być udostępniane przez ruter maskujący, a jakie powinny być udostępniane przez proxy lub traktowane w inny, szczególny sposób.
Po drugie, ponieważ żaden z maskowanych hostów nie jest widoczny, są one w pewnym sensie zabezpieczone przed atakami z zewnątrz. Zapewne upraszcza to konfigurowanie firewalla na hoście maskującym. Nie należy jednak zbytnio ufać maskowaniu. Cała nasza sieć jest tylko tak zabezpieczona jak host maskujący, a więc powinniśmy użyć firewalla, jeżeli bezpieczeństwo jest dla nas istotne.
Po trzecie, maskowanie IP będzie miało pewien wpływ na wydajność sieci. W typowych konfiguracjach prawdopodobnie będzie to ledwo zauważalne. W przypadku gdy mamy wiele aktywnych, zamaskowanych sesji, możemy zauważyć, że przetwarzanie realizowane na maszynie maskującej zaczyna wpływać na przepustowość sieci. Maskowanie IP wymaga wykonywania sporej pracy dla każdego datagramu, porównywalnej z typowym rutingiem.

Pewne usługi sieciowe nie będą działały przy włączonej funkcji maskowania lub będą wymagały wsparcia. Zwykle są to usługi, które opierają się na przychodzących sesjach, takie jak bezpośrednie kanały komunikacyjne (*Direct Communications Channels* – DCC) w IRC-u czy pewne typy usług grupowych wideo i audio. Dla niektórych z nich stworzono specjalne moduły jądra pozwalające rozwiązać problem.

7.2. Konfigurowanie jądra do maskowania IP

Aby użyć funkcji maskowania IP, jądro musi zostać skompilowane z jej obsługą. W czasie konfigurowania jądra serii 2.2 należy wybrać następujące opcje:

```
Networking options  --->
    [*] Network firewalls
    [*] TCP/IP networking
    [*] IP: firewalling
    [*] IP: masquerading
    --- Protocol-specific masquerading support will be built as modules.
    [*] IP: ipautofw masq support
    [*] IP: ICMP masquerading
```

Należy zauważyć, że obsługa maskowania jest dostępna tylko jako moduł jądra. Oznacza to, że w czasie kompilacji jądra należy pamiętać o wykonaniu „make modules” poza zwykłym „make zImage”. Jądra serii 2.4 nie posiadają obsługi maskowania IP jako opcji wybieranej w czasie kompilacji jądra. Dla nich należy wybrać opcję filtrowania pakietów sieciowych:

```
Networking options  --->
    [M] Network packet filtering (replaces ipchains)
```

W czasie kompilacji jąder serii 2.2 powstaje szereg modułów pomocniczych, specyficznych dla protokołów. Niektóre protokoły rozpoczynają działanie od wysyłania żądań na jednym porcie, a potem oczekują na połączenia przychodzące na innym porcie. Protokoły takie zwykle nie mogą być maskowane, gdyż nie ma innego sposobu na powiązanie drugiego połączenia z pierwszym, jak powiązanie ich wewnątrz samych protokołów. Moduły pomocnicze to właśnie robią. W praktyce zaglądają do środka datagramów i umożliwiają działanie maskowania z niektórych protokołów, co w innej sytuacji byłoby niemożliwe. Obsługiwane protokoły to:

Module	**Protocol**
ip_masq_ftp	FTP
ip_masq_irc	IRC
ip_masq_raudio	RealAudio
ip_masq_cuseeme	CU-See-Me
ip_masq_vdolive	For VDO Live
ip_masq_quake	IdSoftware's Quake

Aby uruchomić te moduły, należy ręcznie je załadować za pomocą polecenia *insmod*. Należy zauważyć, że moduły te nie mogą być ładowane przez demona *kerneld*. Każdy z modułów przyjmuje argument, który określa, na jakich portach moduł będzie nasłuchiwał. Moduł RealAudio mógłbyś załadować w następująco.

```
# insmod ip_masq_raudio.o ports=7070,7071,7072
```

Numery portów zależą od protokołu. Dokument mini-HOWTO o maskowaniu IP autorstwa Ambrose Au, podaje więcej szczegółów na temat modułów maskowania IP i omawia, jak je konfigurować.
Pakiet *netfilter* zawiera moduły działające podobnie. Na przykład, aby udostępnić połączenie śledzące sesje FTP, należy załadować moduły *ip_conntrack_ftp* i *ip_nat_ftp.o* i użyć ich.

7.3. Konfigurowanie maskowania IP

Do konfigurowania reguł maskowania IP są używane również polecenia *ipfwadm*, *ipchains* i *iptables*. Reguły maskowania są szczególną klasą reguł filtrujących. Możliwe jest maskowanie jedynie tych datagramów, które są odbierane na jednym interfejsie i rutowane do innego interfejsu. W celu utworzenia reguły maskowania należy skonstruować regułę bardzo podobną do reguły przekazywania dla firewalla, ale z wykorzystaniem specjalnych opcji, które mówią jądru, by maskowało datagram. Polecenie *ipfwadm*

wykorzystuje opcję *–m*, *ipchains* opcję -j MASQ, a *iptables* opcję -j MASQUERADE, by pokazać, że datagram pasujący do reguły powinien być zamaskowany.

Spójrzmy na przykład.

Student informatyki z AGH ma w domu kilka komputerów połączonych w sieć Ethernet. Zdecydował się na użycie jednego z prywatnych, zarezerwowanych adresów sieci. Mieszka razem z innymi studentami, którzy też chcą mieć dostęp do Internetu. Ponieważ ich warunki życiowe są dość skromne, nie mogą pozwolić sobie na stałe połączenie z Internetem, a więc używają zwykłego, komutowanego połączenia PPP. Wszyscy chcieliby współdzielić łącze by pooglądać strony WWW czy pobrać pliki przez FTP bezpośrednio na swoje komputery – maskowanie IP jest tu dobrym rozwiązaniem.
Student konfiguruje najpierw komputer z Linuksem obsługujący łącze komutowane i działający jako ruter dla sieci LAN. Adres IP, jakiego używa przy dzwonieniu, nie jest istotny. Konfiguruje ruter z maskowaniem IP i używa jednego z adresów sieci prywatnej dla swojej sieci LAN: 192.168.1.0. Zapewnia każdemu hostowi w sieci LAN ustawienie domyślnego rutingu tak, by wskazywał na ruter linuksowy. Poniższe polecenia *ipfwadm* wystarczą, by maskowanie zadziałałowtakiej konfiguracji:

```
# ipfwadm -F -p deny
# ipfwadm -F -a accept -m -S 192.168.1.0/24 -D 0/0
```

lub w przypadku *ipchains*:

```
# ipchains -P forward -j deny
# ipchains -A forward -s 192.168.1.0/24 -d 0/0 -j MASQ
```

lub w przypadku *iptables*:

```
# iptables -t nat -P POSTROUTING DROP
# iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

Teraz, gdy tylko jakiś z hostów z sieci LAN spróbuje połączyć się z usługą na hoście zdalnym, jego datagramy zostaną automatycznie zamaskowane przez ruter z Linuksem. Pierwsza reguła w każdym z przykładów zapobiega przed rutowaniem wszelkich innych datagramów i zapewnia pewne bezpieczeństwo. Aby zobaczyć właśnie utworzoną listę reguł maskowania, należy użyć argumentu –l w poleceniu *ipfwadm*.

Aby zobaczyć listę reguł maskowania w przypadku polecenia *ipchains*, należy użyćargument *–L*. Wszelkie reguły, w których pole target ma wartość MASQ, to reguły maskowania. No i aby zobaczyć reguły za pomocą *iptables*, należy użyć:

```
# iptables -t nat -L
```

7.3.1. Ustawianie parametrów czasowych dla maskowania IP

Gdy jest nawiązywane każde nowe połączenie, oprogramowanie maskowania IP tworzy w pamięci powiązanie pomiędzy każdym z hostów w nim uczestniczących. Możliwe jest obejrzenie tych powiązania w dowolnej chwili, zaglądając do pliku */proc/net/ip_masquerade*. Powiązania wygasną po pewnym czasie nieaktywności. Za pomocą polecenia *ipfwadm* można ustawić wartości czasu ich wygaśnięcia. Ogólna składnia jest następująca:

ipfwadm -M -s <*tcp*> <*tcpfin*> <*udp*>

a dla polecenia *ipchains*:

ipchains -M -S <*tcp*> <*tcpfin*> <*udp*>

Implementacja *iptables* wykorzystuje liczniki czasowe o dużo większej wartości i nie pozwala na ich ustawianie. Każda z tych wartości określa licznik czasu używany przez oprogramowanie maskowania IP i jest wyrażona w sekundach.

7.4. Obsługiwanie przeszukiwania serwerów DNS

Obsługiwanie przeszukiwania serwerów nazw z hostów w sieci z maskowaniem IP zawsze stanowiło problem. Istnieją dwa sposoby na dostosowanie DNS-u do środowiska z maskowaniem. Należy powiedzieć każdemu z hostów, żeby używał tego samego DNS-u co ruter i niech maskowanie IP obsługuje ich zapytania DNS. Lub też należy uruchomić linuksowy serwer pamięci podręcznej i ustawić go tak, by każdy z hostów używał go jako swojego DNS-u. Jest to bardziej agresywne działanie, ale o tyle korzystniejsze, że redukuje rozmiar ruchu DNS przesyłanego do łącza internetowego i będzie nieco szybsze, ponieważ większość zapytań będzie obsługiwana z pamięci podręcznej. Wadą tej konfiguracji jest to, że jest bardziej skomplikowana.

7.5. Więcej na temat translacji adresów sieciowych

Oprogramowanie *netfilter* jest w stanie obsłużyć wiele różnych typów translacji adresów sieciowych. Maskowanie IP jest jednym z prostszych jego zastosowań. Możliwe jest na przykład stworzenie takich reguł translacji, które będą tłumaczyć tylko pewne adresy lub zakresy adresów, a reszta pozostanie nietknięta lub będą tłumaczyć adresy na pule adresów, a nie na pojedyncze adresy, tak jak w maskowaniu. W praktyce można użyć polecenia *iptables* do stworzenia reguł translacji, które odwzorowują wszystko, wykorzystując połączenie różnych atrybutów, takich jak adres źródłowy, adres docelowy, typ protokołu, numer portu, itp.

W dokumentacji *netfilter* tłumaczenie adresu źródłowego datagramu nazywa się SNAT (Source NAT). Tłumaczenie adresu docelowego datagramu jest nazywane DNAT (Destination NAT). Tłumaczenie portu TCP lub UDP jest znane pod pojęciem REDIRECT. SNAT, DNAT i REDIRECT to cele, które można używać w poleceniu *iptables* do tworzenia bardziej wyrafinowanych i skomplikowanych reguł. Więcej o tym, jak używać translacji adresów sieciowych można dowiedzieć się z *IPTABLES-HOWTO*.

8. Serwer Proxy

Serwer proxy zachowuje się jako „pośrednik" pomiędzy programem klienckim a serwerem znajdującym się gdzieś w Internecie. Program kliencki nawiązuje połączenie z serwerem proxy zamiast bezpośrednio z serwerem. Serwer proxy odbiera zapytanie od klienta i decyduje czy zapytanie ma zostać przepuszczone czy też należy go odrzucić. Jeżeli zapytanie jest dozwolone wówczas serwer proxy komunikuje się z „rzeczywistym" serwerem i przekazuje otrzymane od klienta zapytanie, a następnie pobiera od serwera odpowiedz przekazując ją następnie do klienta.

Mechanizm proxy nie wymaga żadnego specjalnego hardware'u, natomiast wymagane jest specjalne oprogramowanie dla większości serwisów.

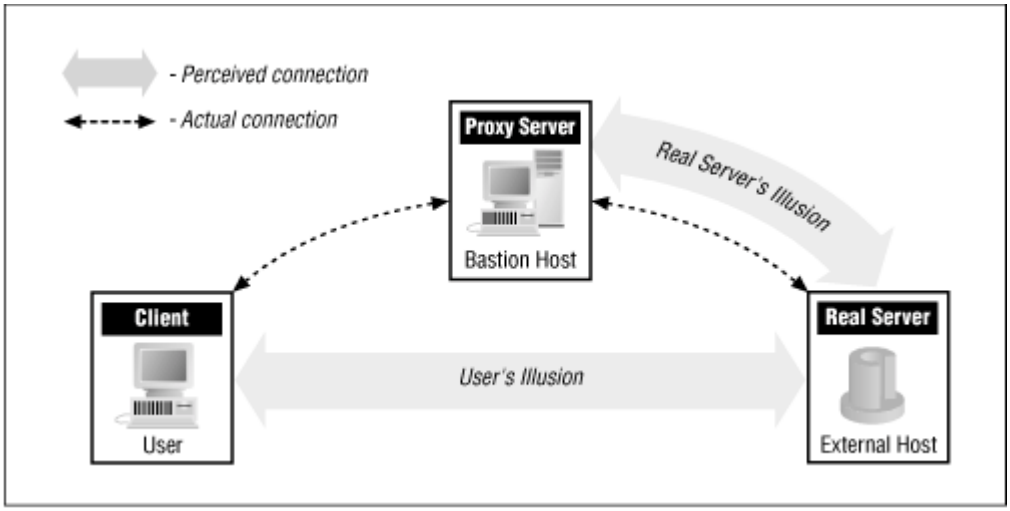
Notka: Systemy proxy są efektywne jedynie wówczas, gdy są wykorzystywane w połączeniu z pewnymi metodami ograniczającymi ruch na poziomie warstwy IP (ang. *IP-level traffic*) pomiędzy klientami a rzeczywistymi serwerami, takimi jak np. rutery ekranujące czy też tzw. hosty dual-homed, nie dokonujące rutowania pakietów. Jeżeli mamy do czynienia z połączeniem pomiędzy klientem a rzeczywistym serwerem na poziomie warstwy IP, klient jest w stanie ominąć system proxy (przypuszczalnie może to zrobić także ktoś z zewnątrz)

8.1. Dlaczego systemy proxy?

Jak wiadomo bezsensownym staje się podłączenie do Internetu, jeżeli użytkownicy nie będą mogli mieć do niego dostępu. Z drugiej strony nie jest bezpieczne jeżeli umożliwimy wolny dostęp dla wszystkich użytkowników. Dlatego musi zostać zastosowany pewien kompromis.

Najczęściej stosowaną metodą jest dostarczenie dla wszystkich użytkowników pojedynczego hosta z dostępem do Internetu. Jakkolwiek nie jest to satysfakcjonujące rozwiązanie ponieważ hosty te nie są transparentne dla użytkowników. Użytkownicy pragnący uzyskać dostęp do serwisów sieciowych nie mogą tego dokonać w sposób bezpośredni. Wymagane jest zalogowanie się do hosta dual-homed, i wykonanie całej pracy z tamtąd, a następnie przesłanie rezultatów z powrotem na własną stację roboczą. Jak wiadomo jest to bardzo uciążliwe, zwłaszcza w przypadku wykorzystywania wielu systemów operacyjnych (np. Linux, Unix na hostach dual-homed, oraz Windows, Macintosh na maszynach klienckich).

Dobrym rozwiązaniem jest zatem konfigurowanie hostów dual-homed wraz z systemem proxy. Zastosowanie systemu proxy powoduje że użytkownik odnosi wrażenie bezpośredniej pracy z serwerem w Internecie (Patrz rysynek poniżej).



Rysunek 10. Zasada działania serwera proxy

Systemy proxy dodatkowo przezwyciężają problemy bezpieczeństwa pracy użytkowników na hostach dual-homed związane z koniecznością logowania się. Połączenia dokonywane są zatem poprzez wymuszone oprogramowanie. Ponieważ oprogramowanie proxy pracuje bez konieczności zalogowanie użytkownika, host jest lepiej zabezpieczony. Dodatkowo poxy zachowuje się jako punk kontrolny, ponieważ nie jest możliwa instalacja niekontrolowanego oprogramowania.

8.1.1. Zalety stosowania serwisów proxy.

- Pozwalają użytkownikom na „bezpośredni" dostęp do Internetu
  Serwisy proxy pozwalają użytkownikowi na dostęp do serwisów sieciowych z własnego systemu, jednakże bez możliwości bezpośredniego przekazywania pakietów pomiędzy użytkownikiem a Internetem. Droga pakietów nie jest bezpośrednia, a uczestniczą w niej hosty dual-homed lub też kombinacja ruterów ekranujących i tzw. hostów bastion,

- Lepsze własności logowania
  Serwisy proxy pozwalają na dokonywanie logowanie na kilka efektywnych sposobów, przez co mamy do czynienia z mniejszą ilością bardziej użytecznych logów.

8.1.2. Wady serwisów proxy

- Serwisy proxy pozostają w tyle za innymi serwisami
  Pomimo iż oprogramowanie proxy jest szeroko dostępne dla starszych i prostszych protokołów (FTP, Telnet) oprogramowanie dla nowych lub też rzadko używanych protokołów jest często trudne do znalezienia. Zazwyczaj mamy do czynienia z pewnym opóźnieniem pomiędzy pojawieniem się nowego serwisu a dostępnością serwerów proxy dla niego.

- Serwisy proxy mogą wymagać różnych serwerów dla każdego z serwisów
  Możliwa jest potrzeba posiadania różnych serwerów dla poszczególnych protokołów.

- Serwisy proxy zazwyczaj wymagają wprowadzenia modyfikacji do klientów, procedur.
  Z powodu wprowadzanych modyfikacji aplikacje takie często nie pracują tak jak zwykłe aplikacje (ang. *nonproxied applications*).

- Serwisy proxy nie nadają się do zastosowania z niektórymi serwisami

- Serwisy proxy nie chronią przed wszystkimi „słabościami" protokołów.
  Jako rozwiązanie bezpieczeństwa, proxing bazuje na zdolności określania które operacje w protokole są bezpieczne. Niestety nie wszystkie protokoły dostarczają prostego sposobu zrobienia tego.

8.2. Typy serwerów proxy

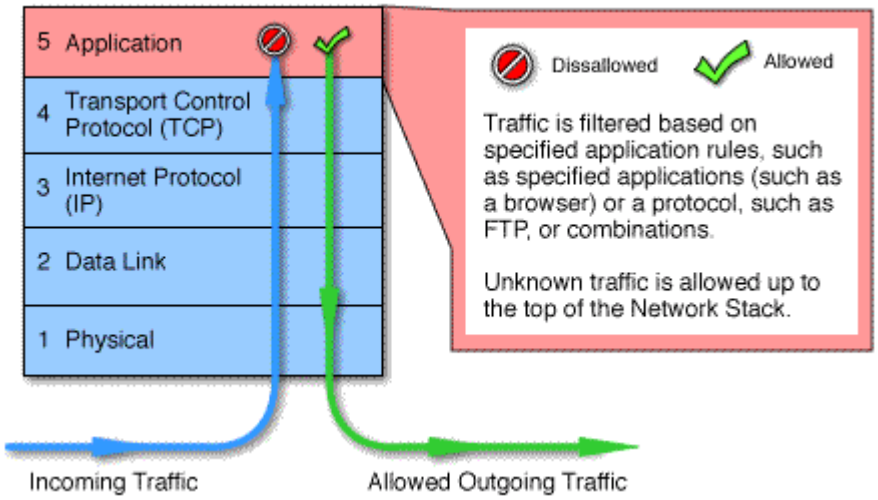
8.2.1. Application-Level Proxy

Application-level proxy to takie proxy, które wie coś o konkretnej aplikacji, dla której świadczy usługę, jak również rozumie i interpretuje komendy w protokole aplikacji.
Generalnie proxy application-level używają zmodyfikowanych procedur celem uzyskania korzyści z wiedzy na temat obsługiwanego przez nie protokołu aplikacji. Proxy tego rodzaju są w stanie pobierać potrzebne informacje z obsługiwanego przez nie protokołu aplikacji.
Proxy tego rodzaju dostarczają standardowej funkcjonalności (sprawdzanie adresów IP) jak również zawansowanej analizy pakietów przechodzących. Przychodzące pakietu są oceniane w oparciu o

zdefiniowaną polityką bezpieczeństwa. Pakiety mogą wejść do sieci wewnętrznej tylko w tedy, gdy są zgodne z wymaganą polityką bezpieczeństwa.
Typowe proxy tego rodzaju mogą dostarczać usług dla aplikacji i protokołów takich jak Telnet, FTP, HTTP, SMTP. Należy zauważyć że odpowiednie proxy muszą zostać zainstalowane dla każdego serwisu warstwy aplikacji.

Zasada działania przedstawia się następująco:
W momencie gdy użytkownik inicjalizuje połączenie z proxy, wówczas proxy dokonuje otwarcia nowego połączenia (innej sesji) do zewnętrznej sieci, tak więc każda aktualna sesja posiada drugą automatycznie generowaną. Pozwala to na pełną kontrolę ruchu odbywającego się na serwerze. Definiowane przez administratorów polityki bezpieczeństwa pozwalają na bardzo elastyczne definiowanie zasad określających obsługę przechodzących pakietów.

Poniższy rysunek przedstawia ideę działania tego rodzaju proxy.



Rysunek 11. Application-Level Proxy

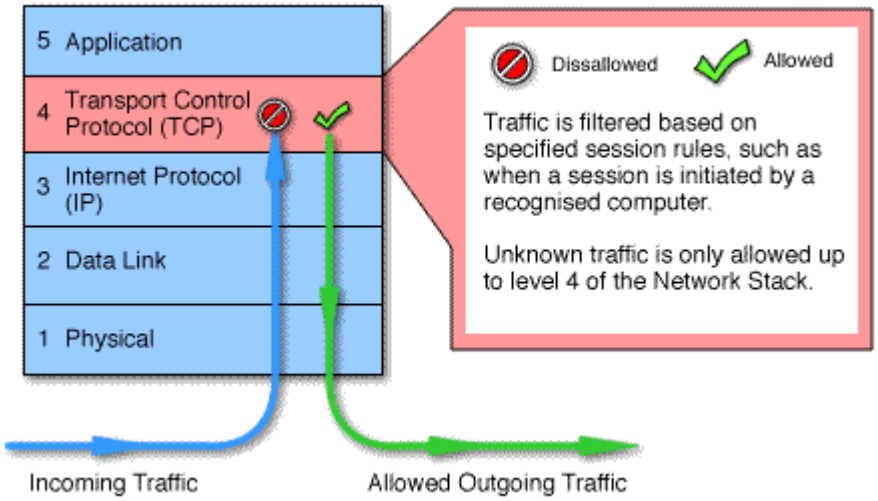
8.2.2. Circuit-Level Proxy

Circuit-level proxy to takie proxy, które tworzy połączenie pomiędzy klientem a serwerem bez interpretacji protokołu aplikacji. Przykładem takiego proxy mogą być obecnie stosowane hybrydowe proxy gateways, które wyglądają od zewnątrz jak proxy, natomiast od środka jak rutery filtrujące. Ponieważ proxy tego rodzaju nie potrafią interpretować protokołu aplikacji potrzebują zazwyczaj zmodyfikowanych klientów, celem uzyskania potrzebnych informacji.

Zaletą takich proxy jest fakt iż dostarczają one usług dla różnych protokołów. Większość serwerów proxy tego typu może zostać przystosowana do obsługi prawie dowolnego protokołu. Jakkolwiek nie każdy protokół może być w łatwy sposób obsługiwany przez tego rodzaju proxy. Wadą tego rodzaju serwerów proxy jest fakt iż dostarczają one bardzo małą kontrolę tego co w takim proxy się dzieje. Podobnie jak filtry pakietów dokonują one jedynie kontroli połączenia na poziomie podstawowym, tzn. adresów źródłowych i docelowych i nie mogą w żaden sposób określać czy komendy przechodzące przez nie są bezpieczne, czy też zgodne z wymaganym protokołem.

Zasada działania tego rodzaju proxy serwerów jest bardzo prosta. Zapytanie internetowe „wędruje" do serwera proxy. Serwer proxy przekazuje zapytanie do Internetu po ty jak dokona zmiany adresu IP. Zewnętrzni użytkownicy widzą zatem jedynie adres IP serwera proxy. Odpowiedź jest pobierana przez proxy i przesyłana z powrotem do klienta. Tym sposobem zewnętrzni użytkownicy nie są w stanie uzyskać dostępu do wewnętrznego systemu.

Poniższy rysunek przedstawia ideę działania tego rodzaju proxy.



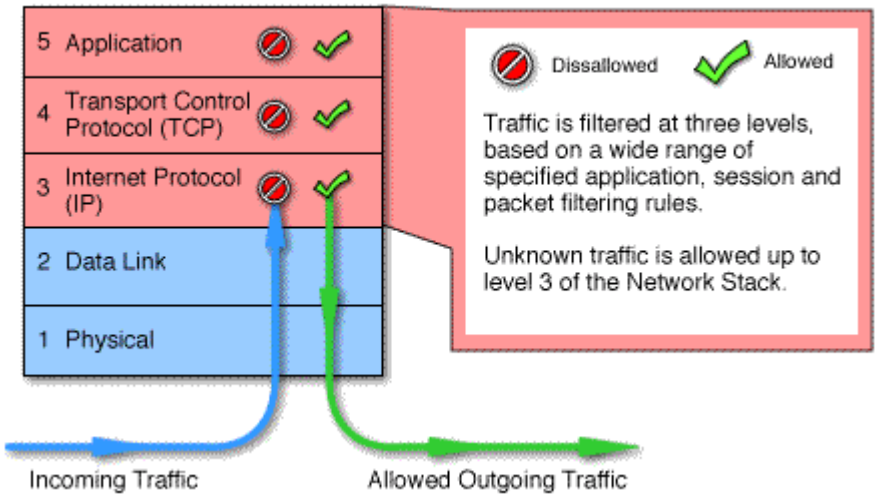
Rysunek 12. Circuit-Level Proxy

8.2.3. Stateful Inspection

Jednym z rodzajów firewall'i są tzw. *stateful multi-layer inspection firewalls*, łączące w sobie trzy typy firewall'i. Dokonują one filtracji pakietów na poziomie warstwy sieciowej, określają czy pakiety sesyjne są legalne, jak również dokonują sprawdzania zawartości pakietów w warstwie aplikacji.
Firewalle tego rodzaju dostarczają wysokiego poziomu bezpieczeństwa, wysokiej wydajności jak również są transparentne dla użytkowników końcowych. Firewalle stateful inspection są kosztowne oraz trudne w

implementacji dlatego też mogą być mniej bezpieczne w porównaniu z prostszymi rozwiązaniami, w przypadku, gdy nie zostaną poprawnie zaimplementowane.
Początkowa analiza jest dokonywana na trzecim poziomie modelu OSI, lecz w odróżnieniu od application-level proxy, używają one zasad biznesowych definiowanych przez użytkowników zamiast używania predefiniowanych informacji aplikacyjnych.

Poniższy rysunek przedstawia ideę działania tego rodzaju firewall'i:



Rysunek 13. Stateful Inspection

8.2.4. Ogólne i dedykowane proxy

Pomimo iż często używane są określenia „application-level" oraz „circuit-level" proxy, istnieje także inny podział serwerów proxy, a mianowicie na ogólne i dedykowane. Proxy dedykowane to takie, które dokonują obsługi pojedynczego protokołu, natomiast ogólne to takie które obsługują wiele protokołów. W praktyce zazwyczaj proxy dedykowane to application-level proxy, natomiast ogólne to circuit-level.

8.3. Przykłady konkretnych rozwiązań

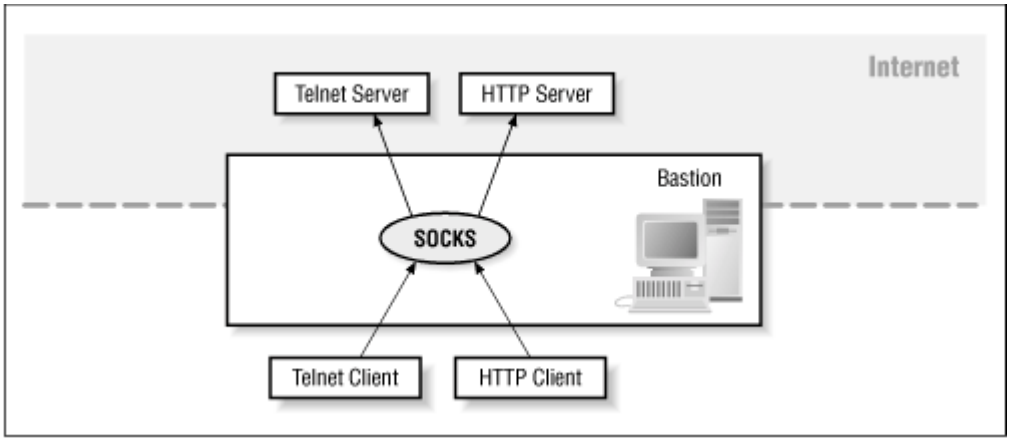
Aby postawić serwer proxy potrzebujemy jednego z niżej wymienionych pakietów:

- SOCKS
- TIS Firewall Toolkit (FWTK)

8.3.1. Użycie SOCKS

SOCKS są bardzo ogólne, dlatego też są aż tak popularne. Istnieje jednak kilka wad o których w dalszej części rozdziału.
Po pierwsze SOCKS dostarczają mechanizmu logowania jednakże większość logów jest tworzona po stronie klienta, co znacznie utrudnia gromadzenie informacji w jednym miejscu w celu późniejszej ich analizy. Kolejną wadą SOCKS jest to, iż mogą współpracować jedynie z klientami bazującymi na protokole TCP, nie wspierają natomiast protokołu UDP (jak np. klienci Archie). W przypadku wykorzystywania klientów bazujących na protokole UDP wymagany jest inny pakiet, np. UDP Packet Relayer dostarczający podobnej funkcjonalności jak SOCKS.

Poniższy rysunek prezentuje idee działania SOCKS.



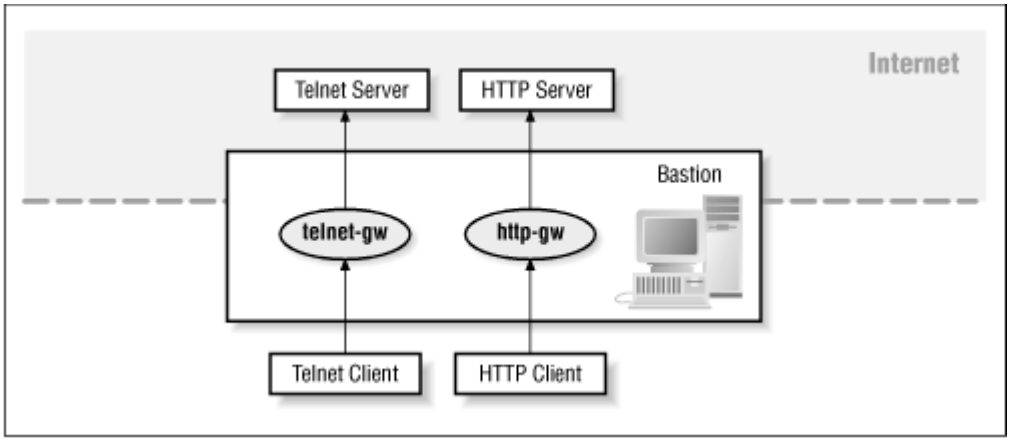
Rysunek 14. Idea działania SOCKS

8.3.2. Użycie TIS Firewall Toolkit

Trusted Information System (http://www.tis.com) jest fragmentem kolekcji programów zaprojektowanych dla firewalli. Program ten udostępnia podobne rzeczy jak SOCK, ale jest zbudowany na innych zasadach. Socks posiada jeden program przeprowadzający wszystkie transakcje z Internetem, TIS natomiast dostarcza jednego programu dla każdego z narzędzi które chcemy użyć w firewall'u.
Dla pokazania kontrastu użyjmy przykładów WWW i dostępu za pomocą telnetu. Używając SOCKS, należy ustawić jeden plik konfiguracyjny i jednego demona. Używając tego pliku tak telnet jak i WWW są dostępne, podobnie jak inne usługi które nie zostały zakazane.
W pakiecie TIS dokonuje się ustawienia jednego demona dla (osobno) WWW i telnetu z osobnymi plikami konfiguracyjnymi. Po zrobieniu tego inne usługi internetowe są zakazane dopóki ich explicite nie ustawimy.

Jeśli demon dla specyficznych usług jest niedostępny (tak jak talk), istnieją ,,plug-in-y" dla demona, ale nie tak elastyczne i łatwe w konfiguracji jak inne narzędzia.

Poniższy rysunek prezentuje idę działania TIS Firewall Toolkit



Rysunek 15. Idea działania TIS Firewall Toolkit

Różnica wygląda na niewielką, jest jednak istotna. Przy kiepsko ustawionym SOCKS serwerze ktoś z wewnątrz może uzyskać większy dostęp do Internetu niż było początkowo planowane. Z pakietem TIS użytkownicy wewnątrz sieci mają jedynie taki dostęp na jaki zezwolił administrator.
SOCKS są łatwiejszy do konfiguracji, łatwiejszy do kompilacji i pozwala na większą elastyczność. TIS jest bardziej bezpieczny, w przypadku konfiguracji użytkowników wewnątrz chronionej sieci. Oba dostarczają całkowitego bezpieczeństwa z zewnątrz.

9. Propozycja przykładowej konfiguracji firewalli (zewnętrznego i wewn€trznego)

Zaproponowana konfiguracja firewalli przeznaczona jest dla następującego przypadku:

- firewall zewnętrzny posiada dwa interfejsy sieciowe: jeden połączony z Internetem, a drugi z siecią graniczną
- publiczne usługi sieciowe (www+smtp+ftp+news) udostępniane są przez maszyny umieszczone w sieci granicznej
- prywatne komputery znajdują się w sieci wewnętrznej i są chronione przez firewall wewnętrzny, który posiada dwa interfejsy: jeden łączacy z siecią prywatną, drugi z siecią graniczną
- komputery sieci granicznej jak i lokalnej za firewallem wewnętrznym są umieszczone za maskaradą IP i posiadają adresy z puli prywatnej klasy C, publicznym adresem dysponuje maszyna, na której postawiony jest firewall zewnętrzny.

Jeśli zawiedzie firewall zewnętrzny, publiczne serwery sieci granicznej będą narażone na atak. Firewall wewnętrzny zabezpiecza sieć wewnętrzną przed zdobytym firewallem zewnętrznym jak i innymi komputerami sieci granicznej.

Stałe symboliczne dla konfiguracji firewalla zewnętrznego:

```
EXTERNAL_INTERFACE=”eth0”        #interfejs firewalla zewnętrznego połączony z Internetem
LOOPBACK_INTERFACE=”lo”          #nazwa interfejsu pętli zwrotnej

EXT_FIR_INT_INTERFACE=”eth1”     #wewnętrzny interfejs firewalla zewnętrznego
EXT_FIR_INT_IPADDR=”192.168.1.1” #wewnętrzny adres firewalla zewnętrznego
INT_FIR_EXT_IPADDR=”192.168.1.2” #zewnętrzny adres zapory dławiącej
DMZ_ADDRESSES=”192.168.1.0/24    #nasz (prywatny) zakres
LAN_ADDRESSES=”192.168.5.1/24”   #nasz (prywatny) zakres
CLASS_A=”10.0.0.0/8”             #sieci prywatne klasy A
CLASS_B=”172.16.0.0/12”          #sieci prywatne klasy B
CLASS_C=”192.168.0.0/16”         #sieci prywatne klasy C
PRIVPORTS=”0:1023”               #zakres uprzywilejowanych portów
UNPRIVPORTS=”1024:65535”         #zakres nieuprzywilejowanych portów
IPADDR=”mój.adres.ip”            #adres pojedynczego dowolnego komputera z sieci granicznej i lokalnej
```

Stałe symboliczne dla konfiguracji firewalla wewnętrznego:

```
INT_FIR_EXT_INTERFACE=”eth0”       #interfejs zewnętrzny firewalla zewnętrznego
INT_FIR_INT_INTERFACE=”eth1”       #interfejs wewnętrzny firewalla wewnętrznego
INT_FIR_EXT_IPADDR=”192.168.1.2”  #adres zewnętrznego interfejsu firewalla wewnętrznego
EXT_FIR_INT_IPADDR=”192.168.1.1”  #adres wewnętrznego interfejsu firewalla zewnętrznego
DMZ_ADDRESSES=”192.168.1.0/24”     #nasz (prywatny)zakres dla sieci granicznej
LAN_IPADDR=”192.168.5.1”           #nasz (prywatny) zakres dla sieci lokalnej
ANYWHERE=”any/0”                   #dowolny adres IP
LOOPBACK=”127.0.0.0/8”             #zarezerwowany zakres adresów pętli
CLASS_A=”10.0.0.0/8”               #sieci prywatne klasy A
CLASS_B=”172.16.0.0/12”            #sieci prywatne klasy B
CLASS_C=”192.168.0.0/16”           #sieci prywatne klasy C
PRIVPORTS=”0:1023”                 #zakres uprzywilejowanych portów
UNPRIVPORTS=”1024:65535”           #zakres nieuprzywilejowanych portów
```

Konfiguracja firewalla zewnętrznego:

```
#usuwamy istniejące reguły ze wszystkich łańcuchów
ipchains –F

#ustawiamy domyślną politykę porzucania pakietów
ipchains –P input DENY
ipchains –P output REJECT
ipchains –P forward REJECT

#maskarada dla sieci granicznej i lokalnej za firewallem wewnętrznym
ipchains –A forward –i $EXTERNAL_INTERFACE –s $DMZ_ADDRESSES –j MASQ
ipchains –A forward –i $ EXTERNAL_INTERFACE –s $LAN_ADDRESSES –j MASQ

#nieograniczony ruch przez interfejs pętli zwrotnej
ipchains –A input –i $LOOPBACK_INTERFACE –j ACCEPT
ipchains –A output –i $LOOPBACK_INTERFACE –j ACCEPT

#blokujemy fałszywe pakiety wysłane rzekomo spod adresu IP zewnętrznego
ipchains –A input –i $EXTERNAL_INTERFACE –s $IPADDR –j DENY –l

#blokujemy pakiety rzekomo pochodzące z prywatnej sieci klasy A
ipchains –A input –i $EXTERNAL_INTERFACE –s $CLASS_A –j DENY
ipchains –A input –i $EXTERNAL_INTERFACE –d $CLASS_A –j DENY
```

```
ipchains –A output –i $EXTERNAL_INTERFACE –s $CLASS_A –j DENY -l
ipchains –A output –i $EXTERNAL_INTERFACE –d $CLASS_A –j DENY -l

#blokujemy pakiety rzekomo pochodzące z prywatnej sieci klasy B
ipchains –A input –i $EXTERNAL_INTERFACE –s $CLASS_B –j DENY
ipchains –A input –i $EXTERNAL_INTERFACE –d $CLASS_B –j DENY
ipchains –A output –i $EXTERNAL_INTERFACE –s $CLASS_B –j DENY -l
ipchains –A output –i $EXTERNAL_INTERFACE –d $CLASS_B –j DENY -l

#blokujemy pakiety rzekomo pochodzące z prywatnej sieci klasy C
ipchains –A input –i $EXTERNAL_INTERFACE –s $CLASS_C –j DENY
ipchains –A input –i $EXTERNAL_INTERFACE –d $CLASS_C –j DENY
ipchains –A output –i $EXTERNAL_INTERFACE –s $CLASS_C –j DENY -l
ipchains –A output –i $EXTERNAL_INTERFACE –d $CLASS_C –j DENY -l

#blokujemy pakiety rzekomo pochodzące z interfejsu pętli zwrotnej
ipchains –A input –i $EXTERNAL_INTERFACE –s $LOOPBACK –j DENY -l
ipchains –A output –i $EXTERNAL_INTERFACE –s $LOOPBACK –j DENY -l

#serwer DNS w firewallu zewnętrznym nasłuchuje na porcie 53 żądań równorzędnego serwera DNS z firewalla
#wewnętrznego
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p udp –s $INT_FIR_EXT_IPADDR 53 \
–d $EXT_FIR_INT_IPADDR 53 –j ACCEPT
ipchains –A output –i $EXT_FIR_INT_INTERFACE –p udp –s $EXT_FIR_INT_IPADDR 53 \
–d $INT_FIR_EXT_IPADDR 53 –j ACCEPT

#lokalne zapytania DNS od klientów do serwera DNS w firewallu wewnętrznym
ipchains –A output –i $EXT_FIR_INT_INTERFACE –p udp \
–s $EXT_FIR_INT_IPADDR $UNPRIVPORTS -d $INT_FIR_EXT_IPADDR 53 –j ACCEPT
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p udp \
–s $INT_FIR_EXT_IPADDR 53 –d $EXT_FIR_INT_IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A output –i $EXT_FIR_INT_INTERFACE –p tcp \
–s $EXT_FIR_INT_IPADDR $UNPRIVPORTS -d $INT_FIR_EXT_IPADDR 53 –j ACCEPT
ipchains –A input –i $EXT_FIR_INT_INTERFACE ! –y –p tcp \
–s $INT_FIR_EXT_IPADDR 53 –d $EXT_FIR_INT_IPADDR $UNPRIVPORTS –j ACCEPT

#przekazywanie zapytań lokalnych do serwera nazw dostawcy usług internetowych
ipchains –A output –i $EXTERNAL_INTERFACE –p udp\
-s $IPADDR 53 –d $NAME_SERVER 53 –j ACCEPT
ipchains –A input –i $EXTERNAL_INTERFACE –p udp\
```

```
-s  $NAME_SERVER 53 –d $IPADDR 53–j ACCEPT

#lokalne zapytania DNS od klientów do serwera
ipchains –A output –i $EXTERNAL_INTERFACE –p udp \
-s $IPADDR $UNPRIVPORTS –d $ANYWHERE 53 –j ACCEPT
ipchains –A input –i $EXTERNAL_INTERFACE –p udp \
-s $ANYWHERE 53 –d $IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A output –i $EXTERNAL_INTERFACE –p tcp \
-s $IPADDR $UNPRIVPORTS –d $ANYWHERE 53 –j ACCEPT
ipchains –A input –i $EXTERNAL_INTERFACE ! –y –p tcp \
-s $ANYWHERE 53 –d $IPADDR $UNPRIVPORTS –j ACCEPT

#lokalny serwer DNS do zdalnego klienta
ipchains –A input –i $EXTERNAL_INTERFACE –p udp \
-s $ANYWHERE $UNPRIVPORTS –d $IPADDR 53 –j ACCEPT
ipchains –A output –i $EXTERNAL_INTERFACE –p udp \
-s $IPADDR 53 –d $ANYWHERE $UNPRIVPORTS –j ACCEPT
ipchains –A input –i $EXTERNAL_INTERFACE –p tcp \
-s $ANYWHERE $UNPRIVPORTS –d $IPADDR 53 –j ACCEPT
ipchains –A output –i $EXTERNAL_INTERFACE ! –y –p tcp \
-s $IPADDR 53 –d $ANYWHERE $UNPRIVPORTS –j ACCEPT

#reguły pozwalajace lokalnym hostom na łączenie się z serwerami WWW w hostach zdalnych, a zdalnym #klientom z serwerem WWW w sieci granicznej
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp \
-s $DMZ_ADDRESSES $UNPRIVPORTS –d $ANYWHERE 80 –j ACCEPT
ipchains –A output –i $EXT_FIR_INT_INTERFACE –p tcp ! –y \
-s $ANYWHERE 80 -d $DMZ_ADDRESSES $UNPRIVPORTS –j ACCEPT
ipchains –A output –i $EXT_FIR_INT_INTERFACE –p tcp \
-s $ANYWHERE $UNPRIVPORTS –d $WEB_SERVER_DMZ_ADDRESS 80 –j ACCEPT
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp ! –y \
-s $WEB_SERVER_DMZ_ADDRESS 80 –d $ANYWHERE $UNPRIVPORTS –j ACCEPT

#łączenie się przez klientów z portem NNTP serwera newsów w strefie granicznej
NEWS_SERVER_DMZ_IPADDR=”192.168.1.6”
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp \
-s <nasze.klienty.grup.dyskusyjnych> $UNPRIVPORTS \
–d $NEWS_SERVER_DMZ_IPADDR 119 –j ACCEPT
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp ! -y\
-s $NEWS_SERVER_DMZ_IPADDR 119
```

```
-d  <nasze.klienty.grup.dyskusyjnych> $UNPRIVPORTS –j ACCEPT

#moduł przekazywania portów
ipmasqadm portfw –a –P tcp –L <nasze.klienty.grup.dyskusyjnych> 119 \
-R $NEWS_SERVER_DMZ_IPADDR 119

#dostęp do zdalnego serwera grup
NEWS_FEED=”<nasz.zdalny.serwer.grup>”
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp \
-s $NEWS_SERVER_DMZ_IPADDR  $UNPRIVPORTS \
–d $NEWS_FEED 119 –j ACCEPT
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp ! -y\
-s $NEWS_FEED 119
-d  $NEWS_SERVER_DMZ_IPADDR $UNPRIVPORTS –j ACCEPT

MAIL_SERVER_DMZ_IPADDR=”192.168.1.8”
#zezwolenie na przychodzące połączenia pocztowe
ipchains –A input –i $EXTERNAL_INTERFACE –p tcp \
-s $ANYWHERE $UNPRIVPORTS –d $IPADDR 25 –j ACCEPT
ipchains –A output –i $EXTERNAL_INTERFACE –p tcp ! -y\
-s $IPADDR 25 -d $ANYWHERE $UNPRIVPORTS –j ACCEPT
ipchains –A output –i $EXT_FIR_INT_INTERFACE –p tcp \
-s $ANYWHERE $UNPRIVPORTS –d $MAIL_SERVER_DMZ_IPADDR 25 –j ACCEPT
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp ! –y \
-s $MAIL_SERVER_DMZ_IPADDR 25 –d $ANYWHERE $UNPRIVPORTS –j ACCEPT
ipmasqadm portfw –a –P tcp –L $IPADDR 25 –R $MAIL_SERVER_DMZ_IPADDR 25

#pozwolenie serwerowi pocztowemu w sieci granicznej na przekazywanie poczty do zdalnych miejsc #przeznaczenia
ipchains –A input –i $EXT_FIR_INT_INTERFACE –p tcp \
-s $MAIL_SERVER_DMZ_IPADDR $UNPRIVPORTS -d $ANYWHERE 25 –j ACCEPT
ipchains –A output –i $EXT_FIR_INT_INTERFACE –p tcp ! –y \
-s $ANYWHERE 25 -d  $MAIL_SERVER_DMZ_IPADDR $UNPRIVPORTS –j ACCEPT
ipchains –A output –i $EXTERNAL_INTERFACE –p tcp \
-s $IPADDR $UNPRIVPORTS –d $ANYWHERE 25 –j ACCEPT
ipchains –A output –i $EXTERNAL_INTERFACE –p tcp ! -y\
-s $ANYWHERE 25 –d $IPADDR $UNPRIVPORTS –j ACCEPT
```

Konfiguracja firewalla wewnętrznego:

```
#usuwamy istniejące reguły ze wszystkich łańcuchów
ipchains –F

#ustawiamy domyślną politykę porzucania pakietów
ipchains –P input REJECT
ipchains –P output REJECT
ipchains –P forward REJECT

#nieograniczony ruch przez interfejs pętli zwrotnej
ipchains –A input –i $LOOPBACK_INTERFACE –j ACCEPT
ipchains –A output –i $LOOPBACK_INTERFACE –j ACCEPT

#odrzucamy sfałszowane pakiety rzekomo pochodzące z interfejsów firewalla wewnętrznego
ipchains –A input –i $INT_FIR_EXT_INTERFACE –s $INT_FIR_EXT_IPADDR –j REJECT –l
ipchains –A input –i $INT_FIR_INT_INTERFACE –s $LAN_IPADDR –j REJECT –l

#odrzucamy pakiety do (lub z) sieci prywatnych klasy A
ipchains –A input –i $INT_FIR_EXT_INTERFACE –s $CLASS_A –j REJECT -l
ipchains –A output –i $INT_FIR_EXT_INTERFACE –d $CLASS_A –j REJECT –l
ipchains –A input –i $INT_FIR_INT_INTERFACE –s $CLASS_A –j REJECT -l
ipchains –A output –i $INT_FIR_INT_INTERFACE –d $CLASS_A –j REJECT –l

#odrzucamy pakiety do (lub z) sieci prywatnych klasy B
ipchains –A input –i $INT_FIR_EXT_INTERFACE –s $CLASS_B –j REJECT -l
ipchains –A output –i $INT_FIR_EXT_INTERFACE –d $CLASS_B –j REJECT –l
ipchains –A input –i $INT_FIR_INT_INTERFACE –s $CLASS_B –j REJECT -l
ipchains –A output –i $INT_FIR_INT_INTERFACE –d $CLASS_B –j REJECT –l

#odrzucamy pakiety z adresem źródłowym zarezerwowanym dla interfejsu pętli zwrotnej
ipchains –A input –i $INT_FIR_EXT_INTERFACE –s $LOOPBACK –j REJECT -l
ipchains –A output –i $INT_FIR_INT_INTERFACE –s $LOOPBACK –j REJECT-l

#buforujący i przekazujący serwer nazw DNS
ipchains –A output –i $INT_FIR_EXT_INTERFACE –p udp \
-s $INT_FIR_EXT_IPADDR 53 –d $DMZ_ADDRESSES 53 –j ACCEPT
ipchains –A input –i $INT_FIR_EXT_INTERFACE –p udp \
-s $DMZ_ADDRESSES 53 –d $INT_FIR_EXT_IPADDR 53 –j ACCEPT
```

```
#lokalne zapytania od klientów do serwera
ipchains –A input –i $INT_FIR_EXT_INTERFACE –p udp \
-s $DMZ_ADDRESSES $UNPRIVPORTS –d $INT_FIR_EXT_IPADDR 53 –j ACCEPT
ipchains –A output –i $INT_FIR_EXT_INTERFACE –p udp \
-s INT_FIR_EXT_IPADDR 53 –d $DMZ_ADDRESSES $UNPRIVPORTS –j ACCEPT
ipchains –A input –i $INT_FIR_EXT_INTERFACE –p tcp \
-s $DMZ_ADDRESSES $UNPRIVPORTS –d $INT_FIR_EXT_IPADDR 53 –j ACCEPT
ipchains –A output –i $INT_FIR_EXT_INTERFACE –y –p tcp \
-s INT_FIR_EXT_IPADDR 53 –d $DMZ_ADDRESSES $UNPRIVPORTS –j ACCEPT

#wychodzące połączenia klienckie od firewalla wewnętrznego do dowolnego serwera WWW
ipchains –A output –i $INT_FIR_EXT_INTERFACE –p tcp \
-s $INT_FIR_EXT_IPADDR $UNPRIVPORTS –d $ANYWHERE 80 –j ACCEPT
ipchains –A input –i $INT_FIR_EXT_INTERFACE –p tcp ! -y\
-s $ANYWHERE 80 –d $INT_FIR_EXT_IPADDR $UNPRIVPORTS  –j ACCEPT

#dostęp lokalnych klientów do zdalnych serwerów grup dyskusyjnych
ipchains –A output –i $INT_FIR_EXT_INTERFACE –p tcp -s $INT_FIR_EXT_IPADDR $UNPRIVPORTS \
-d $NEWS_SERVER_DMZ_IPADDR 119 –j ACCEPT
ipchains –A input –i $INT_FIR_EXT_INTERFACE –p tcp ! -y -s $NEWS_SERVER_DMZ_IPADDR 119 \
-d $INT_FIR_EXT_IPADDR $UNPRIVPORTS –j ACCEPT

#wychodzące żądania klientów FTP
ipchains –A output –i INT_FIR_EXT_INTERFACE –p tcp \
-s $INT_FIR_EXT_IPADDR $UNPRIVPORTS -d $ANYWHERE 21 –j ACCEPT
ipchains –A input –i INT_FIR_EXT_INTERFACE –p tcp ! –y \
-s $ANYWHERE 21 –d $INT_FIR_EXT_IPADDR $UNPRIVPORTS  –j ACCEPT

#odpowiedzi na żądanie utworzenia kanału danych w trybie zwykłym
ipchains –A input –i INT_FIR_EXT_INTERFACE –p tcp \
-s $ANYWHERE 20 –d $INT_FIR_EXT_IPADDR $UNPRIVPORTS  –j ACCEPT
ipchains –A output –i INT_FIR_EXT_INTERFACE –p tcp ! –y \
-s $INT_FIR_EXT_INTERFACE $UNPRIVPORTS -d $ANYWHERE 20 –j ACCEPT

# odpowiedzi na żądanie utworzenia kanału danych w trybie pasywnym
ipchains –A output –i INT_FIR_EXT_INTERFACE –p tcp  \
-s $INT_FIR_EXT_IPADDR $UNPRIVPORTS -d $ANYWHERE  $UNPRIVPORTS –j ACCEPT
ipchains –A input –i INT_FIR_EXT_INTERFACE –p tcp ! –y \
-s $ANYWHERE $UNPRIVPORTS  –d $INT_FIR_EXT_IPADDR $UNPRIVPORTS  –j ACCEPT
```

10. Skrypt

```
#!/usr/bin/perl
# Skrypt dziala w oparciu o logi ipchains jadra 2.2.X
# i przedtsawia liste polaczen nawiazywanych z intranetu
# Administracja systemami komputerowymi, KI, AGH,
# A. Majka, D. Koscielniak, P. Nowak, Informatyka IV, czerwiec 2002
#
# Zalozenia przy ktorych skrypt dziala prawidlowo:   :)
# 1. z tego samego hosta zrodlowego nie mozna wyslac pakietow do roznych hostow,
#    tj. w ramach tego samego polaczenia; konieczne jest nawiazanie nowego polaczenia
# 2. skrypt dziala poprawnie jesli pozycje w logu sa pozycjami z masqaringu (znacznik
MASQ
#    oraz FORWARD)
# 3. roznica 5 minut jest uwzgledniana jako czas pomiedzy ostatnim pakietem danego
#    polaczenia a pierwszym pakietem polaczenia nastepnego
# 4. roznica 5 minut jest ograniczeniem powodujacym zwiekszenie liczby polaczen,
#    gdyz bez niego, wszelkie pakiety i polaczenia zlalyby sie w jedno polaczenie
# 5. statystyki sa generowane tylko dla tych polaczen, ktorych pierwszy pakiet
#    (inicjujacy)znajduje się w logu

use Time::Local qw(timelocal);

print "\n\n\n\n";
print "Firewall logs (ipchains) analiser. Majka, Koscielniak, Nowak (c) 2002\n";
print "All rights reserved to above authors (May change in future).\n";
print "_____________________________________________________________________\n";
print "Processing...\n";


$errorflag = 0;
open(LOGFILE, $ARGV[0]) or $errorflag = 1;

if ($errorflag == 1) {
  print "$0: nie znaleziono pliq\n";
  die;
}


$firstdate = "";
$date = "";
$total_traffic = 0;
$howmany = 0;
$all = 0;
while ( <LOGFILE> ) {
  next unless /Packet log: forward MASQ/i;
  $all++;
  chomp;
  @log = split;
  #znalazlem w wsieci, ze ipchains ma czasami dziwne jazdy, wiec ja je usuwam...
  for (@log) { $_ =~ s/\)/\\\)/g; $_ =~ s/\(/\\\(/g; }
  ($month,$day,$time,$policy,$proto,$ipsource,$ipdest,$tot_len,  $orderID,  $isStart)  =
@log[0,1,2,8,10,11,12,13,15,18];
  if (length($month) eq 0 or
      length($day) eq 0 or
      length($policy) eq 0 or
      length($proto) eq 0 or
      length($ipsource) eq 0 or
      length($ipdest) eq 0 or
      length($tot_len) eq 0 or
```

```
      length($orderID) eq 0 ) {
    next;
  }

  $date = $day . " " . $month . " " . $time;
  if (length($firstdate) == 0) {
    $firstdate = $date;
  }

  $lastdate = $date;
  ($flush, $protocol) = split "=", $proto;
  ($ips, $ports) = split ":", $ipsource;
  ($ipd, $portd) = split ":", $ipdest;
  ($flush, $packetlen) = split "=", $tot_len;
  ($flush, $packetcounter) = split "=", $orderID;

  if (length($ips) eq 0 or
      length($ports) eq 0 or
      length($ipd) eq 0 or
      length($portd) eq 0 or
      length($packetlen) eq 0 or
      length($packetcounter) eq 0) {
    next;
  }

  #taka walidacja na wszelki wypadek ;)
  if ($ips !~ /[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$/ or
      $ipd !~ /[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}$/ or
      $ports !~ /[0-9]{1,5}$/ or
      $portd !~ /[0-9]{1,5}$/ or
      $packetlen !~ /[0-9]+$/ or
      $packetcounter !~ /[0-9]+$/ ) {
    next;
  }

  $connection_id = $ips . " " . $ipd . " " . $portd;
  #jedno wielkie zalozenie: z jednego hosta:portu w sieci NIE MOZNA wyslac dwoch pakietow
  #do roznych hostow, chyba ze rozpoczeto na tym hoscie i na tym porcie NOWE polaczenie
  if ($isStart eq "SYN") {
      $insert_new = 1;
      if (exists $last_connection{$connection_id}) {
      if (length($ARGV[1]) eq 0) {
              #znalezlismy polaczenie z tego hosta - sprawdzamy czy dluzej niz 5 minut
temu
          $index = $last_connection{$connection_id} . " " . $connection_id;
          $diff_t = &calc_diff($connection_last_date{$index}, $date);
          if ($diff_t < 18000) {
                                          $insert_new = 0;
          }
      } else {
          $insert_new = 0;
      }
      }
      if ($insert_new eq 1) {
      $index = $packetcounter . " " . $connection_id;
      $last_connection{$connection_id} = $packetcounter;
      $connection_last_date{$index} = $date;
      $connection_begining{$index} = $date;
      $connection_out{$ips}++;
      $connection_in{$ipd}++;
      } else {
      $connection_last_date{$index} = $date;
      }
  } else {
      unless (exists $last_connection{$connection_id}) {
```

```
      #pakiet dotyczy polaczenia, ktorego poczatek nie byl w logach -> wyrzucamy
      next;
      }
      $index = $last_connection{$connection_id} . " " . $connection_id;
      $connection_last_date{$index} = $date;

  }

  #zliczamy sobie liczbe pakietow wyslanych z hosta
  ++$packetsFrom{$ips};

  #zliczamy liczbe pakietow wyslanych do hosta na zewnatrz
  ++$packetsTo{$ipd};

  #zliczamy liczbe bajtow wogule
  $total_traffic += $packetlen;
  #i liczbe bajtow na host w intranecie
  $traffichost{$ips} += $packetlen;
  #liczba bajtow na host w internecie
  $traffichost_dest{$ipd} += $packetlen;
  #a takze ile pakietow uwzglednilismy w analizie
  $howmany++;
}

print "\nReport generated.\n\n";
print "Found ",$all," packets. The ",$howmany," of them passed validation and was
included in report.\n";

print "\n\nConnection traffic analisys:\n\n";
for (sort keys %connection_begining) {
    ($flush, $host, $host_d, $port_d) = split " ", $_;
    $diff = &calc_diff($connection_begining{$_}, $connection_last_date{$_});
    print "From $connection_begining{$_} during $diff seconds host ",
    $host ," was connected to $host_d:$port_d \n";
}

print "\n\n";
print "Source traffic analisys:\n\n";
for (sort keys %traffichost) {
    print "Host ", $_, " send out ", $traffichost{$_}, " bytes in ", $packetsFrom{$_} ,"
packets and ",$connection_out{$_}," connections.\n";
}

print "\n\nDestination traffic analisys:\n\n";
for (sort keys %traffichost_dest) {
    print "Host ", $_, " received ", $traffichost_dest{$_}, " bytes in ", $packetsTo{$_}
," packets and ",$connection_in{$_}," connections.\n";
}

print "\n\nTotal bytes send out: ",$total_traffic,"\n\n";

sub calc_diff { #pierwsza data jest starsza
    ($day, $moth, $t) = split " ", $_[0];
    ($h, $m, $s) = split ":", $t;

    ($dayn, $mothn, $tn) = split " ", $_[1];
    ($hn, $mn, $sn) = split ":", $tn;

    $month = $monthn = 5;

    $diff = - timelocal($s, $m, $h, $day, $month-1, (localtime)[5]) + timelocal($sn, $mn,
$hn, $dayn, $monthn-1, (localtime)[5]);
    return $diff;
}
```

11. Bibliografia

[1] Robert L. Ziegler „Linux Firewalls”, Robomatic, 2001

[2] D. Brent Chapman & Elizabeth D. Zwicky „Building Internet Firewalls” O’Reilly, First Edition, November 1995.

[3] W. Cheswick, S. Bellovina „Firewalls and Internet Security”, Addison Wesley.

[4] Olaf Kirch „Linux Network Administrator's Guide” (wersja polska), O’Reilly & Associates, 1993-2000

[5] Materiały ze strony : http://www.cisco.com dotyczące głównie Acces-list.

[6] Mark Grennan, mark@grennan.com „Firewall and Proxy Server HOWTO”,2002
(http://www.tldp.org/HOWTO/Firewall-HOWTO.html#toc1)

[7] Rusty Russell „Linux IPCHAINS-HOWTO”, 2000
(http://www.tldp.org/HOWTO/IPCHAINS-HOWTO.html#toc1)

[8] Rusty Russell „Linux iptables HOWTO”, 1999
(http://www.telematik.informatik.uni-karlsruhe.de/lehre/seminare/LinuxSem/downloads/netfilter/iptables-HOWTO.html#toc1)

[9] David A. Ranch „Linux IP Masquerade HOWTO”,2002
(http://www.tldp.org/HOWTO/IP-Masquerade-HOWTO/index.html)