AKADEMIA GÓRNICZO-HUTNICZA

Wydział Elektrotechniki, Automatyki, Informatyki i Elektroniki

KATEDRA INFORMATYKI

Firewall.

Praca zaliczeniowa z przedmiotu Administracja systemem komputerowym.

Wersja **0.1-82** z dnia **13.06.2002**

Kierunek, rok studiów:

Informatyka IV rok

Przedmiot:

Administracja systemem komputerowym.

Temat:

15

Prowadzący zajęcia:

mgr inż. Bogusław Juza

Rok akad: **2001/2002**

Semestr: **letni**

Zespół autorski:

Piotr Kopyt	piotr.kopyt@zst-softel.pl
Michał Kułakowski	kulakow@ceti.pl
Krzysztof Niemiec	krzysztof.niemiec@magellan.net.pl

Kraków, maj 2002

Spis treści

1. Co to jest Firewall ?

Bezpieczeństwo sieci komputerowych, jest czynnikiem na który kładziony jest coraz większy nacisk. Wynika to po części z faktu, że ostatnio znacznie wzrosła liczba prób włamań (często udanych) do systemów komputerowych na których coraz częściej opiera się działanie różnych instytucji. Najlepszym sposobem uniknięcia zagrożeń jakie wiążą się z włamaniem do systemu komputerowego jest uniemożliwienie dostępu do sieci osobom niepowołanym. Do tego celu wykorzystuje się właśnie firewalle. Najczęściej stosowane metody ataku opisano poniżej.

Metoda ataku	Opis	Jak się bronić ?	Rola firewalla
Nieautoryzowany dostęp.	Polega na korzystaniu z usług oferowanych przez użytkowników którzy nie są do tego uprawnieni.	Precyzyjne określenie uprawnień użytkowników. Można np. zabronić dostępu do sieci wszystkim użytkownikom poza wyznaczoną grupą osób.	Firewalle są w stanie znacznie ograniczyć nieautoryzowany dostęp.
Dziury w programach.	Błędy w programach wynikają głównie z faktu, że kiedy powstawały niektóre z nich bezpieczeństwo nie było rzeczą do której przykładano szczególną wagę. Typowym przykładem są tu programy typu rlogin, rexec itp.	Należy wyłączyć wszystkie zbędne usługi które mogą stać się źródłem zagrożenia.	Firewall nie jest w stanie zabezpieczyć systemu przed tego typu atakami.
Odmowa usługi (DOS – deny of service)	Atak tego typu powoduje że usługa przestaje działać lub nie pozwala z siebie korzystać. Ataki te polegają bądź na wysyłaniu przez intruza znacznej liczby zapytań która powoduje przeciążenie (zawieszenie) usługi bądź też na preparowaniu	Uniemożliwienie dotarcia złośliwym pakietom do hosta oraz zapobieżenie obsługi podejrzanych zapytań.	Firewall pozwala znacznie ograniczyć tego typu ataki.
Podszywanie się.	W atakach tego typu intruz udaje niewinnego hosta przesyłając sfałszowane adresy w pakietach IP.	Należy weryfikować wiarygodność datagramów i poleceń. Należy również wyłączyć możliwość rutowania pakietów o złym adresie źródłowym. Dobrą metodą jest także wprowadzenie nieprzewidywalności w obsłudze połączeń np. dynamiczna alokacja portów.	Firewall pozwala znacznie ograniczyć tego typu ataki.

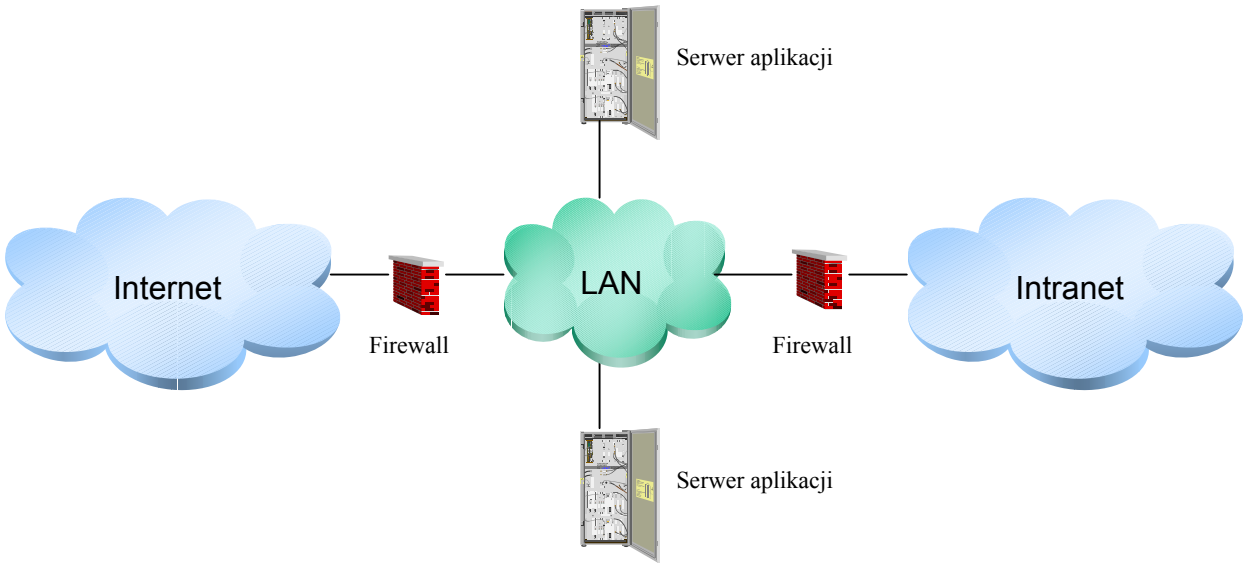
Atak typu SMURF (jest to atak typu DOS)	Atak ten polega na wysłaniu przez intruza wielu requestów PING na adres rozgłoszeniowy danej sieci, przy czym requesty mają spreparowny adres nadawcy tak aby wskazywał on na atakowany komputer. W rezultacie ofiara jest zalewana setkami odpowiedzi PING.	Blokowanie możliwości wysłania pakietu PING request na adres rozgłoszeniowy.	Firewall jest w stanie całkowicie zabezpieczyć przed tego typu atakami.
Podsłuchiwanie.	Jest to bardzo prosta metoda ataku. Polega na nasłuchiwanie przez hosta wszystkich pakietów, także tych które nie są dla niego przeznaczone (karta sieciowa w trybie PROMISCOUS).	Szyfrowanie danych, detekcja sniferów poprzez wykrywanie kart sieciowych pracujących w trybie PROMISCOUS.	Firewall nie jest w stanie zabezpieczyć systemu przed tego typu atakami.

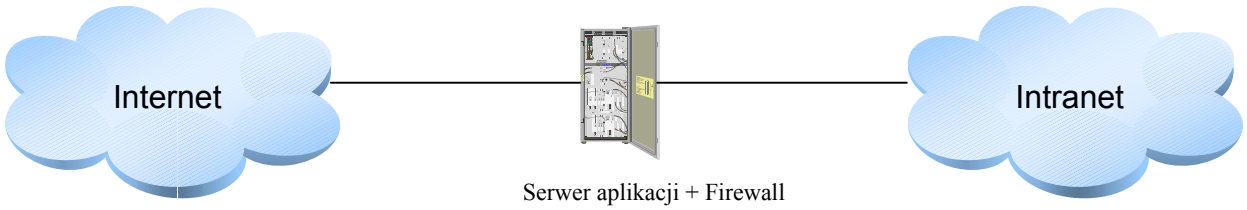
Firewall - „ściana ogniowa" jest terminem wziętym z konstrukcji samochodu. W samochodach firewalle fizycznie oddzielają silnik od pasażerów. To znaczy, że chronią one pasażerów w wypadku gdy silnik zapali się cały czas dostarczając kontroli nad nim

Firewall jest bezpiecznym i zaufanym komputerem który jest umieszczony pomiędzy siecią lokalną a Internetem. Firewall jest skonfigurowany z użyciem reguł określających jaki ruch sieciowy ma być obsługiwany a jaki blokowany.

1.1. Architektury firewalli.

Poniżej przedstawiono dwie podstawowe architektury firewalli. W skład pierwszej wchodzą dwa firewalle, pomiędzy którymi znajdują się serwery sieciowe, takie jak serwer WWW, FTP, NEWS, Application Servers. Konfiguracja taka zapewnia kontrolę nad połączeniami nie tylko Internet <-> Intranet, Internet <-> Serwery Aplikacji, ale również Intranet <-> Serwery Aplikacji co umożliwia realizację wewnętrznej polityki bezpieczeństwa w firmie. Dodatkowo rozwiązanie to zapewnia większe bezpieczeństwo ze względu na podwójną kontrolę dostępu przy połączeniach zewnętrznych. Ze względu na swoje skomplikowanie jest to architektura dedykowana dla dużych firm. W małych sieciach najczęściej stosuje się drugie rozwiązanie polegające na tym, że zarówno firewall jak i serwery aplikacji umieszczone są na jednej maszynie, jest to rozwiązanie tańsze i prostsze w konfiguracji.
Przykład pierwszej konfiguracji będzie opisany w niniejszym opracowaniu.





1.2. Typy firewalli.

Istnieją dwa podstawowe typy firewalii:

- **Firewalle filtrujące**

Firewalle filtrujące działają na poziomie pakietów IP (albo datagramów IP te dwa pojęcia w niniejszej pracy będą używane zamiennie). Są zaprojektowane do kontroli przepływu bazując na adresie źródłowym, docelowym, porcie i typie pakietu (zawartych w każdym z pakietów).

Ten typ firewalli jest bardzo bezpieczny, ale traci wiele typów zapisu. Może zablokować ludziom z dostęp z sieci prywatnej, ale nie powie, kto dostał się do twojego systemu publicznego, ani kto wyszedł z sieci lokalnej do Internetu.

Filtrowanie IP jest mechanizmem decydującym o tym które pakiety IP należy obsłużyć a które należy odrzucić, przy czym przy odrzucaniu pakietów można zdefiniować czy należy w odpowiedzi przesłać

komunikat ICMP do hosta który przysłał pakiet. Firewalle umożliwiają definiowanie wielu kryteriów określających filtrowane pakiety, do najczęstszych należą:

- typ protokołu: TCP, UDP, ICMP itp.
- numer portu (dla TCP/UDP)
- typ pakietu: SYN/ACK, ICMP Echo Request itp.
- adres źródłowy pakietu
- adres docelowy pakietu.

Firewalle filtrujące można dodatkowo podzielić na firewalle ze statyczną (static access list firewall) i dynamiczną (dynamic access list firewall) listą określającą reguły filtrowania.
Linux posiada opcję filtrowania pakietów w jądrach powyżej 1.3.x.

- **Serwery proxy**

Serwery proxy są serwerami działającymi na poziomie protokołu aplikacji (np. HTTP), serwery te nie rutują pakietów, a zamiast tego otwierają nowe połączenie do serwera z którym chce się połączyć host z sieci lokalnej. Cała komunikacja lokalny host <-> zdalny serwer odbywa się poprzez serwer proxy. Serwery proxy oferują znacznie więcej możliwości i są w stanie zapewnić większe bezpieczeństwo niż firewalle IP, gdyż „rozumieją" one zawartości obsługiwanych pakietów. Typowym przykładem jest tutaj serwer proxy WWW (np. squid), użycie takiego serwera zapobiega nadużyciom jakie mogą powstać przy próbie łączenia się z serwerem WWW (usługą na porcie 80) poprzez Telnet.
Pewne usługi mogą być zapewnione tylko przy pomocy serwerów proxy np. FTP w trybie aktywnym. (Aczkolwiek istnieją moduły firewalli IP pozwalające na obsługę trybu aktywnego FTP, ich działanie opiera sie jednak na analizie datagramów, są więc one niczym innym jak tylko serwerami proxy).

2. Narzedzia do konfiguracji Firewalli w systemie LINUX.

Firewall oparty na statycznej liście dostępu jest najczęściej stosowanym firewallem. W zależności od wersji jądra różnią się narzędzia oraz opcje jakie należy ustawić podczas kompilacji jądra.

W jądrach wersji 2.2 należy wybrać następujące opcje:

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

W przypadku jąder 2.4 opcje te mają postać:

```
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
```

Także narzędzia służące do konfiguracji firewalla różnią się w zależności od wersji jądra, dla jąder serii starszych niż 2.2 jest to *ipfwadm*, dla jąder 2.2 i nowszych *ipchains* zaś dla wersji 2.4 *iptables* z pakietu *netfilter*.

Dla każdego hosta można rozpatrywać filtrowanie pakietów w trzech miejscach:

- na wejściu (przekazywanie sterownik Ethernet -> warstwa rutująca)
- wewnątrz warstwy rutującej
- na wyjściu (przekazywanie warstwa rutująca -> sterownik Ethernet)

Dla każdego z tych miejsc można zdefiniować reguły określające zasady filtrowania pakietów, przy każdej regule należy określić miejsce jej stosowania. W programach *ipfwadm* i *ipchains* reguła związana z pakietami wejściowymi znaczona jest słowem kluczowym INPUT, związana z warstwą rutującą słowem FORWARD a związana z wyjściem słowem OUTPUT. W pakiecie *netfilter* zmianie uległy punkty w których następuje filtracja. Reguły Input oraz Output odnoszą się do przekazywania warstwa TCP/UDP <-> warstwa rutująca.

2.1. ipfwadm

Polecenie *ipfwadm* ma następującą składnię:

```
ipfwadm kategoria polecenie parametry [opcje]
```

Kategorie.

Kategoria określa rodzaj definiowanej reguły. Może przyjmować trzy wartości:

`-I` reguła wejściowa

`-O` reguła wyjściowa

`-F` reguła przekazywania.

Polecenia.

`-a [polityka]`	dodanie nowej reguły na koniec listy reguł
`-i [polityka]`	wstawienie nowej reguły na początek listy reguł
`-d [polityka]`	usunięcie istniejącej reguły
`-p`	ustawienie polityki domyślnej
`-l`	wylistowanie wszystkich reguł
`-f`	usuniecie wszystkich reguł

polityka może przybierać jedną z trzech wartości:

`accept` – zezwolenie na odbiór, rutowanie i wysyłanie pasujących pakietów

`deny` – odrzucenie pakietów

`reject` – odrzucenie pakietów, z tym że do hosta który nadesłał pakiet wysyłany jest komunikat ICMP

Paramtery.

`-P protokół`

określa protokół, może przyjmować wartości TCP, UDP, ICMP lub all.

`-S adres/maska [port]`

określa adres źródłowy pakietów do których pasuje reguła, maska podawana jest w postaci liczby całkowitej określającej ilość bitów np. 32 określa maskę postaci 255.255.255.0, domyślnie przyjmowana jest maska o wartości 32. Parametr ten wymaga aby podany był również parametr określający protokół. Numery portów mogą być podane w postaci zakresu np. 20:25. W przypadku protokołu ICMP pole portu określa typ datagramu (np. echo-request)

`-D adres/maska [port]`

określa adres docelowy, reguły identyczne jak w przypadku adresu źródłowego

`-v adres`

paramter ten dotyczy reguł wyjściowych i wejściowych (-I, -O) i określa interfejs sieciowy komputera

`-W nazwa`

ten parametr działa identycznie jak parametr –v z tym że operuje na nazwie interfejsu np. eth0

Argumenty opcjonalne.

`-b`

określa tryb dwukierunkowy, dodanie tego parametru do reguły powoduje to samo co dodanie reguły z zamienionymi wartościami parametrów –S i –D.

`-o`

powoduje że datagramy pasujące do reguły będą zapisywane do logu jądra

`-y`

dodanie tego parametru powoduje że do reguły pasować będą jedynie pakiety próbujące nawiązać połączenie TCP (mające ustawiony bit SYN i wyzerowany ACK)

`-k`

dodanie tego parametru powoduje że do reguły pasować będą jedynie pakiety będące odpowiedzią na próbę nawiązania połączenia TCP (mające ustawiony bit ACK)

Przykładowe wywołanie polecenia *ipfwadm* przedstawiono poniżej:

```
# ipfwadm -F -f
# ipfwadm -F -p deny
# ipfwadm –F –a deny –P tcp –S 0/0 80 –D 192.168.1.0/24 -y
# ipfwadm -F -a accept -P tcp -S 192.168.1.0/24 -D 0/0 80 -b
```

argument –F w powyższych poleceniach mówi że definiowane są reguły przekazywania (FORWARD) pierwsze polecenie czyści wszystkie reguły zapisane w konfiguracji *ipfwadm*, drugie polecenie ustawia politykę domyślną na odrzucanie pakietów, polityka ta będzie stosowana dla wszystkich pakietów które nie pasują do żadnej reguły. Czwarta reguła nakazuje przepuszczać wszystkie pakiety protokołu IP o adresie źródłowym którego pierwsze 24 bity, zgadzają się z adresem 192.168.1.0, dowolnym adresie docelowym (parametr 0/0) i porcie 80. Parametr –b zezwala na ruch pakietów w obie strony. Trzecia reguła pozwala na nawiązywanie połączeń TCP (wysyłanie pakietów z ustawionym znacznikiem SYN) jedynie przez hosty znajdujące się za firewallem. Reguły czytane są kolejno, dlatego reguła blokująca datagramy nawiązujące połączenie znalazła przed regułą akceptującą datagramy na porcie 80. Powyższy zestaw reguł zezwala na przepuszczanie datagramów zawierających pakiety HTTP.

2.2. ipchains

Dla każdego pakietu lista reguł *ipfwadm* przeszukiwana jest sekwencyjnie aż do natrafienia na pasującą regułę. Dla złożonych polityk bezpieczeństwa zawierających znaczną ilość reguł może to powodować

spowolnienie obsługi pakietów przez firewall a tym samym spowolnienie pracy całej sieci lokalnej. Program *ipchains* pozwala na przyspieszenie procesu przeszukiwanie zestawu reguł poprzez możliwość definiowania skoków pomiędzy regułami oraz możliwość definiowania własnych zestawów reguł obok standardowych INPUT, FORWARD i OUTPUT. Zestawy reguł tworzą tzw. łańcuchy. Składnia *ipchains* wygląda następująco:

```
ipchains polecenie określenie_reguły opcje
```

Polecenia

`-A łańcuch`

Dodaje jedną lub kilka reguł na koniec łańcucha

`-I łańcuch numer_reguły`

wstawia jedną lub kilka reguł na początek łańcucha

`-D łańcuch`

usuwa regułę z łańcucha

`-R łańcuch numer_reguły`

zamienia regułę o podanym numerze w łańcuchu

`-C łańcuch`

testuje zadanym łańcuchem datagram opisany regułą

`-L [łańcuch]`

listuje wszystkie reguły łańcucha lub wszystkich łańcuchów

`-F [łańcuch]`

usuwa wszystkie reguły z zadanego łańcucha lub wszystkich łańcuchów

`-Z [łańcuch]`

zeruje liczniki pakietów i bajtów dla zadanego łańcucha lub wszystkich łańcuchów

`-N łańcuch`

tworzenie nowego łańcucha o zadanej nazwie

`-X [łańcuch]`

usunięcie łańcucha zdefiniowanego przez użytkownika lub wszystkich łańcuchów zdefiniowanych przez użytkownika

`-P łańcuch polityka`

ustawia politykę domyślną dla danego łańcucha. Dopuszczalne polityki to ACCEPT, DENY, REJECT, REDIR i RETURN. Polityki ACCEPT, DENY, REJECT mają takie samo znaczenie jak w *ipfwadm,* REDIR oznacza że pakiet powinien zostać przekierowany na inny port firewalla. Polityka ta jest wykorzystywana do konfigurowania transparentnego serwera w3cache, (przekierowania z portu 80 na port serwera np. 8080). RETURN powoduje że kod firewalla powraca do łańcucha który wywołał regułę i kontynuuje działanie począwszy od następnej reguły.

Parametry.

`-p [!] protokół`

określa protokół, dopuszczalne wartości to tcp, udp, icmp lub all. ! oznacza negację. Domyślną wartością jest all.

`-s [!] adres[/maska][!][port]`

określa adres źródłowy pakietu, zasady identyczne jak w *ipfwadm*. ! oznacza negację.

`-d [!] adres[/maska][!][port]`

określa adres docelowy pakietu, zasady identyczne jak dla parametru –s.

`-j cel`

określa akcję jaką należy podjąć kiedy reguła pasuje, można to tłumaczyć jako „jump" (skok). Dopuszczalne cele skoku to ACCEPT, DENY, REJECT, REDIR, RETURN oraz łańcuchy zdefiniowane przez użytkownika. Dodatkowo istnieje jeszcze wartość MASQ nakazująca transformację adresów IP (transformacja adresów omówiona będzie w rozdziale 5). Jeśli parametr ten zostanie pominięty to dla datagramu pasującego do reguły nie zostanie podjęte żadne działanie, zostaną jedynie uaktualnione liczniki datagramów i bajtów.

`-i [!] nazwa_interfejsu`

określa interfejs który przyjął datagram lub przez który datagram zostanie wysłany. ! oznacza negację, + oznacza że do reguły pasował będzie każdy datagram przyjęty lub wysłany przez interfejs którego nazwa zaczyna się od zadanego ciągu np. eth+ pasuje do wszystkich interfejsów Ethernet. Parametr ten jest odpowiednikiem parametrów –V –W *ipfwadm*.

`[!] -f`

reguła ta dotyczy wszystkiego poza pierwszym fragmentem datagramu podzielonego na fragmenty

Opcje.

`-b`

ustawia ruch dwukierunkowy, podobnie jak w *ipfwadm*

`-v`

powoduje wyświetlanie bogatszych informacji przez *ipchains*

`-n`

wyłącza konwersję adresów IP na nazwy

`-l`

powoduje że pasujące do reguły datagramy są logowane przez jądro.

`-o [max_rozmiar]`

powoduje że max_rozmiar bitów pasującego datagramu kopiowane jest do urządzenia „netlink".

`-m wartość_znakowania`

powoduje znakowanie datagramów zadaną wartością. Opcja ta nie jest obecnie wykorzystywana.

`-t maska_and maska_or`

pozwala na operacje na bitach typ usługi (TOS – type of service) w nagłówku IP datagramu. maska_and i maska_or są maskami bitowymi które będą poddawane operacji AND i OR z bitami typu usługi datagramu. Opcja ta pozwala na nadawanie priorytetu rutowanym datagramom.

`-x`

opcja ta wymusza pokazywanie wszystkich wyników bez zaokrągleń

`-y`

podobnie jak w *ipfwadm* opcja ta powoduje że do reguły pasują datagramy z ustawionym bitem SYN i wyzerowanymi bitami ACK i FIN.

Zdefiniowane reguły pozwalające na ruch HTTP, przy użyciu *ipchains* miałyby postać:

```
# ipchains –F forward
# ipchains –P forward DENY
# ipchains –A forward –s 0/0 80 –d 192.168.1.0/24 –p tcp –y -j DENY
# ipchains –A forward –s 192.168.1.0/24 –d 0/0 80 –p tcp –b -j ACCEPT
```

powyższe reguły nie pokazują jednak możliwości jakie dają łańcuchy IP, rozpatrzmy reguły zdefiniowane poniżej zezwalające na ruch WWW:

```
# ipchains –F input
# ipchains –P input DENY
# ipchains –N rdir
# ipchains –A rdir –j REDIRECT 8080 –p tcp –s 192.168.1.0/24 –d 0/0 80
# ipchains –A rdir –j REDIRECT 8443 –p tcp –s 192.168.1.0/24 –d 0/0 443
# ipchains –A input –p tcp –j rdir
```

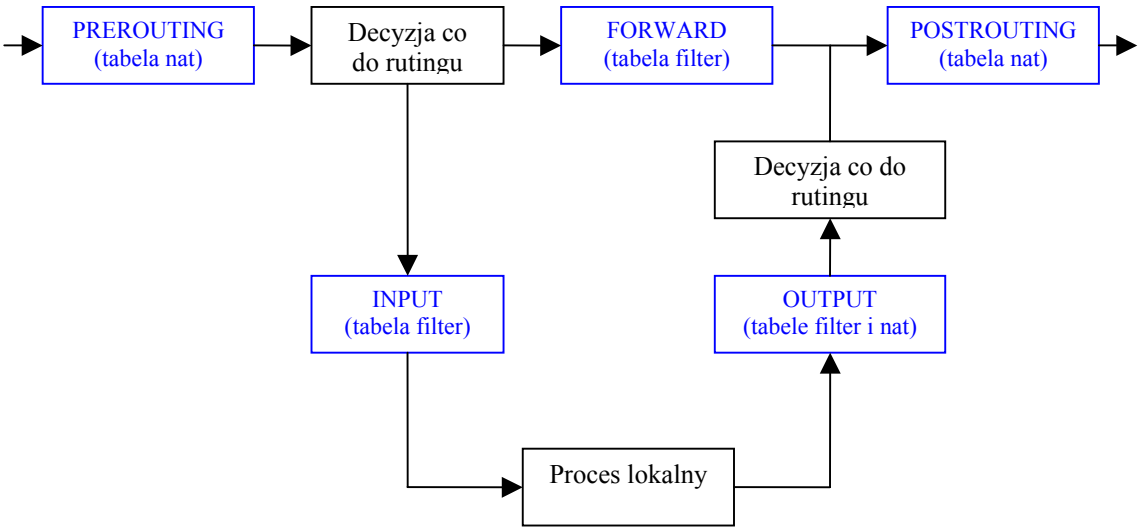
zdefiniowaliśmy tutaj reguły pozwalające na przekierowanie ruchu HTTP i HTTPS do lokalnego transparentnego serwera w3cache. Reguły opisujące przekierowania zgrupowane są w łańcuchu redir wszystkie pakiety wchodzące (reguła input) przekierowywane są do łańcucha redir. Pozostałe pakiety są odrzucane (domyślna polityka DENY). Jest to tylko przykład użycia łańcuchów definiowanych przez użytkownika i nie pokazuje on wszystkich zalet tego mechanizmu, zalety te ujawniają się gdy reguły zdefiniowane są w ten sposób że nie jest konieczne przeszukiwanie całej listy reguł przy sprawdzaniu pojedynczego pakietu.

2.3. netfilter i tabele IP

W netfilter zmieniono nieco architekturę firewalla, główne zmiany dotyczyły punktów w których następuje filtrowanie pakietów. Reguły zdefiniowane ze słowem kluczowym INPUT w *ipfwadm* i *ipchains* dotyczyły zarówno pakietów przeznaczonych dla hosta jak i tych które trafią do warstwy rutujacej, podobnie OUTPUT dotyczyło pakietów wygenerowanych przez hosta jak i przez niego rutowanych. W *iptables* istnieją dwie tablice reguł. Pierwsza *filter* (domyślna) zawiera wbudowane łańcuchy INPUT i OUTPUT dla pakietów przeznaczonych lub wygenerowanych przez lokalnego hosta oraz FORWARD dla pakietów rutowanych przez hosta. Druga tablica *nat* dotyczy reguł stosowanych dla pakietów dla których dokonywana jest translacja adresów IP (Network Address Translation - NAT) i zawiera ona łańcuchy PREROUTING i

ROUTING, dla tablicy *nat* dostępny jest także łańcuch OUTPUT. Dostępna jest także tablica rzadko używana mengle.

Poniżej przedstawiono proces obsługi pakietów przez *netfilter*, procesy za które odpowiedzialny jest *netfilter* zaznaczono na niebiesko z podaniem nazw łańcuchów w których zgrupowane są odpowiednie reguły:



W skład *netfilter* wchodzą moduły *ipfwadm.o, ipchains.o* oraz *ip_tables.o.* Jednocześnie może być załadowany tylko jeden z nich. Modułu *ipfwadm.o* i *ipchains.o* symulują działanie odpowiednio *ipfwadm* i *ipchains.* Załadowanie modułu *ip_tables.o* pozwala na korzystanie z narzędzia *iptables.* Składnia *iptables* wygląda następująco:

```
iptables polecenie okreslenie_reguly rozszerzenia
```

Polecenia.

`-A łańcuch`

Dodaje jedną lub kilka reguł na koniec łańcucha

`-I łańcuch numer_reguły`

wstawia jedną lub kilka reguł na początek łańcucha

`-D łańcuch`

usuwa regułę z łańcucha

`-D łańcuch numer_reguly`

usuwa regułę z łańcucha z pozycji o numerze numer_reguly, reguła na pierwszej pozycji ma numer 1

`-R łańcuch numer_reguły`

zamienia regułę o podanym numerze w łańcuchu

`-C łańcuch`

testuje zadanym łańcuchem datagram opisany regułą

`-L [łańcuch]`

listuje wszystkie reguły łańcucha lub wszystkich łańcuchów

`-F [łańcuch]`

usuwa wszystkie reguły z zadanego łańcucha lub wszystkich łańcuchów

`-Z [łańcuch]`

zeruje liczniki pakietów i bajtów dla zadanego łańcucha lub wszystkich łańcuchów

`-N łańcuch`

tworzenie nowego łańcucha o zadanej nazwie

`-X [łańcuch]`

usunięcie łańcucha zdefiniowanego przez użytkownika lub wszystkich łańcuchów zdefiniowanych przez użytkownika

`-P łańcuch polityka`

ustawia politykę domyślną dla danego łańcucha. Dopuszczalne polityki pokrywają się z dopuszczalnymi celami (podawanymi po -j).

Cele (targets).

ACCEPT

Oznacza akceptację (przepuszczenie) pakietu.

DROP

Oznacza odrzucenie pakietu.

RETURN

Powoduje że kod firewalla powraca do łańcucha który wywołał regułę i kontynuuje działanie począwszy od następnej reguły.

QUEUE

Przekierowuje pakiet do przestrzeni użytkownika w celu dalszego przetworzenia. Aby możliwa była obsługa pakietów w przestrzeni użytkownika konieczny jest moduł ip_queue (jest to moduł eksperymentalny). Dostęp do przekierowanych pakietów odbywa sie za pomocą libipq API (np. perlipq).

LOG

Powoduje że pakiet pasujący do reguły jest logowany (jako komunikat jądra). Dla celu tego można zdefiniować parametry:

--log-level określa poziom komunikatu

--log-prefix określa prefiks dodawany do komunikatu

Cel ten nie jest celem ostatecznym, tzn. po zalogowaniu pakietu kod firewalla powraca do dalszego sprawdzania reguł dla tego pakietu.

MIRROR

Powoduje że pakiet wysyłany jest z powrotem do hosta który go przysłał.

SNAT

Cel ten może być zdefiniowany jedynie dla łańcucha POSTROUTING tablicy nat, powoduje że dla danego pakietu zamieniany jest adres źródłowy. Wartość na którą ma być zmieniony adres definiuje się przy pomocy parametru –to-source.

MASQUERADE

Cel ten podobnie jak SNAT definiowany jest dla łańcucha POSTROUTING tablicy nat, i również powoduje że dla danego pakietu zamieniany jest adres źródłowy. Różnica polega na tym że adres źródłowy zamieniany jest na adres interfejsu do którego skierowany zostanie pakiet. Dlatego cel ten może być stosowany przy dynamicznym przydziale adresów IP.

DNAT

Cel ten może być zdefiniowany jedynie dla tablicy nat, powoduje że dla danego pakietu zamieniany jest adres docelowy. Wartość na którą ma być zmieniony adres definiuje się przy pomocy parametru –to-destination.

Parametry.

`-p [!] protokół`

określa protokół, dopuszczalne wartości to tcp, udp, icmp lub all. ! oznacza negację. Domyślną wartością jest all.

`-s [!] adres[/maska]`

określa adres źródłowy pakietu, zasady podobne jak w *ipfwadm* z tym że nie określa się tutaj portu (służy do tego parametr *sport*). ! oznacza negację.

`-d [!] adres[/maska][!][port]`

określa adres docelowy pakietu, zasady identyczne jak dla parametru –s.

`-j cel`

określa akcję jaką należy podjąć kiedy reguła pasuje, można to tłumaczyć jako „jump” (skok). Dopuszczalne cele skoku to ACCEPT, DROP, QUEUE i RETURN oraz łańcuchy zdefiniowane przez użytkownika. Jeśli parametr ten zostanie pominięty to dla datagramu pasującego do reguły nie zostanie podjęte żadne działanie, zostaną jedynie uaktualnione liczniki datagramów i bajtów.

`-i [!] nazwa_interfejsu`

określa interfejs który przyjął datagram lub przez który datagram zostanie wysłany. ! oznacza negację, + oznacza że do reguły pasował będzie każdy datagram przyjęty lub wysłany przez interfejs którego nazwa zaczyna się od zadanego ciągu np. eth+ pasuje do wszystkich interfejsów Ethernet.

`-o [!] nazwa_interfejsu`

określa interfejs na który datagram zostanie wysłany, składnia identyczna jak dla parametru –i.

`[!] -f`

reguła ta dotyczy wszystkiego poza pierwszym fragmentem datagramu podzielonego na fragmenty

Opcje.

`-v`

powoduje wyświetlanie bogatszych informacji przez *iptables*

`-n`

wyłącza konwersję adresów IP na nazwy

`-x`

opcja ta wymusza pokazywanie wszystkich wyników bez zaokrągleń

`- -numery_wierszy`

powoduje, że przy wyświetlaniu zestawu reguł pokazywane są numery wierszy. Numer wiersza odpowiada pozycji reguły w łańcuchu.

Rozszerzenia.

Funkcjonalność *iptables* można rozszerzyć poprzez moduły bibliotek dzielonych. Istnieją standardowe rozszerzenia udostępniające funkcje *ipchains.* Poniższa lista pokazuje funkcje dostępne przez rozszerzenie.

Rozszerzenia TCP: używane z –m tcp –p tcp

`--sport [!]port[:port]`

określa port z którego musi pochodzić datagram, można również podobnie jak w *ipfwadm* i *ipchains* definiować zakres portów. ! określa negację.

`--dport [!]port[:port]`

określa port docelowy datagramu, zasady identyczne jak dla -sport

`--tcp-flags [!] maska lista`

określa wartości jakie muszą mieć znaczniki pakietu aby ten pasował do reguły. maska to lista oddzielonych przecinkami znaczników które są sprawdzane. lista to lista oddzielonych przecinkami znaczników które muszą być ustawione żeby reguła pasowała. Dopuszczalne wartości znaczników to SYN, ACK, FIN, RST, URG, PSH, ALL i NONE. ! oznacza negację.

`[!] --syn`

powoduje że reguła pasuje do datagramów z ustawionym bitem SYN i wyzerowanymi znacznikami ACK i FIN.

Rozszerzenia UDP: używane z –m udp –p udp

`--sport [!]port[:port]`

zasady identyczne jak dla rozszerzenia TCP.

`--dport [!]port[:port]`

określa port docelowy datagramu, zasady identyczne jak dla –sport

Rozszerzenia ICMP: używane z –m icmp –p icmp

`--icmp-type [!] nazwa_typu`

określa typ komunikatu ICMP pasującego do reguły, niektóre z dopuszczalnych nazw to: echo-request, echo-reply, source-quench, time-exceeded, destination-unreachable.

Rozszerzenia MAC: używane z –m mac

`--mac-source [!] adres`

powoduje że do reguły pasuje datagram z podanym adresem MAC

Rozszerzenia multiportt: używane z –m multiport

Pozwala na definiowanie numerów portów w bardziej wygodny sposób z użyciem –sport i –dport numery portów i zakresy mogą być oddzielone przecinkami np. –dport 20:25,80,443

Rozszerzenia limit: używane z –m limit

`--limit value`

określa przedział czasu w jakim dokonywane jest limitowanie pakietów, domyślną wartością jest 3/godzinę

`--limit-burst value`

określa liczbę pakietów jakie mogą pasować do reguły w jednostce czasu określonej parametrem limit, domyślną wartością jest 5

zapis:

```
# iptables -A FORWARD -m limit -j LOG
```

powoduje, że pierwszych 5 pakietów zostanie zapisanych w logach (załóżmy że zapisywanie do logów realizowane jest przez reguły zgrupowane w łańcuchu LOG), następnie przez 20 minut nie będzie zalogowany żaden pakiet. Po 20 minutach „zwolni się miejsce" dla jednego pakietu, który będzie można zalogować. Jeśli przez 100 minut od chwili zalogowania 5 pakietu nie nadejdzie żaden pakiet pasujący do reguły to „pojemność reguły" znów będzie wynosiła 5, czyli nastąpi powrót do punktu wyjścia. Opcja ta jest wykorzystywana do obrony przed atakami typu DOS (np. SYN FLOOD).

Rozszerzenia state: używane z –m state

`--state value`

value określa stan połączenia do którego należy analizowany pakiety. Możliwe wartości to INVALID oznaczający że pakiet nie pasuje do żadnego znanego połączenia, ESTABLISHED oznacza że pakiet należy do połączenia dla którego zarejestrowano ruch pakietów w obu kierunkach, NEW oznacza że pakiet inicjuje nowe połączenie albo należ do połączenia dla którego nie zarejestrowano ruchu w obu kierunkach, RELATED oznacza że pakiet inicjuje nowe połączenie które jest związane z połączeniem już istniejącym np. transfer danych przy połączeniu FTP lub komunikat błędu ICMP.

Zdefiniowane wcześniej dla *ipfwadm* i *ipchains* reguły przepuszczające ruch HTTP, w przypadku *iptables* będą miały postać:

```
# modprobe ip_tables
# iptables -F FORWARD
# iptables -P FORWARD DROP
# iptables -A FORWARD -m tcp -p tcp -s 0/0 --sport 80 -d 192.168.1.0/24
  --syn -j DROP
# iptables -A FORWARD -m tcp -p tcp -s 192.168.1.0/24 --sport / 80 -d
0/0 -j ACCEPT
# iptables -A FORWARD -m tcp -p tcp -d 192.168.1.0/24 --dport 80 -s 0/0
  -j ACCEPT
```

2.4. Przykład trudnego przypadku

W przedstawionych dotychczas przykładowych konfiguracjach firewalla dotyczących ruchu HTTP nie było żadnych problemów z ustaleniem reguł, jednak nie zawsze tak jest. Typowym przykładem usługi dla której skonfigurowanie firewalla nastręcza trudności jest FTP, a konkretnie tryb aktywny FTP. Jak wiadomo w

protokole FTP istnieje osobne połączenie (port) do komunikacji i osobny port do przesyłu danych. Scenariusz przy przesyłaniu danych w trybie aktywnym jest nastepujący: klient wysyłając do serwera żądanie (wysyłanie na port 21 serwera) do którego obsługi konieczne będzie otwarcie połączenia do przesłania danych wysyła jednocześnie numer portu (polecenie PORT) na którym oczekuje na podjęcie przez serwer połączenia (w tym przypadku to klient otwiera socket i oczekuje na połączenie). I tu właśnie pojawia się problem, gdyż nie wiadomo na jakim porcie klient będzie oczekiwał na połączenie. Nie można tego stwierdzić bez analizy zawartości pakietu IP (czego firewalle filtrujące z reguły nie robią). Istnieją cztery rozwiązania powyższego problemu:

- pierwsze (mało eleganckie) polega na wymuszeniu na wszystkich klientach FTP znajdujących się za firewallem trybu pasywnego FTP
- drugie polega na zastosowaniu serwera proxy FTP, który działa na poziomie protokołu FTP i nie będzie miał problemu z odczytaniem numeru portu na którym klient oczekuje na połączenia
- trzeci, polega na użyciu modułu *ip_conntrack_ftp* który analizuje zawartość pakietów IP i na tej podstawie określa numer portu na jakim klient otworzył socket, rozwiązanie to dotyczy firewalli oparty o *netfilter*.
- wiele serwerów FTP otwiera połączenie do klienta z portu 20 co można wykorzystać przy definiowaniu reguł firewalla

2.5. **Firewall z dynamiczną obsługą ruchu sieciowego**.

Poprzednio opisane konfiguracje firewalli były konfiguracjami raczej statycznymi tzn. raz zdefiniowane reguły nie podlegały zmianie w trakcie pracy firewalla. Często jest to niewystarczające i zachodzi potrzeba dynamicznego modyfikowania reguł przekazywania pakietów. Jako przykładowe narzędzia pozwalające na dynamiczną obsługę ruchu sieciowego omówimy biblioteki *libipq* i *libiptc*.

Moduł Perlipq dla Perla.

Jest to biblioteka dostarczająca API do obsługi pakietów skierowanych do przestrzeni użytkownika poprzez przekierowanie tych pakietów do celu QUEUE. Znajduje się ona w module IPTables::IPv4::IPQueue. Dostępne funkcje to (parametry do funkcji przekazywane są jako hashe):

`new`

Konstruktor, tworzy nowy obiekt będący kolejką dla datagramów przekierowanych do przestrzeni użytkownika, parametry wywołania to

`protocol` – określa protokół, dopuszczalne wartości to IPv4 i IPv6. IPv4 jest wartością domyślną.

`copy_mode` – przyjmuje dwie wartości: IPQ_COPY_META i IPQ_COPY_PACKET. IPQ_COPY_META oznacza że do przestrzeni użytkownika kopiowana jest jedynie „metadata” pakietu (nagłówek Ethernet) dla

IPQ_COPY_PACKET kopiowana jest także zawartość pakietu, w tym przypadku konieczne jest zdefiniowanie parametru copy_range.

`copy_range` – określa w bitach, część zawartości pakietu która ma zostać przekopiowana do przestrzeni użytkownika

`set_mode`

ustawia jeden z trybów kopiowania IPQ_COPY_META lub IPQ_COPY_PACKET.

`get_message`

pobiera pakiet z kolejki, opcjonalnym parametrem wywołania może być [timeout] gdzie timeout określa czas timeout dla operacji, jeśli parametr ten jest równy zero albo jest niezdefiniowany to timeout dla operacji nie istnieje. Funkcja ta zwraca obiekt IPTables::IPv4::IPQueue::Packet. Obiekt ten posiada następujące atrybuty (są to atrybuty read-only):

`packet_id`	ID pakietu
`mark`	oznaczenie nadane przez Netfilter.
`timestamp_sec`	czas nadejścia pakietu (część sekundowa)
`timestamp_usec`	czas nadejścia pakietu (część mikrosekundowa)
`hook`	określa tabelę Netfilter z której nastąpiło przekierowanie do przestrzeni użytkownika
`indev_name`	nazwa interfejsu przez który przyszedł pakiet
`outdev_name`	nazwa interfejsu przez który wyjdzie pakiet.
`hw_protocol`	protokół hardware.
`hw_type`	typ hardware.
`hw_addrlen`	długość adresu hardware'owego.
`hw_addr`	adres hardware'owy
`data_len`	wielkość zawartości pakietu.
`payload`	zawartość pakietu.

`set_verdict`

określa czynności jakie jądro ma powziąć w stosunku do pakietu, parametrami tej funkcji są:

`id` – ID pakietu

`verdict` – określa operację której jądro ma dokonać na pakiecie, przyjmuje jedną z dwóch wartości: NF_DROP lub NF_ACCEPT

`[data_len, buf]` – opcjonalnie określa zmodyfikowana zawartość pakietu.

`close`

niszczy kolejkę pakietów i zwalnia zajęte przez nią zasoby.

`errstr`

Zwraca informacje o błędzie

Poniżej przedstawiono typową funkcję wykorzystującą powyższe metody. Funkcja przekazuje pakiety z powrotem do jądra bez modyfikacji ich.

```
sub main
{
    my ($queue, $msg);

    $queue = new IPTables::IPv4::IPQueue(copy_mode => IPQ_COPY_META)
             or die IPTables::IPv4::IPQueue->errstr;

    while (1) {
       $msg = $queue->get_message()
              or die IPTables::IPv4::IPQueue->errstr;
       print "Issuing verdict on packet " . $msg->packet_id() . "\n";
       $queue->set_verdict($msg->packet_id(), NF_ACCEPT) > 0
       or die IPTables::IPv4::IPQueue->errstr;
    }
}
```

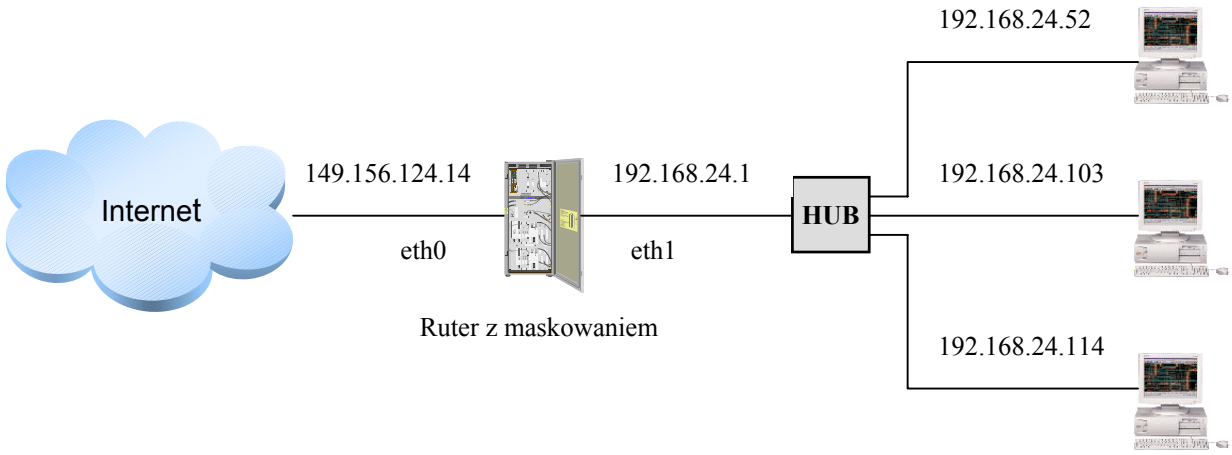
Biblioteki libipq i libiptc dla języka C.

Biblioteka o podobnej jak *Perlipq* funkcjonalności istnieje również dla języka C. Dodatkowo dla języka tego istniej również biblioteka *libiptc* pozwalająca na modyfikowanie reguł zawartych w łańcuchach *iptables*.

3. Maskowanie IP.

Maskowanie IP (ang. IP masquerading) jest jedną z odmian zaawansowanej funkcji sieciowej tzw. translacji adresów sieciowych (NAT –Network Address Translation). Maskowanie pozwala aby hosty z sieci Ethernet były widziane w Internecie pod jednym publicznym adresem IP. Dzięki temu dysponując zaledwie jednym adresem IP można zapewnić dostęp do internetu wielu hostom a zamiany adresów dokonuje w czasie rzeczywistym ruter.

Przykład sieci z maskowaniem IP.



Załóżmy że host 192.168.24.52 chce połączyć się z jakimś hostem znajdującym się w internecie np. 213.180.130.200 (www.onet.pl), wysyła więc pakiet TCP do rutera (załóżmy że ruter z maskowaniem jest gatewayem dla hostów sieci, jest to najczęściej spotykane rozwiązanie), ruter dokonuje translacji adresu IP w pakiecie zamieniając go na adres IP własnego interfejsu który widoczny jest w internecie (149.156.124.14) zapamiętując jednocześnie to połączenie tak by móc dokonać ponownej translacji adresów IP gdy nadejdzie odpowiedź z serwera.

Zastosowanie maskowania IP powoduje że sieć posiada specyficzne właściwości

- żaden z hostów sieci lokalnej nie jest widoczny w internecie, cała sieć Ethernet znajdująca się za ruterem maskującym w internecie widziana jest pod adresem rutera. Z jednej strony ogranicza to możliwości hostów sieci lokalnej np. dostęp do serwera WWW postawionego na jednym z nich będzie możliwy tylko z hostów w sieci lokalnej. Dlatego aby możliwe było korzystanie przez użytkowników z usług WWW, FTP, poczty elektronicznej itp. konieczne jest skonfigurowanie tych usług na maszynie z ruterem. Z drugiej strony podnosi to znacznie bezpieczeństwo hostów w sieci lokalnej, gdyż nie można się z nimi połączyć z Internetu.

- maskowanie IP nie pozostaje bez wpływu na wydajność sieci lokalnej, gdyż dla każdego rutowanego pakietu należy dokonać translacji adresów IP
- ruter z maskowaniem wymaga dodatkowej konfiguracji aby móc korzystać z niektórych usług. Przykładem może być tutaj omawiana wcześniej usługa FTP (tryb aktywny). Jest to realizowane poprzez specjalne moduły (dla FTP jest to moduł *ip_masq_ftp.o*).
- ruter realizujący maskowanie IP powinien być domyślnym gatewayem dla wszystkich hostów w sieci lokalnej, jeśli z jakichś względów gateway musiałby się znajdować po drugiej stronie rutera (patrząc od strony Ethernetu) to konieczne byłoby zastosowanie serwera proxy ARP.
- dodatkowej konfiguracji wymaga także serwer DNS, tak aby zwracał właściwe adresy dla hostów sieci lokalnej. Dobrym rozwiązaniem jest tutaj uruchomienie serwera DNS na maszynie na której znajduje się ruter realizujący maskowanie IP.

3.1. Konfigurowanie maskowania IP.

Konfigurowanie jądra.

Aby wymusić obsługę maskowania IP przez jądro należy je skonfigurować z następującymi opcjami:

dla jąder serii 2.2 obsługa dostępna jest jako moduł jądra

```
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
```

w przypadku jąder serii 2.4 obsługa maskowania nie jest dostępna jako opcja podczas kompilacji jądra. W czasie kompilacji należy wybrać opcję filtrowania pakietów:

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

Do skonfigurowania rutera tak aby obsługiwał maskowanie IP można użyć dowolnego z narzędzi *ipfwadm, ipchains, iptables.*

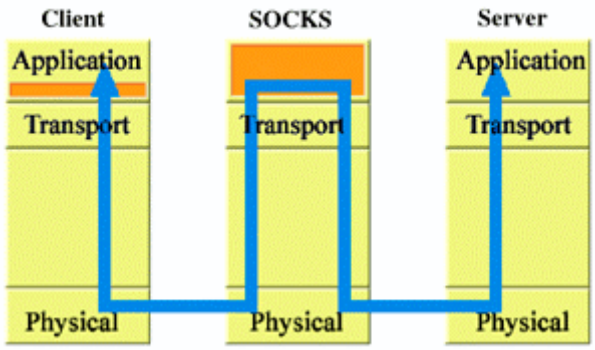
```
# ipfwadm -F -p deny
# ipfwadm -F -a accept -m -S 192.168.24.0/24 -D 0/0

# ipchains -P forward -j deny
# ipchains -A forward -s 192.168.24.0/24 -d 0/0 -j MASQ

# iptables -t nat -P POSTROUTING DROP
# iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

4. Serwery proxy.

Serwery proxy w przeciwieństwie do firewalli filtrujących nie operują na pakietach IP, działają natomiast „warstwę wyżej" na protokole aplikacji (np. HTTP) lub z użyciem dedykowanego protokołu (np. SOCKS). Serwery te można podzielić na specjalizowane serwery proxy np. serwery w3cache (przykładem takiego serwera jest squid) lub serwery „ogólnego przeznaczenia" pozwalające na korzystanie z wielu usług. Na przykładzie serwera SOCKS który należy do tej drugiej grupy serwerów zostaną opisane tego typu serwery.

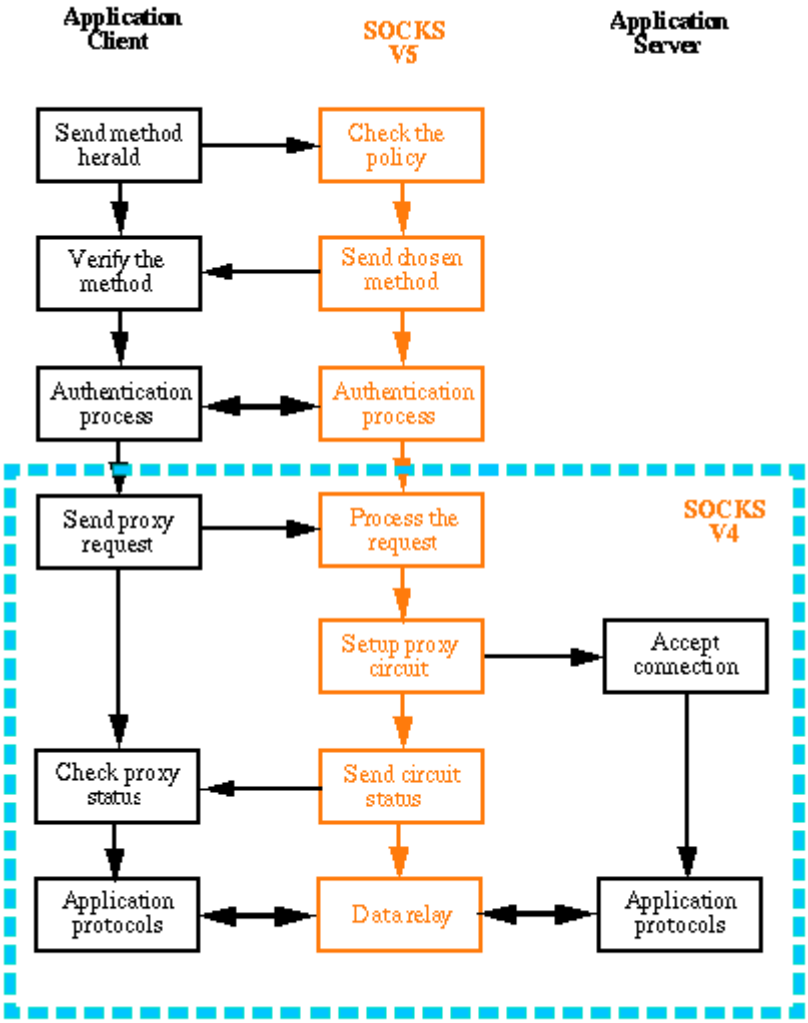
4.1. Serwer SOCKS.

Serwery SOCKS pozwalają na dostęp do wielu usług sieciowych, pozwalają także na kontrolę dostępu a także umożliwiają dostęp zza sieci w której zastosowano maskowanie IP. Pewną niedogodnością jest fakt że dostęp do usług wymaga korzystania przez maszyny klienckie ze specjalnych bibliotek. Biblioteki te działają pomiędzy warstwą aplikacji oraz warstwą transportową. Schematycznie przedstawiono to poniżej.



Kiedy komputer kliencki chce połączyć się z serwerem aplikacji (np. serwerem WWW) znajdującym się za serwerem SOCKS, łączy się z serwerem SOCKS wysyłając polecenie CONNECT w którym znajduje się adres IP zdalnego hosta oraz numer portu. Serwer SOCKS łączy się z hostem zdalnym i „obsługuje" połączenie klient – host zdalny (które notabene składa się z dwóch połączeń). Oczywiście istnieje mechanizm pozwalający na definiowanie dostępu do zdalnych hostów przez hosty sieci lokalnej. Jeśli maszyna kliencka otworzyła socket na którym oczekuje na połączenie ze strony zdalnego serwera aplikacji to do serwera SOCKS wysyłane jest polecenie BIND które informuje o tym że nastąpi połączenie ze strony serwera aplikacji, dzięki temu serwer SOCKS jest w stanie je obsłużyć. Powyższy schemat dotyczy protokołu SOCKSv4, w protokole SOCKSv5 istnieją dodatkowo mechanizmy pozwalające na

autoryzowanie połączenia. Schemat pokazujący wszystkie etapy zestawienia połączeń dla protokołów SOCKSv4 i SOCKSv5 przedstawiono poniżej:



Serwer SOCKSv4 jest często używany jako firewall pozwalający na kontrolę dostępu do Internetu z sieci lokalnej, jeśli jednak potrzebna jest większa kontrola nad dostępem do Internetu to należy zastosować protokół SOCKSv5. W protokole tym w porównaniu z wersją poprzednią wprowadzono kilka dodatkowych „features" :

- *Autoryzacja*

Autoryzacja użytkownika może odbywać się dwiema drogami:

- Poprzez przesłanie do serwera hasła oraz loginu użytkownika, po weryfikacji poprawności przesłanych danych możliwa jest obsługa połączeń danego użytkownika (identyfikowanego

przez ID). Nie jest to zalecana metoda ponieważ nazwa użytkownika i hasło przesyłane są otwartym tekstem bez szyfrowania.

- Poprzez mechanizm GSS-API (Generic Security Service Application Programming Interface), który zapewnia szyfrowanie przesyłanych danych za pomocą zewnętrznych bibliotek szyfrujących (np. Kerberos).

- *Negocjację metod autoryzacji*

Schemat negocjacji metody za pomocą której nastąpi autoryzacja użytkownika jest następujący:

1. Klient wysyła do serwera listę metod autoryzacji, które jest w stanie obsługiwać.
2. Serwer w odpowiedzi wysyła nazwę metody której klient powinien użyć. Metoda ta jest określana na podstawie polityki bezpieczeństwa zdefiniowanej dla serwera, jeśli klient przesłał metody których serwer nie akceptuje połączenie jest zrywane przez serwer.

- *Rozpoznawanie adresów symbolicznych.*

Serwer SOCKSv5 posiada również funkcjonalność serwera DNS, dlatego w poleceniach CONNECT i BIND klient może przesyłać adresy symboliczne zamiast adresów IP. Ułatwia to znacznie konfigurację serwera DNS.

- *Obsługa protokołu UDP.*

Do obsługi „połączeń" UDP serwer SOCKSv5 tworzy virtualny obwód. Istnieją dwie zasadnicze różnice pomiędzy obsługą protokołów TCP i UDP.

- „virtualny obwód" dla połączenia UDP składa się z dwóch par adres-port określających punkty końcowe „połączenia"
- nagłówki pakietów proxy UDP dokonują enkapsulacji przesyłanych danych (w tym adresu docelowego pod który ma być przesłany datagram)

Ponieważ funkcjonalność serwera SOCKSv4 „zawiera się" w funkcjonalności serwera SOCKSv5, poniższy opis dotyczył będzie serwera SOCKSv5.

4.1.1. Konfiguracja serwera SOCKS.

Konfiguracja serwera SOCKSv5 zawarta jest w pliku /etc/socks5.conf. Plik podzielony jest na sześć sekcji.

- *Sekcja hostów „zbanowanych".*

Określa hosty do których zabroniony jest dostęp, wpisy w tej sekcji mają postać:

```
ban source_host source_port
```

host źródłowy i port źródłowy muszą być określone zgodnie ze wzorcem.

- *Sekcja określająca metody autoryzacji*

Określa metody autoryzacji dla poszczególnych hostów. Wpisy tej sekcji mają postać:

```
auth source_host source_port auth_methods
```

w przypadku pominięcia tej sekcji w pliku konfiguracyjnym mechanizm autoryzacji nie jest stosowany przy obsłudze połączeń.

- *Sekcja konfiguracji interfejsów*

Jest to sekcja wymagana jedynie dla serwerów SOCKS które posiadają więcej niż jeden interfejs sieciowy (multi-homed serwers). Określa ona na którym interfejsie należy otwierać połączenie pod określony adres. Zapobiega to podszywaniu się przez hosty z Internetu pod maszyny znajdujące się w sieci lokalnej, ponadto określa interfejs którego należy użyć przy obsłudze polecenia BIND. Wpisy w tej sekcji mają postać:

```
interface hostpattern portpattern interface_address
```

hostpattern może definiować zarówno zdalny jak i lokalny adres IP, jeśli jest to adres lokalny to wpis oznacza że klient o tym adresie musi użyć określonego interfejsu przy połączeniu z serwerem SOCKS w przeciwnym wypadku połączenie jest odrzucane, jeśli zaś jest to adres zdalny to określa on interfejs pod którym dostępny jest ten adres (definiowany jest w ten sposób „ruting").

- *Sekcja zmiennych*

Pozwala nadawać wartości zmiennym, wpisy w tej sekcji mają postać:

```
set variable value
```

- *Sekcja proxy*

Określa hosty z którymi można się połączyć za pośrednictwem innych serwerów proxy. Wpisy tej sekcji mają postać:

```
proxy_type dest_host dest_port proxy_list
```

proxy_type może przybierać jedną z trzech wartości: socks4, sock5, noproxy, proxy_list określa listę serwerów proxy zgodnie ze wzorcem.

- *Sekcja kontroli dostępu*

W sekcji tej znajdują się reguły pozwalające na zdefiniowania zasad dostępu dla poszczególnych adresów. Wpisy w tej sekcji mogą mieć dwojaką postać:

```
permit auth cmd src_host dest_host src_port dest_port [user_list]
 deny auth cmd src_host dest_host src_port dest_port [user_list]
```

gdzie:

auth – określa listę metod autoryzacji koniecznych dla realizacji danego połączenia, parametr ten musi być zgodny ze wzorcem.

cmd – określa komendy jakie może wykonać klient spod adresu src_host na serwerze spod adresu dest_host, parametr ten musi być zgodny ze wzorcem.

src_host, dest_host, dest_port – określają odpowiednio adres źródłowy, adres docelowy i port

user_list – określa użytkowników dla których dostępne jest to połączenie, używane w połączeniu z parametrem auth

Wzorce używane w pliku konfiguracyjnym.

Wzorce adresów IP.

- pasuje do każdego adresu

111. pasuje do adresów podsieci 111.0.0.0/255.0.0.0

111.111. pasuje do adresów podsieci 111.111.0.0/255.255.0.0

111.111.111. pasuje do adresów podsieci 111.111.111.0/255.255.255.0

.domain.name pasuje do adresów hostów kończących się na .domain.name

host.domain.name pasuje do hosta o podanym adresie

Wzorce portów.

ftp określa port usługi FTP (port 21)

80 określa port o numerze 80

- pasuje do wszystkich numerów portów

[20,25] pasuje do numerów portów od 20 do 25 włącznie

(20,25) pasuje do numerów portów 21,22,23,24

Wzorce określające metodę autoryzacji.

n bez autoryzacji

u autoryzacja na podstawie loginu i hasła

k autoryzacja przy pomocy Kerberosa 5

- bez określenia metody autoryzacji

Wzorce poleceń.

- pasuje do wszystkich poleceń

c connect

b bind

u UDP

p ping

t traceroute

Wzorce określające użytkowników.

Aby określić użytkowników należy podać ich listę oddzieloną przecinkami bez spacji

Wzorce określające serwery proxy.

Określają demony proxy które mają zostać użyte. Należy podać ich listę oddzieloną przecinkami bez spacji.

Wzorce określające listę serwerów.

Aby określić listę serwerów należy podać ich nazwy lub adresy IP oddzielone przecinkami bez spacji, opcjonalnie przy każdym adresie po dwukropku można podać numer portu.

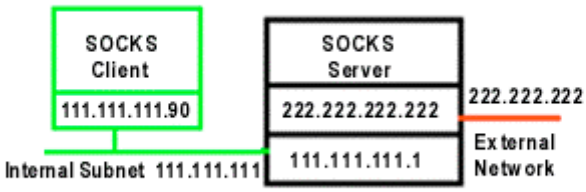
4.1.2. Konfiguracja klientów.

Jak już wspomniano wymagane są specjalne biblioteki dla maszyn klienckich, pozwalające na korzystanie z serwera SOCKS. Biblioteki te operują pomiędzy warstwą aplikacji a warstwą transportową, i są odpowiedzialne m.in. za wysyłanie poleceń CONNECT, BIND oraz za proces autoryzacji użytkownika. Możliwe są trzy drogi umożliwienia klientowi współpracy z serwerem SOCKS:

1. Kompilacja kodu klienta wraz z kodem bibliotek.
2. Użycie funkcji aplikacji pozwalającej na współpracę z serwerem. Niektóre aplikacje (np. przeglądarki WWW, ICQ) mają wbudowane funkcje pozwalające im na współpracę z serwerami SOCKS. Konfiguracja polega na wpisaniu adresu i portu serwera SOCKS (niekiedy także loginu i hasła).
3. Użycie aplikacji runsocks która umożliwia współpracę z serwerem SOCKS bez potrzeby rekompilacji.

4.1.3. Przykładowa konfiguracja.

Poniżej przedstawiona jest przykładowa konfiguracja serwera SOCKS posiadającego dwa interfejsy sieciowe (multi-homed serwer).



Konfiguracja klienta SOCKSv5.

Konfiguracja bibliotek klienta zawarta jest w pliku /etc/libsocks5.conf w tym przypadku w pliku tym powinny się znaleźć następujące dwie linijki:

```
noproxy - 111.111.111. - -
socks5 - - - - 111.111.111.1
```

pierwsza z nich określa podsieć lokalną do której dostęp jest możliwy bez korzystania z proxy, druga zawiera adres serwera proxy5 który powinien zostać użyty do połączenia z hostami w Internecie.

Konfiguracja serwera SOCKSv5.

W pliku /etc/socks2.conf serwera powinny znaleźć się wpisy:

```
auth - - -
interface 111.111.111. - 111.111.111.1
interface - - 222.222.222.222
permit - - 111.111.111. - - -
```

pierwsza linijka określa że dla żadnego połączenia nie jest stosowany mechanizm autoryzacji, druga mówi serwerowi że pod interfejsem o adresie 111.111.111.1 dostępna jest sieć 111.111.111.0/255.255.255.0 trzecia linijka definuje interfejs pod który należy kierować pozostałe pakiety (oprócz definiowania „rutingu”

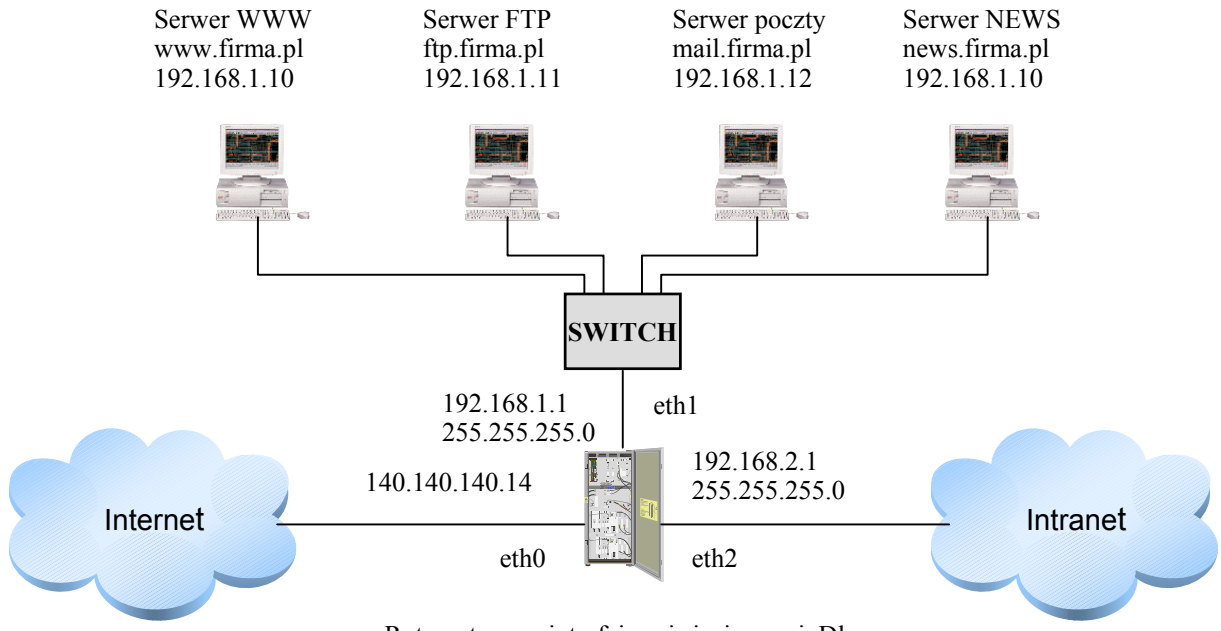
wpisy pełnią także funkcję podnoszące bezpieczeństwo jak to opisano wcześniej). Czwarta linijka zezwala na połączenia ze strony hostów sieci lokalnej. Dodatkowo komputer na którym stoi serwer SOCKS powinien być skonfigurowany tak aby nie zezwalał na rutowanie pakietów („ruting" ma odbywać się za pośrednictwem serwera a nie warstwy transportowej) oraz powinien blokować ruch sieciowy pomiędzy hostami sieci lokalnej i Internetem.

5. Przykład firewalla dla firmy.

W tym rozdziale zajmiemy się prezentacją przykładowego rozwiązania firewalla dla firmy posiadającej sieć lokalną z której powinien być możliwy dostęp do Internetu, dodatkowo firma posiada serwery WWW, FTP, NEWS i poczty elektronicznej.

5.1. Architektura

Poniżej przedstawiono proponowaną konfigurację sieci firmowej. Jak widać dla hostów sieci lokalnej jak i dla serwerów usług sieciowych zastosowano maskowanie IP.



Ruter z trzema interfejsami sieciowymi. Dla eth1 i eth2 realizowane jest maskowanie.

Architektura ta odbiega nieco od architektur przedstawionych w p. 1.1 chociaż jest ona bardzo zbliżona do przypadku z dwoma firewallami. Różnica polega tutaj na zastosowaniu maszyny o trzech interfejsach sieciowych, dzięki temu unika się redundantnego firewalla na linii Internet-Intranet. Przy projektowaniu firewalla przyjęliśmy pewne założenia dotyczące serwerów usług sieciowych. I tak:

Sewer WWW.

Zakładamy że serwerem WWW jest apache obsługujący ruch HTTP i HTTPS. Dodatkowo zakładamy możliwość otwierania socketów do serwera przez applety, do tego celu wydzielimy zakres portów.

Serwer FTP

Serwer FTP jest standardowym serwerem, zakładamy że nie korzysta z opcji szyfrowania transmisji (FTP over SSL). Z tym serwerem związany jest pewien problem. Jak zostało wspomniane wcześniej obsługa

ruchu FTP poprzez firewall sprawia kłopoty przy trybie aktywnym FTP. Rozwiązania tego problemu przedstawiono w punkcie 2.4 jednak niektóre z nich mogą nie działać w połączeniu z maskowaniem IP, dlatego warto tutaj rozpatrzyć możliwość postawienia serwera FTP na publicznym adresie IP.

Serwer poczty.

Niech demonem pocztowym działającym na serwerze poczty będzie demon korzystający z protokołu SMTP, dodatkowo zakładamy że na serwerze pocztowym działa usługa POP3.

Serwer NEWS.

Zakładamy że serwerem NEWS jest serwer NNTP.

Dodatkowo zakładamy, że na ruterze działa serwer DNS który obsługuje konwersje nazw na adresy IP zarówno dla sieci lokalnej, serwerów usług jak i internetu. Serwer ten powinien być skonfigurowany w taki sposób aby dla sieci publicznej zwracał dla wszystkich nazw serwerów usług adres 140.140.140.14, zaś dla sieci lokalnej zwracał adresy lokalne. Zastosowanie maskowania IP przynosi dodatkowe korzyści, po pierwsze za pomocą jednego publicznego adresu IP możemy „obsłużyć" całą sieć, po drugie struktura serwerów usług sieciowych jest niewidoczna z zewnątrz dodatkowo taka konfiguracja jest elastyczna istnieje np. możliwość postawienia kilku maszyn pracujących jako serwery WWW które będą widoczne pod tym samym adresem.

5.2. Konfiguracja

W tej części przedstawiona zostanie konfiguracja całego systemu. System na którym testowano konfigurację różnił się od zaproponowanego rozwiązania tym że posiadał tylko 2 interfejsy sieciowe. Wszystkie przedstawione poniżej polecenia konfiguracyjne odnoszą się do zaproponowanego systemu (z 3 interfejsami sieciowymi) przy założeniu że system jest identyczny jak ten na którym testowano ustawienia. System na którym testowano ustawienia to RedHat 6.0 z jądrem (skompilowanym) 2.4.18. Testy odbywały się w sieci lokalnej DS14.

5.2.1. Kompilacja jądra

Poniżej przedstawiono opcje dotyczące sieci i narzędzi sieciowych które wybrano przy kompilacji jądra 2.4.18 dla systemu (opcje te zapisane są w pliku .config):

```
#
# Networking options
#
CONFIG_PACKET=y
CONFIG_PACKET_MMAP=y
CONFIG_NETLINK_DEV=y
CONFIG_NETFILTER=y
CONFIG_NETFILTER_DEBUG=y
CONFIG_FILTER=y
CONFIG_UNIX=y
CONFIG_INET=y
CONFIG_IP_MULTICAST=y
CONFIG_IP_ADVANCED_ROUTER=y
CONFIG_IP_MULTIPLE_TABLES=y
```

```
CONFIG_IP_ROUTE_FWMARK=y
CONFIG_IP_ROUTE_NAT=y
CONFIG_IP_ROUTE_MULTIPATH=y
CONFIG_IP_ROUTE_TOS=y
CONFIG_IP_ROUTE_VERBOSE=y
CONFIG_IP_ROUTE_LARGE_TABLES=y
CONFIG_IP_PNP=y
CONFIG_IP_PNP_DHCP=y
CONFIG_IP_PNP_BOOTP=y
CONFIG_IP_PNP_RARP=y
CONFIG_NET_IPIP=y
CONFIG_NET_IPGRE=y
CONFIG_NET_IPGRE_BROADCAST=y
CONFIG_IP_MROUTE=y
CONFIG_IP_PIMSM_V1=y
CONFIG_IP_PIMSM_V2=y
CONFIG_ARPD=y
CONFIG_INET_ECN=y
CONFIG_SYN_COOKIES=y

#
#   IP: Netfilter Configuration
#
CONFIG_IP_NF_CONNTRACK=y
CONFIG_IP_NF_FTP=y
CONFIG_IP_NF_IRC=y
CONFIG_IP_NF_QUEUE=y
CONFIG_IP_NF_IPTABLES=y
CONFIG_IP_NF_MATCH_LIMIT=y
CONFIG_IP_NF_MATCH_MAC=y
CONFIG_IP_NF_MATCH_MARK=y
CONFIG_IP_NF_MATCH_MULTIPORT=y
CONFIG_IP_NF_MATCH_TOS=y
CONFIG_IP_NF_MATCH_AH_ESP=m
CONFIG_IP_NF_MATCH_LENGTH=y
CONFIG_IP_NF_MATCH_TTL=y
CONFIG_IP_NF_MATCH_TCPMSS=y
CONFIG_IP_NF_MATCH_STATE=y
CONFIG_IP_NF_MATCH_UNCLEAN=y
CONFIG_IP_NF_MATCH_OWNER=y
CONFIG_IP_NF_FILTER=y
CONFIG_IP_NF_TARGET_REJECT=y
CONFIG_IP_NF_TARGET_MIRROR=y
CONFIG_IP_NF_NAT=y
CONFIG_IP_NF_NAT_NEEDED=y
CONFIG_IP_NF_TARGET_MASQUERADE=y
CONFIG_IP_NF_TARGET_REDIRECT=y
CONFIG_IP_NF_NAT_SNMP_BASIC=m
CONFIG_IP_NF_NAT_IRC=y
CONFIG_IP_NF_NAT_FTP=y
CONFIG_IP_NF_MANGLE=y
CONFIG_IP_NF_TARGET_TOS=y
CONFIG_IP_NF_TARGET_MARK=y
CONFIG_IP_NF_TARGET_LOG=y
CONFIG_IP_NF_TARGET_ULOG=y
CONFIG_IP_NF_TARGET_TCPMSS=y
CONFIG_IPV6=y
```

W sekcji `Ethernet (10 or 100Mbit)` należy wybrać opcje które spowodują wkompilowanie w jądro sterowników dla kart sieciowych (można również wybrać ładowanie sterowników kart sieciowych jako

modułów). Dla kart opartych o popularnie chipsety RTL 8029 i RTL8139 (zastosowane w testowanym systemie) są to opcje (odpowiednio):

```
CONFIG_NE2K_PCI=y
CONFIG_8139CP=y
CONFIG_8139TOO=y
```

5.2.2. Instalacja kart sieciowych.

Sposób instalacji drugiej karty sieciowej zależny jest od tego czy sterowniki zostały wkompilowane w jądro czy też została wybrana opcja ładowania ich jako modułów

Sterowniki wkompilowane w jądro.

Jądro standardowo wykrywa tylko jedną kartę sieciową i aby druga karta została również wykryta należy do jądra przekazać parametr `ether=irq, base_addr, name`. Gdzie irq jest numerem przerwania pod którym karta ma być obsługiwana, base_addr jest adresem początkowym zakresu pamięci przydzielonej karcie a name jest nazwą interfejsu (standardowo jest to nazwa postaci ethN). Parametr można przekazać do jądra poprzez dodanie do pliku /etc/lilo.conf linijki:

```
append=" ether=irq, base_addr, name"
```

po dodaniu tej linijki należy „przeinstalować" lilo poprzez wydanie komendy lilo.

Sterowniki ładowane jako moduły.

W przypadku sterowników ładowanych jako moduły przypisanie kartom adresów i przerwań odbywa się poprzez wpis w pliku /etc/modules.conf, wpisy te mają postać:

```
alias eth0 ne2k-pci
alias eth1 8139cp
options ne2k-pci irq=5 io=0xe400
options 8139 irq=7 io=0xe800
```

Nadawanie interfejsom sieciowym numerów IP odbywa się z użyciem narzędzia *ifconfig*.

5.2.3. Definiowanie rutingu.

Aby możliwe było poprawne przekazywanie pakietów pomiędzy interfejsami konieczne jest zdefiniowanie rutingu. Służy do tego narzędzie *route*. Polecenia konfigurujące ruting znajdują się w pliku /etc/sysconfig/network dla naszego systemu powinny się tam znaleźć odpowiednie wpisy (w testowanym systemie w pliku tym były polecenia definiujące ruting z tym że wymagały one modyfikacji gdyż otrzymana w wyniku ich działania tablica rutingu była nieprawidłowa):

```
# ruting dla adresu lokalnego
route add 127.0.0.1 dev lo
# siec zawierajaca  serwery uslug sieciowych
route add -net 192.168.1.0 netmask 255.255.255.0 dev eth1
# intranet
route add -net 192.168.2.0 netmask 255.255.255.0 dev eth2
# siec zewnetrzna
route add -net 140.140.0.0 netmask 255.255.0.0 dev eth0
# „wyjscie na swiat" przez 140.140.1.1
route add -net default gw 140.140.1.1 dev eth0
```

Dodatkowo należy zezwolić na forwarding pakietów przez jądro (defaultowo opcja ta jest wyłączona), dokonuje się tego poleceniem:

```
echo ‘1’ >> /proc/sys/net/ipv4/ip_forward
```

Polecenie to należy umieścić w dowolnym skrypcie uruchamianym podczas startu systemu (np. w skrypcie definiującym reguły firewalla).

5.2.4. Konfigurowanie firewalli.

Poniżej przedstawiono skrypt konfiguracyjny firewalla, skrypt ten umieszczony został w katalogu /etc/rc.d/init.d, aby skrypt ten był wywoływany podczas startu systemu na odpowiednim poziomie należy w odpowiednim katalogu umieścić link do niego, najczęściej jest to poziom 3 w związku z tym jest to katalog /etc/rc.d/rc3.d

```
#! /bin/sh

# nasze sieci
WEB_SERVERS="192.168.1.0/24"
INTRANET="192.168.2.0/24"
GATEWAY_IP="140.140.140.14"

# nasze serwery
WWW_SERVER="192.168.1.10"
WWW_SERVER_PORTS="80,443"

FTP_SERVER="192.168.1.11"
FTP_SERVER_PORTS="20,21"

MAIL_SERVER="192.168.1.12"
MAIL_SERVER_PORTS="25,110"

NEWS_SERVER="192.168.1.13"
NEWS_SERVER_PORTS="119,563"

# porty gatewaya na których ruch ma być przepuszczany
GATEWAY_TCP_PORTS="22,23,53"
GATEWAY_UDP_PORTS="53"

MASQUERADE_LOG_SYN=" MASQUERADE_LOG_SYN "
MASQUERADE_LOG_FIN=" MASQUERADE_LOG_FIN "
MASQ_INFO="Setting masquerade and firewall rules"

#####################################################################

# skrypt zawierający funkcje biblioteczne
. /etc/rc.d/init.d/functions

# 1 zezwalamy na forwarding pakietów
echo '1' >> /proc/sys/net/ipv4/ip_forward


# 2 pozwalamy na ruch do gatewaya na zdefiniowanych wcześniej portach
# zezwalamy także na obsługę pakietów ICMP
iptables -P INPUT DROP
iptables -A INPUT -m multiport -p tcp -d "$GATEWAY_IP" --dport
"$GATEWAY_TCP_PORTS" -j ACCEPT
iptables -A INPUT -m multiport -p udp -d "$GATEWAY_IP" --dport
"$GATEWAY_UDP_PORTS" -j ACCEPT
iptables -A INPUT -m icmp -p icmp -d "$GATEWAY_IP" -j ACCEPT
```



```
# 3 zabezpieczamy się przed skanowaniem portów przy użyciu najczęściej
# stosowanych technik, logujemy próby skanowania
iptables -t nat -A PREROUTING -p tcp --tcp-flags ALL FIN,URG,PSH
-j LOG --log-prefix "XMAS PORT SCAN: " --log-level warning
iptables -t nat -A PREROUTING -p tcp --tcp-flags ALL NONE
-j LOG --log-prefix "NULL PORT SCAN: " --log-level warning
iptables -t nat -A PREROUTING -p tcp --tcp-flags SYN,RST SYN,RST
-j LOG --log-prefix "SYN/RST PORT SCAN: " --log-level warning
iptables -t nat -A PREROUTING -p tcp --tcp-flags SYN,FIN SYN,FIN
-j LOG --log-prefix "SYN/FIN PORT SCAN: " --log-level warning

iptables -t nat -A PREROUTING -p tcp --tcp-flags ALL FIN,URG,PSH -j DROP
iptables -t nat -A PREROUTING -p tcp --tcp-flags ALL NONE -j DROP
iptables -t nat -A PREROUTING -p tcp --tcp-flags SYN,RST SYN,RST -j DROP
iptables -t nat -A PREROUTING -p tcp --tcp-flags SYN,FIN SYN,FIN -j DROP


# 4 zabezpieczamy się przed podszywaniem się przez hosty z zewnątrz pod
# hosty z naszej sieci
iptables -t nat -A PREROUTING -s "$INTRANET" -i eth0 -j DROP
iptables -t nat -A PREROUTING -s "$WEB_SERVERS" -i eth0 -j DROP

# 5 odrzucamy wszystkie pakiety z Ip różnym od IP naszego gatewaya
# przychodzące na interfejs eth0
iptables -t nat -A PREROUTING -d !"$GATEWAY_IP" -i eth0 -j DROP

# 6 przekierowujemy pakiety skierowane do gatewaya na porty odpowiadające
# usługom oferowanym przez nasze serwery do serwerów, dzięki temu
# serwery są „widoczne” w Internecie pomimo że są za maskaradą
iptables -t nat -A PREROUTING -m multiport -p tcp -d "$GATEWAY_IP" --dport
"$WWW_SERWER_PORTS" -j DNAT --to-destination "$WWW_SERVER"
iptables -t nat -A PREROUTING -m multiport -p tcp -d "$GATEWAY_IP" --dport
"$FTP_SERWER_PORTS" -j DNAT --to-destination "$FTP_SERVER"
iptables -t nat -A PREROUTING -m multiport -p tcp -d "$GATEWAY_IP" --dport
"$MAIL_SERWER_PORTS" -j DNAT --to-destination "$MAIL_SERVER"
iptables -t nat -A PREROUTING -m multiport -p tcp -d "$GATEWAY_IP" --dport
"$NEWS_SERWER_PORTS" -j DNAT --to-destination "$NEWS_SERVER"

# 7 ustawiamy logowanie pakietów przesyłanych pomiędzy naszą siecią
# lokalną a Internetem, logujemy pakiety otwierające połączenie (z usatwionym
# bitem SYN) oraz zamykające je (z ustawionym bitem FIN)
iptables -A FORWARD -m tcp -p tcp -s "$INTRANET" -o eth0 --tcp-flags
SYN,RST,ACK SYN -j LOG --log-level info --log-prefix "$MASQUERADE_LOG_SYN"
iptables -A FORWARD -m tcp -p tcp -s "$INTRANET" -o eth0 --tcp-flags
FIN,RST,ACK FIN,ACK -j LOG --log-level info --log-prefix "$MASQUERADE_LOG_FIN"

# 8 ustawiamy maskaradę
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

success "$MASQ_INFO"
```

W powyższym pliku najpierw zezwalamy na forwarding pakietów IP (sekcja 1). Potem ustalamy reguły dotyczące pakietów przeznaczonych dla gatewaya (sekcja 2).Reguły te nie dotyczą pakietów które przekierowaliśmy do serwerów usług sieciowych, gdyż zgodnie ze schematem z pkt. 2.3 pakiety te nie są przetwarzane przez reguły łańcucha INPUT. Zezwalamy jedynie na połączenia via telnet i ssh do naszego serwera oraz na ruch pakietów ICMP i obsługę DNS. W wersji 1.1.1 iptables, którą posługiwaliśmy się w naszym systemie był błąd polegający na tym że przy powyższych ustawieniach nie były przepuszczane pakiety ICMP echo reply. Reguły zgrupowane w sekcji 3 zabezpieczają gateway przed skanem portów, przy użyciu najczęściej stosowanych technik. Pakiety odpowiadające tym technikom są logowane i odrzucane. W sekcji 4 zabezpieczamy się przed podszywaniem się przez hosty z Internetu pod hosty z naszej sieci. W

sekcji 6 odrzucamy wszystkie pakiety z IP innym niż IP naszego gatewaya przychodzące na interfejs eth0. Następnie przekierowujemy pakiety skierowane na porty odpowiadające usługom sieciowym które oferują nasze serwery do tych serwerów. Dzięki temu serwery te są dostępne w Internecie pomimo tego że znajdują się za maskaradą. W sekcji 7 znajdują się reguły odpowiedzialne za logowanie połączeń z sieci lokalnej do Internetu. Logujemy pakiety otwierające połączenie (z ustawionym bitem SYN) i zamykające połączenie (z ustawionymi bitami FIN i ACK). W sekcji 8 ustawiamy maskowanie dla pakietów skierowanych do interfejsu eth0.

6. Skrypt analizujący logi z maskowania pod systemem Linux.

Poniżej przedstawiono fragment przykładowy pliku zawierającego logi z maskowania w systemie. W przedstawionym uprzednio pliku konfiguracyjnym firewalla ustawiliśmy logowanie pakietów otwierających i zamykających połączenia Intranet-Internet (pakiety te oznaczone są odpowiednio jako MASQUERADE_LOG_SYN i MASQUERADE_LOG_FIN).

```
May 26 02:30:16 optyk masquerade: Setting masquerade and firewall rules
succeeded
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.200 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=911 DF
PROTO=TCP SPT=1107 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.110 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=920 DF
PROTO=TCP SPT=1108 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_FIN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.110 LEN=40 TOS=0x00 PREC=0x00 TTL=127 ID=925 DF
PROTO=TCP SPT=1108 DPT=80 WINDOW=17345 RES=0x00 ACK FIN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.131.42 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=931 DF
PROTO=TCP SPT=1109 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.110 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=936 DF
PROTO=TCP SPT=1110 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_FIN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.110 LEN=40 TOS=0x00 PREC=0x00 TTL=127 ID=944 DF
PROTO=TCP SPT=1110 DPT=80 WINDOW=17249 RES=0x00 ACK FIN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.200 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=945 DF
PROTO=TCP SPT=1111 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.200 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=949 DF
PROTO=TCP SPT=1112 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:51 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.200 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=960 DF
PROTO=TCP SPT=1113 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:52 optyk kernel:  MASQUERADE_LOG_FIN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.131.42 LEN=40 TOS=0x00 PREC=0x00 TTL=127 ID=971 DF
PROTO=TCP SPT=1109 DPT=80 WINDOW=17121 RES=0x00 ACK FIN URGP=0
May 26 02:30:52 optyk kernel:  MASQUERADE_LOG_SYN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.110 LEN=48 TOS=0x00 PREC=0x00 TTL=127 ID=978 DF
PROTO=TCP SPT=1114 DPT=80 WINDOW=16384 RES=0x00 SYN URGP=0
May 26 02:30:52 optyk kernel:  MASQUERADE_LOG_FIN IN=eth1 OUT=eth0
SRC=192.168.1.2 DST=213.180.130.110 LEN=40 TOS=0x00 PREC=0x00 TTL=127 ID=985 DF
PROTO=TCP SPT=1114 DPT=80 WINDOW=17249 RES=0x00 ACK FIN URGP=0
```

Skrypt analizujący logi napisany został w języku Perl. Skrypt parsuje plik zawierający logi jądra i generuje statystykę połączeń. Plik do którego trafiają logi jądra definiowany jest w pliku /etc/syslog.conf . skrypt traktuje połączenia pomiędzy którymi nie upłynęło więcej niż 5 minut jako jedno połączenie. Ponadto można wybrać opcję wyświetlania adresów zdalnych hostów jako numerów IP bądź nazw (wywołanie skryptu z parametrem –n powoduje wyświetlanie nazw, domyślnie wyświetlane są numery IP).

Poniżej przedstawiono kod źródłowy skryptu:

```
#! /usr/bin/perl

use Socket;
use Time::Local qw(timelocal);

$log_file="/var/log/info_messages"

# przedział czasowy (w sekundach) między którym połaczenai do tego samego hosta
# zdalnego na ten sam port są traktowane jako jedno połączenie
$conn_interval = 300;

#określa czy zdalne hosty mają by wyświetlane jako nazwy czy jako adresy IP
$printbyNames = 0;

# tablica hashujaca zawierająca połączania
%connections = ();

# funkcja wyswietlająca wynik działania skryptu
sub printConnections {
    print "Connections:\n";
    my @localIPs=keys(%connections);
    foreach  $localIP (@localIPs) {
       print "local IP: $localIP\n";
       my $oneIPconnnections = $connections{$localIP};
       my @remoteIPs=keys(%$oneIPconnnections);
       foreach  $remoteIP (@remoteIPs) {
          if ( $printbyNames ) {
             my $addr = gethostbyaddr(inet_aton($remoteIP), AF_INET);
             print "  remote host: $addr\n";
          } else {
             print "  remote IP: $remoteIP\n";
          }
          foreach $onePortConnections ($oneIPconnnections->{$remoteIP}) {
             my @remotePorts=keys(%$onePortConnections);
             foreach $port (@remotePorts) {
                print "    remote port: $port\n";
                foreach $onePortConnection ($onePortConnections->{$port}) {
                   foreach $conn (@$onePortConnection) {
                      print "      start:  $conn->{'startDay'}
                                          $conn->{'startMonth'}
                                          $conn->{'startHour'}:
                                          $conn->{'startMin'}:
                                          $conn->{'startSec'}
                                end:      $conn->{'endDay'}
                                          $conn->{'endMonth'}
                                          $conn->{'endHour'}:
                                          $conn->{'endMin'}:
                                          $conn->{'endSec'}\n";
                   }
                }
             }
          }
       }
    }
}

# delta {day1, month1, hour1, min1, sec1, day2, month2, hour2, min2, sec2}
# funkcja obliczająca odstęp pomiędzy podanymi czasami
# wynik zwracany jest w sekundach
sub delta {
    $t1 = timelocal(@_[4], @_[3], @_[2], @_[0], @_[1]);
    $t2 = timelocal(@_[9], @_[8], @_[7], @_[5], @_[6]);
    $t2 - $t1;
}
```



```
# sprawdzamy czy skrypt wywołano z parametrem -n, jeśli tak ustawiamy
# wyświetlanie nazw zamiast adresów IP
if ($ARGV[0] eq "-n") {
    $printbyNames = 1;
}

if (!(open(FILE,"$log_file"))){
    print "\nERROR: Can't open file $log_file \n\n";
}else{
    while ($row=<FILE>) {
       # parsujemy pojedyncza linijkę pliku
       if ($row =~ m/([^ ]+) +([^ ]+) +(..):(..):(..).*MASQUERADE_LOG_(...).*SRC=(\S*)
                                          DST=(\S*) .*SPT=(\S*) DPT=(\S*) .*/ ){
          $day      = $2;
          $month    = $1;
          $hour     = $3;
          $min      = $4;
          $sec      = $5;
          $srcIP    = $7;
          $dstIP    = $8;
          $srcPORT  = $9;
          $dstPORT  = $10;
          $connType = $6;

          if ($connType eq "SYN") {
             # pakiet SYN : otwarcie połączenia
             if ($connections{$srcIP}->{$dstIP}->{$dstPORT}) {
                my $onePortConnections = $connections{$srcIP}->
                                                    {$dstIP}->
                                                    {$dstPORT};
                foreach $conn (@$onePortConnections) {
                   my $del = delta( $conn->{"startDay"},
                                    $conn->{"startMonth"},
                                    $conn->{"startHour"},
                                    $conn->{"startMin"},
                                    $conn->{"startSec"},
                                    $day,
                                    $month,
                                    $hour,
                                    $min,
                                    $sec);
                   if ($del < $conn_interval) {
                      # jeśli znaleźliśmy połączenie otwarte nie wcześniej
                      # niż 5 minut temu anulujemy jego zamknięcie
                      $conn->{"endDay"}   = "-1";
                      $conn->{"endMonth"} = "-1";
                      $conn->{"endHour"}  = "-1";
                      $conn->{"endMin"}   = "-1";
                      $conn->{"endSec"}   = "-1";
                      last;
                   } else {
                      # logujemy nowe połączenie
                      my $connection = {"startDay"   => $day,
                                        "startMonth" => $month,
                                        "startHour"  => $hour,
                                        "startMin"   => $min,
                                        "startSec"   => $sec,
                                        "endDay"     => "-1",
                                        "endMonth"   => "-1",
                                        "endHour"    => "-1",
                                        "endMin"     => "-1",
                                        "endSec"     => "-1"};
                      $len = ++$#{@{$connections{$srcIP}->
                                                 {$dstIP}->
                                                 {$dstPORT}}};
                      $connections{$srcIP}->
```

```
                                    {$dstIP}->
                                    {$dstPORT}->[$len] = $connection;
                        last;
                                                        }
                    }
                } else {
                    my $connection = {"startDay"   => $day,
                                      "startMonth" => $month,
                                      "startHour"  => $hour,
                                      "startMin"   => $min,
                                      "startSec"   => $sec,
                                      "endDay"     => "-1",
                                      "endMonth"   => "-1",
                                      "endHour"    => "-1",
                                      "endMin"     => "-1",
                                      "endSec"     => "-1"};
                    $connections{$srcIP}->{$dstIP}->{$dstPORT} = [$connection];
                }
            } else {
                # pakiet FIN: połączenie jest zamykane
                my $onePortConnections = $connections{$srcIP}->
                                                     {$dstIP}->
                                                     {$dstPORT};
                # szukamy nie zamkniętego połączenia i zamykamy go
                foreach $conn (@$onePortConnections) {
                    if ($conn->{"endDay"} == "-1") {
                        $conn->{"endDay"}   = $day;
                        $conn->{"endMonth"} = $month;
                        $conn->{"endHour"}  = $hour;
                        $conn->{"endMin"}   = $min;
                        $conn->{"endSec"}   = $sec;
                    }
                }
            }
        }
    }
    print "\n\n";
    printConnections;
    close FILE;
}
```

Rezultat działania skryptu wywołanego z parametrem –n przedstawiono poniżej:

```
Connections:
local IP: 192.168.1.2
  remote host: onet.hit.gemius.pl
    remote port: 80
      start: 26 May 02:30:51 end: 26 May 02:30:52
  remote host: f7virt.onet.pl
    remote port: 80
      start: 26 May 02:30:51 end: 26 May 02:30:52
  remote host: f1virt.onet.pl
    remote port: 80
      start: 26 May 02:30:51 end: -1 -1 -1:-1:-1
```

Wartości –1 w ostatniej linijce oznaczają że połączenie nie zostało jeszcze (w momencie uruchomienai skryptu) zamknięte.
W skrypcie wykorzystany został mechanizm wyrażeń regularnych Perla. Wykorzystany został także mechanizm tworzenia anonimowych hashy i tablic. Wszystkie połączenia trzymane są w tablicy hashującej hashowanej po lokalnych numerach IP która zawiera odwołania do tablic hashujących zawierających połączenia których dokonano z danego adresu IP. Tablice te z kolei hashowane są po numerach IP zdalnych hostów z którymi się łączono a zawierają odwołania do tablic hasujących hashowanych po numerach portów i zawierających odwołania do tablic zawierających połączenia do danego portu. Każda komórka takiej tabeli zawiera odwołanie do hasha będącego „rekordem” reprezentującym pojedyncze połączenie.

7. Bibliografia

[1] Olaf Kirch, Terry Dawson, LINUX Podręcznik administratora sieci, O'Reilly 2000

[2] Materiały dostępne na http://www.jtz.org.pl/tlumaczenia.html

[3] Materiały dostępne na http://www.netfilter.org

[4] Materiały dostępne na http://www.socks.nec.com