helion.pl

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Sieci komputerowe. Kompendium



Autor: Karol Krysiak
ISBN: 83-7197-942-8
Format: B5, stron: około 448

- Omówienie teoretycznych podstaw sieci komputerowych
- Szczegółowy opis działania sieci opartych na TCP/IP i innych protokołach
- Najnowsze technologie: sieci bezprzewodowe, protokół IPv6
- Praktyczne wskazówki dotyczące zabezpieczania sieci
- Omówienie wszystkich popularnych usług sieciowych: od HTTP do SNMP

Często zdarza się, że administrator sieci rozpoczynając swoją pierwszą pracę nie jest do niej przygotowany. Jego wiedza jest fragmentaryczna i bardzo teoretyczna, a zetknięcie z rzeczywistymi potrzebami okazuje się dużym zaskoczeniem. Musi szybko uzupełnić braki w swoich wiadomościach, uporządkować je i zdobyć narzędzia do rozwiązywania napotkanych problemów.

Książka, którą trzymasz w ręku, zawiera informacje, które umożliwią Ci szybkie przygotowanie się do pełnienia obowiązków administratora sieci. To źródło informacji, do którego zawsze będziesz mógł wrócić, aby przypomnieć sobie zasadę działania i właściwości charakterystyczne dla wprowadzanej w twojej sieci technologii. Nie jest bowiem prawdą, że administrator sieci musi znać na pamięć całe tomy parametrów sieciowych, możliwe sposoby konfiguracji. Musi mieć źródło, w którym znajdzie potrzebne informacje. Takim źródłem stanie się dla Ciebie ta książka, opisująca m.in.:

- Podstawowe narzędzia administratora sieci
- Topologie i modele budowy sieci
- Najważniejsze technologie stosowane przy budowie sieci
- Sieci światłowodowe i bezprzewodowe
- Standard Ethernet
- Protokoły warstwy internetowej, adresowanie i routing, IPv6
- Usługi warstwy aplikacji: DNS, SMTP, POP, IMAP, FTP, HTTP, SSL, Telnet, SSH i inne
- Protokoły Token Ring, FDDI, IPX, ISDN, PPP, xDSL, Frame Relay, ATM, sieci oparte na telewizji kablowej
- Zagadnienia związane z administracją sieciami LAN
- Sposoby zabezpieczania sieci komputerowych, konfigurację zapór sieciowych i postępowanie w razie wykrycia włamania

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Wstęp 11

Rozdział 1. Sieci komputerowe 23
1.1. Podział sieci komputerowych w zależności od rozmiaru 23
1.2. Topologie sieci komputerowych 24
Topologia sieci 24
Topologia fizyczna 24
Topologia logiczna 25
1.3. Model ISO/OSI 26
1.4. Model protokołu TCP/IP 28

Rozdział 2. Warstwa dostępu do sieci — rodzaje nośników 33
2.1. Najważniejsze technologie 33
2.2. Przewód koncentryczny 35
Zastosowania sieci 10Base-2 38
2.3. Skrętka UTP 39
Wymagania dla instalacji spełniającej założenia CAT-5 44
2.4. Światłowód 45
Budowa światłowodu 45
Zasada działania światłowodu 46
Światłowód wielomodowy 46
Światłowód jednomodowy 47
Złącza światłowodowe 47
Standardy transmisji światłowodowych 49
2.5. Sieci bezprzewodowe. Wireless LAN — standard 802.11 49

Rozdział 3. Warstwa dostępu do sieci — standard Ethernet 59
3.1. Historia 59
3.2. Działanie protokołu 59
Metody transmisji 59
Norma IEEE 802.3 60
Wydajność sieci Ethernet 10 Mb/s 63
3.3. Budowa ramki Ethernet 64
Protokół LLC 66
3.4. Zasady konstruowania sieci Ethernet 68
Reguły dla Ethernetu (10 Mb/s) 68
Reguły dla Fast Ethernetu (100 Mb/s) 72
Reguły dla Gigabit Ethernetu (1 000 Mb/s) 73

3.5. Technologie ... 73
Full-duplex ... 73
MAC Control ... 74
Automatyczne negocjowanie parametrów łącza ... 75
1000Base-T ... 76
VLAN ... 76
Sygnały i kodowanie ... 80
3.6. Protokół ARP — protokół określania adresów ... 82
Proxy-ARP ... 83
Reverse-ARP ... 83
Zapobieganie zdublowaniu adresów IP ... 84
Pakiet protokołu ARP ... 84
Polecenia do manipulacji tablicą ARP ... 86
3.7. Urządzenia sieciowe działające w warstwie dostępu do sieci ... 88
Karta sieciowa ... 88
Modem ... 89
Transceiver ... 89
Konwerter nośników ... 90
Regenerator (repeater) ... 90
Koncentrator (hub) ... 90
Most (bridge) ... 94
Przełącznik (switch) ... 95

Rozdział 4. Warstwa Internetu ... 99
4.1. Protokół IP ... 99
Zadania spełniane przez protokół IP ... 100
Cechy protokołu IP ... 100
Budowa datagramu IP ... 100
4.2. Adresowanie IP ... 103
Klasy adresów w TCP/IP ... 107
Bezklasowe routowanie międzydomenowe (CIDR) ... 107
Adresy specjalne i klasy nieroutowalne ... 112
Nadawanie adresów IP interfejsowi sieciowemu ... 113
4.3. Routowanie datagramów IP ... 117
Tablica routingu ... 120
Polecenia służące do manipulacji tablicą routingu ... 122
Routing źródłowy ... 124
4.4. Protokół ICMP ... 126
Zadania protokołu ICMP ... 126
Format nagłówka ICMP ... 127
Pola Typ i Kod komunikatu ICMP ... 128
Polecenia wykorzystujące protokół ICMP ... 131
4.5. IPv6 — wersja szósta protokołu IP ... 134
Nagłówek IPv6 ... 135
Adres IPv6 ... 136
4.6. Urządzenia pracujące w warstwie Internetu ... 137
Router ... 137

Rozdział 5. Warstwa transportowa ... 139
5.1. Port, gniazdo ... 139
5.2. Protokół UDP ... 141
5.3. Protokół TCP ... 142
tcpdump ... 145
netstat ... 150

Rozdział 6. Usługi warstwy aplikacji ... 155
6.1. DNS ... 155
Rejestrowanie własnej domeny ... 158
Ogólne informacje o serwerach DNS ... 159
Jak to w rzeczywistości działa? ... 160
Konfiguracja hosta ... 162
Rekordy zasobów ... 165
Serwery DNS ... 167
Konfiguracja serwera BIND ... 167
Sterowanie demonem named ... 175
Kwestie bezpieczeństwa ... 179
Format komunikatu DNS ... 182
Programy użytkowe — diagnostyka ... 184
6.2. SMTP ... 192
Serwery SMTP ... 193
Sprawdzanie działania serwera ... 195
Protokół MIME ... 201
Bezpieczeństwo ... 203
6.3. POP ... 204
Sprawdzanie działania serwera ... 205
Serwery POP ... 206
6.4. IMAP ... 207
Sprawdzanie działania serwera ... 207
Serwery IMAP ... 209
6.5. FTP ... 210
Tryby pracy FTP ... 210
Komunikacja z serwerem ... 212
Obsługa programu ftp ... 214
Serwery ... 215
Bezpieczeństwo ... 216
6.6. HTTP ... 217
Protokół HTTP ... 217
Sprawdzanie działania serwera HTTP ... 218
Serwery ... 219
Bezpieczeństwo ... 221
6.7. SSL ... 222
Certyfikaty ... 223
Uproszczona zasada działania SSL ... 223
Długość klucza ... 224
Wykorzystanie pakietu stunnel ... 224
6.8. Telnet ... 225
6.9. SSH ... 225
6.10. Finger ... 227
6.11. Auth ... 227
6.12. NNTP ... 228
6.13. SNMP ... 228
Różnice pomiędzy wersjami SNMP ... 231
Bezpieczeństwo ... 231
6.14. IRC ... 232
6.15. Whois ... 232
6.16. NTP ... 235
6.17. Syslog ... 237

6.18. Bootps, DHCP ... 238
Nagłówek DHCP ... 239
Proces uzyskiwania konfiguracji ... 241
Konfiguracja klientów DHCP ... 242
Serwery DHCP ... 244
6.19. NetBIOS ... 248
Wyszukiwanie nazw NetBIOS ... 250
Optymalizacja ... 251
Bezpieczeństwo ... 252
6.20. Urządzenia sieciowe pracujące w warstwie aplikacji ... 252
Komputer ... 252
Serwer ... 253

Rozdział 7. Inne protokoły ... 255
7.1. Token Ring ... 255
7.2. FDDI ... 256
7.3. IPX/SPX ... 258
Budowa pakietu IPX ... 259
Adresy IPX ... 261
Protokoły używane w IPX ... 261
7.4. ISDN ... 262
7.5. PPP ... 265
Ramka PPP ... 266
Dodatkowe możliwości PPP ... 267
Konfiguracja PPP w Linuksie ... 267
7.6. xDSL ... 273
ADSL ... 274
RADSL ... 275
SDSL ... 275
HDSL ... 276
VDSL ... 276
7.7. Frame Relay ... 276
Opis technologii Frame Relay ... 276
Zasada działania FR ... 277
Format ramki Frame Relay ... 279
Mechanizmy sieci FR ... 279
Parametry transmisji FR ... 282
7.8. ATM ... 283
Właściwości standardu ATM ... 283
Interfejsy ATM ... 284
Rodzaje połączeń w sieciach ATM ... 284
Komórka ATM ... 285
Usługi ATM ... 286
Model ATM ... 287
Klasy ruchu ... 288
Trasowanie ATM ... 290
Dodatkowe możliwości sieci ATM ... 290
7.9. Sieci w gniazdku zasilającym — PLC ... 291
Topologia sieci PLC ... 291
Standardy PLC ... 292
Wady PLC ... 293
7.10. Sieci telewizji kablowych ... 294
Standard MCSN/DOCSIS ... 294

Rozdział 8. Administracja siecią LAN ... 297
8.1. Projektowanie sieci LAN ... 298
Struktura fizyczna sieci ... 298
Struktura logiczna sieci ... 301
8.2. Rozwiązywanie problemów ... 303
Poważna awaria ... 303
Użytkownik ... 304
Rady ... 304
Problemy ... 306
8.3. Narzędzia administratora sieci ... 308
Sniffery ... 310
Analizatory sieci ... 315
SNMP ... 322
Inne ... 322
Testowanie dostępności usług ... 324
Skanery bezpieczeństwa ... 325
Inne narzędzia ... 331
8.4. Wykorzystanie protokołu SNMP ... 331
Konfiguracja agenta snmpd ... 332
Konfiguracja menedżera MRTG ... 333
8.5. Zarządzalne urządzenia aktywne ... 336

Rozdział 9. Bezpieczeństwo ... 341
9.1. Polityka bezpieczeństwa ... 342
9.2. Najważniejsze pojęcia ... 345
9.3. Konstrukcja sieci ... 357
9.4. Rozpoznanie terenu ... 363
Zbieranie danych ... 363
Skanowanie ... 365
Metody ukrywania skanowania ... 369
Identyfikacja systemu operacyjnego ... 372
9.5. Metody włamań ... 377
Uzyskanie dostępu ... 378
Destabilizacja pracy ... 386
9.6. Zagrożenia wewnętrzne ... 390
Wykrywanie snifferów ... 391
Sposoby omijania przełączników ... 392
Zasoby ... 395
9.7. Podsumowanie ... 396
Zachowanie podczas włamania ... 396

Rozdział 10. Firewall ... 399
10.1. Rodzaje firewalli ... 399
Tradycyjne proxy (Traditional proxies) ... 399
Przezroczyste proxy (Transparent proxies) ... 400
Tłumaczenie adresów IP (NAT) ... 400
Filtrowanie pakietów ... 400
10.2. Obsługa filtrowania pakietów w Linuksie ... 401
Ipchains — Linux 2.2 ... 402
Składnia polecenia ipchains ... 403
Iptables — Linux 2.4 ... 405
10.3. Tworzymy firewall ... 407
Podstawy ... 408
Konfiguracja ... 409

Logi systemowe ... 416
Problemy z działaniem firewalla ... 416
Wyłączanie firewalla ... 416
10.4. Dodatkowe funkcje ... 417
Ograniczenia na ICMP ... 417
Jak przepuścić nową usługę na przykładzie Direct Connect ... 419
Ustawianie priorytetów pakietów ... 419
Wykrywanie skanowania za pomocą firewalla ... 421
Wykrywanie NAT ... 422

Skorowidz ... 425

Rozdział 10.

Firewall

W tym rozdziale zajmę się sposobami kompleksowego zabezpieczenia sieci komputerowej za pomocą techniki **filtrowania datagramów** (*firewall*) oraz **tłumaczenia adresów sieciowych** NAT (*Network Address Translation*). Nie przedstawię tutaj gotowego rozwiązania, ale postaram się, abyś zrozumiał, w jaki sposób możesz stworzyć coś takiego dla swojej sieci. Uważam, że absurdem jest ściągnięcie z sieci gotowego skryptu i uruchomienie go. Musisz rozumieć, co robi twój firewall i jak działa — lepiej, aby był prosty, ale żebyś rozumiał dokładnie jego działanie.

Każdy firewall jest inny, tak jak każda sieć jest inaczej zbudowana i różne są charaktery i poziomy wiedzy ich administratorów. Twój firewall rozwija się wraz z tobą, wraz ze zwiększaniem się twojej wiedzy o zagrożeniach i metodach przeciwdziałania im. Zmienia się też wraz z wykrywaniem coraz nowszych zagrożeń. Jest to żywy organizm stanowiący pierwszą linię ochrony dla twojej sieci, poświęć więc mu trochę czasu i zaangażowania.

10.1. Rodzaje firewalli

Poniżej przedstawię bardziej dokładny podział firewalli niż podany w rozdziale 9. Opiszę również ich cechy, na podstawie których jako administrator sieci będziesz wybierał odpowiedni rodzaj do potrzebnego ci zastosowania. Oczywiście dobór konkretnego firewalla z dostępnych na rynku pozostawiam już tobie, zwłaszcza że tryb wydawania książki jest długi, a zmiany możliwości produktów dostępnych na rynku następują bardzo szybko.

Tradycyjne proxy (Traditional proxies)

Jest to rodzaj firewalla pośredniczącego; pakiety z sieci prywatnej nigdy nie wychodzą do Internetu i *vice versa*. Adresy IP w sieci prywatnej powinny być z klas nieroutowalnych. Jedynym sposobem połączenia się z Internetem jest wywołanie firewalla, ponieważ jest on jedyną maszyną mogącą łączyć się równocześnie z obiema sieciami. Urucha-

miamy jest na nim program zwany *proxy*, który tego dokonuje. Dla każdej usługi, która ma być dostępna z Internetu, na firewallu musi być uruchomiony osobny program pośredniczący w jej świadczeniu.

Komputery w sieci wewnętrznej muszą być specjalnie skonfigurowane w celu uzyskania dostępu do wybranych usług. Przykładowo, aby ściągnąć stronę WWW, muszą połączyć się z firewallem na port 8080, zalogować się i zażądać potrzebnej strony. W tym momencie uruchamia się odpowiedni program pośredniczący, ściąga potrzebne dane i przekazuje do odpowiedniego komputera w sieci lokalnej.

Przezroczyste proxy (Transparent proxies)

Jest to drugi rodzaj firewalla pośredniczącego; pakiety z sieci prywatnej również nigdy nie wychodzą do Internetu i *vice versa*. Adresy IP w sieci prywatnej powinny być z klas nieroutowalnych. Jedyną drogą połączenia się z Internetem jest wywołanie firewalla, ponieważ jest on jedyną maszyną, mogącą łączyć się równocześnie z obiema sieciami. Uruchamiamy na nim program zwany *transparent proxy*, który tego dokonuje. System operacyjny zamiast dokonać routingu pakietów do Internetu, kieruje je do tego programu. Dla każdej usługi, która ma być dostępna z Internetu, na firewallu musi być uruchomiony osobny program pośredniczący w świadczeniu takiej usługi.

Przezroczyste proxy jest bardziej wygodne dla użytkowników i dla administratora. Klient nie musi wiedzieć o użyciu oprogramowania typu proxy i nie musi być specjalnie konfigurowany. Na przykład firewall jest skonfigurowany do przekierowywania (np. za pomocą polecenia `ipchains`) wszystkich połączeń do portu 80 na port 8080, na którym pracuje *tranparent proxy*. Przy próbie pobrania dowolnej strony WWW transmisja przekierowywana jest na port 8080, a następnie wszystko odbywa się tak, jak w poprzednim przykładzie.

Tłumaczenie adresów IP (NAT)

Pakiety z sieci prywatnej nigdy nie wychodzą do Internetu bez specjalnej obróbki i *vice versa*. Adresy IP w sieci prywatnej powinny być z klas nieroutowalnych. W tym przypadku używamy specjalnych funkcji zmieniających niektóre pola transportowanych pakietów (adres źródłowy, port źródłowy). Zostało to już opisane wcześniej.

Filtrowanie pakietów

W tym przypadku nasza sieć jest częścią Internetu, pakiety mogą poruszać się bez zmian poprzez obie sieci. Firewall na podstawie nagłówków protokołów podejmuje decyzję, czy przepuścić dany pakiet, czy nie. Filtrowanie pakietów zastosowano, aby ograniczyć dostęp z Internetu tylko do naszych wewnętrznych serwerów i uniemożliwić dostęp do komputerów użytkowników. Jeżeli firewall ma możliwość śledzenia sesji protokołów warstwy transportowej, nazywamy go **firewallem z inspekcją stanu** (*statefull instpection*); czasami spotyka się w literaturze również pojęcie firewalla obwodów.

Ten typ firewalla ma najmniejsze możliwości kontroli i autoryzacji dostępu, ale jest równocześnie najbardziej elastyczny i wprowadza najmniej ograniczeń dla użytkowników z sieci wewnętrznej. Nie ma potrzeby uruchamiania specjalnych programów pośredniczących.

10.2. Obsługa filtrowania pakietów w Linuksie

Linux jest typowym systemem sieciowym; możliwości filtrowania pakietów pojawiły się w nim bardzo wcześnie, bo już w wersji 1.1. Alan Cox po prostu w 1994 roku przeniósł z BSD kod używanego tam ipfw. Jos Vos rozbudował ten kod i w jądrach wersji 2.0 pojawiło się narzędzie `ipfwadm` do kontroli reguł filtrowania. W 1998 roku dla wersji 2.2 Paul Russell i Michael Neuling mocno zmienili kod jądra i wprowadzili nowe narzędzie `ipchains`. W 1999 roku dla jąder w wersji 2.4 kod podsystemu sieciowego został przepisany od nowa, wprowadzono wiele przydatnych możliwości: zaawansowany routing (również w oparciu o adres źródła pakietu), zarządzanie pasmem oraz całkiem nową „ścianę ogniową". Do zarządzania firewallem zostało przeznaczone nowe polecenie `iptables`. Musisz mieć włączony (wkompilowany) w jądrze moduł *netfilter* oraz zainstalowany pakiet *iptables*.

Tabela 10.1. *Porównanie funkcjonalności ipchains i iptables*

	Ipchains	Iptables
Możliwe kryteria selekcji pakietów	Adres źródłowy IP	
	Adres docelowy IP	
	Protokół (TCP,UDP) i port lub zakres portów	
	Protokół ICMP i komunikat	
	Zaistnienie fragmentacji	
	Wartości bitów TOS	
	Pakiety z flagą SYN	
	Możliwość określania reguł symetrycznych (odwrotnych) `-y`	
		Adres sprzętowy interfejsu MAC
		Dowolna kombinacja flag TCP
		Kryterium częstotliwości nadchodzących pakietów (zabezpieczenie przeciwko DoS)
		Właściciel pakietu w systemie
Funkcjonalność		Inspekcja stanu, śledzenie sesji dla protokołów TCP, UDP, ICMP
	Uproszczona translacja adresów, tzw. *Masquerading*	Pełna funkcjonalność NAT
	Możliwość logowania pakietów	Możliwość konfigurowania wyglądu (i poziomu) logów
		Obsługa IPv6

Pamiętaj, że pakiet *ipchains* nie jest już rozwijany, a do *iptables* ciągle pojawiają się nowe dodatki. Jest to główny powód, poza o wiele większą funkcjonalnością, dla którego skupię się na tym pakiecie. Jeśli brak jest funkcji, których potrzebujesz, spróbuj zajrzeć do zbioru dodatków do nazwie **Patch-o-Matic**; listę najnowszych można przeczytać pod adresem *http://www.netfilter.org/documentation/pomlist/pom-extra.html*. Najciekawsze dodatki umożliwiają ograniczanie liczby połączeń z jednego IP, selekcjonowanie pakietów w zależności od opcji IP, określanie rozmiaru pakietu.

Ipchains — Linux 2.2

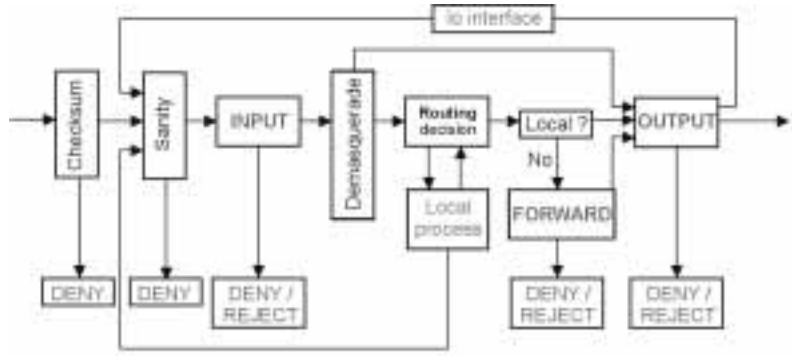
W tym podrozdziale zamieszczam skrócony opis obiegu datagramów IP w jądrach wersji 2.2. Następny podrozdział jest poświęcony projektowi netfilter i dokładniej opisuje działanie `iptables` używanego w jądrach wersji 2.4.

`Ipchains` jest linuksowym poleceniem używanym do konfiguracji reguł firewalla i tłumaczenia adresów IP (NAT). Zastąpiło ono starsze `ipfwadm`. Aby móc wykorzystać komputer z systemem operacyjnym Linuks do filtrowania pakietów, należy skompilować jądro systemu z opcjami (dla jądra z serii 2.2):

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

Aby stwierdzić, czy jądro ma prawidłowo wkompilowaną obsługę filtrowania datagramów, należy sprawdzić, czy w katalogu */proc/net/* istnieje plik *ip_fwchains*, zawierający konfigurację reguł firewalla.

Przy konfigurowaniu firewalla za pomocą polecenia `ipchains`, podstawowym pojęciem jest **łańcuch** (*chain*). Łańcuch jest to zbiór reguł filtrujących, na podstawie których podejmowana jest decyzja, czy pakiet zostanie przepuszczony, czy też usunięty. Istnieją trzy standardowe łańcuchy: *input*, *forward* i *output*, ponadto użytkownik może tworzyć własne łańcuchy.



Rysunek 10.1. *Obieg pakietów w Linuksie 2.2*

Droga datagramów IP poprzez firewall jest bardzo skomplikowana. Na samym początku drogi pakietu sprawdzana jest suma kontrolna datagramów (*checksum*), następnie testowane są one pod kątem deformacji (*sanity*). Później pakiety przechodzą przez łańcuch wejściowy (*input chain*) i jeśli trzeba, podlegają odwrotnemu procesowi NAT

(*demasquerade*), czyli adres routera usuwany jest z pola „adres docelowy" i zastępowany adresem IP docelowego komputera w sieci prywatnej. Dalej na podstawie tablicy routingu (*routing decision*) lub protokołu routującego podejmowana jest decyzja o dalszym losie pakietu. Procesy lokalne (*local process*) mogą odbierać pakiety po etapie routowania i wysyłać pakiety poprzez etap routowania i bezpośrednio łańcuch wyjściowy (*output chain*). Jeśli pakiet nie został utworzony przez proces lokalny, jest dodatkowo sprawdzany w łańcuchu przejściowym (*forward chain*). Jeśli proces lokalny będzie się komunikował z innym procesem lokalnym, kieruje pakiet przez łańcuch wyjściowy (*output chains*) do interfejsu **lo** (*lo interface - loopback*). Każdy z pakietów opuszczających komputer musi zostać sprawdzony przez reguły zawarte w łańcuchu wyjściowym (*output chain*).

Składnia polecenia ipchains

Pokrótce przedstawię podstawową składnię polecenia `ipchains`. Dokładny opis pozostałych opcji znajduje się w publikacji *http://www.tldp.org/HOWTO/IPCHAINS-HOWTO.html*

Po poleceniu `ipchains` następuje opcja określająca jego działanie:

- `-F` (*Flush*) — usuwa wpisy z wymienionego później łańcucha,
- `-A` (*Append*) — dodaje nową regułę do łańcucha,
- `-I` (*Insert*) — wstawia nową regułę do łańcucha na podane miejsce,
- `-D` (*Delete*) — usuwa regułę z łańcucha.

Następnie występują wpisy określające źródło i cel transmisji:

- `-s` (*source*) — źródło,
- `-d` (*destination*) — cel.

Adresy mogą być podawane jako nr IP lub nazwa oraz jako zakresy, np.

- `-s 192.168.23.24` — adres źródłowy równy 192.168.23.24,
- `-d 192.168.23.0/24` — adres źródłowy pochodzący z sieci 192.168.23.0 o 24-bitowej masce,
- `-d 192.168.23.0/255.255.255.0` — zapis równoważny poprzedniemu,
- `-s 0.0.0.0/0` — dowolny adres.

Większość opcji umożliwia zastosowanie negacji logicznej, na przykład `-s ! localhost` oznacza wszystkie komputery poza lokalnym. Protokół może być podawany jako numer pobrany z pliku */etc/protocols* lub nazwa (*tcp*, *udp*, *icmp*), nie jest ważne, czy użyjesz małych, czy też wielkich liter:

- `-p udp` — protokół UDP,

Można również podawać numer portu, dla którego układamy regułę:

`-p TCP -s 0.0.0.0/0 23` — ruch protokołem TCP z dowolnego adresu z portu 23 (telnet).

Możemy definować również zakresy portów:

`-p TCP -s 192.168.1.6 20:23` — ruch protokołem TCP z adresu 192.168.1.6 z portów o numerach 20, 21, 22 i 23,

`-p UDP -d 192.168.1.0/24 1024:` — ruch protokołem UDP do komputerów z sieci 192.168.1.0/24 na wysokie porty (porty o numerach 1024 i więcej).

Równie dobrze możemy podać nazwę portu: `-p TCP -s 0.0.0.0/0 www`.

Określamy interfejs sieciowy, którego dotyczy reguła:

`-i eth0` — karta sieciowa eth0,

`-i eth+` — oznacza wszystkie interfejsy zaczynające się na *eth*.

Aby włączyć zapisywanie pakietów do logów systemowych, musimy dodać do reguły flagę `-l`.

Flaga `-j` specyfikuje, co należy wykonać z pakietem pasującym do reguły. Jeśli nie ma tej flagi, to reguła jest używana do prostego zliczania pakietów ją spełniających. Działaniami, które możemy zlecić, są:

`-j ACCEPT` — podejmuje decyzję o przepuszczeniu pakietu pasującego do reguły,

`-j DENY` — natychmiast likwiduje pakiety pasujące do reguły,

`-j REJECT` — likwiduje pakiety, wysyłając do nadawcy komunikat ICMP o nieosiągalności celu,

`-j MASQ` — stosuje uproszczoną odmianę NAT, nazywaną czasem „maskaradą" (jądro musi być skompilowane z *IP Masquerading enabled*); opcja prawidłowa jedynie dla łańcucha *forward*,

`-j REDIRECT numer_portu` — przekierowuje pakiet na lokalny port; opcja musi być użyta w łańcuchu INPUT i można jej używać jedynie dla protokołów TCP i UDP,

`-j RETURN` — powoduje natychmiastowe osiągnięcie końca łańcucha.

Ustalamy politykę dla łańcucha. Polityka określa, co należy zrobić z pakietem nie pasującym do żadnej z zawartych w nim reguł:

```
ipchains -P input DENY
```

Przedstawię teraz przykładową konfigurację; należy ją traktować jedynie jako przykład podany w celu przedstawienia sposobu konstruowania reguł firewalla:

```
# Wyczyszczenie łańcuchów w celu pozbycia się potencjalnych pozostałości
  /sbin/ipchains -F input
  /sbin/ipchains -F output
  /sbin/ipchains -F forward
```

```
# Przekierowanie ruchu na port używany przez usługę przezroczystego Proxy
  /sbin/ipchains -A input -p tcp -s 192.1.2.0/24 -d 0.0.0.0/0 80 -j REDIRECT 8080

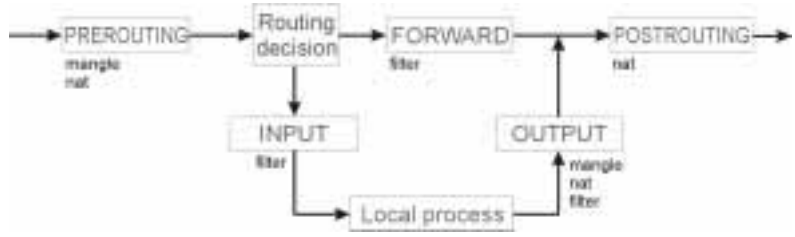
# Utworzenie własnego łańcucha o nazwie my-chain
  /sbin/ipchains -N my-chain
  # Dodawanie reguł do łańcucha my-chain
  # Zezwolenie na transmisje z serwerów SMTP
  /sbin/ipchains -A my-chain -s 0.0.0.0/0 smtp -d 192.1.2.10 1024: -j ACCEPT
  # Zezwolenie na transmisje do serwerów SMTP
  /sbin/ipchains -A my-chain -s 192.1.2.10 -d 0.0.0.0/0 smtp -j ACCEPT
  # Zezwolenie na transmisje z serwerów WWW
  /sbin/ipchains -A my-chain -s 0.0.0.0/0 www -d 192.1.2.11 1024: -j ACCEPT
  # Zezwolenie na transmisje do serwerów WWW
  /sbin/ipchains -A my-chain -s 192.1.2.0/24 1024: -d 0.0.0.0/0 www -j ACCEPT
  # Zezwolenie na transmisje z serwerów DNS
  /sbin/ipchains -A my-chain -p UDP -s 0.0.0.0/0 dns -d 192.1.2.0/24 -j ACCEPT
# Przekierowanie pakietów z sieci lokalnej do łańcucha my-chain
  /sbin/ipchains -A input -s 192.1.2.0/24 -d 0.0.0.0/0 -j my-chain
# Konfiguracja NAT
  # nie dokonuj NAT na ruchu z sieci lokalnej do sieci lokalnej
  /sbin/ipchains -A forward -s 192.1.2.0/24 -d 192.1.2.0/24 -j ACCEPT
  # nie dokonuj NAT dla ruchu z zewnątrz
  /sbin/ipchains -A forward -s 24.94.1.0/24 -d 0.0.0.0/0 -j ACCEPT
  # dokonuj NAT dla ruchu z sieci wewnętrznej na zewnątrz
  /sbin/ipchains -A forward -s 192.1.2.0/24 -d 0.0.0.0/0 -j MASQ

# Nie pozwól na jakikolwiek inny ruch
  /sbin/ipchains -P my-chain input DENY
```

Iptables — Linux 2.4

Polskie tłumaczenie dokumentu *HOWTO do iptables* znajdziesz na stronie *http://mr0vka.eu.org/tlumaczenia/index.html*. Powinieneś się z nim zapoznać, zanim zaczniesz czytać dalszy ciąg tego działu. Przeczytaj również pozostałe tłumaczenia pojawiające się na stronie **Łukasza Bromirskiego** *http://mr0vka.eu.org* — naprawdę warto. Zajrzyj również na stronę domową pakietu netfilter *http://www.netfilter.org*. Taaaak. Wiedziałem, że nie będzie ci się chciało czytać takiej ilości tekstu. Dobrze, zatem postaram się omówić pokrótce, jak wędrują pakiety w jądrze 2.4.

Podstawowym pojęciem jest **łańcuch** (*chain*). Łańcuch jest to zbiór reguł filtrujących. Istnieją trzy standardowe łańcuchy: *input, forward* i *output* oraz łańcuchy dodatkowe, *prerouting* i *forward*. Ponadto użytkownik może tworzyć własne łańcuchy. Łańcuch jest listą reguł, do których po kolei jest porównywany pakiet. Jeśli pakiet pasuje do którejś z reguł, wykonywana jest akcja zdefiniowana wewnątrz niej. Reguły definiujemy właśnie poleceniem `iptables`. Jeśli pakiet nie pasuje do żadnej reguły, wykonywana jest akcja zdefiniowana w **polityce** danego łańcucha. Tylko łańcuchy podstawowe i dodatkowe mają własną politykę, łańcuchy utworzone przez użytkownika nie mają własnej polityki, a pakiet po przejściu przez taki łańcuch wraca do miejsca (do łańcucha), z którego został wywołany.



Rysunek 10.2. *Obieg pakietów w Linuksie 2.4*

Jeśli porównasz ten schemat z algorytmem dla *ipchains*, zuważysz, jak bardzo został on uproszczony i logicznie przemyślany. Pakiet sprawdzany jest najpierw w łańcuchu **PREROUTING**. Następnie podejmowana jest decyzja *Routing decision*, czy pakiet skierowany jest do komputera lokalnego. Przechodzi wtedy przez łańcuch **INPUT** i dociera do procesów lokalnych *Local process*. Drugą możliwością jest, że pakiet jest skierowany do jednej z sieci, do których dany komputer jest routerem. W takim przypadku podejmowana jest decyzja o routingu danego pakietu (w uproszczeniu — definiowana karta sieciowa, przez którą należy dany pakiet dalej przesłać) i pakiet trafia do łańcucha **FORWARD**. Pakiety wygenerowane przez procesy lokalne (np. przeglądarkę WWW wysyłającą zapytanie o stronę) przechodzą przez łańcuch **OUTPUT**. Następnie wszystkie pakiety przechodzą przez łańcuch **POSTROUTING** i są wysyłane na karty sieciowe.

Przedstawione powyżej łańcuchy są dodatkowo rozdzielone pomiędzy tablice. zawierające niektóre z nich. Pierwszą jest tabela *mangle*, służąca do zmieniania zawartości pakietów przechodzących przez nasz firewall. Następnie pakiety przechodzą przez tabelę *nat*, w której definiujemy różne rodzaje translacji adresów NAT. Ostatnią jest najczęściej używana tabela *filter*. Definiujemy w niej reguły filtrowania pakietów, czyli podstawowy kod naszego firewalla. Nie wszystkie łańcuchy istnieją w poszczególnych tabelach.

Do tabeli *mangle* (polecenie: `iptables -t mangle ...`) należą łańcuchy:

- PREROUTING,
- OUTPUT.

Do tabeli *nat* (polecenie: `iptables -t nat ...`) należą łańcuchy:

- PREROUTING - DNAT (*destination nat*) podmieniany jest adres docelowy pakietów,
- POSTROUTING - SNAT (*source nat*) podmieniany jest adres źródłowy pakietów,
- OUTPUT - DNAT dla pakietów generowanych przez procesy lokalne.

Domyślną tabelą jest *filter* (polecenie: `iptables ...` — nie trzeba używać przełącznika `-t`), należą do niej łańcuchy:

- INPUT,
- OUTPUT,
- FORWARD.

Nie będę omawiał składni `iptables`, jest ona bardzo rozbudowana i zajęłaby zbyt dużo miejsca. Ponadto zostało to bardzo dobrze zrobione w wymienionym wcześniej dokumencie HOWTO (z polskim tłumaczeniem). Podczas tworzenia firewalla postaram się omówić używane przeze mnie konstrukcje. Jeśli coś wyda ci się niejasne, zajrzyj do HOWTO lub na stronę manuala (`man iptables`).

10.3. Tworzymy firewall

Postanowiłem omówić tworzenie firewalla na konkretnym, praktycznym przykładzie. Ponieważ jednak niezbyt dobrym pomysłem na początek byłoby tworzenie rozbudowanego firewalla, przedstawię często spotykaną sytuację.

Zakładamy małą sieć lokalną złożoną, podłączoną poprzez Ethernet do Internetu. Oczywiście rzadko się zdarza, aby nasze łącze stałe było skrętką. Jednak ze względu na przejrzystość, przedstawię takie rozwiązanie. Tak czy inaczej jest to tylko szablon i aby go zastosować w praktyce, będziesz go musiał poddać znacznym modyfikacjom. Aby zminimalizować koszty, utworzymy firewall na tej samej maszynie, co router — najczęściej tak się właśnie robi, konstruując rozwiązanie oparte na Linuksie. Takie rozwiązanie daje nam dużą elastyczność, możemy korzystać z możliwości nowego kodu sieciowego jądra 2.4: zarządzać pasmem przydzielanym komputerom w sieci lokalnej i konkretnym usługom, stosować routing źródłowy i łączyć to wszystko z rozbudowanymi możliwościami firewalla opartego na `iptables`. Zawsze najlepszym wyjściem jest, gdy firewall jest dedykowanym serwerem, na którym nie ma uruchomionych żadnych dodatkowych usług. Dzięki temu nie ma możliwości (przynajmniej teoretycznie) włamania do niego. Na maszynę spełniającą tego typu zadania dla niewielkiej sieci często wystarczy stare 486DX i jakieś 16 MB RAM-u. Zdobycie tego typu „maszyny" nie powinno nastręczać wielkich kłopotów.

Jeśli nie masz dysku twardego, to możesz poszukać jakiejś jednodyskietkowej dystrybucji Linuksa lub BSD przystosowanej do pełnienia roli routera i firewalla.



Jednodyskietkowe dystrybucje przeznaczone do tworzenia routera i firewalla:

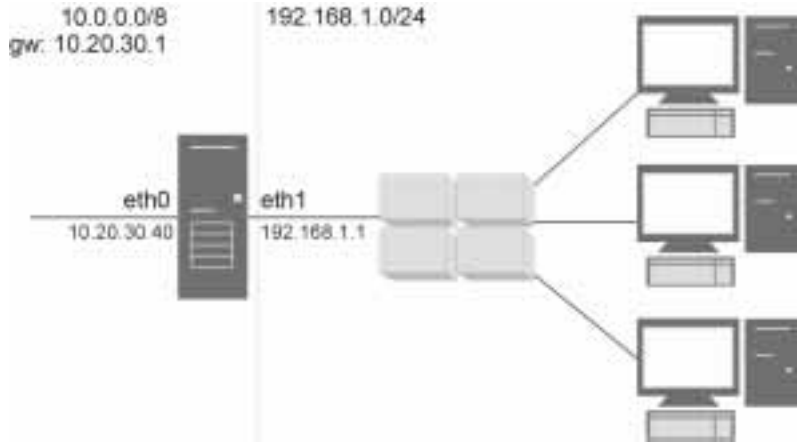
- *http://www.freesco.org*,
- *http://www.freesco.arx.pl/linux/* — polska strona dystrybucji Freesco przystosowanej do **SDI**,
- *http://www.linuxrouter.org*,
- *http://www.coyotelinux.com*,
- *http://www.floppyfw.org*.

Jednak wtedy najpewniej nie będziesz mógł skorzystać z konfiguracji opisanej poniżej — wersje jednodyskietkowe Linuksów są wykonywane najczęściej na jądrach serii 2.2. Możesz wtedy korzystać jedynie z polecenia `ipchains`. Jeśli chodzi o wybór dystrybucji, to nie jest to zbyt ważny element, wybierz najbardziej wygodną dla siebie. Jeśli

dystrybucję będziesz często aktualizował, powyłączasz wszystkie usługi sieciowe, to zabezpieczenie powinno być wystarczające. Oczywiście dyskusyjne jest pozostawienie działającego demona SSH do administracji. Jeśli jest to konieczne, zainstaluj jak najnowszą wersję stabilną i ciągle dbaj o szybką aktualizację źródeł, które oczywiście samodzielnie skompilujesz. Nie używaj pakietu binarnego otrzymanego z dystrybucją. Ponadto postaraj się wytyczyć ścieżki, z których będziesz korzystał podczas administracji i podczas implementacji firewalla pozwól na dostęp do SSH tylko z konkretnych adresów IP. Jeśli podajesz adresy w sieci lokalnej, zdefiniuj adresy MAC kart sieciowych zamiast adresów IP.

Podstawy

Router (i firewall), który konfigurujemy, ma dwie karty sieciowe; jedną — **eth0** — na zewnątrz z numerem IP przydzielonym przez naszego dostawcę usług i drugą — **eth1** — do naszej sieci lokalnej, której IP sami sobie konfigurujemy. Jeśli masz łącze SDI, to najprawdopodobniej musisz wymienić wpis eth0 na ppp0. Cały opis przedstawię dla dystrybucji Slackware, jednak jak już napisałem, nie jest ważne, jakiej dystrybucji użyjesz, ale jak ją skonfigurujesz. Nigdy nie należy pozostawiać konfiguracji domyślnej.



Rysunek 10.3. *Sieć, dla której napiszemy firewall*

eth0

przydzielony numer IP: 10.20.30.40
z maską 8 bitów czyli: 255.0.0.0
gateway (router): 10.20.30.1

eth1

przyjmujemy numer IP: 192.168.1.1
z maską 24 bity 255.255.255.0

Przykładowy komputer w sieci wewnętrznej konfigurujemy:

numer IP: 192.168.1.2
z maską 255.255.255.0

gateway (router): 192.168.1.1
serwery DNS ustawiamy na zalecane przez dostawcę usług.

W przypadku Slackware'a wyremowałem (postawiłem na początku wiersza #) ze skryptu */etc/rc.d/rc.inet1* wszystko poza wierszem:

```
# poczatek skryptu /etc/rc.d/rc.inet1
HOSTNAME=`cat /etc/HOSTNAME`
```

Następnie konfigurujemy standardowe urządzenie lo, dopisując do tego pliku:

```
/sbin/ifconfig lo 127.0.0.1
/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo
```

W moim przypadku włączamy potrzebne moduły do kart sieciowych 3Com:

```
/sbin/modprobe 3c509
```

A następnie przechodzimy do konfiguracji tychże kart sieciowych i routingu.

```
/sbin/ifconfig eth0 10.20.30.40 netmask 255.0.0.0 broadcast 10.255.255.255 up
/sbin/route add default gw 10.20.30.1
/sbin/ifconfig eth1 192.168.1.1 netmask 255.255.255.0 broadcast 192.168.1.255 up
```

Dalej musimy włączyć NAT dla pakietów z sieci lokalnej wychodzących na zewnątrz.

```
modprobe ip_conntrack_irc
modprobe ip_nat_irc
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to-source=10.20.30.40
echo "1" > /proc/sys/net/ipv4/ip_forward
/etc/rc.d/firewall.start
/usr/local/sbin/sshd
```

`Poleceniami modprobe` włączamy moduły do irca (jeśli będą nam potrzebne). `Iptables` uruchamia SNAT pakietów, który podmienia adres źródłowy pakietów z naszej sieci wewnętrznej na adres 10.20.30.40. Oczywiście odbywa się to w łańcuchu POSTROUTING w tabeli nat, co widać w strukturze tego polecenia. W tym momencie pakiety z naszego przykładowego komputera o adresie 192.168.1.2, wychodząc w Internet (z karty eth0) wydają się pochodzić z adresu 10.20.30.40. Aby to zadziałało, musimy jeszcze uruchomić proces routingu — dzieje się to w następnym wierszu, gdzie zmieniamy odpowiednią wartość w katalogu */proc*. Następnie uruchamiamy nasz skrypt *firewall.start*. Na koniec uruchamiamy demona ssh (najlepiej w wersji 2), abyśmy mogli administrować naszym routerem z zewnątrz. Jest to koniec naszego skryptu */etc/rc.d/rc.inet1*, uruchamiającego podstawowe funkcje sieciowe. W tym momencie zaczynamy pisanie skryptu firewalla */etc/rc.d/firewall.start*.

Konfiguracja

Podstawową zasadą przy pisaniu reguł firewalla jest:



najpierw stawiamy barierę nie do przebycia dla wszystkich pakietów, a następnie wybijamy w niej przejścia tylko dla takiego ruchu, który chcemy przepuścić.

W skrócie: co nie jest dozwolone — jest zabronione, czyli po pierwsze wybieramy połączenia do naszego routera z Internetu, które będą możliwe. Poza tym tak naprawdę powinniśmy również ściśle zdefiniować rodzaje usług w Internecie, z których mogą korzystać komputery znajdujące się w sieci lokalnej, oraz połączenia, które komputery z sieci lokalnej mogą wykonywać do routera. Ponieważ omawiamy najprostszy przypadek, nie będę na razie komplikował sytuacji i założę, że kierownictwo firmy nie wykazało zrozumienia i użytkownicy mają mieć dostęp do wszystkich usług udostępnianych w Internecie.

Zagrożenia wynikające z takiego postępowania opisywałem w poprzednich rozdziałach. Aby przypomnieć, podam przykład, gdy na komputerze w sieci lokalnej pojawi się jakiś trojan. Przy silnie restrykcyjnym firewallu, gdyby trojan próbował nawiązać połączenie na zewnątrz, zostałoby ono uniemożliwione przez firewall i mielibyśmy o tym informację w logach systemowych. Oczywiście istnieją trojany wykorzystujące np. port 80 (http), który raczej musimy przepuścić. Oznacza to, że na komputerach lokalnych nie możemy zaniedbać podstawowych zasad bezpieczeństwa i musimy zainstalować jakieś (często uaktualniane) oprogramowanie antywirusowe.

Tak więc zaczynamy. W katalogu */etc/rc.d/* tworzymy sobie skrypt *firewall.start*, który jest uruchamiany z pliku *rc.inet1*. Dla większości dystrybucji Linuksa możemy uruchamianie firewalla umieścić w pliku */etc/rc.d/rc.local*. Moja konfiguracja została oparta na opisie ze strony: *http://www.sns.ias.edu/~jns/security/iptables* — oczywiście dokonałem w niej dużych zmian. Nie będę tworzył skryptu sparametryzowanego — jest on zbyt mało przejrzysty dla celów demonstracji, a ponadto gdy zaglądamy do niego po kilku miesiącach, jest trudniejszy w analizie. Przed poleceniami `iptables` występującymi w skrypcie powinieneś dopisać pełną bezwzględną ścieżkę dostępu, np. `/usr/local/sbin/iptables` ...

```
#!/bin/sh
#Poczatek skryptu /etc/rc.d/firewall.start
modprobe ip_tables
modprobe ip_conntrack
modprobe ip_conntrack_ftp
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

Najpierw próbujemy ładować moduły, które mogą się nam przydać. Bardzo ważnym modułem jest **conntrack**, umożliwiający śledzenie połączeń. Dzięki niemu możemy stwierdzić, które pakiety nie są poprawnymi pakietami danego połączenia. Moduł conntrack potrafi śledzić, poza oczywistym protokołem TCP, który jest protokołem połączeniowym, również protokoły UDP i ICMP. Lista aktualnie śledzonych połączeń znajduje się w pliku */proc/net/ip_conntrack*. W ostatnich 3 wierszach ustawiliśmy politykę podstawowych łańcuchów na DROP — w tym momencie nasz router nie odbiera i nie przepuszcza żadnych pakietów, możesz to sprawdzić. Teraz wykonamy wstępną konfigurację parametrów jądra wpływających na działanie stosu TCP/IP w Linuksie. Jeśli z uwagą przeczytałeś wcześniejsze rozdziały, a zwłaszcza ten dotyczący bezpieczeństwa, będziesz rozumiał, co robią te parametry i przeciwko jakim atakom są zabezpieczeniem.

```
# (1) Wyłączamy odpowiedzi na pingi
/bin/echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_all
```

```
# (2) Ochrona przed atakiem typu Smurf
/bin/echo "1" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

# (3) Nie akceptujemy datagramów IP z opcją "source route"
/bin/echo "0" > /proc/sys/net/ipv4/conf/all/accept_source_route

# (4) Nie przyjmujemy pakietow ICMP rediect, które mogą zmienić naszą tablicę
routingu
/bin/echo "0" > /proc/sys/net/ipv4/conf/all/accept_redirects

# (5) Włączamy ochronę przed komunikatami ICMP error
/bin/echo "1" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses

# (6) Wlacza logowanie dziwnych (spoofed, source routed, redirects) pakietow
/bin/echo "1" > /proc/sys/net/ipv4/conf/all/log_martians

# (7) Wszystkie karty nie beda przyjmowaly pakietow z sieci innych niz te
# z tablicy routingu, jest to ochrona przed spoofingiem
echo "1" > /proc/sys/net/ipv4/conf/all/rp_filter
```

Ponieważ w jądrach serii 2.4 automatyczna defragmentacja przyjmowanych pakietów jest włączana domyślnie, już nie musimy tego uruchamiać. W wierszu 1 informujemy nasz router, aby nie odpowiadał na zapytania ICMP echo — czyli router nie będzie odpowiadał na pingi. Warto się nad tym zastanowić, ponieważ postępowanie takie jest niezgodne z zaleceniami IETF. Podaję ten wpis w celach edukacyjnych, ponieważ może się zdarzyć, że będziesz chciał go użyć.

W wierszu 2 uniemożliwiamy odpowiedzi ICMP *Echo reply* na pakiety rozgłoszeniowe (broadcast), w efekcie czego bronimy się przed atakiem typu **Smurf.** Atak ten polega na tym, że atakujący wysyła pakiety ICMP *echo request* skierowane na adresy broadcastowe naszej sieci. Wszystkie komputery naszej sieci odpowiadają na takie pakiety, co najczęściej powoduje silne przeciążenie sieci. Bardzo często adres źródłowy takiego pakietu jest zmieniony i ustawiony na rzeczywistą ofiarę, która otrzymuje nagle dużo pakietów z odpowiedziami typu *echo-reply*. Jeśli w ten sposób zostanie zaatakowanych kilka podsieci, łącze ofiary może ulec zapchaniu.

W wierszu 3 powiadamiamy jądro systemu, aby nie akceptowało pakietów *source routed*. Atakujący może użyć tego typu pakietów, aby udawać, że jest z wnętrza naszej sieci, lecz ruch ten byłby kierowany z powrotem wzdłuż ścieżki, po której do nas dotarł. Opcje te rzadko są używane w zgodnych z prawem celach.

W wierszu 4 zabraniamy zmieniania naszej tablicy routingu na podstawie pakietów ICMP redirect.

W wierszu 5 powodujemy, że nasz host nie będzie reagował na fałszywe komunikaty o błędach.

Wiersz 6 powoduje, że informacja o wszelakich dziwnych (podejrzanych) pakietach znajdzie się w logach systemowych. Może to być niebezpieczne, ponieważ nie mamy wpływu na częstotliwość zapisów i zmasowany atak takimi pakietami może przepełnić nasze logi.

Standardowym mechanizmem jest ignorowanie pakietów pochodzących z sieci nie znajdujących się w naszej tablicy routingu, co odbywa się w wierszu 7. Oznacza to, że jeśli na karcie eth1 jest zdefiniowana sieć 192.168.1.0/24, to nie przyjmie ona pakietów z adresu 10.1.23.32. Oczywiście karta eth0 będzie przyjmowała pakiety ze wszystkich adresów, ponieważ jest poprzez nią dostęp do naszej domyślnej trasy (routera) — również pakiety z adresem źródłowym naszej karty eth1. Musimy więc zadbać za pomocą filtrowania na firewallu, aby nie istniała taka możliwość. Piszemy zatem dalszy ciąg skryptu:

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

Pierwsze co umożliwiamy, to ruch na urządzeniu lo (*lopback*), które jest wykorzystywane przez różne procesy lokalne. Temu ruchowi nie stawiamy żadnych ograniczeń.

Poniżej tworzymy nowy łańcuch o nazwie *syn-flood*, który będzie zabezpieczał nasz router przed atakami typu DoS. Oczywiście jest bardzo prawdopodobne, że będziesz musiał dobrać (najczęściej eksperymentalnie) czułość działania modułu limit.

```
iptables -N syn-flood
iptables -A INPUT -i eth0 -p tcp --syn -j syn-flood
iptables -A syn-flood -m limit --limit 1/s --limit-burst 4 -j RETURN
iptables -A syn-flood -j LOG --log-level debug --log-prefix "SYN-FLOOD: "
iptables -A syn-flood -j DROP
```

W pierwszym wierszu tworzymy łańcuch o wymyślonej przez nas nazwie *syn-flood*. W drugim wierszu powodujemy, że wszystkie pakiety protokołu TCP z ustawioną flagą SYN pojawiające się na karcie sieciowej eth0 i wędrujące przez łańcuch INPUT (skierowane bezpośrednio do naszego routera) zostaną przekierowane do łańcucha *syn-flood*. Pakiety z ustawioną flagą SYN (i w poprawnym połączeniu również ack) służą do nawiązywania połączenia. W wierszu 3 zaczynamy „zliczanie" tych pakietów. Jeżeli pojawi się więcej niż jeden na sekundę pakiet nawiązujący połączenie bezpośrednio z naszym routerem, to po czterech takich pakietach reguła 3 przestanie działać i przerzucać pakiety spowrotem (`-j RETURN`) do łańcucha INPUT. W takim przypadku zostaną wykonane następne reguły. Reguła 4 spowoduje, że informacja o takim pakiecie pojawi się w logach systemowych z przedrostkiem: „SYN-FLOOD: ". Reguła 5 spowoduje, że pakiet taki zostanie skasowany bez żadnej dodatkowej reakcji ze strony naszego hosta. Przydałoby się jeszcze ustawić ograniczenie *limit* na ilość pakietów zapisywaną do logów, bo w przypadku ataku DoS może nam się zapełnić przestrzeń dyskowa, zwłaszcza że system może nie nadążyć z zapisywaniem zdarzeń do logów. Oto poprawna postać wiersza nr 4:

```
iptables -A syn-flood -m limit --limit 1/s --limit-burst 4 -j LOG --log-level debug
--log-prefix "SYN-FLOOD: "
```

W podanym wierszu też możemy poeksperymentować z wartościami parametrów dotyczących ograniczenia limit.

```
iptables -A INPUT -i eth0 -p icmp -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o eth0 -p icmp -m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Wiersze te zapewniają nam przepuszczanie wcześniej nawiązanych połączeń protokołu ICMP (wiersz 1) i nawiązywanie połączeń pochodzących z naszego hosta (wiersz 2). Oczywiście mowa o połączeniach do i z routera, ponieważ łańcuchy INPUT i OUTPUT dotyczą jedynie lokalnego hosta.

```
iptables -A INPUT -i eth0 -p tcp ! --syn -m state --state NEW -j LOG --log-level
debug --log-prefix "NEW: "
iptables -A INPUT -i eth0 -p tcp ! --syn -m state --state NEW -j DROP
```

Powyższe wiersze wynikają z pewnej niekonsekwencji w implementacji modułu *conntrack* służącego do śledzenia połączeń w pakiecie netfilter. Moduł ten pakiety TCP bez flagi syn, a z ustawioną flagą ack uznaje za pakiety NEW, czyli nawiązujące połączenie. W pierwszym z powyższych wierszy logujemy (zapisujemy do logów) pakiety uznane za NEW (`--state NEW`), nie posiadające ustawionej flagi SYN (`! --syn`). W kolejnym wierszu odrzucamy takie pakiety.

```
iptables -A INPUT -i eth0 -f -j LOG --log-level debug --log-prefix "FRAGMENTS: "
iptables -A INPUT -i eth0 -f -j DROP
```

Podobna konstrukcja zastosowana jest w podanych regułach. Pakiety sfragmentowane `-f` na wszelki wypadek (atak Jolt2) porzucamy, oczywiście uprzednio je logując. Jeśli w logach systemowych będziemy mieli informacje o usuniętych poprawnych połączeniach posługujących się sfragmentowanymi pakietami, to będziemy musieli wyłączyć te reguły.

Poniżej mamy małe zabezpieczenie anty-spoofingowe (przeciw podszywaniu się pod inne adresy niż własne). W sumie niepotrzebne, ale nigdy nie wiadomo, czy nasz stos TCP/IP nie ma jeszcze nie wykrytych błędów. Ponadto stosując dodatkowo logowanie takich pakietów, uzyskamy informację o próbach spoofingu.

```
# Pakiety z adresem źródłowym ustawionym na nasz
iptables -A INPUT -i eth0 -s 10.20.30.40 -j DROP    # atak Land
# Pakiety z adresow nieroutowalnych, multicast i zarezerwowanych
# iptables -A INPUT -i eth0 -s 10.0.0.0/8 -j DROP   # class A
iptables -A INPUT -i eth0 -s 172.16.0.0/12 -j DROP  # class B
iptables -A INPUT -i eth0 -s 192.168.0.0/16 -j DROP # class C
iptables -A INPUT -i eth0 -s 224.0.0.0/4 -j DROP    # multicast
iptables -A INPUT -i eth0 -d 224.0.0.0/4 -j DROP    # multicast
iptables -A INPUT -i eth0 -s 240.0.0.0/5 -j DROP    # reserved
iptables -A INPUT -i eth0 -s 127.0.0.0/8 -j DROP    # lo
iptables -A INPUT -i eth1 -p udp -d 192.168.1.255 --dport 137:138 -j DROP
```

Powyższe reguły nie powinny nastręczać wątpliwości. Na karcie eth0 nie powinny się pojawić pakiety z pochodzące z:

- wiersz 2 — naszego własnego adresu, charakterystyczne dla ataku Land,
- wiersze 4 – 9 — adresów zarezerwowanych; wyremowany został zakres 10.0.0.0/8, w naszym przykładzie używany przez kartę eth0,
- wiersz 10 — adresu interfejsu wewnętrznego lo.

Niestety, najczęściej komputery w naszej sieci lokalnej będą używały protokołu NetBIOS enkapsulowanego w TCP/IP. Protokół ten silnie „śmieci" — wysyła wiele broadcastów IP protokołu UDP na porty 137 i 138. Abyśmy nie mieli logów systemowych

zarzuconych informacjami o działalności tego protokołu, wpisujemy ostatnią z powyższych reguł. Oczywiście możemy (a nawet powinniśmy) dodać podobne reguły dotyczące karty eth1. W końcu dochodzimy do reguł określających, jakie połączenia mogą być nawiązane bezpośrednio z naszym routerem.

```
# łańcuch INPUT
iptables -A INPUT -p tcp --sport 1024: --dport  22 -m state --state NEW -j ACCEPT
iptables -A INPUT -p tcp --sport 1024: --dport 113 -m state --state NEW -j REJECT
--reject-with icmp-port-unreachable
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A INPUT -j LOG --log-level debug --log-prefix "INPUT: "
iptables -A INPUT -j DROP
```

Ponieważ w powyższych regułach nie zdefiniowałem interfejsu sieciowego, będą one działały na obu kartach sieciowych w naszym routerze. W drugim wierszu definiujemy regułę mówiącą: pakiety protokołu tcp pochodzące z portów powyżej 1023 skierowane na port 22 (ssh) i uznane za NEW, czyli nawiązujące połączenie, akceptujemy. Pozostałe pakiety dla połączenia nawiązanego za pomocą pakietu przepuszczonego przez regułę nr 2 zostaną przepuszczone przez regułę w wierszu 4. Mówi ona: pakiety, które uznasz za należące (ESTABLISHED) lub związane (RELATED) z już istniejącym połączeniem, zaakceptuj. To jest właśnie zaleta firewalli z inspekcją stanu (moduł conntrack); możemy podczas konfiguracji firewalla operować pojęciami: „nawiązane połączenie", „poprawne połączenie", „pakiet nawiązujący połączenie" itp. Pakiety nie zaakceptowane przez te reguły zostaną zalogowane (reguła nr 5) i usunięte (reguła nr 6). Oczywiście gdyby nie było reguły nr 6, to zadziałałaby polityka łańcucha INPUT, wcześniej ustawiona również na DROP… ale trochę ostrożności nikomu nie zaszkodzi.

Gdybyśmy uruchomili na naszym routerze serwer WWW (czego oczywiście ze względów bezpieczeństwa nie polecam) wystarczyłoby skopiować regułę nr 2 i wkleić ją poniżej tej reguły (w wierszu poprzedzającym regułę 4) i zmienić wartość portu 22 na 80. Przypominam, że numery portów znajdziesz w pliku */etc/services*.

Jak wcześniej pisałem, powinieneś zdecydować, z jakich adresów IP będziesz administrował swoim routerem i skonfigurować swój firewall tak, aby pozwalał jedynie na połączenia z tych adresów. Załóżmy, że będziesz łączył się protokołem SSH2 jedynie z komputera o adresie 192.168.1.2, znajdującego się w twojej sieci lokalnej. Bardziej restrykcyjna reguła, w miejsce wiersza drugiego, będzie miała postać:

```
iptables -A INPUT -i eth1 -p tcp -s 192.168.1.2 --sport 1024: --dport 22 -m state
--state NEW -j ACCEPT
```

Możesz również wykorzystać możliwość definiowania adresu MAC komputera uprawnionego do połączenia; po dokładniejszy opis zajrzyj do manuala lub HOWTO. Do wytłumaczenia pozostaje reguła nr 3. Bardzo często różne serwery usług (np. ftp) pytają hosta, z którego się łączymy, o nasze dane, za pomocą usługi ident działającej na porcie 113. Jeżeli nie byłoby reguły 3, przykładowy serwer FTP wysłałby pakiet na port 113 naszego routera i czekał na odpowiedź. Ponieważ pakiet ten zostałby usunięty przez firewalla, nie otrzymałby żadnej odpowiedzi. W końcu otrzymalibyśmy zgodę na połączenie z serwerem FTP, ale czas oczekiwania byłby dosyć długi. W wierszu 3 powodujemy, że na zapytania na port 113 protokołu TCP generowany jest pakiet protokołu ICMP z komunikatem: *icmp-port-unreachable*. Serwer FTP jest poinformowany,

że na naszym hoście usługa ident nie działa i już bez dalszej zwłoki pozwala nam się zalogować. Zwróć również uwagę, że dla protokołu UDP nie zostały zdefiniowane żadne reguły, ponieważ nie udostępniamy żadnych usług z niego korzystających. Tak naprawdę to udostępniamy jedynie ssh.

```
# lancuch OUTPUT
iptables -A OUTPUT -m state --state ! INVALID -j ACCEPT
iptables -A OUTPUT -j LOG --log-level debug --log-prefix "OUTPUT: "
iptables -A OUTPUT -j DROP
```

W tym łańcuchu również powinniśmy dokładnie zdefiniować połączenia, które można nawiązywać z naszego routera. Jednak w ten sposób sami sobie ograniczalibyśmy funkcjonalność naszego routera, a może się zdarzyć, że będziemy mieli ochotę trochę poeksperymentować z pakietami. Jednak oczywiście w normalnych warunkach zalecam ustawienie bardziej restrykcyjnych reguł na łańcuch OUTPUT. Reguła w wierszu drugim akceptuje wszystkie pakiety należące do poprawnych połączeń. Składnia `--state INVALID` oznacza wszystkie wadliwe pakiety, nie należące do poprawnego połączenia. Składnia `--state ! INVALID` jest przeczeniem, oznacza więc wszystkie poprawne pakiety. Reguła 3 powoduje logowanie odrzuconych pakietów, a reguła 4 — ich usunięcie.

```
# łańcuch forward
iptables -A FORWARD -i eth1 -p tcp -s 192.168.1.0/24 --sport 1024: -m state --state
NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -i eth1 -p udp -s 192.168.1.0/24 --sport 1024: -m state --state
NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -o eth1 -p tcp -d 192.168.1.0/24 --dport 1024: -m state --state
ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -o eth1 -p udp -d 192.168.1.0/24 --dport 1024: -m state --state
ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -p icmp -m state --state ! INVALID -j ACCEPT
iptables -A FORWARD -j LOG --log-level debug --log-prefix "FORWARD: "
iptables -A FORWARD -j DROP
```

W łańcuchu FORWARD definiujemy reguły dotyczące pakietów wędrujących poprzez nasz router. Reguły 2 i 3 oznaczają: dla połączeń protokołów TCP i UDP wchodzących na kartę eth1 (`-i eth1`) z adresów sieci lokalnej 192.168.1.0/24 z portów powyżej 1023 przepuść pakiety nawiązujące połączenia (NEW), pakiety należące do istniejących połączeń (ESTABLISHED) i pakiety związane z istniejącymi połączeniami (RELATED). Oznacza to, że z sieci lokalnej możemy nawiązywać dowolne połączenia. Reguły 4 i 5 oznaczają: dla protokołów TCP i UDP wychodzących z karty eth1 (`-o eth1`) do sieci lokalnej 192.168.1.0/24 na porty powyżej 1023 przepuść pakiety należące do istniejących połączeń (ESTABLISHED) i pakiety związane z istniejącymi połączeniami (RELATED). W domyśle nie przepuszczaj pakietów nawiązujących połączenia do sieci lokalnej. Reguła 6 pozwala na niczym nie ograniczony ruch pakietów protokołu ICMP należących do poprawnych połączeń. Jeśli pojawią się jakieś pakiety nie należące do istniejącego połączenia, nie zostaną one przepuszczone. Ostatnie dwie reguły standardowo logują i wycinają wszystkie pozostałe pakiety.

I oto mamy działający i w podstawowym stopniu zabezpieczający naszą sieć firewall. Jego dostosowanie do potrzeb rzeczywistej sieci pozostawiam oczywiście tobie. Nie podawałem tutaj wszystkich aspektów jego konstrukcji, na które powinieneś zwrócić uwagę, ponieważ chciałem zachować jasną strukturę konstrukcji firewalla. Jak pisałem na początku tego rozdziału, nie jest to gotowy do zastosowania skrypt. Jest to pewien

przykład, który miał ci pokazać zasady tworzenia firewalla. Możesz się na nim oprzeć jak na szkielecie, ale musisz go obudować własnymi regułami. Powinieneś przeczytać ponownie rozdział dotyczący bezpieczeństwa, wypisując sobie zagrożenia, przed którymi powinien obronić twoją sieć firewall, a następnie zaimplementować tę ochronę w formie poleceń w skrypcie. Warto zajrzeć na stronę domową projektu netfilter i poprzeglądać dokumentację w poszukiwaniu przykładów, różnych nietypowych rozwiązań itp. Kilka ciekawych zagadnień poruszę w następnym podrozdziale.

Logi systemowe

Po uruchomieniu skryptu */etc/rc.d/firewall.start* mamy działający poprawnie firewall. Przydałoby się, abyśmy informacje o działaniu firewalla mieli w logach. W tym celu wpisujemy na koniec pliku */etc/syslog.conf* następujący kod:

```
*.*          /dev/tty12
*.*          /var/log/wszystko
```

i oczywiście restartujemy demona *syslogd*:

```
# killall -HUP syslogd
```

od tego momentu w pliku */var/log/wszystko* (oraz na konsoli 12 — *Alt+F12*) będziemy mieli wszystkie logi systemowe. Należy uważać, aby logi nam nie przepełniły partycji, czyli należy poprawnie skonfigurować *logrotate,* ale to oczywiście znajdziecie już na stronach dotyczących konfiguracji Linuksa.

Problemy z działaniem firewalla

Należy obserwować logi systemu i patrzeć, co jest wycinane przez nasze reguły. Każdy wycięty pakiet będzie opatrzony przedrostkiem, który zdefiniowaliśmy w regułach, np. `--log-prefix "FORWARD: "`. Dzięki temu łatwo zorientujemy się, w którym miejscu dzieje coś niedobrego. Pamiętaj, że niektóre z usuwanych pakietów nie są logowane. Jeśli użyjesz:

```
# iptables -L -n
```

otrzymasz listę uruchomionych reguł. Sprawdź, czy nie wygląda inaczej, niż byś się spodziewał. Jeśli do podanego polecenia dodasz przełącznik `-v`, otrzymasz listing wraz z licznikiem pakietów i bajtów objętych przez daną regułę. Sprawdź te ilości, zwracając uwagę na reguły usuwające pakiety (DROP) bez ich wcześniejszego zalogowania. Jeżeli nie uda ci się znaleźć nieprawidłowo działającej reguły, dopisz reguły logujące wszystkie pakiety usuwane przez twój firewall.

Wyłączanie firewalla

Czasem istnieje potrzeba szybkiego wyłączenia firewalla. Poniżej zamieszczam zawartość skryptu czyszczącego ustawienia naszego firewalla. Pamiętaj, że po użyciu tego skryptu nadal działa NAT uruchomiony w skrypcie */etc/rc.d/rc.inet1*, czyli komputery z sieci lokalnej mają ciągle dostęp do Internetu. Dzięki temu użytkownicy nie powinni zauważyć tej operacji.

```
/etc/rc.d/firewall.stop
iptables -F
iptables -t mangle -F
iptables -P FORWARD ACCEPT
iptables -P INPUT ACCEPT
iptables -P OUTPUT ACCEPT
iptables -X syn-flood
/bin/echo "0" > /proc/sys/net/ipv4/icmp_echo_ignore_all
/bin/echo "0" > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts
/bin/echo "1" > /proc/sys/net/ipv4/conf/all/accept_source_route
/bin/echo "1" > /proc/sys/net/ipv4/conf/all/accept_redirects
/bin/echo "0" > /proc/sys/net/ipv4/icmp_ignore_bogus_error_responses
/bin/echo "0" > /proc/sys/net/ipv4/conf/all/rp_filter;
/bin/echo "0" > /proc/sys/net/ipv4/conf/all/log_martians
```

10.4. Dodatkowe funkcje

Ograniczenia na ICMP

Protokół ICMP jest protokołem kontrolnym Internetu, ma bardzo duże możliwości, a niestety nie zostały w nim zaimplementowane żadne mechanizmy związane z bezpieczeństwem. Ponadto protokół ten może być wykorzystany do różnorodnych ataków oraz w celu zapewnienia komunikacji różnych mało przyjaznych programów. Omówię teraz pokrótce zalecenia dotyczące ograniczania ruchu i potencjalne wykorzystanie poszczególnych komunikatów ICMP. Należy tak skonfigurować swój firewall, aby jedynie komputery udostępniające usługi do Internetu mogły używać komunikatów ICMP. Warto za pomocą zarządzania pasmem (w Linuksie 2.4 polecenie `tc`) określić maksymalne pasmo dostępne dla ruchu ICMP jako niewielką część całkowitego. Więcej informacji o zarządzaniu pasmem w systemie Linux znajdziesz pod w dokumencie „Kształtowanie Ruchu i Zaawansowany Routing" pod adresem *http://www.mr0vka.eu.org/docs*; oryginał tego i nowsze dokumenty znajdziesz *http://lartc.org*.

ICMP Echo

Nie należy uniemożliwiać ruchu tych pakietów. Oczywiście decyzja i tak zawsze należy do ciebie. Dobrym pomysłem jest filtrowanie pakietów tego typu skierowanych na adres rozgłoszeniowy. Można wprowadzić bardziej restrykcyjną politykę i zezwolić na używanie tego komunikatu jedynie pracownikom technicznym pionu utrzymania sieci. Ciekawym pomysłem jest ustawienie niestandardowego i dosyć niskiego maksymalnego rozmiaru pakietów przepuszczanych przez firewall i przeszkolenie pracowników w używaniu polecenia `ping` w celu generowania pakietów o takich właśnie rozmiarach.

ICMP Echo reply

Czasami stosowana jest metoda komunikacji z wykorzystaniem tych komunikatów. Nie będą one odpowiedzią na zapytania z wnętrza DMZ, ale przedostaną się poprzez firewalle nie dokonujące inspekcji stanu dla ICMP.

ICMP Timestamp

Może zostać wykorzystany do wykrywania komputerów. Należy zupełnie blokować.

ICMP Information request

Może zostać wykorzystany do wykrywania komputerów. Należy zupełnie blokować.

ICMP Parameter problem

Sfałszowany uszkodzony pakiet, wchodząc do naszej sieci, może wywołać taką odpowiedź naszego serwera, która daje pewną informację atakującemu.

ICMP Redirect

Tego typu komunikaty, jeśli już mogą się pojawić, to jedynie pomiędzy routerami. Może się zdarzyć, że zmienimy router w naszej sieci (zadziała jakiś system awaryjny). Jeśli zmieni się również tablica routingu poprzedniego routera, to będzie on informował za pomocą takich komunikatów komputery próbujące nadal wysyłać swoje pakiety przez niego o zmianie domyślnej bramy. Najczęściej możesz zupełnie zablokować takie komunikaty bez szkody dla swojej sieci.

ICMP Source quench

Nie są (jak na razie) niebezpieczne. Powinny być wysyłane jedynie w odpowiedzi na zbyt duży transfer.

ICMP Time exceeded

Komunikaty z kodem 0 używane są do wykrywania routerów; wystarczy wysyłać pakiety z coraz mniejszym parametrem TTL, a router powinien odpowiedzieć właśnie tym komunikatem z ustawionym kodem 0.

Jeśli zaczniemy wysyłać pofragmentowane pakiety z brakującym jednym z fragmentów, to badany host odpowie nam tym komunikatem z ustawionym kodem 1. Może to służyć do wykrywania hostów w sieci.

Nie należy blokować tych komunikatów.

ICMP Parameter problem

Może być wykorzystywany do wykrywania hostów; wystarczy wysyłać pakiety z niewłaściwymi wartościami, a komputery, które je odbiorą, powinny odpowiadać tym komunikatem.

ICMP Address request, Address reply

Oba komunikaty należy blokować. Nie ma powodu, dla którego miałyby z zewnątrz sieci docierać do nas takie komunikaty.

Jak przepuścić nową usługę na przykładzie Direct Connect

Podam tutaj przykład przepuszczania przez firewalla usług typu komunikatory, narzędzia do wymiany plików P2P. Podana wcześniej konfiguracja firewalla nie pozwoli nam na używanie popularnego w Polsce programu DirectConnect na komputerach lokalnych. Załóżmy, że naszym zadaniem jest uruchomienie dostępu do tej usługi. Możemy próbować uruchomić klienta DC i obserwować w logach, jakie pakiety są usuwane przez nasz firewall. Jednak dla niektórych protokołów może to być żmudne zadanie, więc o wiele prościej jest udać się na stronę producenta lub na jedną z polskich stron poświęconych DC. Znajdziemy tam opis potrzebnych ustawień. Przy DC ustawionym w tryb pasywny transfery z komputera lokalnego odbywają się protokołem TCP z wysokich portów na porty 411,412. Pozostaje nam zapisanie tego w postaci reguł iptables:

```
-p tcp -s 192.168.1.2 --sport 1024: -d IP --dport 411:412
-p tcp -d 192.168.1.2 --dport 1024: -s IP --sport 411:412
```

Okazuje się, że nasz firewall nie powinien ograniczać takich połączeń. Niestety, okazało się, że nadal są problemy z pracą aplikacji. Sprawdźmy więc, co jest usuwane przez firewall. Po obserwacji logów okazuje się, że należy dodać reguły przepuszczające również połączenia korzystające z UDP.

```
iptables -A FORWARD -i eth1 -p udp -s 192.168.1.0/30 --sport 411:413 -m state
--state NEW,ESTABLISHED,RELATED -j ACCEPT
iptables -A FORWARD -o eth1 -p udp -d 192.168.1.0/30 --sport 411:413 -m state
--state ESTABLISHED,RELATED -j ACCEPT
```

Jest to przykład rozwiązywania problemów tego typu; myślę, że nie raz będziesz zmuszony do rekonfiguracji swojego firewalla, choćby w celu przepuszczania komunikacji popularnych „czatów" z portali internetowych, opartych na komunikacji nawiązywanej na wysokie porty. Podam jeszcze jeden bardzo przydatny przykład:

```
# Unreal Tournament
iptables -A FORWARD -i eth1 -p udp -s 192.168.1.0/24 --sport 1024: --dport 7777
-m state --state NEW,ESTABLISHED,RELATED -j ACCEPT
```

Ustawianie priorytetów pakietów

Nieźle byłoby zapewnić naszym pakietom jakiś priorytet w Internecie. Ponieważ TCP/IP nie jest protokołem implementującym zbyt dobrze **QoS** (*Quality of Service*) oraz w rzeczywistości tylko niektóre routery w sieci spełniają założenia RFC, więc nie spodziewaj się cudów, ale nie zaszkodzi poustawiać bity **TOS** (*Type of Service*) w nagłówku protokołu IP. Parametry podane dalej zgodne są z zaleceniami RFC 1060/1349. Po wpisaniu `iptables -m tos -h` otrzymujemy listę proponowanych wartości bitów TOS:

```
                    dec (hex)
Minimize-Delay       16 (0x10)
Maximize-Throughput   8 (0x08)
Maximize-Reliability  4 (0x04)
Minimize-Cost         2 (0x02)
Normal-Service        0 (0x00)
```

Pierwsze trzy bity pola TOS (bity pierwszeństwa) są obecnie nie używane. Najważniejsze są następne cztery bity tego pola, ostatni nie jest używany. Poprawny pakiet powinien mieć ustawiony (o wartości 1) jedynie jeden z tych bitów. Propozycja ustawień dla poszczególnych aplikacji została zawarta w RFC 1340.

Tabela 10.2. *Zalecane wartości pola TOS*

Aplikacja	0 0 0	Minimalizacja opóźnień	Maksymalizacja szybkości	Maksymalizacja poprawności	Minimalizacja kosztów	0	Wartość szesnastkowa
		Minimize Delay	*Maximize Throughput*	*Maximize Reliability*	*Minimize Cost*		
Telnet, Rlogin, Ssh	0 0 0	1	0	0	0	0	0x10
FTP							
kontrola	0 0 0	1	0	0	0	0	0x10
dane	0 0 0	0	1	0	0	0	0x08
Dane masowe	0 0 0	0	1	0	0	0	0x08
TFTP	0 0 0	1	0	0	0	0	0x10
SMTP							
faza komend	0 0 0	1	0	0	0	0	0x10
faza danych	0 0 0	0	1	0	0	0	0x08
DNS							
pytanie UDP	0 0 0	1	0	0	0	0	0x10
pytanie TCP	0 0 0	0	0	0	0	0	0x00
transfer strefy	0 0 0	0	1	0	0	0	0x08
ICMP							
błąd	0 0 0	0	0	0	0	0	0x00
zapytanie	0 0 0	0	0	0	0	0	0x00
ICMP odpowiedź	Taka sama wartość, jak w odpowiadającym zapytaniu						
SNMP	0 0 0	0	0	1	0	0	0x04
BOOTP	0 0 0	0	0	0	0	0	0x00
NNTP	0 0 0	0	0	0	1	0	0x02

Na podstawie tych wartości dopisujemy na końcu naszego skryptu `/etc/rc.d/firewall.start` następujące wiersze:

```
# ustawiamy odpowiednio bity TOS w naglowku IP
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 20 -j TOS --set-
tos 8
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 21 -j TOS --set-
tos 16
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 22 -j TOS --set-
tos 16
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 23 -j TOS --set-
tos 16
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 25 -j TOS --set-
tos 16
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 53 -j TOS --set-
tos 16
iptables -t mangle -A PREROUTING -p udp -s 192.168.1.0/24 --dport 53 -j TOS --set-
tos 16
```

```
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 80 -j TOS --set-
tos 8
iptables -t mangle -A PREROUTING -p udp -s 192.168.1.0/24 --sport 411:413 -j TOS --
set-tos 16 # DC
iptables -t mangle -A PREROUTING -p tcp -s 192.168.1.0/24 --dport 411:413 -j TOS --
set-tos 8 # DC
```

Ostatnie dwa wiersze dotyczą transmisji Direct Connect i są dobrane eksperymentalnie. Aby podejrzeć zawartość tabeli *mangle*, wpisujemy: `iptables -t mangle -L`. Dla połączeń z komputera lokalnego również możemy poustawiać TOS.

```
iptables -t mangle -A OUTPUT -p tcp --dport 20 -j TOS --set-tos 8
iptables -t mangle -A OUTPUT -p tcp --dport 21 -j TOS --set-tos 16
iptables -t mangle -A OUTPUT -p tcp --dport 22 -j TOS --set-tos 16
iptables -t mangle -A OUTPUT -p tcp --dport 23 -j TOS --set-tos 16
iptables -t mangle -A OUTPUT -p tcp --dport 25 -j TOS --set-tos 16
iptables -t mangle -A OUTPUT -p tcp --dport 53 -j TOS --set-tos 16
iptables -t mangle -A OUTPUT -p udp --dport 53 -j TOS --set-tos 16
iptables -t mangle -A OUTPUT -p tcp --dport 80 -j TOS --set-tos 8
```

Wykrywanie skanowania za pomocą firewalla

Możemy utworzyć reguły, które będą nam raportowały w logach fakt skanowania naszego routera. Ponadto możemy wydzielić rodzaje skanów, aby mieć informację o metodach używanych przez agresora. Poniższe wpisy dodajemy przed łańcuchem *syn-flood* w pliku */etc/rc.d/firewall.start*.

```
# skan NULL musimy wykrywać osobno
iptables -A INPUT -p tcp --tcp-flags ALL NONE        -m limit --limit 10/s --limit-
burst 4 -j LOG --log-level debug --log-prefix "SKAN_NULL: "
iptables -A INPUT -p tcp --tcp-flags ALL NONE        -j DROP
# wszystkie pakiety uznane za NEW bez flagi SYN są podejrzane
iptables -N skany
iptables -A INPUT -i eth0 -p tcp ! --syn -m state --state NEW -j skany
iptables -A skany -p tcp --tcp-flags ALL RST         -m limit --limit 10/s --limit-
burst 4 -j LOG --log-level debug --log-prefix "SKAN_INVERSE: "
iptables -A skany -p tcp --tcp-flags ALL RST         -j DROP
iptables -A skany -p tcp --tcp-flags ALL ACK         -m limit --limit 10/s --limit-
burst 4 -j LOG --log-level debug --log-prefix "SKAN_TCP_PING: "
iptables -A skany -p tcp --tcp-flags ALL ACK         -j DROP
iptables -A skany -p tcp --tcp-flags ALL FIN         -m limit --limit 10/s --limit-
burst 4 -j LOG --log-level debug --log-prefix "SKAN_FIN: "
iptables -A skany -p tcp --tcp-flags ALL FIN         -j DROP
iptables -A skany -p tcp --tcp-flags ALL FIN,PSH,URG -m limit --limit 10/s --limit-
burst 4 -j LOG --log-level debug --log-prefix "SKAN_XMAS-NMAP: "
iptables -A skany -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
iptables -A skany -p tcp                             -m limit --limit 10/s --limit-
burst 4 -j LOG --log-level debug --log-prefix "SKAN_INNE: "
iptables -A skany -j DROP
```

Skan SYN zostanie wykryty przez łańcuch *syn-flood*. Ostatnia reguła zapewnia nam logowanie wszystkich innych podejrzanych pakietów. Aby nie spowodować zapchania logów systemowych, zostały dodane parametry `-m limit`, dzięki którym zapisywana będzie tylko część skanujących nas pakietów. Oczywiście, jeśli agresor ustawi politykę

skanowania w programie nmap np. na „Paranoid", to zostanie wykryty jedynie pierwszy pakiet, a pozostałe nie przekroczą częstotliwości ustawionej w module limit. Z tego powodu powinieneś zwracać uwagę na każdy wpis w logach, jednak pamiętaj, że dosyć często zdarzają się pakiety należące do uszkodzonych połączeń, które mogą powodować fałszywy alarm. Uwzględnij również, że używanie zaawansowanych modułów typu limit powoduje większe zużywanie zasobów systemowych.

Wykrywanie NAT

Postanowiłem opisać metody wykrywania, czy ktoś w naszej sieci nie udostępnia łącza, stosując NAT. Postaram się podać metodę, która w przypadku wstąpienia na drogę prawną może stanowić dowód. Przedstawię również metody zapobiegania wykryciu, które może zastosować użytkownik łącza.

Teraz nastąpi krótkie omówienie metod wykrywania uruchomionej translacji adresów NAT. Teoretycznie proces NAT jest nie do wykrycia, w rzeczywistości jest wiele metod, którymi można się posłużyć, aby go wykryć. Można podejrzeć zawartość pakietów TCP i stwierdzić na przykład, że wewnątrz nich adres źródłowy (np. przy protokole ICQ) komputera jest ustawiony na inny niż adres źródłowy datagramu IP. Choć takie działanie leży oczywiście w zakresie możliwości administratora sieci, jednak z punktu widzenia prawa jest nielegalne (nienaruszalność korespondencji, zachowanie prywatności) i nie może być użyte jako dowód. Ogólnym wnioskiem z różnych dyskusji jest, że administrator może podglądać jedynie nagłówki protokołów IP i TCP, UDP, ICMP. Można oczywiście na ich podstawie wnioskować o dokonywanej maskaradzie, jednak nie będzie to stuprocentowy dowód. Przykładowo maskarada wykonywana przez jądro Linuksa serii 2.2 dokonuje połączeń z pewnego ściśle określonego zakresu portów zdefiniowanych w pliku */usr/src/linux/include/net/ip_masq.h* (dla jądra rozpakowanego w katalogu */usr/src/linux*) w wierszach o treści:

```
#define PORT_MASQ_BEGIN 61000
#define PORT_MASQ_END (PORT_MASQ_BEGIN+4096)
```

Oczywiście wystarczy zmienić te wartości i przekompilować jądro, aby zakres był zupełnie inny. W nowym jądrze 2.4 możesz teraz używać portów 61000 – 65095, nawet jeśli działa NAT. Kod maskarady zakładał do tej pory, że ten zakres portów jest zarezerwowany do jego użytku, więc programy nie mogły go używać. Z moich obserwacji wynika, że w jądrach 2.4 NAT został rozwiązany poprawnie i alokuje porty od 1024 wzwyż — czyli tak, jak powinien to robić normalny, pojedynczy komputer. Dla procesów lokalnych również alokuje kolejne porty powyżej 1024 i nie powoduje to konfliktów z portami zajętymi dla NAT-u.

Maskarada zmienia wartości nagłówka IP (adres) i protokołów niższych TCP/UDP (port), co powoduje zmianę sum kontrolnych obu protokołów. Niektóre rodzaje oprogramowania NAT nie zmieniały tej sumy. Oczywiście aby to wykryć, również musisz przeczytać cały pakiet, do czego nie masz prawa. Czyli ponownie posiadasz informację o używaniu NAT, ale nie może ona być pełnoprawnym dowodem. Ponadto są różne metody oparte na wykrywaniu systemu operacyjnego (*fingerprinting*) i porównaniu z np. rodzajem przeglądarki WWW raportowanej przez protokół HTTP. Przykładowo jeśli host zachowuje się jak Linux, a przeglądarką podaną w nagłówkach HTTP jest „MS

IE", to sprawa jest prosta. Ta metoda również jest bezprawna, ponieważ musiałbyś odczytywać prywatną transmisję. Myślę, że na podstawie analizy działania protokołów TCP/IP i protokołów warstw wyższych można wymyślić jeszcze wiele metod na wykrywanie NAT.

Ciekawą metodą jest sprawdzanie, czy badany host ma uruchomiony proces routingu, który jest potrzebny do poprawnego działania NAT. Załóżmy, że adres podejrzanego komputera to 192.168.1.100. Na swoim routerze konfigurujesz jakąś sieć, do której testowany komputer rzekomo będzie routerem.

```
# route add -net 1.2.3.0/24 gw 192.168.1.100 eth1
# ping 1.2.3.4
```

Równocześnie (na drugiej konsoli) uruchamiasz sniffer. Jeśli okaże się, że testowany komputer odpowiedział na ping do ciebie, będzie to oznaczało, że ma uruchomiony routing. Jeśli nie miałby routingu, nie powinien reagować w żaden sposób na pakiety nie przeznaczone dla niego.

Jest jeszcze jeden, niemal pewny sposób na wykrycie NAT. Po prostu nasz firewall jest oczywiście routerem, a każdy router ma obowiązek zmniejszać pole TTL w nagłówku IP o 1. Uwzględniając, że komputery z systemem Windows wysyłają pakiety z TTL = 128, to pakiety te, wychodząc z karty eth0 routera, będą miały już ustawiony TTL = 127. Można to prosto sprawdzić:

```
# tcpdump -v -n -i eth0
```

Pakiety generowane lokalnie przez nasz router mają ustawioną charakterystyczną dla Linuksa wartość TTL = 64. Aby zwiększyć wartość pola TTL o jeden, czyli ukryć fakt przechodzenia pakietów przez router, na końcu naszego skryptu konfigurującego firewalla dodajemy wiersz:

```
# Zwiększamy o 1 pole TTL pakietów przechodzących przez nasz NAT
iptables -t mangle -A PREROUTING -s 192.168.1.0/24 -j TTL --ttl-inc 1
```

Zmiana wartości TTL odbywa się w tabeli *mangle* (`-t mangle`) w łańcuchu PREROUTING. Nie możemy wykorzystać łańcucha OUTPUT tej tabeli (innych łańcuchów ona nie posiada), ponieważ zmieniałby on tylko pakiety wychodzące bezpośrednio z routera, a nie przechodzące przez niego. Ograniczamy pakiety zmieniane przez regułę do pochodzących z komputerów w sieci lokalnej 192.168.1.0/24, na których dokonujemy procesu NAT. Niestety, najprawdopodobniej nie sprowadzi się to do wpisania dodatkowej reguły do pliku konfigurującego firewalla. Najczęściej w dystrybucjach Linuksa pakiet iptables jest bez łaty (*patch*) udostępniającej opcję `-j TTL`. Aby ją włączyć, będziemy musieli pobrać źródła pakietu iptables, zainstalować łaty na jądro Linuksa zawarte w pakiecie, skompilować i zainstalować nowe jądro oraz skompilować samo iptables. Oczywiście wcześniej najlepiej jest usunąć pakiet iptables, który mieliśmy zainstalowany wraz z naszą dystrybucją Linuksa.

Pakiet iptables ściągamy ze strony domowej projektu netfilter — *http://www.netfilter.org*. Należy zajrzeć do FAQ i z podanych tam adresów ściągnąć iptables. Następnie przeczytać rozdział dotyczący **patch-o-matic**. Jest to system dodatków do iptables; jednym z nich jest właśnie łata TTL. Rozpakowujemy iptables i kompilujemy patche:

```
$ tar -xvzf iptables-1.2.4.tar.bz2
$ cd iptables-1.2.4
# make patch-o-matic
```

Otrzymamy listę pytań, najpierw o położenie źródeł jądra, na które chcemy zainstalować dodatki, a następnie o to, czy chcemy użyć kolejnych patchy zawartych w patch-o-matic. Powinniście wybrać te związane z TTL (może również inne, jeśli spodobają się ci ich możliwości), a następnie skompilować i zainstalować samo iptables:

```
# make
# make install
```

Później czeka nas konfiguracja, kompilacja i instalacja nowego jądra, ale tego już nie będę omawiał. Pamiętaj jedynie, aby przy konfiguracji jądra zaznaczyć (choćby jako moduły) elementy dotyczące iptables (a zwłaszcza TTL). Oczywiście nie wszystkie musisz wkompilowywać bezpośrednio w jądro, ale to już zależy od ciebie.