Administracja Systemem Linux/Unix

Uniwersytet Marii Curie-Skłodowskiej
Wydział Matematyki, Fizyki i Informatyki
Instytut Informatyki

Administracja Systemem Linux/Unix

Bogdan Księżopolski, Damian Rusinek

Lublin 2012

Instytut Informatyki UMCS
Lublin 2012

Bogdan Księżopolski, Damian Rusinek

Administracja Systemem Linux/Unix

Recenzent: Andrzej Bobyk

Opracowanie techniczne: Marcin Denkowski
Projekt okładki: Agnieszka Kuśmierska

Praca współfinansowana ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Publikacja bezpłatna dostępna on-line na stronach
Instytutu Informatyki UMCS: informatyka.umcs.lublin.pl.

Wydawca
Uniwersytet Marii Curie-Skłodowskiej w Lublinie
Instytut Informatyki
pl. Marii Curie-Skłodowskiej 1, 20-031 Lublin
Redaktor serii: prof. dr hab. Paweł Mikołajczak
www: informatyka.umcs.lublin.pl
email: dyrii@hektor.umcs.lublin.pl

Druk
FIGARO Group Sp. z o.o. z siedzibą w Rykach
ul. Warszawska 10
08-500 Ryki
www: www.figaro.pl

ISBN: 978-83-62773-24-4

Spis treści

Rozdział 1

Wstęp

1.1. O czym jest ta książka?

Dzięki tej książce czytelnik będzie mógł poznać system Linux zarówno ze strony użytkownika, jak również administratora. Pierwszy rozdział zawiera wstęp do książki, który właśnie czytasz. Drugi rozdział dotyczy podstaw systemu Linux i przeznaczony jest dla początkujących użytkowników systemu. Trzeci rozdział przedstawia techniki przetwarzania tekstu, które są przydatne w codziennej pracy z systemem. Rozdział czwarty opisuje zagadnienie tworzenia skryptów powłoki BASH. Tworzenie skryptów powłoki, jest istotną umiejętnością w pracy administratora systemu Linux. Rozdział piąty opisuję podstawową oraz zaawansowaną konfigurację serwera WWW na podstawie oprogramowania *apache*. Szósty rozdział dotyczy zagadnień tworzenia tzw. zapór ogniowych oraz translacji adresów sieciowych przy pomocy oprogramowania *iptables*. Siódmy rozdział przedstawia zagadnienie tzw. skanowania jako techniki monitorowania serwerów sieciowych.

Wszystkie fragmentu kodu oraz przykłady przedstawione w pierwszych czterech rozdziałach, zostały napisane oraz sprawdzone w dystrybucji Debian GNU/Linux 5.0. Kolejne trzy rozdziały książki, bazują na dystrybucji Ubuntu 11.04, która również wywodzi się z dystrybucji Debian.

Rozdział 2

Podstawy systemu Linux

W niniejszym rozdziale omówimy podstawowe komendy systemu Linux, dzięki którym będziemy mogli poruszać się po systemie, a także zbierać informację o nim i o jego elementach.

Dowiemy się również, jak rozpocząć pracę z systemem Linux zaraz po zalogowaniu.

2.1. Rozpoczęcie pracy

Po uruchomieniu systemu użytkownik musi zalogować się na swoje konto za pomocą nazwy użytkownika (inaczej loginu).

```
login as:
```

Rysunek 2.1. Okno logowania

Po wpisaniu loginu należy podać również swoje hasło. Zarówno w przypadku loginu oraz hasła wielkość liter ma znaczenie.

Po poprawnym zalogowaniu się na swoje konto naszym oczom ukazuje się znak zachęty (ang. prompt), którego przykładowy wygląd jest widoczny na rysunku 2.2.

```
user@host:/etc$
```

Rysunek 2.2. Znak zachęty

Powyższy znak zachęty składa się z nazwy użytkownika *user*, nazwy komputera *host* oraz scieżki do aktywnego katalogu; w tym przypadku aktywnym folderem jest */etc*. Ostatni znak $ może mieć postać hasha #, jeśli efektywny UID jest równy 0 (użytkownik ma prawa roota) lub znaku dolara $, jeśli użytkownik nie ma praw roota.

Jeśli aktywnym katalogiem jest folder domowy użytkownika (zwykle */home/nazwa_użytkownika*), to ścieżka jest zastępowana znakiem tyldy ∼.

Wygląd znaku zachęty można zmienić za pomocą zmiany zmiennej środowiskowej PS1. Zmienne środowiskowe omówimy w następnym rozdziale.

2.2. Praca z konsolą

Użytkownik po zalogowaniu się wykorzystuje jedną z 7 konsoli wirtualnych, udostępnionych przez system. Ostatnia - siódma - konsola jest zarezerwowana dla sesji środowiska graficznego X Window System. Pozostałe sześć konsoli to konsole tekstowe.

Użytkownik ma możliwość przełączania się między konsolami za pomocą dwóch metod. Pierwsza z nich to specjalne kombinacje klawiszy, druga to specjalna komenda.

2.2.1. Przełączanie się między konsolami kombinacją klawiszy

W celu przełączenia się na inną konsolę należy wcisnąć klawisz *ALT* oraz klawisz funkcyjny *Fn*, gdzie *n* jest numerem konsoli. Dodatkowo między sąsiadującymi konsolami można się przełączać za pomocą kombinacji *ALT* i *strzałka w lewo/prawo* [1].

Metoda ta umożliwia przełączenie się z konsoli tekstowej na dowolną inną. Jednakże w przypadku sesji środowiska graficznego X kombinacje te nie działają. W tym przypadku należy użyć kombinacji klawiszy *CTRL*, *ALT* i klawisz funkcyjny z numerem konsoli, na którą chcemy się przełączyć. Jest to jedyny skrót, który działa w środowisku X.

W środowisku graficznym można korzystać z konsoli okienkowych, czyli bez potrzeby przełączania się do konsoli tekstowej. Bardzo wygodnym narzędziem do pracy z konsolami jest **yakuake**. W kolejnych rozdziałach pokażemy jak w łatwy sposób instalować pakiety (programy i narzędzia).

2.2.2. Przełączanie się między konsolami za pomocą komendy

Innym sposobem przełączania się między konsolami jest skorzystanie z komendy chvt. Przykładowe użycie zostało zaprezentowane na rysunku 2.3.

```
chvt numer_konsoli
```

Rysunek 2.3. Przełączanie się między konsolami

Załóżmy, że pracujemy na konsoli nr 3 i chcemy przejść na konsolę nr 4. Mamy wtedy 3 możliwości:
— wykorzystanie kombinacji klawiszy *ALT + F4*;
— wykorzystanie kombinacji klawiszy *ALT* + ->;
— użycie komendy *chvt 4*.

2.2.3. Przewijanie tekstu w konsoli

Konsole uruchomione w sesji środowiska graficznego posiadają paski przewijania, dzięki którym możemy przewinąć i podejrzeć wiersze, które zostały przesunięte za ekran konsoli przez nowe wiersze.

W przypadku konsoli tekstowych włączanych za pomocą kombinacji ALT + F1-F6 nie ma paska i do przewijania wierszy należy wykorzystać dwie kombinacje klawiszy:
— w celu pokazania wierszy, które zostały przesunięte poza górną krawędź ekranu musimy skorzystać z kombinacji **SHIFT** + **PAGE UP**;
— w celu pokazania wierszy, które zostały przesunięte poza dolną krawędź ekranu (np. przez skorzystanie z powyższej kombinacji) musimy skorzystać z kombinacji **SHIFT** + PAGE DOWN.

Kombinacje klawiszy SHIFT + PAGE UP/DOWN nie przewijają wierszy w obrębie aplikacji uruchomionych z poziomu konsoli. W tym celu należy korzystać z mechanizmów przewijania, jakie dostarczają dane aplikacje.

2.3. Informacje o użytkowniku

W tym podrozdziale omówimy komendy, które operują na kontach użytkowników, w szczególności na koncie użytkownika, który jest aktualnie zalogowany.

2.3.1. Przedstaw się

W celu sprawdzenia nazwy użytkownika należy skorzystać z komendy *whoami*. Przykład wykorzystania przedstawiono na rysunku 2.4.

```
user@host:~$ whoami
user
```

Rysunek 2.4. Sprawdzenie nazwy zalogowanego użytkownika

Komenda ta może wydawać się nielogiczna, gdyż przecież każdy użytkownik wie, na jakie konto się logował. Sens tej komendy wróci, gdy zajmiemy się skryptami BASH. Skrypty nie wiedzą, który użytkownik jest zalogowany na systemie i dzięki tej komendzie mogą uzyskać jego nazwę. Przykładowe sytuacje, gdy ta informacja jest potrzebna to:
— stworzenie folderu użytkownika, w którym będą przechowywane dane pochodzące z uruchomionej aplikacji;
— wykonanie różnych operacji w zależności od tego, który użytkownik uruchamia skrypt.

2.3.2. Zmiana hasła

Żeby zmienić hasło, użytkownik musi skorzystać z komendy *passwd*. Przykład zmiany hasła został przedstawiony na rysunku 2.5.

```
user@host:~$ passwd
Zmienianie hasła dla user.
Bieżące hasło UNIX:
Podaj nowe hasło UNIX:
Ponownie podaj hasło UNIX:
passwd: hasło zostało zmienione
```

Rysunek 2.5. Zmiana hasła

Jak widać na rysunku 2.5 po uruchomieniu procesu zmiany hasła, należy podać stare hasło. Jest to zabezpieczenie przed sesją pozostawioną przez użytkownika. Gdyby obca osoba podeszła do stanowiska i nie byłoby tego zabezpieczenia, to mogłaby ona zmienić hasło zalogowanemu użytkownikowi.

Następnie należy podać nowe hasło i je potwierdzić. Wynika to z faktu, że hasła podczas wpisywania nie są wyświetlane na ekranie. To również jest zabezpieczenie

przed obcymi osobami, które mogłyby je podejrzeć. Gdy nowe hasła się zgadzają, to wyświetlona zostanie informacja, że hasło zostało zmienione.

2.3.3. Lista zalogowanych użytkowników

W celu podejrzenia listy użytkowników, którzy są aktualnie zalogowani w systemie, musimy skorzystać z komendy *who*. Przykładowy wynik komendy jest przedstawiony na rysunku 2.6.

```
user   pts/0        2011-09-30 14:08 (IP)
```

Rysunek 2.6. Lista aktywnych użytkowników

Lista użytkowników składa się z czterech kolumn. Pierwsza z nich to nazwa użytkownika, druga to terminal, na którym użytkownik jest zalogowany, trzecia to data logowania, zaś ostatnia to komentarz (w tym przypadku jest to IP, z którego użytkownik jest zdalnie zalogowany). W celu dodania nagłówków kolumn należy skorzystać z flagi -H.

```
user@host:~$ who -H
UŻYTKOWNIK   TERM         CZAS                        KOMENTARZ
user             pts/0         2011-09-30 14:08   (IP)
```

Rysunek 2.7. Lista aktywnych użytkowników z tytułami kolumn

2.3.4. Lista zalogowanych użytkowników i operacje przez nich wykonane

Komenda *who* wyświetla podstawowe informacje o zalogowanych użytkownikach. Jeśli potrzebujemy bardziej szczegółowych informacji to powinniśmy skorzystać z komendy *w*. Wynik komendy został przedstawiony na rysunku 2.8.

```
user@host:~$ w
 14:42:17 up 8 days,  2:22, 1 user, load average: 0,00, 0,00, 0,00
USER     TTY      FROM        LOGIN      IDLE    JCPU   PCPU  WHAT
user   pts/0    IP          14:08      0.00s   0.36s  0.00s  w
```

Rysunek 2.8. Wynik komendy *w*

Komenda w na początku wyświetla informacje o systemie takie jak:
— aktualny czas systemu;
— okres przez jaki system działa;
— ilość zalogowanych użytkowników.

Poniżej wyświetlona jest ośmiokolumnowa lista użytkowników, gdzie:
— kolumna USER to nazwa użytkownika;
— kolumna TTY to terminal, na którym użytkownik pracuje;
— kolumna FROM zawiera adres IP, z którego użytkownik jest zalogowany zdalnie lub jest pusty, gdy użytkownik jest zalogowany lokalnie;
— kolumna LOGIN to data zalogowania się do systemu;
— kolumna IDLE to okres, przez jaki użytkownik był nieaktywny;
— kolumna JCPU to całkowity czas procesora przydzielony na wykonanie zadań użytkownika;
— kolumna PCPU to całkowity czas procesora przydzielony na wykonanie zewnętrznych procesów;
— kolumna WHAT to nazwa aktualnie wykonywanego procesu.

Jak widać na rysunku 2.8, użytkownik *user* zalogował się o godzinie *14:08* i aktualnie wyświetlił listę zalogowanych użytkowników (komenda *w*).

2.3.5. Ostatnie logowanie

W celu sprawdzenia ostatniej daty logowania użytkownika, który nie musi być zalogowany, musimy skorzystać z komendy *last*.

```
user@CSS:~$ last
user pts/0       IP           Fri Sep 30 14:08   still logged in
user pts/0       IP           Thu Sep 29 13:13 - 17:44  (04:31)
user pts/0       IP           Thu Sep 29 13:11 - 13:12  (00:00)
reboot system boot 2.6.26-2-686 Thu Sep 22 12:20 - 15:14 (8+02:53)
user pts/0       IP           Fri Sep 16 01:58 - 01:58  (00:00)
reboot system boot 2.6.26-2-686 Thu Sep 15 13:10 - 15:14 (15+02:03)
```

Rysunek 2.9. Wynik komendy *last*

Użycie komendy bez żadnego argumentu wyświetla listę ostatnich logowań do systemu. Ja widać na rysunku 2.9 ostatnio do systemu logował się tylko użytkownik *user*. Daty logowania oraz czas zalogowania znajdują się w ostatniej kolumnie.

Jeśli chcemy sprawdzić, kiedy ostatnio logował się konkretny użytkownik, to jako argument musimy podać jego nazwę.

```
user@CSS:~$ last bogdan
bogdan   pts/0        host       Thu Sep  1 15:42 - 15:47  (00:04)
bogdan   pts/0        host       Thu Sep  1 15:41 - 15:41  (00:00)
```

Rysunek 2.10. Wynik komendy *last* dla jednego użytkownika

2.4. Informacje o systemie

Informacje o systemie możemy ogólnie podzielić na dwa rodzaje: informacje, które się nie zmieniają, takie jak np. zainstalowany sprzęt czy dystrybucja oraz te, które się ciągle zmieniają, takie jak uruchomione procesy, zużycie pamięci czy procesora.

Na początku rozdziału napisaliśmy, że wszystkie fragmentu kodu i zrzuty ekranu pochodzą z dystrybucji Debian GNU/Linux 5.0. W celu sprawdzenia jaka dystrybucja systemu Linux jest zainstalowana należy użyć komendy *head* na pliku konfiguracyjnym */etc/issue*, w sposób przedstawiony na rysunku 2.11.

```
head -n1 /etc/issue
```

Rysunek 2.11. Składnia komendy sprawdzającej zainstalowaną dystrybucję systemu Linux

Składnia komendy head zostanie omówiona w następnym rozdziale.

2.4.1. Procesy

Podstawową komendą do wyświetlenia listy uruchomionych procesów jest komenda *ps*. Uruchamiając tę komendę bez żadnych flag, otrzymamy prostą listę procesów uruchomionych przez użytkownika, na którego się zalogowaliśmy.

W celu wyświetlenia wszystkich procesów wszystkich użytkowników wraz wykorzystaniem pamięci i procesora należy użyć dodatkowych flag *aux*.

```
ps aux
```

Rysunek 2.12. Komenda wyświetlająca rozszerzoną listę uruchomionych procesów

W przypadku, gdy chcemy wyświetlić procesy jednego użytkownika, to musimy skorzystać z flagi *-u*.

```
ps -u user
```

Rysunek 2.13. Wszystkie procesy użytkownika *user*

Komenda *ps* wyświetla stan procesów z danej chwili, w której została uruchomiona. W celu wyświetlenia listy procesów, która jest co chwilę odświeżana, należy użyć komendy *top*. Wynik tej komendy jest przedstawiony na rysunku 2.14.

Komenda *top* w skrócie wyświetla informację o aktualnym stanie systemu. Wyświetla ilość tasków z podziałem na ich aktualny stan, informacje o pamięci operacyjnej, przestrzeni wymiany oraz o zużyciu procesora.

Dodatkowo wyświetlana jest lista procesów, która jest co chwilę odświeżana. Znajdują się na niej podstawowe informację dotyczące każdego z procesów, takie jak

```
top - 14:42:24 up 14 days,  2:22,  1 user,  load average: 0.00, 0.00, 0.00
Tasks:  75 total,   1 running,  74 sleeping,   0 stopped,   0 zombie
Cpu(s):  0.3%us,  0.3%sy,  0.0%ni, 99.3%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:    254676k total,   235412k used,    19264k free,    95632k buffers
Swap:   746980k total,     1312k used,   745668k free,    39720k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
12156 damian    20   0  2392 1100  876 R  0.7  0.4   0:00.02 top
    1 root      20   0  2104  684  588 S  0.0  0.3   0:14.00 init
    2 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 migration/0
    4 root      15  -5     0    0    0 S  0.0  0.0   0:09.74 ksoftirqd/0
    5 root      RT  -5     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
    6 root      15  -5     0    0    0 S  0.0  0.0   0:46.32 events/0
    7 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 khelper
   39 root      15  -5     0    0    0 S  0.0  0.0   0:01.32 kblockd/0
   41 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kacpid
   42 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kacpi_notify
  113 root      15  -5     0    0    0 S  0.0  0.0   0:00.08 kseriod
  146 root      20   0     0    0    0 S  0.0  0.0   0:00.00 pdflush
  147 root      20   0     0    0    0 S  0.0  0.0   2:13.98 pdflush
  148 root      15  -5     0    0    0 S  0.0  0.0   0:00.16 kswapd0
  149 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 aio/0
  599 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ksuspend_usbd
  600 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 khubd
  635 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ata/0
  636 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 ata_aux
  822 root      15  -5     0    0    0 S  0.0  0.0   0:00.42 reiserfs/0
  898 root      16  -4  2288  784  488 S  0.0  0.3   0:00.34 udevd
 1323 root      15  -5     0    0    0 S  0.0  0.0   0:00.00 kpsmoused
 1760 root      20   0  2184  600  480 S  0.0  0.2   0:11.74 dhclient3
 1785 daemon    20   0  1896  508  416 S  0.0  0.2   0:00.00 portmap
```

Rysunek 2.14. Wynik komendy *top*

identyfikator, nazwa, czy użycie pamięci oraz procesora. Oprócz nich lista zawiera również dodatkowe informacje o procesie, jak np. priorytet (kolumna PR), czy stan procesu (kolumna S).

Proces może znajdować się w jednym z następujących stanów:

— R (ang. running) - proces aktualnie wykonywany;
— S (ang. sleeping) - proces w stanie spoczynku;
— T (ang. stopped) - proces zatrzymany *Z (ang. zombie) - proces skończony, który czeka aż jego pozostałości zostaną usunięte z pamięci.*

Czasami zdarzy się sytuacja, kiedy program się zawiesi i przestanie odpowiadać. Wtedy jedynym wyjściem jest zakończenie jego działania przez zabicie jego procesu. Do tego służy komenda wysyłająca sygnały do procesów o nazwie *kill*.

```
kill identyfikator_procesu
```

Rysunek 2.15. Komenda kończąca proces

Najprostszym sposobem zabicia procesu jest komenda przedstawiona na rysunku 2.15. Jeśli nie określimy typu wysyłanego sygnału, to domyślnie wysyłany

jest sygnał TERM, kończący proces. Identyfikator procesu można pobrać z listy wyświetlonej przez komendę *ps* lub *top*.

Niektóre procesy przechwytują i ignorują sygnał TERM. Wtedy należy przesłać sygnał, który nie może zostać przechwycony. W tym celu jako jedną z flag należy dodać -9. Przykład jest przedstawiony na rysunku 2.16.

```
kill -9 identyfikator_procesu
```

Rysunek 2.16. Komenda wymuszająca zakończenie procesu

2.5. Poruszanie się po strukturze katalogów

Główną komendą do poruszania się po strukturze katalogów jest komenda *cd*.

```
cd ścieżka
```

Rysunek 2.17. Składnia komendy cd

Argument *ścieżka* może być ścieżką bezwzględną lub względną. Ścieżka bezwzględna rozpoczyna się od znaku slash / i zawiera wszystkie foldery począwszy od folderu głównego systemu.

Znak slash / jest utożsamiany z folderem głównym systemu. W przeciwieństwie do systemu Windows, w systemie Linux foldery rozdzielane są znakiem slash /, a nie backslash \.

Ścieżka względna rozpoczyna się od folderu w którym aktualnie się znajdujemy. Stąd, chcąc przejść do folderu, który jest podfolderem aktualnie odwiedzanego folderu, nie musimy pisać całej bezwzględnej ścieżki, a jedynie nazwę podfolderu.

Załóżmy, że znajdujemy się w folderze */etc* i chcemy sie dostać do folderu */etc/init.d*

— W przypadku ścieżki bezwzględnej będzie to komenda *cd /etc/init.d*

— W przypadku ścieżki względnej będzie to po prostu *cd init.d* lub *cd ./init.d*

Kropka reprezentuje folder, w którym aktualnie się znajdujemy.

Za pomocą ścieżki względnej możemy również przechodzić do folderów nadrzędnych. Aby to zrobić, należy wpisać dwie kropki ... Na przykład chcąc się dostać z folderu */etc* do folderu */var* należy użyć komendy *cd ../var*

2.6. Zmienne środowiskowe

Zmienne środowiskowe zwane również systemowymi to zmienne, które przechowują wartości zdefiniowane zarówno przez system, jak i przez użytkowników oraz uruchamiane skrypty. Można je porównać do zmiennych globalnych, czyli są

dostępne w całym systemie i w całym systemie przechowują tę samą wartość. Posiadają swoją unikalną nazwę, za pomocą której są rozróżniane.

Wielkość liter w nazwach zmiennych ma znaczenie. Zmienna *ZMIENNA* oraz *Zmienna* to dwie różne zmienne.

W systemie istnieją predefiniowane zmienne, których nazwy są zarezerwowane. Dlatego podczas tworzenia nowych zmiennych trzeba uważać, żeby nie próbować zapisać wartości istniejącej zmiennej. Tych zmiennych nie można zmienić, ale istnieją również predefiniowane zmienne, których wartości można zmieniać, np. PATH. Wartości tych zmiennych należy modyfikować ostrożnie.

Przykładowe predefiniowane zmienne to:
— HOME - ścieżka do folderu domowego zalogowanego użytkownika;
— PATH - wskazuje folderu, w których należy szukać plików binarnych;
— PS1 - sposób wyświetlania znaku zachęty;
— EDITOR - nazwa domyślnego edytora tekstu;
— SHELL - ścieżka do powłoki zalogowanego użytkownika;
— i inne.

Wszystkie zmienne środowiskowe można wyświetlić za pomocą komendy *env*.

2.6.1. Tworzenie zmiennych środowiskowych

W celu stworzenia lub zmiany wartości zmiennej środowiskowej należy wpisać w konsoli jej nazwę, znak równości (=) oraz wartość w cudzysłowie.

```
NAZWA_ZMIENNEJ="WARTOŚĆ"
```

Rysunek 2.18. Ustalenie wartości zmiennej środowiskowej

Przypisując wartość do zmiennej, należy pamiętać o tym, aby nie umieszczać spacji między znakiem równości a nazwą zmiennej oraz cudzysłowem.

Tak zapisana zmienna będzie dostępna w aktualnej powłoce oraz będzie mogła być wykorzystywana w skryptach wykonywanych w tej powłoce. Aby zmienna była dostępna również w jej podpowłokach, należy ją wyeksportować. W zależności od rodzaju powłoki robimy to za pomocą różnych komend:
— **bash** - komenda *export*;
— **sh** - komenda *export*;
— **tcsh** - komenda *setenv*.

W ten sposób utworzymy zmienną, z której będziemy mogli korzystać w wielu powłokach, ale tylko do momentu wylogowania się.

2.6.2. Zapisywanie zmiennych środowiskowych na stałe

Każda powłoka systemu posiada swoje pliki startowe, które są uruchamiane po zalogowaniu się użytkownika. Jako, że w tym skrypcie wykorzystywana jest powłoka BASH, to właśnie na jej przykładzie pokażemy jak zapisać zmienną środowiskową na stałe z wykorzystanie plików startowych powłoki.

Pliki, których nazwy rozpoczynają się od kropki, są ukryte. W niektórych systemach nie ma pliku *.bash_ profile*. W BASHu pliki startowe znajdują się w folderze

domowym użytkownika i nazywają się *.bash_profile* oraz *.bashrc*. Jeśli przyjrzymy się ich zawartości, to zauważymy, że już są w niej ustalane niektóre zmienne środowiskowe, np. PS1. W celu zapisania na stałe naszej zmiennej, musimy dopisać do pliku komendę, która ją ustawia - przykład na rysunku 2.19.

```
export NAZWA_ZMIENNEJ="WARTOŚĆ"
```

Rysunek 2.19. Ustalenie wartości zmiennej środowiskowej

Dzięki tej linii system będzie wykonywał tę komendę po każdorazowym zalogowaniu się użytkownika, dzięki czemu zmienna będzie dostępna już na stałe.

Pliki startowe można wykorzystać do wywoływania dowolnych komend, czy skryptów podczas logowania. Częstym przypadkiem wykorzystania tego mechanizmu jest ustawianie aliasów. Jeśli otworzymy ponownie plik *.bashrc*, to zauważymy linie podobne do przedstawionej na rysunku 2.20.

```
alias ls='ls --color=auto'
```

Rysunek 2.20. Przykład definiowania aliasu

W tym przypadku alias jest ustawiony na komendę *ls*, który powoduje nadpisanie istniejącej komendy *ls* jej pokolorowaną wersją, czyli *ls –color=auto*. Użytkownik może dowolnie ustalać swoje aliasy i analogicznie do zmiennych środowiskowych umieszczać w plikach startowych, żeby były dostępne po wylogowaniu się.

2.6.3. Pobieranie wartości zmiennych środowiskowych

W celu pobrania wartości zmiennej należy poprzedzić jej nazwę znakiem dolara $. Samo pobranie wartości zmiennej nic nam nie da i nie będziemy mogli się przekonać czy działa, dlatego użyjemy komendy *echo*, która jako argument przyjmuje wartość, którą wyświetla na ekranie. Przykład został przedstawiony na rysunku 2.21.

```
user@host:~$ MOJA_ZMIENNA="test"
user@host:~$ echo $MOJA_ZMIENNA
test
```

Rysunek 2.21. Pobieranie wartości zmiennej

Szczególnym przypadkiem jest wyświetlenie wartość zmiennej np. połączonej z innym ciągiem znaków. Jeśli do do zmiennej MOJA_ZMIENNA chcielibyśmy dopisać ciąg znaków *owy*, to musimy skorzystać z nawiasów klamrowych. Na rysunku 2.22 został przedstawiony przykład poprawnego pobrania wartości zmiennej w takiej sytuacji. Przykład na rysunku 2.23 nie zadziała, gdyż w tym przypadku

będziemy próbowali pobrać wartość zmiennej MOJA_ZMIENNAowy, która nie jest zdefiniowana.

```
echo ${MOJA_ZMIENNA}owy
```

Rysunek 2.22. Poprawna konkatenacja zmiennej z ciągiem znaków

```
echo $MOJA_ZMIENNAowy
```

Rysunek 2.23. Błędna konkatenacja zmiennej z ciągiem znaków

2.6.4. Usuwanie zmiennych środowiskowych

Istnieją dwie możliwości usunięcia zmiennych środowiskowych. Pierwszą z nich jest nie tyle trwałe usunięcie, co jej wyczyszczenie. W tym celu należy wyeksportować zmienna z wartością równą pustemu stringowi - rysunek 2.24.

```
export NAZWA_ZMIENNEJ=""
```

Rysunek 2.24. Wyczyszczenie zmiennej środowiskowej

W celu trwałego usunięcia zmiennej należy skorzystać z komendy *unset* (dla powłoki *tcsh* komenda *unsetenv*) i jako parametr podać nazwę zmiennej - rysunek 2.25.

```
unset NAZWA_ZMIENNEJ
```

Rysunek 2.25. Usunięcie zmiennej środowiskowej

2.6.5. Zmiana wyglądu znaku zachęty

Jako przykład wykorzystania zmiennych środowiskowych zajmiemy się zmienną PS1, która służy do zdefiniowania wyglądu znaku zachęty. W tym celu należy nadać nową wartość zmiennej PS1. Oczywiście musimy mieć zestaw znaków specjalnych, dzięki którym wyświetlimy w znaku zachęty informacje, takie jak nazwa użytkownika, host, ścieżka aktualnego pliku, itp.

Przykładowe znaki specjalne, z których możemy korzystać to:
— $\backslash w$ - bieżący katalog;
— $\backslash a$ - dzwonek systemowy;
— $\backslash t$ - czas w formacie gg:mm:ss;

— \d - data w formacie dzień tygodnia - miesiąc - dzień miesiąca;
— \s - nazwa powłoki;
— \u - nazwa użytkownika;
— \h - nazwa hosta do pierwszej kropki;
— \XXX - szesnastkowa liczba XXX.

Dla przykładu sprawdźmy, jak będzie wyglądał znak zachęty po zmianie wartości zmiennej PS1 na podaną na rysunku 2.26.

```
PS1='\d \t \u@\h Tak, jaśnie Panie?'
```

Rysunek 2.26. Zmiana znaku zachęty

Znak zachęty powinien wyglądać tak jak na rysunku 2.27. Żeby powrócić do poprzedniego stanu należy ustawić wartość PS1 tak, jak na rysunku 2.28.

```
pią paź 07 13:00:43 user@host Tak, jaśnie Panie?
```

Rysunek 2.27. Zmieniony znak zachęty

```
PS1='\u@\h:\w'
```

Rysunek 2.28. Powrót do domyślnego wyglądu znaku zachęty

2.7. Instalowanie pakietów

Każdy użytkownik podczas pracy z systemem Linux prędzej czy później będzie musiał zainstalować jakieś narzędzie, pakiet, itp. Przypomnijmy, że skrypt był napisany w oparciu o dystrybucję Debian. Ma to znaczenie w przypadku wykorzystywanych narzędzi do instalowania i zarządzania pakietami oraz samego pakietu. Więcej na ten temat opowiemy w trakcie omawiania tych elementów.

W przypadku środowiska graficznego sprawa jest prosta, gdyż istnieją nakładki graficzne (np. Synaptic), które w dużej mierze ułatwiają zarządzanie pakietami. Dlatego zajmiemy się zarządzaniem pakietami z poziomu konsoli.

2.7.1. Przejście na konto roota

Zwykły użytkownik nie ma uprawnień do instalowania lub odinstalowywania pakietów. Żeby móc to robić, trzeba uzyskać prawa administratora, czyli użytkownika root. Można to zrobić w dwojaki sposób.

Pierwszy sposób to komenda *su*. Jest to polecenie, które bez wylogowywania aktualnego użytkownika zaloguje go na konto administratora. Komenda w celu potwierdzenia tożsamości prosi o podanie hasła użytkownika root.

Po zalogowaniu się na konto administratora pracujemy w odrębnej powłoce, więc żeby wrócić do powłoki użytkownika, na którego byliśmy zalogowani, musimy skorzystać z polecenia *exit*.

W niektórych dystrybucjach z rodziny Debiana polecenia *su* nie ma.

Innym sposobem jest skorzystanie z komendy *sudo*. Jest to komenda, która pozwala na wykonanie innych poleceń z prawami użytkownika root, ale bez konieczności logowania się na jego konto.

Aby użytkownik mógł korzystać z tego polecenia, musi być dodany do listy *sudoers*, która znajduje się w pliku */etc/sudoers*.

W celu skorzystania z tej komendy należy poprzedzić wywołanie innej komendy tym poleceniem. Przykładem może być próba wyświetlenia treści pliku */etc/sudoers*. Żeby to zrobić, trzeba mieć prawa root'a. Przykład ten został przedstawiony na rysunku 2.29.

```
user@host:~$ cat /etc/sudoers
cat: /etc/sudoers: Brak dostępu
user@host:~$ sudo cat /etc/sudoers
[sudo] password for user:
# /etc/sudoers
...
```

Rysunek 2.29. Wykorzystanie polecenia *sudo*

Polecenie *sudo* wymaga podania hasła. Jest to hasło aktualnie zalogowanego użytkownika, a nie root'a. Dodatkowo, użytkownik jest o nie pytany raz w danej powłoce, także jeśli drugi raz będziemy chcieli wyświetlić zawartość pliku */etc/sudoers/*, to system już nie będzie pytał o hasło.

Jeśli nie masz hasła do konta root, a Twój login nie znajduje się w pliku */etc/sudoers*, to musisz poprosić kogoś, kto ma prawa roota, żeby Twój login do tego pliku dopisał.

2.7.2. Pakiety

Pakiet to skompilowane źródło (kod), dzięki czemu jego instalacja jest prosta i szybka. Istnieje również możliwość kompilowania źródeł programów bezpośrednio na swoim komputerze, lecz trwa to dłużej i często pojawiają się błędy związane np. z brakiem lub niezgodnością bibliotek. Dlatego też dla użytkowników, dla których domyślna konfiguracja programów skompilowana w pakietach jest wystarczająca, pakiety są idealnym rozwiązaniem.

W zależności od dystrybucji rozróżnia się kilka rodzajów pakietów:

— **RPM** - występujące w rodzinie dystrybucji Red Hat;
— **DEB** - występujące w Debianie (np. popularne Ubuntu jest oparte na Debianie);
— **TGZ** - spakowane archiwum TAR, które jest wykorzystywane np. przez Slackware.

2.7.3. Niskopoziomowy *dpkg*

dpkg to niskopoziomowy menadżer pakietów, który jest obecny w naszej dystrybucji, czyli Debianie. Dzięki niemu możemy instalować pobrane pakiety. Odpowiednikiem *dpkg* dla pakietów RPM jest program *rpm* obecny w dystrybucjach z rodziny Red Hat.

Żeby zainstalować pakiet należy użyć flagi *-i* - rysunek 2.30.

```
dpkg -i pakiet.deb
```

Rysunek 2.30. Instalowanie pakietu DEB

W celu usunięcia pakietu należy użyć flagi *-r* - rysunek 2.31.

```
dpkg -r pakiet.deb
```

Rysunek 2.31. Usunięcie pakietu DEB

2.7.4. APT

W celu zainstalowania pakietu, w pprzypadku korzystania z dpkg, musimy najpierw znaleźc ten pakiet. Jest to dosyć uciążliwe, więc żeby temu zaradzić, stworzono APT. APT służy do automatycznego zarządzania pakietami. Jego wielką zaletą jest to, że potrafi sam odnaleźć interesujący nas pakiet, zainstalować go oraz pobrać
i zainstalować zależności wymagane przez ten pakiet.

Żeby korzystać z APT, należy użyć polecenia *apt-get*.

Na początek przyjrzyjmy się plikowi */etc/apt/sources.list*. Fragment jego przykładowej treści zaprezentowano na rysunku 2.32.

```
...
deb http://security.debian.org/ lenny/updates main
deb-src http://security.debian.org/ lenny/updates main

deb http://volatile.debian.org/debian-volatile lenny/volatile main
deb-src http://volatile.debian.org/debian-volatile lenny/volatile main
...
```

Rysunek 2.32. Fragment pliku */etc/sources.list*

Plik ten zawiera ścieżki do repozytoriów z pakietami. Repozytorium to centralny magazyn pakietów, w którym dostęp do wszystkich pakietów jest równie łatwy. W repozytorium przechowuje się bieżące dane, dokumenty, bez archiwum oraz kopii.

Istnieje wiele repozytoriów, które:

— mogą być oficjalne lub nie;
— mogą zawierać wersje stabilne lub beta;
— mogą być tworzone przez różne firmy, w celu przechowywania ich produktów (np. Mozilla);
— itp.

Użytkownik może edytować listę repozytoriów wedle uznania, aczkolwiek trzeba to robić ostrożnie. Są bowiem repozytoria, które przechowują najnowsze (często codziennie aktualizowane) wersje beta programów, które jeszcze nie są sprawdzone pod względem niezawodności lub bezpieczeństwa. Niektórzy świadomie instalują najnowsze wersje beta, aby sprawdzić najnowsze *wodotryski* w narzędziach.

Kiedy już skończyliśmy edytować plik ze źródłami repozytoriów, należy je odświeżyć, żeby system pobrał listę pakietów w nich udostępnionych. W tym celu korzystamy z akcji *update*, tak jak na rysunku 2.33. Po podaniu hasła, APT ściągnie najnowsze dane o pakietach, a po zakończeniu wyświetli informację: *Czytanie list pakietów... Gotowe.*

```
sudo apt-get update
```

Rysunek 2.33. Pobranie informacji o najnowszych pakietach

Instalacja

Po zaktualizowaniu listy pakietów możemy już je instalować. W tym celu należy skorzystać z akcji *install* oraz podać nazwę pakietu. Wcześniej wspomniałem o wygodnym narzędziu *yakuake*, które w bardzo prosty sposób zastępuje nam wiele okien terminala w środowisku graficznym (więcej na temat tego programu na stronie *http://yakuake.kde.org/*). Na rysunku 2.34 przedstawiono proces instalacji pakietu *yakuake*.

Po wpisaniu komendy *sudo apt-get install yakuake* system przedstawi nam listę pakietów, wymaganych do poprawnego działania narzędzia, które chcemy zainstalować (na rysunku lista została skrócona) oraz poinformuje o ilości instalowanych (lub aktualizowanych) pakietów, o rozmiarze danych do pobrania oraz rozmiarze wymaganego miejsca na dysku. Na końcu jesteśmy proszeni o potwierdzenie tych zmian, po czym następuje właściwa instalacja.

Wyszukiwanie

Oczywiście często bywa tak, że pakiet o podanej przez nas nazwie nie istnieje, a nie pamiętamy dokładnie jego nazwy. Wtedy z pomocą przychodzi inny program z rodziny apt - *apt-cache*. Żeby wyszukać pakiety w używanych przez nas repozytoriach, należy skorzystać z opcji *search*. Przykład jest przedstawiony na rysunku 2.35.

Aktualizacja pakietów

Większość programów jest rozwijana i publikowane są ich nowsze wersje. APT sam zatroszczy się o znalezienie nowszych wersji programów w repozytorium i ich zaktualizowanie. My musimy jedynie skorzystać z opcji *upgrade* programu *apt-get*.

Odinstalowywanie pakietów

Na koniec warto jeszcze poznać sposób na odinstalowanie pakietu.

```
user@host:~$ sudo apt-get install yakuake
Czytanie list pakietów... Gotowe
Budowanie drzewa zależności
Odczyt informacji o stanie... Gotowe
Zostaną zainstalowane następujące dodatkowe pakiety:
  aspell aspell-en dbus dbus-x11 esound-clients esound-common
  fam fontconfig hicolor-icon-theme kdelibs-data kdelibs4c2a
  konsole libakode2 libart-2.0-2 libarts1-akode libarts1c2a
  libartsc0  libasound2 libaspell15 libaudio2 libaudiofile0 ...
Sugerowane pakiety:
  aspell-doc spellutils perl-suid ghostscript khelpcenter
  libasound2-plugins nas esound jackd libjasper-runtime
  liblcms-utils libqt3-mt-psql libqt3-mt-mysql libqt3-mt-odbc
  libraw1394-doc  speex gksu kdebase-bin kdebase-runtime
  ktsuss sux
Zostaną zainstalowane następujące NOWE pakiety:
  aspell aspell-en dbus dbus-x11 esound-clients esound-common
  fam fontconfig hicolor-icon-theme kdelibs-data kdelibs4c2a
  konsole libakode2 libart-2.0-2 libarts1-akode libarts1c2a
  libartsc0 libasound2 libaspell15 libaudio2 ... yakuake
0 aktualizowanych, 74 nowo instalowanych, 0 usuwanych
i 65 nieaktualizowanych.
Konieczne pobranie 32,9MB archiwów.
Po tej operacji zostanie dodatkowo użyte 100MB miejsca na dysku.
Kontynuować [T/n]?
```

Rysunek 2.34. Instalacja pakietu *yakuake*

```
user@host:~$ apt-cache search yaku
libnet-dns-fingerprint-perl - library to determine DNS
 server vendor, product and version
yakuake - a Quake-style terminal emulator based on KDE
 Konsole technology
```

Rysunek 2.35. Wyszykiwanie pakietów

Do tego służy opcja *remove* programu *apt-get*. Wcześniej jednak warto sprawdzić, jakie pakiety są zainstalowane w systemie. W tej sytuacji należy skorzystać z wcześniej omówionego *dpkg* i flagi *-l*. Przykład został przedstawiony na rysunku 2.37. Przeszukujemy w nim listę zainstalowanych pakietów i wybieramy te, których wiersze zawierają ciąg znaków *tar*. Następnie usuwamy wybrany pakiet.

Program *grep* służy do przeszukiwania tekstu. Poznamy go lepiej w kolejnych rozdziałach.

```
user@host:~$ sudo apt-get upgrade
[sudo] password for user:
Czytanie list pakietów... Gotowe
Budowanie drzewa zależności
Odczyt informacji o stanie... Gotowe
Następujące pakiety zostały zatrzymane:
  bind9-host dnsutils libbind9-50 libisccc50 libisccfg50 liblwres50
Następujące pakiety zostaną zaktualizowane:
  apache2-mpm-prefork apache2-utils apache2.2-common aptitude
  base-files dhcp3-client dhcp3-common dpkg exim4 exim4-base
  exim4-config exim4-daemon-light klibc-utils libapache2-mod-php5
  libapr1 libc6 libc6-dev libc6-i686 libcups2 libcurl3 libfreetype6
  libldap-2.4-2 libmysqlclient15off ... mysql-server-5.0
59 aktualizowanych, 0 nowo instalowanych, 0 usuwanych
 i 6 nieaktualizowanych.
Konieczne pobranie 124MB archiwów.
Po tej operacji zostanie dodatkowo użyte 1294kB miejsca na dysku.
Kontynuować [T/n]?
```

Rysunek 2.36. Aktualizacja pakietów

```
user@localhost:~$ dpkg -l | grep tar
ii  tar   1.20-1+lenny1    GNU version of the tar archiving utility
user@CSS:~$ sudo apt-get remove tar
[sudo] password for user:
Czytanie list pakietów... Gotowe
Budowanie drzewa zależności
Odczyt informacji o stanie... Gotowe
Następujące pakiety zostaną
USUNIĘTE:
  tar
UWAGA: Zostaną usunięte następujące istotne pakiety.
Nie powinno się tego robić, chyba że dokładnie wiesz co robisz!
  tar
0 aktualizowanych, 0 nowo instalowanych, 1 usuwanych
 i 65 nieaktualizowanych.
Po tej operacji zostanie zwolnione 2302kB miejsca na dysku.
Zaraz zrobisz coś potencjalnie szkodliwego.
Aby kontynuować wpisz zdanie "Tak, rób jak mówię!"
```

Rysunek 2.37. Usuwanie pakietów

2.8. Strumienie

Strumienie to kanały komunikacji między uruchomionymi programami a otoczeniem, np. klawiaturą, czy ekranem. W systemie Linux istnieją 3 standardowe strumienie danych:
— standardowy strumień wejścia (**stdin**);
— standardowy strumień wyjścia (**stdout**);
— standardowy strumień błędów (**stderr**).

Każdy strumień ma swój liczbowy identyfikator zwany deskryptorem:
— **stdin** - deskryptor 0;
— **stdout** - deskryptor 1;
— **stderr** - deskryptor 2.

Każdy uruchomiony proces ma zdefiniowane wszystkie 3 strumienie. Domyślnie strumieniem wejścia jest klawiatura, zaś wyjścia i błędów - ekran.

Główna własność strumieni to możliwość ich przekierowywania. Dzięki temu, dane mogą być wczytywane z różnych źródeł oraz wysyłane do różnych urządzeń, plików, itp.

2.8.1. Strumienie i komenda cat

Zastosowanie strumieni omówimy na przykładzie wcześniej wspomnianej komendy **cat**.

Jeśli uruchomimy tę komendę bez parametru, to system czeka aż wpiszemy dane, gdyż strumieniem wejścia jest klawiatura, a następnie wyświetli to co wpisaliśmy na standardowe wyjście, czyli powieli tekst na ekranie.

```
user@user-desktop:~$ cat
wpisany tekst
wpisany tekst
```

Rysunek 2.38. Komenda **cat** bez parametru

2.8.1.1. Strumień wejścia

Żeby komenda czytała dane z pliku, trzeba go przekazać jako parametr. Innym sposobem jest przekierowanie strumienia wejścia.

Strumień wejścia przekierowujemy za pomocą operatora <. Jak widać na rysunku 2.39, komenda **cat** zachowuje się tak samo po ustawieniu pliku jako strumień wejścia, jak po przekazaniu pliku jako parametr.

2.8.1.2. Strumień wyjścia

Strumień wyjścia można przekierować na dwa sposoby.

Pierwszy sposób polega na wyczyszczeniu pliku, do którego strumień jest przekierowany, a następnie zapisanie wyniku. W tym celu wykorzystuje się operator >. Przykład przedstawiono na rysunku 2.40. Na początku plik wynikowy zawiera

```
user@user-desktop:~$ cat plik.txt
To jest przykladowy plik
który ma wiele linii
i bedzie wyswietlany na ekranie
user@user-desktop:~$ cat < plik.txt
To jest przykladowy plik
który ma wiele linii
i bedzie wyswietlany na ekranie
user@user-desktop:~$
```

Rysunek 2.39. Przekierowanie strumienia wejścia

```
user@user-desktop:~$ cat wynik.txt
Dane poczatkowe w pliku
user@user-desktop:~$ cat plik.txt > wynik.txt
user@user-desktop:~$ cat wynik.txt
To jest przykladowy plik
który ma wiele linii
i bedzie wyswietlany na ekranie
user@user-desktop:~$
```

Rysunek 2.40. Przekierowanie strumienia wejścia - nadpisanie pliku

dane, które są wyświetlone. Po wykonaniu komendy wynik został przekierowany do pliku, który uprzednio został wyczyszczony.

Drugi sposób polega na dopisaniu danych do pliku za pomocą operatora >>. Przykład przedstawiono na rysunku 2.41.

```
user@user-desktop:~$ cat wynik.txt
Dane poczatkowe w pliku
user@user-desktop:~$ cat plik.txt >> wynik.txt
user@user-desktop:~$ cat wynik.txt
Dane poczatkowe w pliku
To jest przykladowy plik
który ma wiele linii
i bedzie wyswietlany na ekranie
user@user-desktop:~$
```

Rysunek 2.41. Przekierowanie strumienia wyjścia - dopisanie do pliku

2.8.1.3. Strumień błędu

Strumień błędy służy do zwracania informacji o błędach. Ponieważ dane wynikowe oraz błędy mają całkiem inny charakter, stworzono dla nich 2 niezależne strumienie mimo, że domyślnie są one zdefiniowane tak samo - ekran.

Do przekierowywania strumienia błędu wykorzystuje się operator 2 >. Cyfra 2 jest deskryptorem strumienia, dlatego też wcześniej omówiony operator > jest równoważny operatorowi 1 >.

Po co przekierowywać strumień błędów?

Jak wiadomo oba strumienie wejścia i błędu przekierowywane są domyślnie na ekran. Ponadto często na wyjście błędu wysyłane są ostrzeżenia, które nie blokują wykonania programu, ale *zaśmiecają* ekran. Chcąc się ich pozbyć, przekierowujemy standardowe wyjście błędu, żeby na ekranie został tylko wynik. Oczywiście, można również przekierować strumień wyjścia do pliku, jeśli chcemy zachować wynik. Wtedy na ekranie pojawią się tylko błędy.

Nasuwa się kolejne pytanie: **Gdzie przekierowywać nieinteresujące nas treści błędów?**

Można przekierować je do jakiegoś tymczasowego pliku, a później go usunąć. Jest jednak wygodniejsze wyjście. Tak jak wcześniej wspomniano, strumienie można przekierowywać na urządzenia. W systemie Linux istnieje urządzenie **/dev/null** zwane *czarną dziurą*. Wszelkie dane przekierowane na to urządzenie zostaną zignorowane i utracone.

2.8.1.4. Przykłady wykorzystania

Wykorzystując komendy **cat**, **echo** oraz strumienie, można wykonać kilka przydatnych operacji.

1. **Stworzyć lub wyczyścić plik**
   Komenda nadpisuje plik pustym łańcuchem znaków.
   ```
   user@user-desktop:~$ echo "" > plik.txt
   ```
2. **Skopiować plik**
   Komenda równoważna z **cp plik1.txt plik2.txt**.
   ```
   user@user-desktop:~$ cat plik1.txt > plik2.txt
   ```
3. **Scalić pliku**
   Komenda tworzy nowy plik i zapisuje w nim treść trzech plików z parametrów.
   ```
   user@user-desktop:~$ cat plik1.txt plik2.txt plik3.txt >
    plik_scalony.txt
   ```

2.9. Potoki

Potoki podobnie jak strumienie służą do komunikacji, jednakże w przeciwieństwie do nich, stronami komunikującymi się są procesy. Idea potoku polega na tym, że wynik jednej operacji staje się danymi wejściowymi następnej.

W linuksowych powłokach systemowych używa się symbolu | (pionowej linii), aby połączyć dwa lub więcej procesów w potok. Potok może mieć dowolną dłu-

gość, tzn. że można dane można *przepuścić* przez wiele operacji łącząc je w potok. Przykład użycia został zaprezentowany na rysunku 2.42.

```
ps -a | sort | uniq | grep -v sh
```

Rysunek 2.42. Przykład zastosowania potoków

Wykorzystano tutaj komendę **ps -a**, która wyświetla listę uruchomionych procesów, która następnie została posortowana. Później komenda **uniq** usunęła powtarzające się linie, a **grep** wyszukał linie, które nie zawierają w sobie łańcucha znaków *sh*.

Flaga **-v** w komendzie **grep** powoduje, że linie, które spełniają wyrażenie, są odrzucane.

2.9.1. Potoki i wyświetlenie dowolnego fragmentu pliku

Wykorzystując potoki oraz komendy **head** i **tail**, łatwo można stworzyć komendę, która wyświetli dowolne linie z pliku.

Przykładowo, jeśli chcemy wyświetlić linie o numerach 101-105, najpierw należy wyświetlić pierwsze 105 linii, a następnie 5 ostatnich z wcześniej wybranych.

```
head -105 plik.txt | tail -5
```

Rysunek 2.43. Wyświetlenie linii o numerach 101-105 z pliku plik.txt

Rozdział 3

Przetwarzanie tekstu

Przetwarzanie tekstu to jedna z najczęściej wykonywanych czynności zarówno przed administratorów systemu, jak i przez zwykłych użytkowników. Jako zwykły użytkownik systemu jesteśmy przyzwyczajeni do edytorów graficznych, w które wyposażone są środowiska graficzne systemu (czyli konsola nr 7). W środowisku graficznym proste przetwarzanie tekstu jest bardzo wygodne. Dzięki myszce, którą w szybki sposób możemy zaznaczać tekst oraz podstawowym skrótom klawiszowym typu CTRL + C, CTRL + V czy CTRL + X edycja tekstu jest prosta i bardzo szybka.

Są jednak przypadki, gdy przetwarzanie tekstu w edytorach graficznych jest utrudnione lub niemożliwe. Dzieje się tak na przykład w przypadku większych ilości danych (np. plików o dużych rozmiarach). Praca z tekstem w edytorze graficznym może być niemożliwa, gdyż zwyczajnie nie będzie on miał wystarczających zasobów do przetwarzania całego pliku jednocześnie. Innym przykładem jest skomplikowane przetwarzanie tekstu. Uciążliwe jest przetwarzanie wyeksportowanego z bazy danych pliku CSV (ang. Comma Separated Values) w zwykłym edytorze tekstu. Ponadto jeśli przetwarzanie jest skomplikowanie, to i w arkuszach kalkulacyjnych jest to żmudna i uciążliwa praca.

Wtedy z pomocą przychodzą edytory strumieni tekstu (SED), czy nawet języki programowania służące do przetwarzania tekstu. Ich zadanie polega na ułatwieniu przetwarzania tekstu. Dużego znaczenia dla użytkownika nabierają wtedy, gdy wykorzystane zostaną do wytworzenia skryptu, który będzie przeprowadzał skomplikowane operacje na strumieniu danych. Warto przeznaczyć czas na napisanie skryptu, który wykonuje nawet proste operacje. Wielu użytkowników wychodzi z założenia, że nie ma sensu uczyć się edytorów strumieni danych oraz marnować czasu na pisanie skryptu, gdyż operacje, które chcą wykonać są proste. Dopiero później, kiedy okazuje się, że te proste operacje są wykonywane często, napisanie skryptu zaoszczędziłoby czas użytkownikowi za każdym razem, kiedy skrypt jest wykorzystywany.

W niniejszym rozdziale omówione zostaną podstawy przetwarzania tekstu oraz wyżej wymienione narzędzia wspomagające zarówno proste jak i zaawansowane przetwarzanie tekstu.

Poznamy również podstawowe operacje na plikach tekstowych, a także polecenia, które służą do wyszukiwania plików.

3.1. Podstawowe operacje na plikach tekstowych

Na początku musimy poznać podstawowe metody przetwarzania tekstu, tj. przeglądanie treści plików lub jej fragmentów, czy na przykład przeszukiwanie plików. Wspomnimy również o strumieniach które mogą być traktowane jako źródło danych (zamiast pliku) oraz potokach, które będą przekazywały wyniki jednej komendy do drugiej komendy.

3.1.1. Przeglądanie pliku

Najprostszym sposobem na podejrzenie pliku jest użycie komendy **cat**. Przyjmuje ona jako parametr jeden lub więcej plików (a dokładnie ścieżek do nich)

oddzielonych spacjami. Wynikiem metody jest wyświetlenie treści tych plików na standardowym wyjściu, czyli na ekranie konsoli.

```
cat plik1.txt plik2.txt
```

Rysunek 3.1. Wyświetlenie treści dwóch plików

W przypadku wyświetlania kodów źródłowych przydatna jest flaga **-n**, która numeruje linie.

Komenda **cat** w połączeniu z przekierowywaniem strumieni uzyskuje nowe możliwości, o których bliżej za chwilę, podczas omawiania strumieni.

3.1.2. Przeglądanie fragmentów pliku

Komenda **cat** staje się bezużyteczna w przypadku przeglądania dużych plików, czy innych ilości danych np. wyników wywołania komend. Prostym przykładem jest przejrzenie listy zainstalowanych pakietów w systemie - **dpkg -l**. Lista ta jest bardzo długa i zwykle nie mieści się w oknie konsoli, nawet uwzględniając jej przewijanie w górę.

W takich przypadkach pomocne stają się komendy, które umożliwiają przeglądanie fragmentów dużych ilości danych. Należą do nich:

— **head** oraz **tail**, które wyświetlają fragment pliku,

— **more** oraz **less**, które umożliwiają przeglądanie całego pliku fragmentami.

Różnica między komendą **head** a **tail** polega na tym, że **head** wyświetla dowolną (domyślnie 10) ilość linii z początku pliku, podczas gdy **tail** wyświetla linie z końca pliku. Jeśli chcemy wyświetlić inną ilość linii niż domyślna, to musimy jako opcję przekazać liczbę, np. **head -100 plik.txt**.

Na rysunku 3.2 wykorzystano potoki, które omówimy za chwilę.

Mówiliśmy wcześniej o wyświetleniu fragmentu pliku. Oczywiście fragment pliku może pochodzić z dowolnej jego części; niekoniecznie z początku lub końca. Jak zatem wyświetlić fragment ze środka pliku? Oczywiście łącząc obie komendy **head** i **tail**. O tym jak to zrobić powiemy przy okazji omawiania potoków.

Druga para komend to **more** i **less**. W przeciwieństwie do poprzedniej pary komend, nie wyświetlają one jednego fragmentu danych, lecz umożliwiają ich przejrzenie fragmentarycznie. Można to porównać do przewracania stron w książce.

Komenda **more** umożliwia jedynie przewijanie tekstu w dół bez możliwości powrotu.

Natomiast komenda **less** jest jej ulepszoną wersją, gdyż umożliwia przewijanie tekstu w gorę i w dół oraz na prawo i lewo, jeśli tekst zawiera długie linie.

Podczas przeglądania danych za pomocą obu powyższych komend, wciśnięcie **spacji** spowoduje przejście w dół o jedną stroną, tzn. że na samej górze będzie linia, która wcześniej była na samym dole.

Ponadto, żeby zakończyć przeglądanie, należy wcisnąć klawisz **q** (ang. quit).

```
user@user-desktop:~$ dpkg -l | head
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf...
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                                  Version
+++-=======================================-
ii  accountsservice                        0.6.14-1git1
ii  acl                                    2.2.51-3
ii  acpi-support                           0.138
ii  acpid                                  1:2.0.10-1ubuntu2
ii  adduser                                3.112+nmu1ubuntu5
user@user-desktop:~$ dpkg -l | tail
ii  yelp-xsl                               3.2.0-0ubuntu1
ii  zeitgeist                              0.8.2-1
ii  zeitgeist-core                         0.8.2-1
ii  zeitgeist-datahub                      0.7.0-0ubuntu4
ii  zeitgeist-extension-fts                0.0.13-0ubuntu1
ii  zenity                                 3.2.0-0ubuntu1
ii  zenity-common                          3.2.0-0ubuntu1
ii  zip                                    3.0-4
ii  zlib1g                                 1:1.2.3.4.dfsg-3ubuntu3
ii  zlib1g-dev                             1:1.2.3.4.dfsg-3ubuntu3
user@user-desktop:~$
```

Rysunek 3.2. Wyświetlenie dziesięciu pierwszych oraz dziesięciu ostatnich linii z listy zainstalowanych pakietów

3.2. Wyszukiwanie plików

Konfiguracja systemu Linux często wymaga odnalezienia plików konfiguracyjnych, które zamierzamy edytować, gdyż nie dla wszystkich pakietów istnieją komendy ułatwiające ich konfigurację.

Wiele plików ma swoje stałe miejsce w systemie (np. plik z listą użytkowników */etc/passwd*), aczkolwiek są również pliki konfiguracyjne, które mogą znajdować się w kilku miejscach w strukturze katalogów. Jest to uzależnione od pakietów, których dotyczą, jak również od dystrybucji systemu.

Na przykład interpreter PHP, który zwykle instaluje się razem z serwerem WWW, może przechowywać swoje pliki konfiguracyjne w folderach */etc/php5/...*, */usr/local/php5/...*, itd.

3.2.1. Shell pattern (tzw. maska)

Polecenia do wyszukiwania plików, które zostaną omówione w tym rozdziale, przyjmują jako argument wzorce (tzw. maski), na podstawie których odnajdują pliki, np. o podanej nazwie.

?	Dopasowuje dowolny znak, który może wystąpić dokładnie jeden raz.
*	Dopasowuje dowolne znaki, które mogą wystąpić dowolną ilość razy (również zero).
[*abcd*]	Dopasowuje jeden znak z podanego zbioru. Znak ña początku neguje zbiór, czyli dopasowuje wszystkie znaki, za wyjątkiem podanych.
\	Usuwa specjalną funkcję metaznaku, przed którym stoi.

Tablica 3.1. Tabela wieloznaczników

Maski te mogą być wzbogacone o pewne metaznaki (tzw. wieloznaczniki), które jednak różnią się od tych, przedstawionych podczas omawiania wzorców regularnych. Maski są prostsze niż wyrażenia regularne, a do ich konstrukcji wykorzystuje się jedynie 4 wieloznaczniki, wymienione w tabeli 3.1.

3.2.2. Proste wyszukiwanie plików

W systemie Linux istnieją komendy, które przeszukują zachowaną wcześniej strukturę katalogów i plików. Plusem tego rozwiązania jest szybkość przeszukiwania takiej bazy. Minusem jest potrzeba aktualizowania takiej bazy po większych zmianach w strukturze plików.

W przypadku plików konfiguracyjnych należy aktualizować bazę po zainstalowaniu nowych pakietów, żeby pliki z nimi powiązane zostały one dodane do bazy.

3.2.2.1. locate

Najprostszym sposobem wyszukiwania plików jest skorzystanie z polecenia **locate** lub jego nowszej wersji **mlocate**.

```
locate nazwa-pliku
```

Rysunek 3.3. Składnia komendy locate

Polecenie przyjmuję maskę nazwy szukanego pliku i wyświetla listę odnalezionych plików wraz z ich bezwzględnymi ścieżkami.

Jeśli polecenie nie odnalazło ani jednego pliki, a wydaje Ci się, że plik o tej nazwie istnieje w systemie, spróbuj zaktualizować bazę i wywołać polecenie jeszcze raz.

W celu zaktualizowania bazy plików, z której korzystają polecenia **locate** i **mlocate**, należy skorzystać z polecenia **updatedb**, które należy uruchomić z prawami administratora.

Jeśli podczas próby aktualizacji napotkasz błąd o treści *updatedb: can not open a temporary file for '/var/lib/mlocate/mlocate.db'*, to znaczy, że próbujesz zaktualizować bazę nie posiadając uprawnień administratora.

```
user@host:~# locate php.ini
/etc/php5/apache2/php.ini
/etc/php5/cli/php.ini
/usr/share/doc/php5-common/examples/php.ini-development
/usr/share/php5/php.ini-production
/usr/share/php5/php.ini-production-dist
/usr/share/php5/php.ini-production.cli
```

Rysunek 3.4. Wynik komendy locate dla pliku php.ini

Do wyszukiwania plików binarnych, źródłowych oraz plików z manuala można użyć również polecenia **whereis**, którego składnia jest taka sama, jak polecenia **locate**.

3.2.3. Zaawansowane wyszukiwanie plików - find

W poprzednim podrozdziale omówiliśmy polecenia, które w prosty i szybki sposób znajdują pliki o podanej nazwie. Jeśli chcemy wyszukać pliki uwzględniając inne kryteria, z pomocą przychodzi polecenie **find**. Jest to zaawansowana wyszukiwarka plików, w której możemy uwzględnić wiele różnych kryteriów wyszukiwania, które dodatkowo można łączyć w logiczne warunki.

Omówimy podstawowe kryteria wyszukiwania plików za pomocą polecenia **find**. Dokładny opis polecenia oraz jego opcji znajduje się w manualu, który można podejrzeć za pomocą komendy **man find**.

```
find [scieżka] [wyrażenie wyszukiwania]
```

Rysunek 3.5. Składnia komendy find

Składnia polecenia została przedstawiona na 3.5.

Pierwszy argument - ścieżka - określa folder, który chcemy przeszukać. Wyszukiwanie odbywa się w wybranym folderze oraz w jego podfolderach rekursywnie. Ścieżka jest argumentem opcjonalnym i jeśli nie zostanie podana, to program przeszukuje folder, w którym jest uruchomiony (wynik polecenia **pwd**).

Drugim argumentem jest wyrażenie wyszukiwania, składające się z kryteriów wyszukiwania, które mogą być połączone logicznymi operatorami, a także z akcji, jakie należy podjąć po znalezieniu plików (np. -print).

Listę głównych kryteriów przedstawiono w tabeli 3.2. Wiele z nich przyjmuje wartości liczbowe, np. rozmiar pliku. W przypadku tych kryteriów można określić, czy wyszukiwany plik ma mieć wartość mniejszą, większą lub równą podanej.
W tym celu należy skorzystać z następującej składni:
— **+n** (np. -size +100), gdy szukamy plików o wartości większej niż podana,
— **-n** (np. -size -100), gdy szukamy plików o wartości mniejszej niż podana,
— **n** (np. -size 100), gdy szukamy plików o wartości takiej samej jak podana.

-name *'maska'*	Nazwa pasuje do podanej maski.
-iname *'maska'*	Działa tak jak **name**, ale w trybie *case-insensitive*, czyli nie zwraca uwagi na wielkość liter.
-path *'maska'*	Ścieżka do pliku pasuje do podanej maski.
-ipath *'maska'*	Działa tak jak **path**, ale w trybie *case-insensitive*.
-amin *n*	Ostatni dostęp do pliku był **n** minut temu.
-anewer *plik*	Ostatni dostęp do pliku nastąpił po ostatniej modyfikacji pliku **plik**.
-atime *n*	Ostatni dostęp do pliku był **n*24** godzin temu.
-cmin *n*	Status pliku został zmieniony **n** minut temu.
-cnewer *plik*	Ostatnia zmiana statusu pliku nastąpiła po ostatniej modyfikacji pliku **plik**.
-ctime *n*	Ostatnia zmiana statusu pliku była **n*24** godzin temu.
-empty	Plik jest plikiem zwykłym lub folderem oraz jest pusty.
-false	Zawsze zwraca fałsz.
-true	Zawsze zwraca prawdę.
-gid *n*	Pliki, których identyfikator grupy jest równy **n**.
-gname *name*	Pliki, którego grupa nazywa się **name**.
-uid *n*	Pliki, których identyfikator użytkownika jest równy **n**.
-uname *name*	Pliki, którego użytkownik nazywa się **name**.
-links *n*	Plik ma **n** linków.
-lname *'maska'*	Pliki będące linkiem symbolicznym do pliku, którego nazwa pasuje do maski.
-ilname *'maska'*	Działa tak, jak **-lname**, ale w trybie *case-insensitive*.
-mmin *n*	Plik był ostatnio zmieniony **n** minut temu.
-mtime *n*	Plik był ostatnio zmieniony **n*24** godzin temu.
-newer *plik*	Ostatnia modyfikacja pliku nastąpiła po ostatniej modyfikacji pliku **plik**.
-perm *prawa*	Pliki, które mają prawa dostępu równe **prawa**. Prawa mogą być zapisane w systemie ósemkowym (np. 644) lub symbolicznym (np. -u+w).
-regexp *wzorzec*	Pliki, których ścieżka może być dopasowana do wzorca regularnego.

Tablica 3.2. Tabela głównych kryteriów wyszukiwania dla polecenia find

-size *n[bckw]*	Pliki, które zajmują **n** jednostek pamięci. Jednostkami mogą być 512-bajtowe **b**loki (domyślnie), bajty (**c**), **k**ilobajty lub dwubajtowe sło**w**a.
-file *typ*	Pliki, które są typu **typ**. Typem może być: — **d** - folder, — **f** - zwykły plik, — **l** - link symboliczny, — **s** - gniazdo.

Tablica 3.3. Tabela głównych kryteriów wyszukiwania dla polecenia find - kontynuacja

(wyrażenie)	Zmiana priorytetu sprawdzenia warunku (wymuszenie pierwszeństwa).
! wyrażenie	Negacja wyrażenia.
-not wyrażenie	Działa tak samo jak *! wyrażenie*.
wyrażenie1 wyrażenie2	Koniunkcja (warunek AND) dwóch wyrażeń. Jeśli pierwsze wyrażenie jest fałszywe, drugie nie jest sprawdzane.
wyrażenie1 -a wyrażenie2	Działa tak samo jak *wyrażenie1 wyrażenie2*.
wyrażenie1 -and wyrażenie2	Działa tak samo jak *wyrażenie1 wyrażenie2*.
wyrażenie1 -o wyrażenie2	Alternatywa (warunek OR) dwóch wyrażeń. Jeśli pierwsze wrażenie jest prawdziwe, drugie nie jest sprawdzane.
wyrażenie1 -or wyrażenie2	Działa tak samo jak *wyrażenie1 -o wyrażenie2*.
wyrażenie1, wyrażenie2	Lista wyrażeń. Wszystkie wyrażenia z listy są sprawdzane, przy czym jako wynik przyjmowany jest wynik ostatniego wyrażenia.

Tablica 3.4. Tabela wyrażeń

Kryteria można łączyć logicznymi warunkami. Do tego służą wyrażenia przedstawione w tabeli 3.4.

Polecenie **find** daje możliwość wykonania pewnych akcji ze znalezionymi plikami. Lista podstawowych akcji została przedstawiona w tabeli 3.6. Ich składnia jest analogiczna do składni kryteriów. Ponadto akcje również mogą przyjmować parametry.

Przykłady

Przykład 1. *Wyszukaj wszystkie pliki w systemie, których rozmiar mieści się w przedziale 2-3GB i wyświetl ich listę wraz z danymi prezentowanymi za pomocą polecenia* ***ls****.*

Rozwiązanie:

```
find / -size +2GB -and -size -3GB -exec ls -la {} \;
```

-exec *komenda ;*	Wywołuje *komendę*, do której przekazywane są wszystkie następujące po niej argumenty, dopóki nie wystąpi znak *';'*. W liście argumentów można użyć znaków , które zostaną zastąpione ścieżkami do znalezionych plików podczas wykonywania komendy. Czasami, wymienione znaki specjalne trzeba poprzedzić znakiem \ lub zawrzeć komendę w cudzysłowach, żeby nie została zinterpretowana przez powłokę jako polecenie wyższego rzędu (równoważnego z poleceniem **find**).
-ok *komenda ;*	Działa tak samo, jak *-exec*, przy czym prosi o potwierdzenie przez użytkownika poprzez standardowe wejście. Jeśli odpowiedź użytkownika jest różna od **y** i **Y**, to komenda nie jest wykonywana.
-ls	Wyświetla listę znalezionych plików na standardowe wyjście, tak samo jak polecenie **ls**.
-fls *plik*	Działa tak samo jak *-ls*, przy czym listę zapisuje do pliku *plik*.
-print	Wyświetla listę ścieżek do znalezionych plików.
-fprint *plik*	Zapisuję listę znalezionych plików do pliku *plik*.
-print0	Wyświetla listę ścieżek do znalezionych plików, przy czym każdy plik oddziela znakiem *NULL*, a nie znakiem nowej linii. Dzięki temu programy, które przyjmują na wejściu listę plików z polecenia **find** będą mogły poprawnie rozdzielić pliki, które w nazwie zawierają znaki nowej linii.
-fprint0 *plik*	Działa tak jak *-print0*, przy czym listę zapisuje do pliku *plik*.

Tablica 3.5. Tabela głównych akcji wykonywanych na znalezionych plikach

-printf *format*	Wyświetla listę znalezionych plików według formatu, który może zawierać znaki specjalne poprzedzone \ oraz dyrektywy **%**. W formacie można wykorzystać następujące przykładowe znaki specjalne: — **a** - dzwonek systemowy, — **b** - znak *backspace*, — **c** - anuluje dalsze wyświetlanie listy według tego formatu, — **n** - znak nowej linii, — **t** - znak tabulacji, — **v** - znak tabulacji pionowej, — \\ - znak *backslash* '\', — **XXX** - znak, którego ósemkowa reprezentacja to *XXX*. Najczęściej używane dyrektywy, to: — **%%** - znak procent *%*, — **%d** - poziom zagnieżdżenia pliku względem folderu, w którym polecenie zostało uruchomione, — **%f** - nazwa pliku, — **%h** - ścieżka do pliku bez jego nazwy, — **%k** - rozmiar pliku w kilobajtach, — **%m** - prawa dostępu (w systemie ósemkowym), — **%p** - ścieżka do pliku, — **%s** - rozmiar pliku w bajtach, — **%t** - data ostatniej modyfikacji pliku, — **%u** - nazwa użytkownika, który jest właścicielem pliku.

Tablica 3.6. Tabela znaków specjalnych oraz dyrektyw akcji *printf*

Przedział uzyskujemy za pomocą koniunkcji *and*.

Przykład 2. *Wyszukaj wszystkie pliki w systemie, które mają rozszerzenie* ***log*** *i skopiuj je specjalnie utworzonego folderu* ***logs*** *w Twoim folderze domowym.*

Rozwiązanie:

```
find / -type f -iname "*.log" -exec cp {} $HOME/logs/ \;
```

Najpierw sprawdzamy, czy plik jest zwykłym plikiem, by następnie sprawdzić jego rozszerzenie bez względu na wielkość znaków. Jako komendy używamy funkcji kopiującej. Musimy podawać ścieżki bezwzględne, żeby móc skorzystać z tego polecenia w każdym miejscu w systemie.

Przykład 3. *Przypisz do użytkownika* ***user*** *wszystkie pliki, które nie mają przydzielonego użytkownika.*

Rozwiązanie:

```
find . -nouser -exec chown user {} \;
```

Przykład 4. *Przypisz wszystkie foldery użytkownika* ***user*** *użytkownikowi* ***pawel****.*

Rozwiązanie:

```
find / -type d -user user -exec chown pawel {} \;
```

Przykład 5. *Usuń pliki systemu kontroli wersji CVS z aktualnego folderu i jego podfolderów.*

Rozwiązanie:

```
find . -type d -name CVS -exec rm -R {} \;
```

System CVS wszystkie swoje dane przechowuje w folderach o nazwie CVS we wszystkich folderach projektu. Stąd szukamy tylko folderów i o nazwie CVS, gdzie wielkość liter ma znaczenie. Skoro są to foldery, to podczas ich usuwania musimy dodać flagę **-R**, żeby włączyć usuwanie rekursywne.

Przykład 6. *Wyświetl listę plików modyfikowanych po dacie 11 stycznia 2012 roku. Niech będzie wyświetlona w formacie data modyfikacji - nazwa pliku (bez całej ścieżki).*

Rozwiązanie:
W celu wyszukania plików modyfikowanych przed lub po konkretnej dacie można skorzystać z warunku **-newer** lub jego negacji. Jako parametr przyjmuje on ścieżkę do pliku. Trik polega na tym, żeby stworzyć plik z konkretną datą modyfikacji. Robi się to za pomocą polecenia **touch** z wykorzystaniem flagi **-d**.

```
touch -d "11 January 2012 00:00:00" znak_czasu
find . -newer znak_czasu -printf "%t - %f\n"
rm znak_czasu
```

Po wyświetleniu znalezionych plików należy po sobie posprzątać, czyli usunąć plik **znak_czasu**.

Przykład 7. *Usuń wszystkie puste pliki z aktualnego folderu i jego podfolderów, zagłębiając się maksymalnie do 3 poziomów niżej w strukturze drzewa folderów.*

Rozwiązanie:
Innymi słowy chodzi o usunięcie pustych plików z aktualnego folderu oraz z jego pod-, podpod- oraz podpodpodfolderów, ale plików bardziej zagłębionych już nie brać pod uwagę.

```
find . -empty -maxdepth 4 -exec rm {} \;
```

Poziom nr 1 odpowiada aktualnemu folderowi, dlatego maksymalne zagłębienie to 4, a nie 3.

Zadania

Zadanie 1. *Odszukaj w swoim katalogu domowym wszystkie pliki zwykłe, mające w nazwie ciąg znaków **log**, wyświetl ich nazwy i zawartość.*
*Wskazówka: użyj **printf** oraz **exec**.*

Zadanie 2. *Wyświetl nazwy wszystkich plików (i katalogów) w systemie, które są Twoją własnością.*

Zadanie 3. *Odszukaj w katalogu **/var/log** wszystkie pliki, mniejsze niż 7000 bajtów.*

Zadanie 4. *Znajdź w swoim katalogu domowym wszystkie pliki, które nie były używane w ciągu ostatnich 7 dni.*

Zadanie 5. *Wyświetl opis plików (nazwa, prawa, atrybuty, itp.) plików w katalogu **/etc**, które są większe niż 10 bajtów i mają więcej niż 1 dowiązanie (link).*

Zadanie 6. *Znajdź w swoim katalogu domowym (bez podkatalogów) wszystkie pliki zwykłe, mniejsze niż 100 bajtów, które były modyfikowane w ciągu ostatnich 7 dni i wyświetl 5 pierwszych linii każdego z nich.*

Zadanie 7. *Usuń z systemu wszystkie pliki o rozszerzeniu **out**, które nie były używane przez ostatni miesi. Niech polecenie pyta użytkownika o potwierdzenie usunięcia pliku.*

3.3. Wyrażenia regularne

Wyrażenia regularne to ciągi znaków zwane wzorcami, które określają zbiór pewnych łańcuchów znaków. Są one bardzo powszechnie wykorzystywane w informatyce, przede wszystkim do przeszukiwania tekstu. Dzięki wyrażeniom regularnym możemy określić zbiór pewnych łańcuchów, które chcemy odnaleźć w tekście bez konieczności wyszukiwania każdego z nich osobno.

W celu zobrazowania potęgi wyrażeń regularnych spójrzmy na przykład, w którym chcemy zamienić w pewnym pliku wszystkie wystąpienia adresów IP (v4) ciągiem *xxx.xxx.xxx.xxx*, czyli po prostu chcemy je ukryć.

Jak wiadomo adres IP (v4) składa się z czterech liczb połączonych kropkami (np. 192.168.0.1). Każda z tych liczb jest jedno-, dwu- lub trzycyfrowa.

```
[1-9][0-9]{0,2}(\.[1-9][0-9]{0,2}){3}
```

Rysunek 3.6. Wzorzec odszukujący wszystkie wystąpienia adresów IP (v4)

Wykorzystując wyrażenia regularne wszystkie adresy IP (v4) możemy odnaleźć za pomocą wzorca przedstawionego na rysunku 3.6.

Jeśli nie rozumiesz jak wzorzec na rysunku 3.6 odnajduje adresy IP, nie martw się. Wyjaśnimy to w dalszej części.

W systemie Linux wykorzystywane są dwa rodzaje wyrażeń regularnych: **BRE** (ang. Basic Regular Expression) oraz wersja rozszerzona **ERE** (ang. Extended Regular Expression) [2]. Niektóre komendy korzystają z wersji **BRE**, np. **grep**, zaś inne z wersji **ERE**, np. egrep.

3.3.1. BRE

W wersji podstawowej większość znaków oznacza swoje odpowiedniki literalne, czyli np. znak **A** z oznacza dosłownie znak **A** w tekście. Dodatkowo istnieje zestaw znaków specjalnych (metaznaków) przedstawionych w tabeli 3.7.

Dodatkowo można korzystać z predefiniowanych grup znaków. Zostały one przedstawione w tabeli 3.8.

Jeśli chcemy skorzystać z konkretnej grupy, musimy ją umieścić w metaznaku oznaczającym zbiór, czyli jeśli chcemy znaleźć cyfrę (**[:digit:]**), to musimy to zapisać w następujący sposób **[[:digit:]]**. Oczywiście, w tworzonym zbiorze mogą wystąpić również inne znaki, np. **[[:digit:]A-Fa-f]**.

Przykłady

Przykład 1. *Podaj wyrażenie, które będzie wyszukiwało tagi HTML. Tagi HTML rozpoczynają się od znaku < i litery oraz kończą na znaku >, zaś w środku mogą się znajdować dowolne znaki oprócz >.*

Rozwiązanie:

```
<[a-zA-Z][^>]*>
```

Szukamy łańcuchów, które rozpoczynają się od < i kończą na >. Za to odpowiedzialne są pierwszy i ostatni znak. Następnie drugi znak musi być literą (małą lub dużą). W środku zaś może być dowolna ilość znaków, które nie są znakiem kończącym, czyli są różne od >.

Do testowania wyrażeń regularnych BRE można wykorzystać komendę **grep**. Służy ona do przeszukiwania pliku i dokładniej zostanie omówiona później.
W celu sprawdzenie wyrażenia regularnego należy użyć następującej składni **grep *wyrażenie* nazwa_pliku**.
W tym celu potrzebujemy jeszcze pliku. Najprościej będzie ściągnąć stronę internetową na dysk za pomocą komendy wget - np. **wget www.onet.pl**. Ściągnie ona na dysk plik o nazwie **index.html**.
Teraz wystarczy użyć komendy **grep *wyrażenie* index.html**.

Przykład 2. *Podaj wyrażenie, które znajdzie odnośniki bezwzględne.*

.	Dopasowuje jeden dowolny znak. Niektóre programy lub funkcję korzystające z wyrażeń BRE nie dopasowują znaku nowej linii ($\backslash n$). Jeśli kropka znajduje się w nawiasach kwadratowych, to traci swoją funkcję i dopasowuje tylko kropkę, np. $[t.k]$ dopasowuje jeden ze znaków: t, k lub kropkę.
[ ]	Dopasowuje jeden znak ze zbioru znajdującego się w nawiasach. Zbiór jest ciągiem znaków, np. $[abcd1234.,]$. Dla ułatwienia dodano znak $-$, który określa przedział znaków. Na przykład, gdy chcemy odnaleźć jedną małą literę (bez polskich znaków) możemy użyć przedziału $[a-z]$. Dodatkowo, przedziały można łączyć, dlatego wzorzec dopasowujący jedną małą lub dużą literę albo cyfrę będzie wyglądać $[a-zA-Z0-9]$. Jeśli chcemy w zbiorze zawrzeć również myślnik $-$, to musimy go umieścić na początku lub końcu, tj. $[-a-zA-Z0-9]$. To samo tyczy się znaków [oraz].
[^]	Działa analogicznie do poprzednika z tą różnicą, że odrzuca wszystkie znalezione znaki. Na przykład wzorzec $[\hat{}A-Z]$ dopasuje wszystkie znaki, które nie są dużą literą.
^	Oznacza początek przeszukiwanego tekstu lub linii, jeśli tekst składa się z wielu linii. Na przykład wzorzec $\hat{}root$ dopasuje wszystkie linie rozpoczynające się od znaków *root*.
$	Znak oznaczający koniec tekstu lub linii.
\(\)	Zapamiętuje część wyrażenia (zwaną również podwyrażeniem lub blokiem), która znajduje się między nawiasami. Zapamiętany blok można następnie wykorzystać ponownie (patrz $\backslash n$).
$\backslash n$	Zawiera to co zostało zapamiętane w n-tej części wyrażenia przy pomocy nawiasów opisanych powyżej. Wartość n jest cyfrą od 1 do 9. Niektóre narzędzia wykorzystujące BRE umożliwiają zapamiętanie większej ilości podwyrażeń.
$*$	Dopasowuje element poprzedzający 0 lub więcej razy. Na przykład wyrażenie **ab*c** dopasowuje ciągi znaków **ac**, **abc**, **abbc**, itd. Oczywiście element poprzedzający nie musi być literałem, może to być na przykład zbiór elementów, czyli np. **a[bcde]*f**.
$\backslash\{m,n\backslash\}$	Dopasowuje element poprzedzający od **m** do **n** razy. Jeśli wrócisz do przykładu z adresami IP, to zauważysz, że tam jest użyty ten metaznak. Na przykład, $a\backslash\{3,5\backslash\}$ Dopasowuje tylko **aaa**, **aaaa**, oraz **aaaaa**. Jeśli druga zmienna zostanie pominięta, to wyrażenie będzie dopasowywało ciąg, aż napotka niepasujący znak (nie będzie ograniczenia górnego na ilość wystąpień).
$\backslash < a$	Dopasuje te słowa, które zaczynają się na literę **a**. Metaznak $\backslash <$ reprezentuje początek słowa.
$a\backslash >$	Dopasuje te słowa, które kończą się na literę **a**. Metaznak $\backslash >$ reprezentuje koniec słowa.

Tablica 3.7. Tabela metaznaków w wersji BRE

[: *alnum* :]	znaki alfanumeryczne [a-zA-Z0-9]
[: *alpha* :]	znaki alfabetyczne [a-zA-Z]
[: *digit* :]	cyfry [0-9]
[: *lower* :]	małe litery [a-z]
[: *upper* :]	duże litery[A-Z]
[: *blank* :]	spacja oraz tabulator
[: *cntrl* :]	znaki kontrolne
[: *graph* :]	znaki drukowalne bez odstępów
[: *print* :]	znaki drukowalne z odstępami
[: *punct* :]	znaki drukowalne bez odstępów, liter oraz cyfr
[: *space* :]	wszystkie znaki odstępu
[: *xdigit* :]	cyfry w systemie heksadecymalnym [0-9a-fA-F]

Tablica 3.8. Tabela grup znaków

Odnośnik bezwzględny jest adresem URL, który zawiera cały schemat, czyli rozpoczyna się od protokołu, który może być szyfrowany (https) lub nie (http).

Rozwiązanie:

```
https\{0,1\}://[^"]*
```

Na początku szukamy ciągu znaków rozpoczynającego się od **http**. Następnie może, ale nie musi wystąpić litera **s**. Później szukamy **://**, a na końcu może wystąpić dowolny ciąg znaków za wyjątkiem znaku ", który kończy odnośnik w języku HTML.

Przykład 3. *Stwórz plik z listą studentów, gdzie każdy wiersz zawiera imię, drugie imię oraz nazwisko oddzielone jedną spacją.*

Podaj wyrażenie, które odnajdzie studentów, którzy mają takie samo pierwsze oraz drugie imię. Zakładamy, że imiona i nazwiska rozpoczynają się od dużej litery i nie zawierają polskich znaków.

Rozwiązanie:

```
\([A-Z][a-z]*\) \1 [A-Z][a-z]*
```

Najpierw szukamy pierwszego imienia, czyli ciągu znaków rozpoczynającego się od dużej litery i zawierającego dowolną ilość małych liter. Znalezione imię zachowujemy w schowku nr 1, którego wykorzystujemy po spacji. Na końcu szukamy jeszcze nazwiska według tej samej zasady co imię.

Komenda **grep** w wyrażeniu **[a-z]** domyślnie uwzględnia polskie znaki.

Zadania

Zadanie 1. *Podaj wyrażenie, które znajdzie tytuł strony w kodzie HTML.*

Zadanie 2. *Podejrzyj plik /var/log/apache2/error.log. Przykładowy wpis:*

```
[Thu Feb 23 00:05:53 2012] [error] [client 127.0.0.1] Treść błędu
```

( )	Ten metaznak istnieje w standardzie BRE, jednakże w ERE jego funkcja została zmieniona. Ma on na celu zgrupowanie wielu elementów w jeden element. Jest to bardzo przydatne w połączeniu z metaznakiem |.
|	Metaznak wprowadza alternatywę do wyrażeń regularnych. Dopasowuje jeden z dwóch elementów, które są po obu stronach znaku. Na przykład wyrażenie $abc\|def$ dopasowuje ciąg abc lub ciąg def. Jeśli zaś chcemy dopasować ciągi $abcf$ lub $adef$, musimy skorzystać z grupowania omówionego powyżej. Wtedy wyrażenie będzie wyglądać następująco $a(bc\|de)f$.
+	Dopasowuje co najmniej 1 wystąpienie elementu występującego po metaznaku. Działa tak samo jak $\{1,\}$
?	Dopasowuje wyrażenie, w którym element poprzedzający może wystąpić 0 lub 1 raz. Działa tak samo jak $\{0,1\}$.

Tablica 3.9. Tabela nowych metaznaków w wersji ERE

Podaj wyrażenie, które znajdzie wpisy z dnia 1 lutego 2012 roku. Pamiętaj o tym, że data wpisu jest zawsze na początku pliku.

Zadanie 3. *Podaj wyrażenie, które znajdzie polski kod pocztowy.*

3.3.2. ERE

Standard ERE (ang. Extended Regular Expression) był rozwijany ze standardem BRE. Wprowadza on kilka nowych metaznaków oraz zasad tworzenia wyrażeń.

3.3.3. Zmiany

1. W standardzie ERE znak \ powoduje, że znak występujący po nim traci swoje właściwości metaznaku, czyli na przykład \$ nie oznacza już końca wiersza tylko zwykły znak dolara.
2. Metaznaki, które w standardzie BRE były poprzedzone znakiem \ (np. \{ \}) w ERE, są zastąpione odpowiednikami bez \ (np. { }).
3. Usunięty został metaznak $\backslash n$.
4. Dodano nowe metaznaki wyszczególnione w tabeli 3.9.

Przykłady

Odpowiednikiem komendy **grep** dla standardu ERE jest **egrep**, z którego można korzystać sprawdzając działanie wyrażeń regularnych.

Przykład 1. *Przepisz wyrażenie BRE odnajdujące odnośniki bezwzględne, używając metaznaków z ERE.*

Rozwiązanie:

```
https?://[^"]*
```

Wyrażenie $\backslash\{0,1\backslash\}$ można zastąpić metaznakiem ?.

Przykład 2. *Podaj wyrażenie, które dopasuje protokół FTP lub HTTP szyfrowany lub nieszyfrowany.*

Rozwiązanie:

```
(http|ftp)s?://
```

Alternatywa dopasowuje odpowiedni protokół. Następnie za pomocą ? dopasowujemy literkę **s**, która odpowiada za szyfrowanie.

Przykład 3. *Podaj wyrażenie, które dopasuje liczbę rzeczywistą.*

Rozwiązanie:

```
-?[1-9][0-9]*(\.[0-9]+)?
```

Liczba rzeczywista może być dodatnia lub ujemna, stąd znak zapytania przy minusie na początku. Jeśli minus nie występuje, to zakładamy, że liczba jest dodatnia. Następnie musi być co najmniej 1 cyfra, która jest różna od zera, gdyż liczba nie może zaczynać się od zera. Następnie może, ale nie musi wystąpić część rzeczywista, która jest oddzielona kropką i zawiera co najmniej 1 cyfrę.

Przykład 4. *Podaj wyrażenie regularne rozpoznające komentarz C: /* ... */.*

Rozwiązanie:

```
/\*([^*]|(\*+[^/]))*\*+/
```

Komentarz rozpoczyna się sekwencją /∗ zaś kończy co najmniej 1 gwiazdką i znakiem /. W środku zaś mogą być wszystkie znaki oprócz gwiazdki albo wiele gwiazdek, po których nie występuje znak /.

Zadania

Zadanie 1. *Podaj wyrażenie regularne rozpoznające słowa, w których występuje parzysta liczba wystąpień litery a.*

Zadanie 2. *Podaj wyrażenie regularne, które rozpoznaje ciąg znaków składający się z parzystej liczby wystąpień znaku a i znaku b.*

Zadanie 3. *Podaj wyrażenie, które sprawdza poprawność hasła. Hasło musi się składać z co najmniej 8 znaków i nie więcej niż 16 znaków oraz musi w nim wystąpić co najmniej jedna mała i duża litera oraz cyfra.*

3.4. Przeszukiwanie tekstu

Przeszukiwanie tekstu jest bardzo częstym zajęciem w systemie Linux. Szczególnie, gdy chcemy się dowiedzieć pewnych informacji o aktualnym stanie systemu, np. sprawdzić czy pewien użytkownik jest zalogowany lub czy pewien program jest uruchomiony albo przeszukać pliku konfiguracyjne, np. sprawdzić jaka jest maksymalna wielkość pliku, który może zostać wgrany na serwer.

Najprostszy sposób przeszukania treści pliku polega na otwarciu go w edytorze tekstu konsolowym (np. mcedit) lub graficznym (np. gedit) i skorzystanie z wbudowanej funkcji szukania. Jednakże jeśli plik jest duży lub chcemy jedynie

znaleźć wyszukiwany tekst (np. sprawdzić, czy istnieje) bez potrzeby edytowania go, prostsze mogą się okazać komendy, które przeszukują treść plików wykorzystując wyrażenia regularne.

3.4.1. grep

Komenda **grep** służy do przeszukiwania treści pliku. Wykorzystywaliśmy ją już podczas sprawdzania poprawności wyrażeń regularnych budowanych według składni BRE.

```
grep wzorzec ścieżka_do_pliku
```

Rysunek 3.7. Składnia komendy **grep**

Pierwszym argumentem polecenia jest wzorzec regularny BRE. Jeśli wzorzec nie jest prostym ciągiem znaków (np. zawiera spacje), to należy ująć go w cudzysłów.

Drugim, opcjonalnym argumentem jest ścieżka do pliku, który ma być przeszukany. Jeśli drugi argument nie wystąpi, to polecenie będzie przeszukiwało standardowe wejście.

Przydatne flagi

— -i - Traktuje wyrażenie regularne jako *case-insensitive*, czyli nie zwraca uwagi na wielkość znaków.
— -e - Określa wyrażenie regularne. Może być wykorzystane do podania wielu wzorców lub do wzorca rozpoczynającego się od −.
— -f - Wczytuje z pliku wzorce oddzielone znakiem nowej linii.
— -c - Wyświetla ilość linii, które zostały dopasowane do wzorca.
— -v - Odwrócenie wzorca. Znajduje linie, które nie spełniają podanego wzorca.
— -E - Traktuje wyrażenie regularne jako rozszerzone (ERE).
— -r - Przeszukuje pliki w folderze oraz wszystkich jego podfolderach rekursywnie.
— -F - Traktuje wzorzec jako zwykły ciąg znaków, który już nie jest wyrażeniem regularnym.

Polecenie **grep** jest bardzo często wykorzystywane w potokach.

Przykłady

Przykład 1. *Podaj komendę, która sprawdzi, czy użytkownik* ***user*** *jest zalogowany w systemie.*

Rozwiązanie:

```
w -h | grep ^user
```

Przykład 2. *Podaj komendę, która wyświetli te linie z pliku* ***dane.txt****, które zawierają kropkę.*

Rozwiązanie:

```
grep [.] dane.txt
grep -F . dane.txt
```

W wyrażeniu regularnym nie można użyć samej kropki, gdyż będzie ona dopasowywała każdy znak. Dlatego kropkę trzeba ująć w jednoelementowy zbiór lub skorzystać z flagi **-F**.

Przykład 3. *Podaj komendę, która wyświetli te linie z pliku **dane.txt**, które nie zawierają imienia **Anna** bez względu na wielkość liter.*

Rozwiązanie:

```
grep -i -v anna dane.txt
```

Przykład 4. *Podaj komendę, która sprawdzi, czy program **program** jest uruchomiony.*

Rozwiązanie:

```
ps haux | grep program
```

Jest to sposób szybkiego sprawdzenia, czy **program** jest uruchomiony. Nie daje to 100% pewności, gdyż może być na przykład inny uruchomiony program o podobnej nazwie, np. znajduje **program-dev**, podczas gdy **program** nie jest uruchomiony.

Zadania

Zadanie 1. *Podaj komendę, która sprawdzi, ile razy w pliku występuje słowo mamona.*

Zadanie 2. *Podaj komendę, która wyświetli linie, które rozpoczynają się od słów **Log** lub **Lag**.*

Zadanie 3. *Podaj komendę, która wyświetli linie, które kończą się literą **a**.*

Zadanie 4. *Podaj komendę, która wyświetli linie, które zawierają słowa kończące się literą **a**.*

Zadanie 5. *Podaj komendę, która wyświetli linie, które zawierają słowa rozpoczynające się literą **a** oraz w których przedostatnia litera jest różna od **n** i **d**.*

3.4.2. Inne warianty polecenia grep

W celu ułatwienia przeszukiwania plików stworzono trzy alternatywy polecenia grep.

Polecenie **egrep** jest odpowiednikiem **grep -E** i traktuje wzorzec, jako rozszerzone wyrażenie regularne (ERE).

Polecenie **fgrep** jest odpowiednikiem **grep -F** i traktuje wzorzec jako zwykły ciąg znaków, w którym każdy znak odpowiada swojemu literałowi.

Polecenie **rgrep** jest odpowiednikiem **grep -r** i szuka wzorca we wszystkich plikach w danym folderze oraz jego podfolderach rekursywnie.

3.5. Edytor SED

Najprostszym sposobem na wprowadzenie zmian w plikach jest wykorzystanie domyślnych - konsolowych (np. vi) lub graficznych (np. gedit) - edytorów tekstu. Z drugiej strony jest to czasochłonny sposób, nawet gdy wykorzystujemy dodatkowe funkcje, takiej jak np. *Znajdź i Zamień*. Jeszcze gorszym przypadkiem jest sytuacja, gdy podobne operacje musimy wykonać na wielu plikach. Wtedy musielibyśmy otworzyć i edytować każdy plik z osobna.

Rozwiązaniem powyższego problemu jest SED (ang. stream edytor), czyli strumieniowy edytor tekstu. Jest to narzędzie stworzone do edycji tekstu w trybie wsadowym, w odróżnieniu do wcześniej wspomnianych edytorów, które pracują w trybie edycji interaktywnej.

SED jest idealnym rozwiązaniem wcześniej wspomnianej sytuacji, w której chcemy wykonać szereg analogicznych operacji modyfikujących plik lub zbiór plików. Należy wtedy napisać skrypt edytujący i wywołać go dla każdego pliku.

Edytor SED jest najczęściej wykorzystywany do prostych modyfikacji tekstu, np. zamiany lub usunięcia fragmentów tekstu, ale może również być wykorzystywany do bardziej skomplikowanych operacji, które również poznamy w tej sekcji.

SED jest uruchamiany za pomocą komendy o tej samej nazwie.

```
sed [-n] 'polecenie' [nazwa_pliku ... ...]
sed [-n] -e 'polecenie' ... [nazwa_pliku ... ...]
sed [-n] -f plik_z_poleceniami ... [nazwa_pliku ... ...]
```

Rysunek 3.8. Składnia komendy sed

Opcje polecenia:

— **-n** - zablokowanie wyświetlania edytowanych wierszy (przydatne, gdy korzystamy z polecenia **p**);
— **-e 'polecenie'** - określenie polecenia edycji pliku (jeśli polecenie jest jedno, to nie trzeba pisac flagi **-e**);
— **-f plik_z_poleceniami** - ścieżka do pliku, w którym znajdują się polecenia edycji (przydatne w sytuacji, gdy wykonujemy wiele poleceń edycji).

Proces edycji działa w ten sposób, że edytor wczytuje kolejny wiersz z pliku lub ze standardowego wejścia, jeśli plik nie został określony, wykonuje na tym wierszu wszystkie polecenia edycji i wypisuje na standardowe wyjście wynik, jeśli nie została użyta flaga **-n**.

Edytor SED nie zmienia zawartości plików, lecz wypisuje wynik na standardowe wyjście. Dlatego jeśli chcemy zapisać wynik do pliku, musimy przekierować do niego standardowe wyjście lub skorzystać z przyrostka **w**.

3.5.1. Polecenie zastąpienia

Najczęściej używanym poleceniem edycji jest zastąpienie oznaczane literką **s** (ang. substitute), którego składania jest następująca *s/wyrażenie/zastąpienie/przyrostek*, gdzie *wyrażenie* jest wyrażeniem regularnym, które dopasowujemy do

wiersza, zaś wynik zastępujemy drugą częścią polecenia - *zastąpienie*. Znaki / oddzielające części polecenia mogą być zastąpione dowolnymi znakami, przy czym ważna jest konsekwencja, czyli zamiana wszystkich znaków / na nowy znak, np. ;. Jest to przydatne w sytuacji, gdy w *wyrażeniu* lub *zastąpieniu* występuje właśnie znak / (np. podczas pracy ze ścieżkami do plików).

Ostatnim elementem w poleceniu edycji jest przyrostek. W przypadku zastąpienia można dodać przyrostek **g**, dzięki czemu polecenie będzie wykonane dla wszystkich wystąpień szukanego wyrażenia w wierszu, a nie tylko dla pierwszego z nich.

W przypadku, gdy chcemy wykonać wiele poleceń edycji, możemy to zrobić na kilka sposobów.

```
sed -e 's/wyr1/zast1/' -e 's/wyr2/zast2/' -e 's/wyr3/zast3/' plik
sed 's/wyr1/zast1/;s/wyr2/zast2/;s/wyr3/zast3/' plik
sed 's/wyr1/zast1/' plik | sed 's/wyr2/zast2/' | sed 's/wyr3/zast3/'
```

Wszystkie powyższe sposoby działają tak samo. Ponadto, można wszystkie polecenia edycji umieścić w pliku i wykorzystać flagę **-f**.

3.5.2. Inne polecenia

W edytorze *sed* można wykorzystać również inne polecenia oprócz zastąpienia. Przykładowe z nich to:

— = - numerowanie linii (*sed = plik*);
— */wyrażenie/ N* - dodanie do wiersza, który dopasowano do wyrażenia wiersza, który występuje po nim;
— */wyrażenie/ d* - usunięcie wierszy, w których dopasowano wyrażenie;
— */wyrażenie/ p* - wyświetlenie wierszy, w których dopasowano wyrażenie (zwykle używane z flagą **-n**);
— */wyrażenie/ q* - zakończenie przetwarzania pliku, gdy wystąpi wiersz, w którym można dopasować wyrażenie;
— */wyrażenie/a text* - dodanie nowej linii o treści *aaa* po linii, w której dopasowano wyrażenie;
— */wyrażenie/i text* - dodanie nowej linii o treści *aaa* przed linią, w której dopasowano wyrażenie;
— */wyrażenie/c text* - zastąpienie całej linii, w której dopasowano wyrażenie nową linią o treści *aaa*;
— *y/abcd/ABCD* - zastąpienie znaków z pierwszej listy znakiem z drugiej listy o tym samym numerze porządkowym (np. małe c jest zastępowane dużym C).

3.5.3. Sed w potokach

Polecenie *sed* jest najczęściej wykorzystywane w potoku jako pośrednik, który ma za zadanie wczytać tekst i przekształcić go w ten sposób, żeby mógł być wykorzystany przez kolejne polecenie. Na przykład, gdy pewien program potrzebuje listy używanych przez użytkowników powłok, to możemy ją przygotować za pomocą

prostego polecenia edycji oraz dodatkowej komendy *uniq*, która usuwa powtarzające się linie. Przykładem przygotowania tych danych wejściowych jest komenda 3.5.3.

```
sed 's/.*://' /etc/passwd | uniq -u
```

Rysunek 3.9. Przygotowanie listy powłok wykorzystywanych przez użytkowników systemu

W tym przypadku pobieramy dane z pliku **/etc/passwd**, w którym jest lista użytkowników oraz ich dane - m. in. wykorzystywana powłoka, która jest ostatnim elementem wiersza, gdzie separatorem jest dwukropek. Dlatego *sed*em usuwamy wszystkie znaki wraz z ostatnim dwukropkiem. Dodatkowo musimy usunąć powtarzające się wiersze, dlatego używamy polecenia *uniq*, które domyślnie usuwa powtarzające się, sąsiednie linie. Flaga **-u** usuwa z poprzedniego zdania słowo *sąsiednie*, czyli po prostu zwracane są wszystkie unikalne wiersze.

3.5.4. Wsteczne odwołanie

W poleceniu zastępowania można korzystać ze wstecznego odwołania analogicznie go wyrażeń regularnych BRE. Wszystkie części *wyrażenia* ujęte w nawiasy \(\) będą miały odpowiednika w *zastąpieniu*. Odpowiedniki te będą dostępne za pomocą wyrażenia $\backslash n$, gdzie n jest numerem części ujętej w nawiasy w szukanym *wyrażeniu*. W miejsce odpowiedników zostaną wstawione łańcuchy znaków, które dopasuje wyrażenie regularne.

```
sed 's/\(imie\) \(nazwisko\)/\2\1/' plik
```

Rysunek 3.10. Zamiana miejscami imienia i nazwiska

Oczywiście, podobnie jak w wyrażeniach BRE, wsteczne odwołanie można również wykorzystywać w obrębie *wyrażenia* szukania.

3.5.5. Znak &

Szczególnym przypadkiem wstecznego odwołania jest wykorzystanie znaku &. Znak ten przechowuje ciąg znaków, który został dopasowany przez całe szukane *wyrażenie*. Stąd, może być wykorzystany tylko w *zastąpieniu*.

```
sed 's/wyrażenie/&&/' plik
```

Rysunek 3.11. Przykład wykorzystania znaku & do duplikowania szukanego tekstu

Znak & może być zastąpiony odpowiednim odwołaniem wstecznym.

```
sed 's/\(wyrażenie\)/\1/' plik
```

Rysunek 3.12. Zastąpienie znaku % odwołaniem wstecznym

3.5.6. Zastąpienie wybranego wystąpienia

Na początku omawiania *zastąpienia* wspomnieliśmy o fladze **g**, która modyfikowała polecenie w ten sposób, że zastępowane były wszystkie wystąpienia dopasowane przez szukane wyrażenie.

Oprócz tego, *sed* umożliwia określenie konkretnego wystąpienia, które ma być zastąpione.

```
sed 's/wyrażenie/zastąpienie/10' plik
```

Rysunek 3.13. Zastąpienie tylko 10 wystąpienia *wyrażenia*

Dodatkowo, można wybrać od którego dopasowania wyrażenia edytor ma zastępować odnaleziony ciąg znaków. Na przykład, jeśli chcemy wyświetlić tylko 10 pierwszych znaków każdego wiersza, a pozostałe zastąpić znakiem gwiazdki (*), to możemy skorzystać z poniższego polecenia.

```
sed 's/./*/11g' plik
```

Rysunek 3.14. Wyświetlenie tylko początku wiersza

3.5.7. Zapisanie wyniku do pliku

Edytor *sed* domyślnie nie modyfikuje plików, z których wczytuje wiersze. Jeśli chcemy zapisać nową, zmodyfikowaną postać pliku, to możemy przekierować standardowe wyjście do pliku.

Innym sposobem na zapisanie wyniku do pliku jest skorzystanie z flagi **w**.

```
sed 's/./*/11gw plik.out' plik
```

Rysunek 3.15. Zapisanie wyniku do pliku

Różnica polega na tym, że przekierowanie wyjścia zapisze do pliku treść całego pliku wejściowego, również z niezmienionymi wierszami. Wykorzystanie flagi **w** spowoduje zapisanie do pliku tylko tych wierszy, które zostały zmodyfikowane oraz wciąż wyświetli pełną treść zmodyfikowanego pliku na standardowym wyjściu.

3.5.8. Wybieranie wierszy

Edytor *sed* umożliwia ograniczenie zbioru wierszy, które mają być zmienione. Oczywiście częściowo robi to na przykład szukane wyrażenie w poleceniu zastępowania, jednakże możliwa jest jeszcze selekcja na wyższym poziomie.

Zakres wierszy można ograniczyć wykorzystując następujące sposoby:
— wybranie linii o konkretnym numerze;
— wybranie zakresu linii z przedziału określonego dwoma numerami;
— wybranie linii, które dopasowują podane wyrażenie;
— wybranie zakresu linii od początku pliku do linii, która dopasowuje podane wyrażenie;
— wybranie zakresu linii od linii, która dopasowuje podane wyrażenie do końca pliku;
— wybranie zakresu linii od linii, która dopasowuje pierwsze podane wyrażenie do linii, która dopasowuje drugie podane wyrażenie.

Pierwszy sposób polega po prostu na podaniu numeru linii, którą należy zmodyfikować zgodnie z podanym poleceniem.

```
sed '25 s/./*/11g' plik
```

Rysunek 3.16. Ukrycie znaków numer 11 i następnych, ale tylko w 25 wierszu

W drugim przypadku należy podać dwie liczby oddzielone przecinkiem, które określają początek i koniec przedziału, który zostanie wyedytowany zgodnie z następującym po nich poleceniem.

```
sed '15,25 s/./*/11g' plik
```

Rysunek 3.17. Ukrycie znaków numer 11 i następnych w wierszach o numerach od 15 do 25 włącznie

Adresowanie za pomocą wyrażenia polega na tym, że edytor próbuje do każdego wiersza dopasować podane wyrażenie, i jeśli mu się to uda to może wykonać polecenie na tym wierszu.

```
sed '/\/bin\/sh/ s/./*/11g' plik
```

Rysunek 3.18. Ukrycie znaków numer 11 i następnych w wierszach użytkowników, który korzystają z powłoki */bin/sh*

W przypadku wyrażeń, które adresują wiersze nie można zastąpić znaków / innym znakiem drukowanlnym, tak jak ma to miejsce w poleceniach. Znak separatora można zmienić, gdy pierwszym znakiem wyrażenia jest \. Wtedy drugi, dowolny znak drukowalny staje się separatorem.

```
sed '\_/bin/sh_ s/./*/11g' plik
```

Rysunek 3.19. Zmiana separatora

Jeśli chcemy w wyrażeniu skorzystać ze znaku / jako literału, można go poprzedzić znakiem \ bez potrzeby zmiany znaku separatora.

Najbardziej rozbudowanym przypadkiem jest określenie przedziału wierszy za pomocą dwóch wyrażeń regularnych. Polega on na podaniu dwóch wyrażeń regularnych oddzielonych przecinkiem. Edytor napotykając wiersz, w którym może dopasować pierwsze wyrażenie zaczyna wykonywać podane polecenia edycji na tym i następnych wierszach, aż spotka wiersz, do którego będzie mógł dopasować drugie wyrażenie (na nim również wykonuje polecenie edycji). Proces ten jest powtarzany, aż edytor napotka koniec pliku.

```
sed '/bash$/,/sh$/ s/./*/g' plik
```

Rysunek 3.20. Ukrycie wierszy z przedziałów określonych wyrażeniami

Zamiast jednego z wyrażeń można użyć liczby określającej numer wiersza. Szczególną rolę gra znak $, który reprezentuje koniec pliku.

```
sed '1,/koniec$/ s/./*/g' plik
```

Rysunek 3.21. Ukrycie wierszy od początku pliku do wiersza z ciągkiem znaków *koniec*

```
sed '/poczatek/,$ s/./*/g' plik
```

Rysunek 3.22. Ukrycie wierszy od wiersza z ciągiem znaków *poczatek* do końca pliku

3.5.9. Grupowanie poleceń

W przypadku, gdy na tym samym zbiorze wierszy (również jednoelementowym) chcemy wykonać kilka operacji możemy skorzystać z grupowania poleceń.

Użycie wielu poleceń (za pomocą flagi **-e** lub **-f**) z tym samym wyrażeniem wybierającym wiersze nie zadziała w tym przypadku, gdyż na kolejne polecenia przekazywany jest wynik poprzedniego polecenia, więc adresowanie w kolejnych poleceniach może nie wybrać zmodyfikowanych wierszy.

Na przykład jeśli wybieramy wiersze, które zawierają ciąg znaków *tego szukamy* i w pierwszym poleceniu zastępujemy go ciągiem *to wstawiamy*, to drugie polecenie już nie dopasuje tego wiersza, gdyż nie odnajdzie w nim *tego szukamy*.

Żeby temu zaradzić, wprowadza się grupowanie poleceń. Składa się ono z jednego wyrażenia wybierającego wierszę oraz wielu poleceń edycji ujętych w klamry { }.

```
sed -n '
/szukamy/ {
     s/szukamy/nowy tekst/
     # a tu możemy wstawić komentarz
     p
}
' plik
```

Rysunek 3.23. Przykładowe użycie grupowania poleceń

W zaprezentowanym przykładzie bierzemy pod uwagę wiersze, które zawierają ciąg znaków *szukamy*, a następnie zamieniamy go na *nowy tekst* i wyświetlamy ten zmieniony wiersz.

Jeśli skorzystalibyśmy z dwóch odrębnych poleceń z tym samym wyrażeniem wybierającym wiersze, to żaden wiersz nie zostałby wyświetlony, gdyż drugie polecenie (wyświetlające) nie dopasowałoby żadnego wiersza.

3.5.10. Odwrócenie adresu

W poleceniach edytujących można skorzystać również ze znaku !, który odwraca adres polecenia. Innymi słowy polega to na tym, że podane polecenie jest zastosowane do wszystkich wierszy, które nie są dopasowane przez wyrażenie adresujące według wszystkich, wcześniej opisanych reguł.

Odwrócenie adresu przydaje się w sytuacji, gdy określenie wierszy, które mają być pozostawione bez zmian jest łatwiejsze albo, gdy jest to możliwe w przeciwieństwie do określenia wierszy, które mają być edytowane.

```
sed '/zostawiamy/ ! s/./*/g' plik
```

Rysunek 3.24. Ukrycie tylko tych wierszy, w których nie ma ciągu znaków *zostawiamy*

Przykłady

Przykład 1. *Sprawdź, ile jest linii kodu w pliku* ***plik.sh*** *usuwając komentarze, czyli linie rozpoczynające się od znaku # oraz zliczając niepuste linie.*

Rozwiązanie:

```
sed 's/^#.*//'  f1 f2 f3 | grep -v '^$' | wc -l
```

Przykład 2. *Wyświetl informacje o użytkownikach z grupy o identyfikatorze* ***125****.*

Rozwiązanie:

```
sed -n '/^[^:]*:[^:]*:[^:]*:125/ p' /etc/passwd
```

Na początku wyłączamy domyślne wyświetlanie wierszy (**-n**). Następnie sprawdzamy czwarty element każdego wiersza przyjmując, że dwukropek jest separatorem. Tak jest skonstruowany plik */etc/passwd*, który przechowuje informacje o użytkownikach systemu. Sprawdzany element jest identyfikatorem grupy użytkownika, więc porównujemy go z oczekiwaną liczbą 125. Akcja, jaką podejmujemy, to po prostu **p**rint.

Przykład 3. *Wyświetl treść pliku z ponumerowanymi liniami, ale bez dodatkowych wierszy (ładnie sformatowane).*

Rozwiązanie:

```
sed = plik | sed 'N;s/\n/:  /'
```

Na początku używamy numerowania linii, w które wyposażony jest *sed*. Numerowanie to jednak polega na dodaniu przed każdą linią nowej linii z samym numerem. Dlatego później korzystamy jeszcze z 2 poleceń. Jako, że *sed* działa na wierszach, to bierze pod uwagę wszystkie znaki, dopóki nie napotka znaku nowej linii. Polecenie **N** dokleja do linii aktualnie przetwarzanej linię, która występuje po niej. Na końcu połączenie linii (znak nowej linii), które już istnieje w przetwarzanym wierszu zamieniamy na ciąg znaków "*:* ".

Zadania

Zadanie 1. *Usuń wszystkie białe znaki (spacje i tabulacje) występujące na końcu wierszy.*

Zadanie 2. *Wyświetl pierwsze 10 linii pliku na 4 różne sposoby.*

Zadanie 3. *W pliku* ***test*** *niektóre wiersze zawierają jedynie liczby w systemie szesnastkowym. Znajdź je i zamień małe literki a-f na ich duże odpowiedniki.*

Zadanie 4. *Usuń wszystkie linie między znakami { oraz } zastępując je napisem *** DELETED ***.*

Zadanie 5. *W pliku tekstowym* ***plik.txt*** *dodaj pustą linię przed każdym wierszem, który rozpoczyna się dużą literą.*

Rozdział 4

Skrypty BASH

Zanim zaczniemy pracę ze skryptami BASH, wytłumaczymy, czym one właściwie są.

BASH (ang. Bourne Again SHell) to interpreter poleceń zgodny z *sh*, który został stworzony przez Briana Foxa i Cheta Rameya. Łączy on w sobie zalety C-shella (*csh*) oraz shella Korna (*ksh*). BASH to nie tylko bardzo popularna powłoka używana w systemach Linux, ale również skryptowy język programowania.

Skrypty BASH to zbiór poleceń, które wykonujemy w konsoli powłoki BASH wzbogacony o zmienne, czy instrukcje sterujące (konstrukcja *if*, pętle, itp.). Są one umieszczone w zwykłym pliku tekstowym, w którym pierwsza linia jest specjalnym komentarzem, określającym ścieżkę do interpretera, który ma wykonać skrypt (np. *#!/bin/bash*). Wykonanie skryptu polega na przetłumaczeniu kolejnych poleceń na język zrozumiały dla procesora.

Typowym zastosowaniem skryptów jest zautomatyzowanie często wykorzystywanych sekwencji poleceń do administracji systemem, konfiguracji i wykonywania pojedynczych programów, tworzenia kopii zapasowych itp.

Skrypty mogą być również sterowane z zewnątrz za pomocą parametrów wywołania, które omówimy dokładniej w następnych rozdziałach.

4.1. Pierwsze kroki

4.1.1. Zanim zaczniemy pisać skrypt

Przed napisaniem pierwszego skryptu należy odnaleźć ścieżkę do interpretera, który będzie wykonywał nasze skrypty. Można go zlokalizować za pomocą polecenia *whereis*.

```
whereis bash
```

Najczęstszym folderem, w którym występuje interpreter BASH jest */bin/bash*. Jeżeli nie możemy odnaleźć ścieżki do interpretera, to być może nie jest on zainstalowany. Wtedy należy go zainstalować zgodnie z opisem w sekcji 2.7.

Kiedy odnajdziemy ścieżkę do interpretera (w tej książce będzie przyjęta */bin/bash*), możemy rozpocząć pisanie skryptu. Na początku tworzymy plik za pomocą polecenia *touch*. Nazwa pliku jest dowolna, jednakże do pliku można dodać rozszerzenie **.sh*, żeby zaznaczyć, że jest to skrypt powłoki. Dodanie rozszerzenia ma charakter jedynie informacyjny.

```
touch hello.sh
```

Po stworzeniu pliku należy go otworzyć za pomocą dowolnego edytora tekstu, graficznego (np. kate, gedit) lub konsolowego (np. vi, mcedit). Następnie należy dodać w pierwszej linii specjalny komentarz *#!/sciezka_do_interpretera*, czyli w naszym przypadku *#!/bin/bash*, który pozwoli uruchamiać nasze skrypty tak, jak uruchamiane są polecenia systemowe.

Specjalny komentarz w pierwszej linii nie jest wymagany do uruchomienia skryptu. Jeśli go pominiemy, będziemy musieli określić ścieżkę do interpretera podczas uruchamiania skryptu. Więcej na ten temat w sekcji 4.1.3.1.

```
1 #!/bin/bash
```

Skrypt 4.1. Pusty skrypt powłoki

Tak przygotowany plik jest już skryptem powłoki i w tej książce każdy skrypt będzie się rozpoczynał taką linią.

4.1.2. *Hello World* w BASHu

Wiedząc już jak tworzyć skrypty, przyjrzyjmy się klasycznemu *Hello World* w BASHu.

```
1 #!/bin/bash
2 echo Hello World
```

Skrypt 4.2. Hello World

Wykorzystujemy w nim polecenie *echo*, które wyświetla na standardowym wyjściu wartość parametru, którym w tym przypadku jest ciąg znaków *Hello World*.

Jeśli w konsoli wpiszemy to samo polecenie (*echo Hello World*), to efekt będzie taki sam, jak po uruchomieniu skryptu, o czym w następnej sekcji. Ten prosty skrypt pokazuje, że skrypty BASH to po prostu zbiór poleceń zapisanych w jednym pliku.

Oczywiście, skrypty okażą się jeszcze bardziej przydatne, gdy dodamy do nich instrukcje sterujące.

4.1.3. Uruchamianie skryptów

Mając gotowy skrypt *Hello World*, zapiszmy go w pliku **hello.sh**. Rozszerzenie ma wyłącznie charakter informacyjny. Nie określa ono, w jaki sposób plik zostanie wykonany - taką informację umieściliśmy w specjalnym komentarzu w pierwszym wierszu skryptu.

Każdy plik ma określone prawa dla właściciela (u), grupy właściciela (g) oraz pozostałych użytkowników (o). Każde z nich może być dowolną kombinacją trzech typów: prawo do odczytu (r), prawo do zapisu (w) oraz prawo do wykonywania (x). Jak sama nazwa wskazuje, musimy nadać sobie (właścicielowi pliku) prawo do wykonywania pliku (skryptu). Robimy to za pomocą polecenia 4.1.

```
chmod u+x plik
```

Rysunek 4.1. Nadanie właścicielowi prawa do wykonywania pliku

Po nadaniu prawa do wykonywania pliku, uruchamiamy go prostym poleceniem 4.2.

```
./hello.sh
```

Rysunek 4.2. Uruchomienie skryptu ze specjalnym komentarzem oraz prawami do wykonywania pliku

4.1.3.1. Uruchamianie skryptów bez specjalnego komentarza

Mówiliśmy wcześniej o tym, że specjalny komentarz ze ścieżką do interpretera nie jest wymagany w skrypcie. Jeśli go nie umieścimy, to system nie będzie miał informacji, za pomocą którego interpretera wykonać skrypt.

Dlatego podczas uruchamiania skryptu musimy sami określić, z którego interpretera chcemy skorzystać. Wtedy nie musimy posiadać uprawnień do wykonania pliku, a jedynie prawo do jego odczytania. Przykład uruchomienia skryptu 4.4 przedstawiono na 4.3.

```
1 echo Hello World
```

Skrypt 4.3. Hello World bez specjalnego komentarza

```
/bin/bash hello.sh
```

Rysunek 4.3. Uruchomienie skryptu bez specjalnego komentarza

4.1.3.2. Uruchamianie skryptów bez podawania ścieżki

W przypadku, gdy w skrypcie użyjemy specjalnego komentarza określającego interpreter, uruchamiamy go za pomocą polecenia *./skrypt* pod warunkiem, że znajdujemy się w katalogu, w którym umieszczony jest skrypt. Jeśli chcemy uruchomić skrypt z innego katalogu, to musimy podać pełną ścieżkę do pliku - względną lub bezwzględną. Staje to się niewygodne, gdy chcemy mieć prosty dostęp do skryptu bez względu na miejsce, w którym się znajduje skrypt oraz użytkownik.

W tym celu tworzy się katalog (np. w folderze domowym użytkownika, bądź w jednym z folderów wspólnych użytkowników), w którym umieszcza się stworzone skrypty. Następnie, dodaje się jego ścieżkę bezwzględną do zmiennej środowiskowej **PATH** (4.4).

```
user@localhost:~$ PATH=$PATH:/sciezka_do_folderu
```

Rysunek 4.4. Aktualizacja zmiennej środowiskowej **PATH**

Po tym zabiegu możemy uruchamiać skrypty tak samo, jak polecenia systemowe (np. *hello.sh*) pod warunkiem, że mamy prawo do jego wykonywania.

Zmienna **PATH** przechowuje ścieżki oddzielone dwukropkiem ":"do katalogów, które przeszukiwane są podczas uruchamiania poleceń systemowych.

4.2. Komentarze

Podczas tworzenia pierwszego skryptu poznaliśmy specjalny komentarz, który zaczyna się dwoma znakami #!. Jeśli po znaku # nie umieścimy wykrzyknika, to interpreter potraktuję dalszą część linii (lub całą linię, gdy # jest pierwszym znakiem) jako komentarz.

```
1 #!/bin/bash
2 #komentarz od poczatku linii
3 echo Hello World #komentarz za poleceniem
```

Skrypt 4.4. Hello World

4.3. Zmienne

W skryptach BASH zmienne są deklarowane oraz definiowane tak, jak przedstawiono w przykładzie 4.5. Zadeklarowana została zmienna o nazwie *liczba* oraz przypisano jej wartość liczbową 11.

```
1 #!/bin/bash
2 # deklaracja i definicja zmiennej
3 liczba=11
```

Skrypt 4.5. Hello World

Między nazwą zmiennej, znakiem równości, a wartością nie może być spacji. Żeby sprawdzić, co się stanie, wpiszmy w konsoli polecenia z 4.7. Interpreter potraktuje spację jako koniec polecenia i spróbuje wykonać w przypadku *liczba= 11* polecenie *liczba=*. Jeśli wartość zmiennej ma zawierać w sobie spacje, to można ująć wartość w cudzysłów lub poprzedzić każde wystąpienie spacji znakiem \.

```
1 #!/bin/bash
2 liczba=11 # poprawna definicja
3 liczba = 11 # niepoprawna definicja
4 liczba =11 # niepoprawna definicja
5 liczba= 11 # niepoprawna definicja
6
7 # poprawna definicja zmiennej ze spacjami
8 ciag_znakow=" aa bb"
9 # poprawna definicja zmiennej ze spacjami
10 ciag_znakow=\ aa\ bb
```

Skrypt 4.6. Poprawna i niepoprawne definicje zmiennych

Zmienne w skryptach BASH nie mają określonego typu. Powłoka przypisuje typ zmiennej na podstawie przypisywanej wartości. Jeśli zmiennej *liczba* przypisano by w kolejnej linii wartość *jakis tekst*, to jej typ zmieniłby się na ciąg znaków.

Nazwy zmiennych mogą składać się z liter z alfabetu łacińskiego, znaku podkreślenia _ oraz cyfr, pod warunkiem, że nie są one pierwszym znakiem w nazwie zmiennej (np. nazwa *12zmienna* jest nieprawidłowa). Ponadto wielkość liter w nazwie zmiennych ma znaczenie, czyli zmienne *liczba* oraz *Liczba* to dwie różne zmienne.

W celu pobrania wartości zmiennej, należy poprzedzić jej nazwę znakiem dolara $. Na przykład wartość 11 z powyższej zmiennej można pobrać za pomocą polecenia *$liczba*.

```
1 #!/bin/bash
2 #   definicja   zmiennej
3 STRING="Hello World"
4 INNY_STRING=Hello\ World
5 #wyswietlenie   wartosci   zmiennej
6 echo $STRING
7
8 #wyswietlenie   dlugosci   wartosci   zmiennej
9 echo ${#INNY_STRING}
```

Skrypt 4.7. Pobranie i wyświetlenie wartości zmiennej

Problem może stwarzać konkatenacja (łączenie) stringów. Przyjrzyjmy się przykładowi 4.8. Przedstawiono w nim próbę konkatenacji stringów, która nie działa zgodnie z naszymi oczekiwaniami.

Konkatenacja polega na zapisaniu dwóch wartości jedna po drugiej bez żadnego znaku łączącego (np. *zmienna=qaabbb"*).

```
1 #!/bin/bash
2 moja_zmienna="tresc zmiennej moja_zmienna "
3 #teraz   chcemy   dopisac   do   zmiennej   tekst   "NOWA "
4 moja_zmienna=$moja_zmiennaNOWA
5
6 #   wartosc   zmiennej   moja_zmienna   jest   pustym   stringiem
7 echo $moja_zmienna
```

Skrypt 4.8. Konkatenacja stringów - Problem

Dlaczego więc konkatenacja przedstawiona na przykładzie 4.8 nie działa?

Dzieje się tak dlatego, że interpreter pobierając wartość dowolnej zmiennej, czyta jej nazwę tak długo, aż napotka znak, który nie może być włączony do nazwy zmiennej (np. znak $, ", spację, itp.). Stąd, na przedstawionym problemie interpreter próbuje przypisać do zmiennej *moja_zmienna* wartość zmiennej *moja_zmiennaNOWA*, która jest niezdefiniowana i jej wartość jest pustym stringiem.

```
1 #!/bin/bash
2 moja_zmienna="tresc zmiennej moja_zmienna "
3
4 #teraz   chcemy   dopisac   do   zmiennej   tekst   "NOWA "
5
```

```
6  # pierwsza wersja
7  moja_zmienna=${moja_zmienna}NOWA
8  echo $moja_zmienna
9
10 # druga wersja
11 moja_zmienna=$moja_zmienna"NOWA"
12 echo $moja_zmienna
13
14 # trzecia wersja
15 moja_zmienna="$moja_zmienna"NOWA
16 echo $moja_zmienna
```

Skrypt 4.9. Konkatenacja stringów - Rozwiązanie

Przykłady rozwiązań powyższego problemu zaprezentowano na 4.9. W celu ograniczenia nazwy zmiennej, której wartość chcemy pobrać, można jej identyfikator (bez znaku $) ująć w klamry { }.
Nie zmieniają one nazwy zmiennej, a jedynie ją ograniczają, gdyż identyfikator nie może zawierać znaku klamry. Ten sposób będzie również przydatny w przypadku pobierania wartości argumentów wywoływania skryptu.

Innym sposobem na rozwiązanie powyższego problemu skorzystanie ze znaku cudzysłowu. On również ogranicza nazwę zmiennej, przy czym należy w cudzysłowie zawrzeć również znak $. Tak jak pokazano na przykładzie 4.9 można z niego skorzystać a dwa różne sposoby.

4.3.1. Zmienne tablicowe

W skryptach BASH można również deklarować zmienne tablicowe.

```
1  #!/bin/bash
2
3  # Tablica z trzema elementami
4  TABLICA=( "Pierwsza wartosc" Druga Trzecia )
5
6  # Pobranie wartosci n-tego elementu tablicy
7  echo ${TABLICA[n]}
8  # na przyklad
9  echo ${TABLICA[11]}
10
11 # Pobranie wartosci pierwszego elementu tablicy
12 echo ${TABLICA[0]}
13 # lub
14 echo $TABLICA
15
16 # Pobranie wszystkich elementow polaczonych w jeden
       string
17 echo ${TABLICA[@]}
18
19 # Pobranie ilosci wszystkich elementow tablicy
20 echo ${# TABLICA [@]}
```

Skrypt 4.10. Zmienne tablicowe

Tak jak widać na powyższym przykładzie, zmienna tablicowa (bez indeksu) wskazuje na wartość kryjącą się pod indexem numer 0. Ponadto, index oznacza wszystkie indeksy tablicy i może zostać wykorzystany do konkatenacji wszystkich wartości w tablicy, bądź do zliczenia jej elementów.

4.3.2. Argumenty wywołania skryptu

Do skryptów BASH, tak jak do poleceń systemowych można przekazywać wartości podczas ich wywoływania. Na przykład, gdy chcemy podejrzeć treść pliku **plik_tekstowy**, to wywołujemy polecenie *cat plik_tekstowy*. W tym przypadku wartość *plik_tekstowy* jest argumentem wywołania skryptu, czyli wartością przekazaną do skryptu, z której można korzystać.

```
1 #!/bin/bash
2 echo Witaj $1 $2
```

Skrypt 4.11. Wykorzystanie argumentów wywołania skryptu

Spójrzmy na przykład 4.11. Wykorzystano w nim dwa argumenty wywołania skryptu. Ich wartości pobiera się ze zmiennych **$n**, gdzie n określa numer porządkowy argumentu. Wynik wywołania skryptu zaprezentowano na rysunku 4.5.

```
user@host:~# ./witaj Jan Kowalski
Witaj Jan Kowalski
```

Rysunek 4.5. Uruchomienie skryptu z arumentami wywołania

Za pomocą polecenia **$n** można dostać się do dziewięciu (1-9) pierwszych argumentów wywołania skryptu. W przypadku, gdy do skryptu przekazywana jest większa ilość argumentów, można się do nich odwołać wykorzystując wcześniej omówiony sposób z klamrami, np. *$11* to wartość 11. argumentu.

W skryptach BASH można wykorzystać również następujące zmienne specjalne:

— **$0** - nazwa uruchomionego skryptu;
— **$@** - zwraca wszystkie przekazane argumenty połączone w jeden string;
— **$?** - kod powrotu ostatniego polecenia;
— **$$** - PID procesu;
— **$#** - ilość przekazanych argumentów.

4.4. Instrukcje i wyrażenia

Tak jak w innych językach programowania, również w BASHu nie może zabraknąć wyrażeń (logicznych, arytmetycznych) oraz instrukcji sterujących. W następnych sekcjach poznamy ich

4.4.1. Wywołanie polecenia a pobranie jego wartości

Podczas tworzenia skryptu, korzystamy z innych poleceń. Na przykład skrypt, który szuka określonych plików i wyświetla ich treść, będzie najpewniej korzystał z komend *find* oraz *cat*. Na przykładzie 4.12 przedstawiono skrypt, który wyświetla treść wszystkich plików w aktualnym folderze.

```
1 #!/bin/bash
2
3 cat ./*
```

Skrypt 4.12. Przykład wywołania polecenia w skrypcie

Czasem jednak nie chcemy od razu wyświetlać treści plików na standardowe wyjście, tylko przechować ją do wykorzystania w dalszej części skryptu. Żeby to zrobić, wykorzystujemy znak grawis ’‘’ (ang. grave accent) , który nie występuje w języku polskim, zaś w innych językach (np. francuski, portugalski) umieszcza się go nad literką, żeby zaznaczyć akcent samogłoski krótkiej o intonacji opadającej. Znak ten na klawiaturze zwykle występuje pod znakiem tyldy ’ ’.

Na przykładzie 4.13 przedstawiono wykorzystanie znaku grawis.

```
1 #!/bin/bash
2
3 #Zapisujemy tresc plikow do zmiennej
4 TRESC=‘cat ./*‘
5 #W tym momencie na ekranie nic sie nie pojawilo
6
7 #Tutaj mozemy wykonac operacje na tresci
8 # plikow ze zmiennej TRESC
9 TRESC=‘echo $TRESC | sed ’s/wyrazenie/zastapienie/’‘
10
11 #A na koncu wyswietlamy zawartosc
12 echo $TRESC
```

Skrypt 4.13. Przykład wywołania polecenia w skrypcie

Skrypty, oprócz zwracania danych na standardowe wyjście, mogą również zwracać pewną wartość - kod powrotu (ang. exit status) - w momencie, gdy kończą działanie. Służy do tego polecenie *exit*, którego składnia została przedstawiona na rysunku 4.6.

Komenda *exit* służy do zakończenia działania skryptu w dowolnym momencie. Jako opcjonalny parametr można przekazać status, który zostanie przekazany wyżej do skryptu lub powłoki, która wywołała ten skrypt. Statusem może być dowolna

```
exit [status]
```

Rysunek 4.6. Składnia polecenia *exit*

liczba naturalna jednobajtowa. W przypadku, gdy status nie zostanie określony, to przyjmuje wartość 0.

Najczęściej statusów używa się do zwracania kodu błędu, jaki wystąpił w skrypcie. Stąd, wartość 0 oznacza, że wszystko przebiegło pomyślnie i nie wystąpił żaden błąd. W przeciwnym razie, gdy wystąpił błąd, skrypt wywołujący komendę może na podstawie jego wartości podjąć odpowiednie kroki.

Teraz pojawia się pytanie, jak pobrać kod powrotu zwracany przez skrypt?

W poprzedniej sekcji opisywaliśmy zmienne specjalne, a wśród nich zmienną **$?**, która zwraca kod powrotu ostatniego polecenia. To właśnie za pomocą tej zmiennej będziemy sprawdzać, jaki był status wywoływanych poleceń. Przykładowe sprawdzenie działania zmiennej przedstawiono na 4.14.

```
1  #----------   plik   exit0   ------
2  #!/bin/bash
3  exit
4  #------------------------------
5
6  #----------   plik   exit1   ------
7  #!/bin/bash
8  exit 1
9  #------------------------------
10
11
12 #----------   plik   test   ------
13 #!/bin/bash
14
15 ./exit0
16 ./exit1
17
18 #Status z ostatniego polecenia, czyli exit1
19 echo $? # wyswietli 1
20
21 ./exit0
22 echo $? # wyswietli 0
23
24 zm0=`./exit0`
25 echo $zm0 # wyswietli pusty string
26 zm1=`./exit1`
27 echo $? # wyswietli 1
28
29 #------------------------------
```

Skrypt 4.14. Wykorzystanie zmiennej **$?**

W przypadku zmiennej **$?** brane są pod uwagę wszystkie polecenia, również te otoczone znakiem grawis.

4.4.2. Pobieranie danych od użytkownika

Pobieranie danych od użytkownika i wykorzystywanie ich w skrypcie wprowadza do niego interaktywność.
Zwykle dane, które pochodzą od użytkownika, wprowadza się do skryptu za pomocą argumentów wywołania, aczkolwiek są sytuacje, w których skrypt prowadzi dialog z użytkownikiem.

Częstym przypadkiem interakcji z użytkownikiem jest potwierdzenie wykonania operacji. Skrypt wyświetla użytkownikowi opis operacji, którą chce wykonać i prosi o wpisanie na przykład literki **t** od tak lub **y** od yes, żeby mógł wykonać tę operację. Każdy inny znak anuluje wykonanie operacji.

W celu pobrania danych od użytkownika w czasie wykonywania skryptu należy skorzystać z polecenia *read* (4.7).

```
read nazwa_zmiennej
```

Rysunek 4.7. Składnia polecenia *read*

Polecenie *read* wczytuje do zmiennej przekazanej jako argument treść linii wpisanej przez użytkownika aż do napotkania znaku nowej linii.

Można również jednym poleceniem *read* wczytać wartości do wielu zmiennych (4.15).

```
1 #!/bin/bash
2
3 echo "Podaj cztery wyrazy:"
4 read a b c d
5
6 echo "Pierwszy wyraz to: "$a
7 echo "Drugi wyraz to: "$b
8 echo "Trzeci wyraz to: "$c
9 echo "Czwarty wyraz to: "$d
```

Skrypt 4.15. Wczytywanie wielu zmiennych

Komenda *read* staje się bardzo użyteczna, gdy wykorzystujemy ją wspólnie z instrukcjami sterującymi, które omówimy w następnych sekcjach. Omówimy wtedy również przykłady wykorzystania polecenia *read*, w szczególności przykład z potwierdzeniem operacji.

4.4.3. Wyrażenia arytmetyczne

W skryptach BASH istnieje specjalny mechanizm do obliczania wyrażeń arytmetycznych, które operują na zmiennych całkowitych.

Mechanizm obliczania wyrażeń arytmetycznych nie przeprowadza kontroli przepełnienia. Spróbuj napisać skrypt, który zwraca ujemny kod powrotu i wyświetlić go w innym skrypcie. Będzie to liczba naturalna z przedziału 0-255.

W celu zrozumienia działania oraz potrzeby wprowadzenia mechanizmu do obliczania wyrażeń arytmetycznych spójrzmy na przykład 4.16.

```
1  #!/bin/bash
2
3  NUMER=1
4  # Niepoprawne zwiekszenie licznika
5  NUMER=$NUMER+1
6  echo $NUMER # Wyswietli 1+1, czyli konkatenacje stringow
7
8  NUMER=1
9  # Poprawne zwiekszenie licznika
10
11 # Pierwsza wersja
12 NUMER=$((NUMER+1))
13 echo $NUMER # Wyswietli 2
14
15 # Druga wersja
16 NUMER=$[NUMER+1]
17 echo $NUMER # Wyswietli 3
18
19 # Sposob na zwiekszenie wartosci o 1
20 TMP=$((NUMER++))
21 # lub
22 echo $((NUMER++))
23 # Teraz $NUMER zostal zwiekszony ,
24 # mimo ze nie przypisalismy go tej zmiennej zadnej
     wartosci
25 echo $NUMER # Wyswietli 4
26 # Analogicznie mozna skorzystac z -- do zmniejszenia
27 # wartosci o 1
```

Skrypt 4.16. Obliczanie wyrażeń arytmetycznych

Pierwsza próba zwiększenia licznika nie powiodła się, gdyż BASH domyślnie nie traktuje znaków +,-,*, itp. jako działania arytmetyczne, tylko jako zwykłe znaki. Dlatego w pierwszym przypadku BASH połączył wartość zmiennej **NUMER** z ciągiem znaków *+1* i zapisał co zmiennej **NUMER** nowy ciąg znaków *1+1*.

Dlatego wprowadzono **$(( ))** oraz **$[ ]**, w których umieszcza się wyrażenie. BASH widząc wyrażenie zawarte w tych znakach traktuje je, jako wyrażenie arytmetyczne i oblicza według reguł arytmetycznych.

Oprócz wyżej wymienionych znaków można skorzystać z polecenia *let*. Na przykład *let NUMER=NUMER+1*.

4.4.4. Wyrażenia logiczne

Innym, równie ważnym rodzajem wyrażeń, są wyrażenia logiczne. Są one wykorzystywane przede wszystkim w instrukcjach sterujących, które wykonują różne polecenia zależnie od wyniku wyrażenia logicznego.

Wyrażenia logiczne oblicza się za pomocą polecenia *test* lub odpowiednika z nawiasami kwadratowymi. Przykłady wyrażeń logicznych zaprezentowano w skrypcie 4.17.

```
1  #!/bin/bash
2
3  NUMER=1
4
5  # Wykorzystanie polecenia test
6  test $NUMER=1
7  test -x skrypt
8
9  # lub odpowiedniki nawiasowe
10 # Istotne sa spacje za pierwszym i przed drugim
       nawiasem
11 [ $NUMER = 1 ]
12 [ -x skrypt ]
```

Skrypt 4.17. Obliczanie wyrażeń logicznych

Polecenie test może wykorzystywać dwa rodzaje operatorów: unarne i binarne. Operatory unarne działają na jednym operandzie (np. **-x**), zaś binarne wymagają dwóch operandów (np. =). Wszystkie operatory zostały wymienione w tabeli 4.1.

4.4.5. Instrukcje sterujące

Omawiając wyrażenia arytmetyczne, a szczególnie logiczne, wspominaliśmy o tym, że są one przydatne do sterowania przebiegiem wykonywania skryptu. Oczywiście same wyrażenia nie są w stanie sterować przebiegiem, a są jedynie wykorzystywane przez instrukcje sterujące, które zostaną omówione w tej sekcji.

4.4.5.1. Instrukcja *if*

Instrukcja warunkowa *if* sprawdza, czy warunek jest prawdziwy i w tej sytuacji wykonuje zestaw instrukcji znajdujący się po słowie kluczowym *then*, zaś w przeciwnym przypadku wykonuje zestaw instrukcji znajdujący się za słowem kluczowym *else*. Składnię instrukcji warunkowej *if* przedstawiono w skrypcie 4.18.

```
1  #!/bin/bash
2
3  # Pierwsza wersja
4  if warunek
5  then
6    instrukcje
7  else
8    instrukcje
9  fi
```

-a plik	Sprawdza, czy podany plik istnieje.
-b plik	Sprawdza, czy plik istnieje i jest blokowym plikiem specjalnym.
-c plik	Sprawdza, czy plik istnieje i jest plikiem znakowym.
-e plik	Sprawdza, czy plik istnieje.
-h plik	Sprawdza, czy plik istnieje i jest linkiem symbolicznym.
-f plik	Sprawdza, czy plik istnieje i jest plikiem zwykłym.
-p plik	Sprawdza, czy plik jest łączem nazwanym.
-N plik	Sprawdza, czy plik istnieje i był zmieniany od czasu jego ostatniego odczytu.
-d plik	Sprawdza, czy plik istnieje i jest katalogiem.
-r plik	Sprawdza, czy użytkownik, który uruchamia skrypt, może czytać plik.
-w plik	Sprawdza, czy użytkownik, który uruchamia skrypt, może zapisywać do pliku.
-x plik	Sprawdza, czy użytkownik, który uruchamia skrypt, może wykonać plik.
plik1 -nt plik2	Sprawdza, czy plik1 jest nowszy od pliku2.
plik1 -ot plik2	Sprawdza, czy plik1 jest starszy od pliku2.
wyr1 = wyr2	Sprawdza, czy wyrażenia są równe.
wyr1 != wyr2	Sprawdza, czy wyrażenia są różne.
-n wyr	Sprawdza, czy wyrażenie ma długość większą niż 0.
-z wyr	Sprawdza, czy wyrażenie ma zerową długość.
wyr1 -lt wyr2	Sprawdza, czy wyrażenie 1 jest mniejsze niż wyrażenie 2.
wyr1 -gt wyr2	Sprawdza, czy wyrażenie 1 jest większe niż wyrażenie 2.
wyr1 -ge wyr2	Sprawdza, czy wyrażenie 1 jest równe lub większe niż wyrażenie 2.
wyr1 -le wyr2	Sprawdza, czy wyrażenie 1 jest równe lub mniejsze niż wyrażenie 2.

Tablica 4.1. Tabela operatorów

```
10
11 # Wersja bez konstrukcji else
12 if warunek
13 then
14   instrukcje
15 fi
16
17 # Wersja z if i then w jednej linii
18 if warunek; then
19   instrukcje
20 fi
21
22 # Wersja w jednej linii
23 if warunek; then instrukcje; else instrukcje; fi
24
25 # Uzycie elif
26 if warunek; then
27   instukcje
28 elif warunek; then
29   instrukcje
30 elif warunek; then
31   instrukcje
32 else
33   instrukcje
34 fi
```

Skrypt 4.18. Składnia instrukcji *if*

Na przykładzie 4.18 widać kilka różnych wersji instrukcji *if*. Pierwsza z nich zawiera wszystkie słowa kluczowe w nowych liniach, a także instrukcje wykonywane, gdy warunek jest prawdziwy oraz instrukcje do wykonania, gdy warunek jest fałszywy.

Kolejne wersje to modyfikacje pierwszej. W drugiej wersji nie ma instrukcji, które wykonywane są, gdy warunek jest fałszywy, czyli jednym słowem - części *else*, która jest opcjonalna.

Kolejne dwie wersje zawierają kilka słów kluczowych w jednej linii. W tej sytuacji, przed każdym kolejnym słowem kluczowym trzeba umieścić znak średnika ';'. Na przykładzie są one umieszczone po warunku w wersji trzeciej oraz po warunku i instrukcjach w wersji czwartej.

Dodatkową konstrukcją, z której skorzystano na ostatnim przykładzie, jest *elif*, która jest odpowiednikiem *else if*. Wprowadza ona do jednej instrukcji *if* sprawdzanie wielu warunków, czyli możliwość wyboru odpowiedniego wariantu przebiegu (zestawu instrukcji) w zależności od wyniku wielu warunków logicznych.

```
1 #!/bin/bash
2
3 if [ -x ./skrypt ]
4 then
5   ./skrypt
6 else
```

```
7   echo "Permission denied"
8 fi
```

Skrypt 4.19. Przykład użycia wyrażenia logicznego w instrukcji *if*

W instrukcji sterującej *if* możemy zaprezentować przykładowe wykorzystanie wyrażeń logicznych. Spójrzmy na przykład 4.19. Instrukcja warunkowa sprawdza, czy istnieją prawa do wykonania pliku **skrypt**, który znajduje się w aktywnym katalogu. Jeśli prawa istnieją, to wykonuje plik, a jeśli nie to wyświetla informację o braku uprawnień.

4.4.5.2. Pętle *while* i *until*

Istotnymi instrukcjami sterującymi są pętle. Ich zadaniem jest wielokrotne wykonanie pewnego zestawu instrukcji. Ilość obrotów pętli zależy od jej warunku.

Pętla *while* wykonuje polecenia, dopóki jej warunek jest prawdziwy, zaś pętla *until* działa dokładnie odwrotnie, czyli wykonuje polecenia, dopóki warunek jest fałszywy. Składnia obu pętli została przedstawiona na 4.20.

```
1 #!/bin/bash
2
3 while warunek
4 do
5   instrukcje
6 done
7
8 until warunek; do
9   instrukcje
10 done
```

Skrypt 4.20. Składnia pętli *while* i *until*.

Przykłady wykorzystania pętli przedstawiono na 4.21.

Oba przykłady działają tak samo, czyli wyświetlają 10 razy ciąg znaków **Odliczamy: x**, gdzie x to liczba od 10 do 0. Różnica polega na konstrukcji warunku. Pętla *while* wykonuje instrukcję tak długo, jak wartość zmiennej **NUMER** większe lub równa 0. Z drugiej strony pętla *until* wykonuje instrukcje aż do momentu, kiedy wartość zmiennej **NUMER** będzie mniejsza lub równa 0.

```
1 #!/bin/bash
2
3 NUMER=10;
4 while [ $NUMER -ge 0 ]; do
5   echo "Odliczamy: $NUMER"
6   NUMER=$((NUMER-1))
7 done
8
9 NUMER=10;
10 until [ $NUMER -lt 0 ]; do
11   echo "Odliczamy: $NUMER"
12   NUMER=$((NUMER-1))
```

```
13 done
```

Skrypt 4.21. Przykłady wykorzystania pętli *while* i *until*.

4.4.5.3. Pętla *for*

Pętla *for* posiada dwie postacie. Pierwsza z nich operuje na wszystkich elementach przekazanej listy, zaś druga ma formę zbliżoną do pętli *for* z języka C.

Składnia pierwszej formy pętli *for* została przedstawiona na 4.22.

```
1 #!/bin/bash
2
3 for ZMIENNA in LISTA; do
4     # Tutaj jest dostepna zmienna ZMIENNA,
5     # ktora przechowuje kolejne wartosci z listy LISTA
6   instrukcje;
7 done
```

Skrypt 4.22. Składnia pętli *for* dla listy

Lista wartości jest listą argumentów, która ma składnię analogiczną do argumentów wywołania skryptu. Są to kolejne wartości oddzielone znakiem spacji. Lista może być również zbudowana z elementów tablicy. Na przykładzie 4.23 przedstawiono kilka sposobów przekazywania listy do pętli *for*.

```
1 #!/bin/bash
2
3 # Przekazywanie bezposrednie
4 for ZMIENNA in jeden dwa trzy cztery; do
5   echo $ZMIENNA
6 done
7
8 # Przekazywanie przez zmienna
9 LISTA="a b c d"
10 for ZMIENNA in $LISTA; do
11   echo $ZMIENNA
12 done
13
14 # Przekazywanie przez tablice
15 TABLICA=( a b c d )
16 for ZMIENNA in ${TABLICA[@]}; do
17   echo $ZMIENNA
18 done
19
20 # Przekazywanie plikow (pliki z biezacego katalogu)
21 for ZMIENNA in *; do
22   echo $ZMIENNA
23 done
24
25 # Przekazywanie plikow (pliki HTML z biezacego
       katalogu)
```

```
26 for ZMIENNA in *.html; do
27   echo $ZMIENNA
28 done
29
30 # Przekazywanie wyniku polecenia
31 for ZMIENNA in `ls`; do
32   echo $ZMIENNA
33 done
34 # lub
35 for ZMIENNA in $(ls); do
36   echo $ZMIENNA
37 done
```

Skrypt 4.23. Różne rodzaje listy w pętli *for*

Druga postać pętli *for* została przedstawiona na 4.24. Instrukcja początkowa jest wykonywana tylko raz, przed uruchomieniem pętli. Następnie pętla jest wykonywana, dopóki spełniony jest warunek, a po każdym obrocie wykonywana jest instrukcja końcowa.

```
1 #!/bin/bash
2
3 for ((instr. poczatkowa; warunek; instr. koncowa)); do
4   instrukcje;
5 done
```

Skrypt 4.24. Składnia złożonej pętli *for*

Kilka przykładów wykorzystania złożonej pętli *for* zostało przedstawionych na 4.25.

```
1 #!/bin/bash
2
3 for ((i=0; i<=10; i=$i+1)); do
4   echo "Wartosc zmiennej i="$i
5 done
6
7 i=0
8 for ((;i <= 10;)); do
9   echo "Wartosc zmiennej i="$i
10  i=$((i+1))
11 done
12
13 for ((i=0, j=0; i<=10 && j<=10; i=$i+1, j=$j+2)); do
14   echo "Wartosc zmiennej i="$i
15   echo "Wartosc zmiennej j="$j
16 done
```

Skrypt 4.25. Przykłady wykorzystania złożonej pętli *for*

4.4.5.4. Intrukcje *break* i *continue*

Omawiając pętle, warto przedstawić dwie instrukcje, które również sterują przebiegiem pętli. Pierwsza z nich to *break*, która kończy działanie pętli i przechodzi do następnego polecenia za pętlą. Druga to *continue*, która kończy aktualny obrót pętli i przechodzi na początek pętli, czyli do sprawdzenia warunku. Przykład zastosowania tych instrukcji przedstawiono na 4.26.

```
1  #!/bin/bash
2
3  for ((i=0; i<=10; i=$i+1)); do
4    echo "Wartosc zmiennej i="$i
5    if [ $i = 5 ]; then
6      break;
7    fi
8  done
9
10 for ((i=0; i<=10; i=$i+1)); do
11   if [ $(($i%2)) = 0 ]; then
12     continue;
13   fi
14   echo "Wartosc zmiennej i="$i
15 done
```

Skrypt 4.26. Przykłady wykorzystania instrukcji *break* oraz *continue*

4.4.5.5. Instrukcja *case*

Instrukcja *case* działa podobnie jak kaskada *ifów*, czyli instrukcja *if* z wieloma wariantami *elif*. Różnica polega na tym, że w *case* do wielu wariantów dopasowywane jest to samo wyrażenie.

Składnia instrukcji *case* jest przedstawiona na 4.27. Wzorce mają budowę analogiczną do wzorców plików, czyli może to być zarówno konkretna wartość, jak również maska zawierająca znaki * i ?. Jeśli instrukcja *case* dopasuje wartość wyrażenia do dowolnego wzorca, wykonane zostaną instrukcje przypisane do tego wzorca. W przeciwnym wypadku wykonane zostaną instrukcje domyślne, oznaczone symbolem gwiazdki *).

```
1  #!/bin/bash
2
3  case WYRAZENIE in
4    "WZORZEC1") intrukcje ;;
5    "WZORZEC2") intrukcje ;;
6    "WZORZEC3") intrukcje ;;
7    *) instrukcje domyslne
8  esac
```

Skrypt 4.27. Składnia instrukcji *case*

Przykład wykorzystania instrukcji *case* został przedstawiony na 4.34. Wyświetla on informacje o wszystkich plikach w bieżącym katalogu na podstawie ich rozszerzenia.

```
1  #!/bin/bash
2
3  for PLIK in *; do
4  echo "Plik: $PLIK"
5  case $PLIK in
6    *.txt) echo "Plik tekstowy" ;;
7    *.sh) echo "Skrypt BASH" ;;
8    *.html) echo "Plik HTML";;
9    *) echo "Inny plik"
10 esac
11 echo
12 done
```

Skrypt 4.28. Przykład wykorzystania instrukcji *case*

4.5. Przykłady z życia wzięte

W tej sekcji przedstawimy kilka skryptów, które mogą być przydatne dla użytkowników systemu Linux. Pomogą one również zrozumieć konstrukcje wyrażeń oraz instrukcji sterujących, a także ich wspólne wykorzystanie.

Skrypt 4.29. Wyświetlenie dowolnego przedziału linii w pliku

```
1  #!/bin/bash
2  sed -n $1,$2p $3
3
4  #!/bin/bash
5  ILE_LINII=$(($2-$1+1))
6  head -$2 $3 | tail -$ILE_LINII
```

Skrypt 4.30. Usunięcie plików niewykonywalnych z katalogu podanego jako parametr

```
1  #!/bin/bash
2  if [ $# -ne 1 ]
3  then
4    echo "Niepoprawna liczba argumentow. "
5    echo "Uzycie: $0 katalog"
6    exit 1
7  fi
8
9  if [ -d $1 ]
10 then
```

```
11    BIEZACY_KATALOG=`pwd`
12    cd $1
13    for plik in *
14    do
15      if [ -x $plik ]
16      then
17        continue
18      else
19        rm $plik -f
20      fi
21    done
22    cd $BIEZACY_KATALOG
23  else
24    echo "$1 nie jest katalogiem!"
25    exit 1
26  fi
```

Skrypt 4.31. Usunięcie pustych plików z katalogu podanego jako parametr oraz stworzenie listy usuniętych plikow

```
1  #!/bin/bash
2
3  if [ $# -ne 2 ]
4  then
5    echo "Niepoprawna liczba argumentow. "
6    echo "Uzycie: $0 katalog raport"
7    exit 1
8  fi
9
10 if [ ! -d $1 ]
11 then
12   echo "$1 nie jest katalogiem!"
13   exit 1
14 fi
15
16 BIEZACY_KATALOG=`pwd`
17 cd $1
18 for PLIK in *
19 do
20   if [[ ! -s $PLIK ]] && [[ -f $PLIK ]] && [[ ! -L $PLIK ]]
21   then
22     echo "Usunieto plik $PLIK." >> $BIEZACY_KATALOG/$2
23     rm $PLIK -f
24   fi
25 done
26 cd $BIEZACY_KATALOG
```

Skrypt 4.32. Skrypt wczytuje liczby z pliku i wypisuje ich maksimum, minimum oraz sumę

```
#!/bin/bash

if [ $# -lt 1 ]; then
  echo "Podaj nazwe pliku!"
  exit
fi

if [ ! -f "$1" ]; then
  echo "Plik $1 nie istnieje!"
  exit
fi

ILOSC_WIERSZY=`cat "$1" | wc -l`
if [ $ILOSC_WIERSZY -eq 0 ]; then
  echo "Plik $1 jest pusty!"
  exit
fi

MINIMUM=`sed -n '1,1p' $1`
MAXIMUM=$MINIMUM
SUMA=0

while [ $ILOSC_WIERSZY -gt 0 ]; do
  LICZBA=`sed -n $ILOSC_WIERSZY,${ILOSC_WIERSZY}p $1`

  SUMA=$[$SUMA+$LICZBA]

  if [ $LICZBA -gt $MAXIMUM ]; then
    MAXIMUM=$LICZBA
  elif [ $LICZBA -lt $MINIMUM ]; then
    MINIMUM=$LICZBA
  fi

  ILOSC_WIERSZY=$[$ILOSC_WIERSZY-1]
done

echo "Minimum: $MINIMUM"
echo "Maksimum: $MAXIMUM"
echo "Suma: $SUMA"
```

Skrypt 4.33. Skrypt oblicza liczbę Fibbonacciego, której numer jest podany jako parametr

```
#!/bin/bash

if [ $1 -lt 1 ]; then
  echo "Prosze podac numer wiekszy od zera!"
  exit
```

```
6  fi
7
8  if [ $1 -le 2 ]; then
9    echo 1
10   exit
11 fi
12
13 NR=$[$1-2]
14 P=1
15 PP=1
16
17 while [ $NR -gt 0 ]; do
18   TMP=$(($P+$PP))
19   P=$PP
20   PP=$TMP
21
22   NR=$(( $NR-1 ))
23 done
24
25 echo $TMP
```

Skrypt 4.34. Zmiana liter z dużych na małe w plikach znajdujących się w katalogu podanym w parametrze

```
1  #!/bin/bash
2
3  if [ ! -d $1 ]; then
4    echo "Plik $1 nie jest katalogiem"
5    exit 1
6  fi
7
8  BIEZACY_KATALOG=`pwd`
9
10 cd $1
11 for file in *; do
12     mv ${file} `echo ${file} | tr "A-Z" "a-z"`
13 done
14 cd $BIEZACY_KATALOG
```

Rozdział 5

Administracja serwerem WWW - Apache

Jednym z najważniejszych zadań dla administratora systemu Linux jest konfiguracja serwera WWW. Zadanie to dotyczy prawie każdego administratora, ponieważ praktycznie w każdej organizacji jest uruchomiony Web serwer. Poprawna konfiguracja serwera nie ogranicza się do otwarcia odpowiednich portów na zaporze ogniowej, należy dokładnie przeanalizować parametry wydajnościowe oraz przede wszystkim zagadnienia związane z jego bezpieczeństwem. Dalsza konfiguracja serwera będzie prezentowana dla systemu Linux, dystrybucji Ubuntu 11.04.

5.1. Instalacja serwera Apache

W pierwszym kroku należy zainstalować Web serwer. W systemach z pakietami typu *deb* oraz poprawnie skonfigurowaną aplikacją *apt* należy wykonać następującą komendę:

```
# apt-get install apache2
```

Rysunek 5.1. Instalacja apache2

Weryfikację zainstalowanej wersji serwera można wykonać przy pomocy komendy:

```
# apt-cache search apache2
```

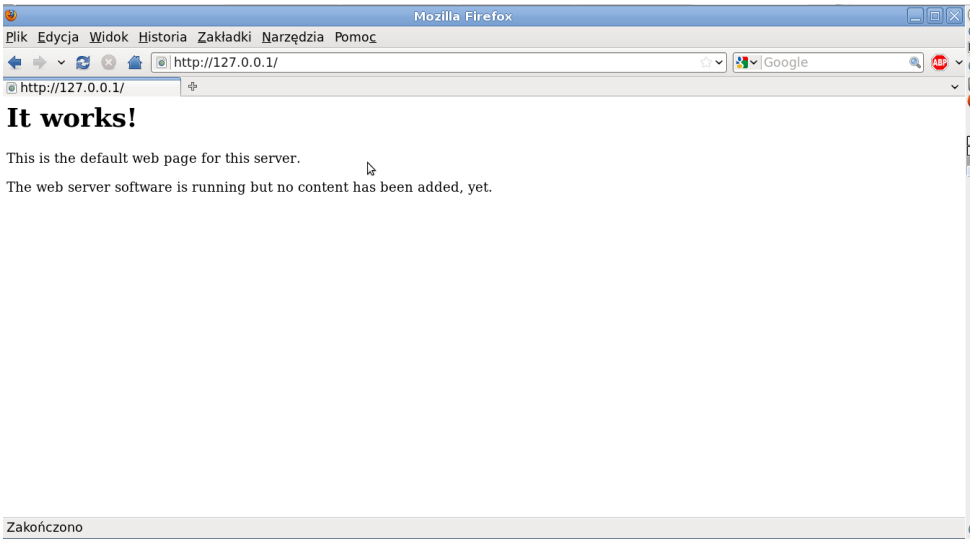
Rysunek 5.2. Weryfikacja zainstalowania pakietu apache2

lub

```
# dpkg --list | grep apache2
```

Rysunek 5.3. Weryfikacja zainstalowania pakietu apache2 - dpkg

Z zainstalowaniem serwera apache wiąże się utworzenie szeregu plików i katalogów (które utworzone są automatycznie w procesie instalacji). W tabeli 5.1 przedstawione są podstawowe pliki oraz katalogi. Standardowe ustawienie serwera apache powoduje, że po wykonaniu komend instalacyjnych serwer automatycznie jest uruchamiany i umożliwiony jest dostęp do niego przez lokalny interfejs. Na rysunku 5.4 przedstawiona jest strona startowa serwera apache2.



Rysunek 5.4. Strona startowa serwera apache2

Tablica 5.1. Pliki oraz katalogi powiązane z serwerem Apache

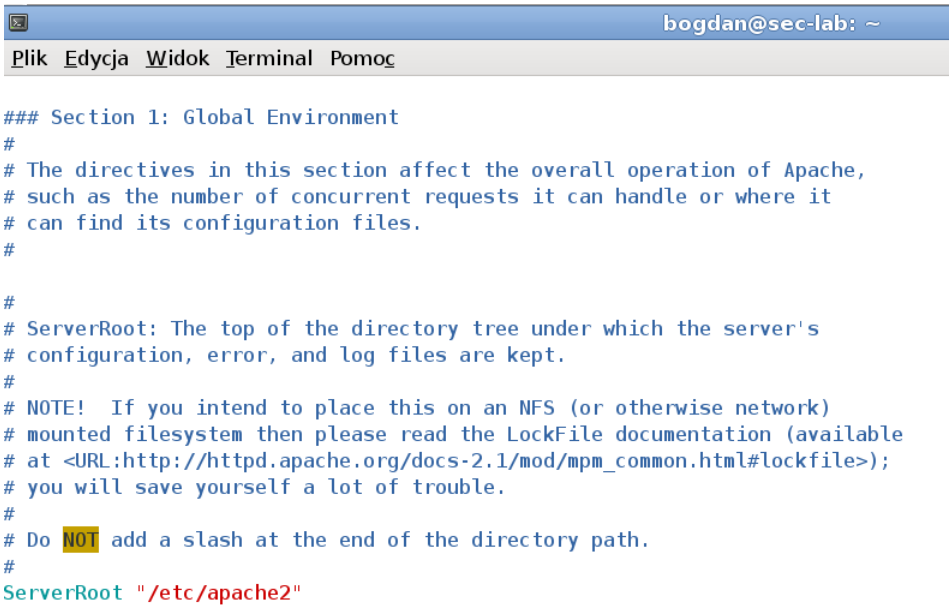
Nazwa Pliku/Katalogu	**Opis**
/etc/apach2	katalog zawierający pliki konfiguracyjne serwera
/usr/sbin/apache2	plik binarny serwera
/etc/apache2/apache2.conf	główny plik konfiguracyjny serwera
/etc/apache2/mods-available	dostępne moduły serwera
/etc/apache2/mods-enabled	włączone moduły serwera jako linki symboliczne do dostępnych modułów
/etc/apache2/conf.d	katalog zawierający pliki konfiguracyjne serwera przeznaczony do dodatkowej konfiguracji
/etc/apache2/sites-available	dostępne dodatkowe konfiguracje systemu, początkowo zawierające konfiguracje wirtualnych hostów
/etc/apache2/sites-enabled	włączone dodatkowe konfiguracje systemu jako linki symboliczne do plików zawartych w katalogu *sites-available*
/var/run/apache2.pid	numer id procesu
/var/log/apache2	zapisy kontrolne serwera apache (logi)
/var/www/	katalog stron html
/etc/apache2/envvar	zmienne środowiskowe serwera

5.2. Główny plik konfiguracyjny - apache2.conf

Podstawowa konfiguracja Web serwera sprowadza się do edycji zmiennych określonych w pliku *apache2.conf*. W dalszej części zostaną omówione najbardziej istotne parametry konfiguracyjne

5.2.1. ServerRoot

Jest to zmienna określająca katalog zawierający plik konfiguracyjne Web serwera 5.5 . Standardowo jest to */etc/apache2*. Jeżeli katalog ten zostanie zmieniony wówczas należy przekopiować wszystkie plik i katalogi z katalogu domyślnego.



Rysunek 5.5. Zmienna ServerRoot

5.2.2. MPM - Moduł przetwarzania równoległego

Apache HTTP Server ma być wydajny i elastyczny, który może pracować na bardzo wielu platformach w wielu różnych środowiskach. Apache zbudowany jest modułowo. Konstrukcja ta umożliwia administratorowi wybór tych funkcji, które zostaną włączone do serwera. Wybór ten należy dokonać poprzez wybranie modułów apacha podczas jego kompilacji lub w czasie wykonywania. Jednym z modułów jest moduł MPM (ang. Multi-Processing Modules), który jest odpowiedzialny za równoległe przetwarzanie zapytań. W ramach tego głównego modułu można wyróżnić jego trzy odmiany: `prefork, worker, event`. `Prefork` odwołuje się do serwera WWW, który obsługuje żądania w sposób podobny do Apache 1.3. Jest to najlepszy MPM służący do izolowania żądań do serwera, dzięki czemu każde

realizowane żądanie nie wpływa jakiekolwiek inne. `Worker` realizuje serwera jako proces multi-hybrydowy oraz wielowątkowy. Dzięki użyciu wątków do realizowania żądań, serwer jest w stanie obsłużyć dużą liczbę żądań przy mniejszym zużyciu zasobów systemowych niż w przypadku ich realizacji, bazując wyłącznie na procesach. `Event` ma na celu umożliwienie realizacji większej ilości żądań w sposób jednoczesny. Wykonywane to jest przy pomocy wątków pomocniczych, które wykonuję część obliczeń za wątki, które przekazały do nich zadanie. Taki podział zadań pozwala na obsługę nowych żadań zwalniając główne wątki.

Każda z nich jest definiowana przy pomocy poniżej opisanych parametrów oraz przedstawionych na rysunku 5.6:

— **StartServers** - parametr ustawia liczbę procesów potomnych tworzonych w czasie uruchamiania; liczba procesów jest dynamicznie kontrolowana w zależności od obciążenia, z reguły nie ma powodów do zmiany tego parametru;
— **MinSpareThreads** - parametr wyznaczający minimalną liczbę bezczynnych procesów potomnych, które nie przetwarzają żądań;
— **MaxSpareThreads** - parametr wyznaczający maksymalną liczbę bezczynnych procesów potomnych, które nie przetwarzają żądań.
— **ThreadLimit** - parametr ten określa maksymalną skonfigurowaną wartość parametru *ThreadsPerChild* na czas trwania procesu Apache;
— **ThreadsPerChild** - parametr ten określa liczbę wątków tworzonych przez każdy proces potomny; wątki te tworzone są na starcie i w innym czasie nie są tworzone nowe;
— **MaxClients** - parametr ten ustawia limit liczby jednoczesnych żądań, które będą obsłużone przez serwer; wszelkie próby połączenia ponad ustalony limit będą ustawiane w kolejce;
— **MaxRequestsPerChild** - parametr ten wyznacza limit na liczbę zapytań, które zostaną obsłużone przez proces potomny.

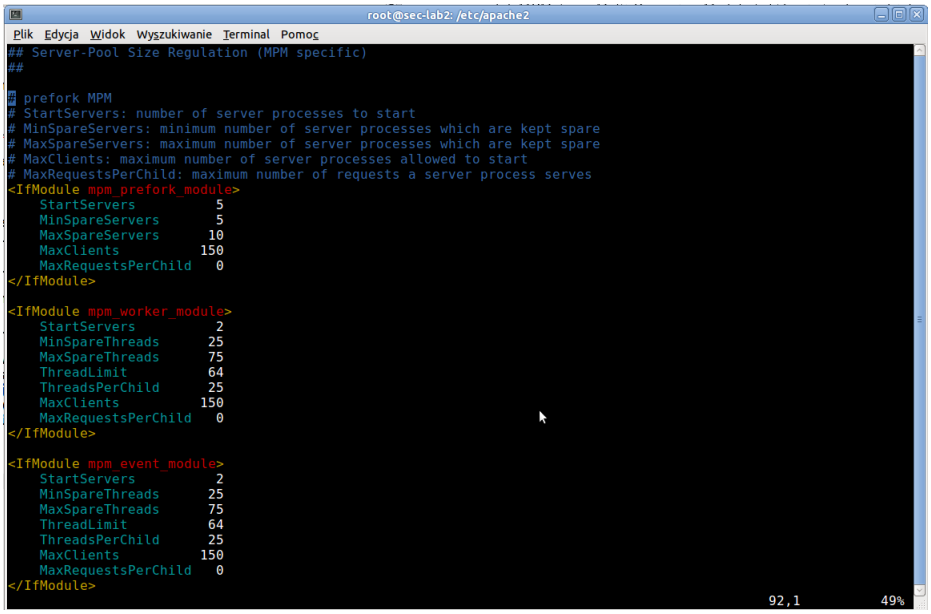
5.2.3. Nazwa użytkownika oraz grupy

Kolejnym parametrem określanym w głównym pliku konfiguracyjnym jest nazwa użytkownika oraz grupy powiązaną z serwerem Apache (Rys.5.7):

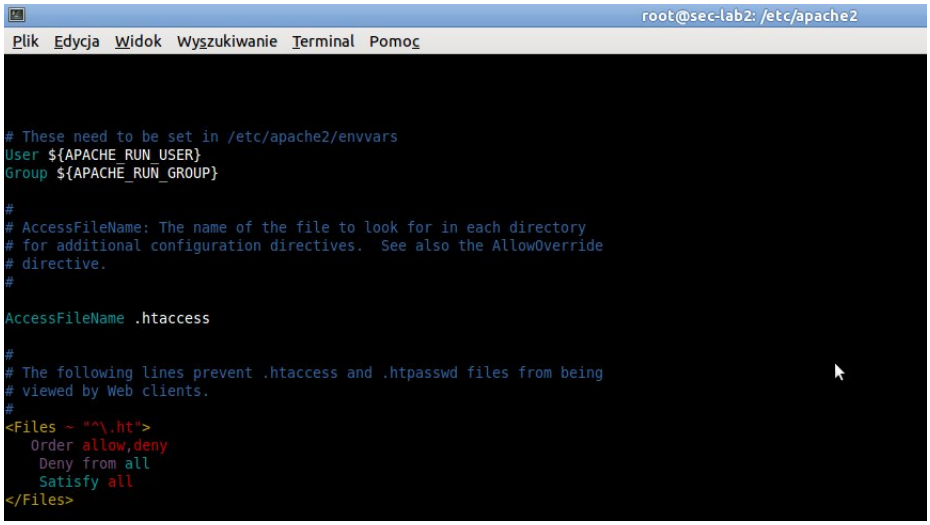
— **User ${APACHE_RUN_USER}** - nazwa użytkownika;
— **Group ${APACHE_RUN_GROUP}** - nazwa grupy.

W wersji apache2, parametry te są określane w pliku `/etc/apache2/envvars` (Rys.5.8), gdzie znajdują się wszelkie zmienne środowiskowe powiązane z serwerem:

— **export APACHE_RUN_USER=www-data** - przypisanie wartości (www-data) do zmiennej określającej nazwę użytkownika reprezentującego serwer apache oraz eksport danych;
— **export APACHE_RUN_GROUP=www-data** - przypisanie wartości (www-data) do zmiennej określającej nazwę użytkownika reprezentującego serwer apache.



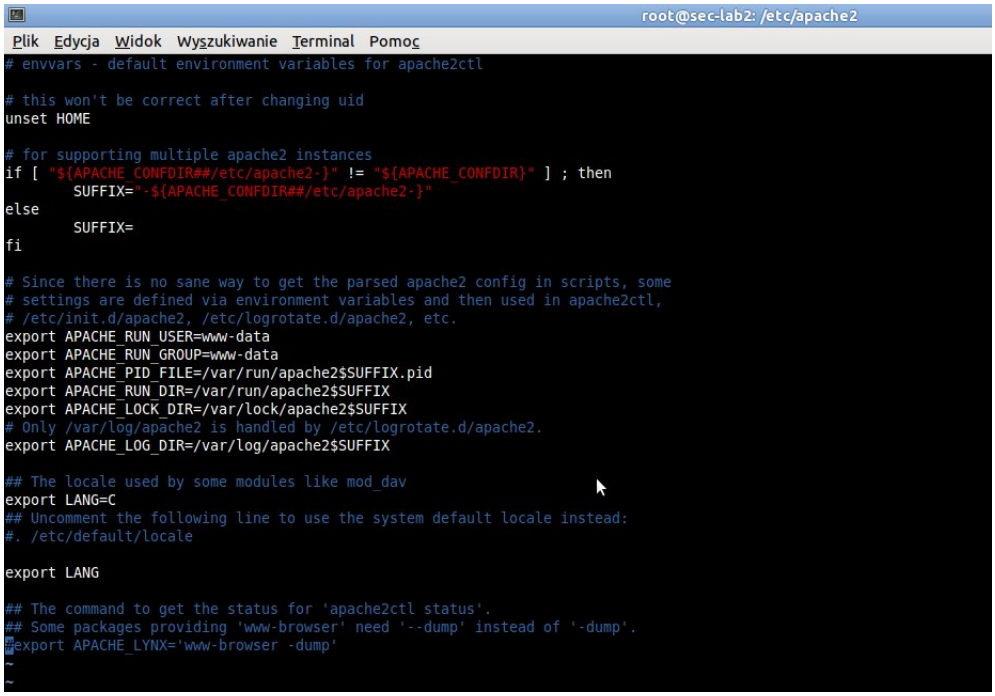
Rysunek 5.6. Zminne do modułów MPM



Rysunek 5.7. Zmienne serwera apache2

5.2.4. Plik .htaccess

Plik `.htaccess` umożliwiają dokonywanie zmian konfiguracyjnych dla poszczególnych katalogów. Plik zawierający jeden lub więcej dyrektyw konfiguracyjnych jest umieszczony w określonym katalogu dokumentów, wtedy dyrektywy mają zastosowanie do tego katalogu i wszystkich jego podkatalogów. Przykładowa konfigu-



Rysunek 5.8. Plik konfiguracyjny envvars

racja tego pliku może dotyczyć utworzenia systemu autoryzacji dostępu do danego drzewa katalogu realizowanego przez serwer Apache. W tym celu należy utworzyć plik `.htaccess` i umieści w katalogu, do którego dostęp będzie poprzedzany autoryzacją przy pomocy nazwy użytkownika i hasła. Przykładowa budowa tego pliku przedstawiona jest niżej.

```
AuthType Basic
AuthName "Podaj haslo - strefa 2"
AuthUserFile ".htpas"
require valid-user
```

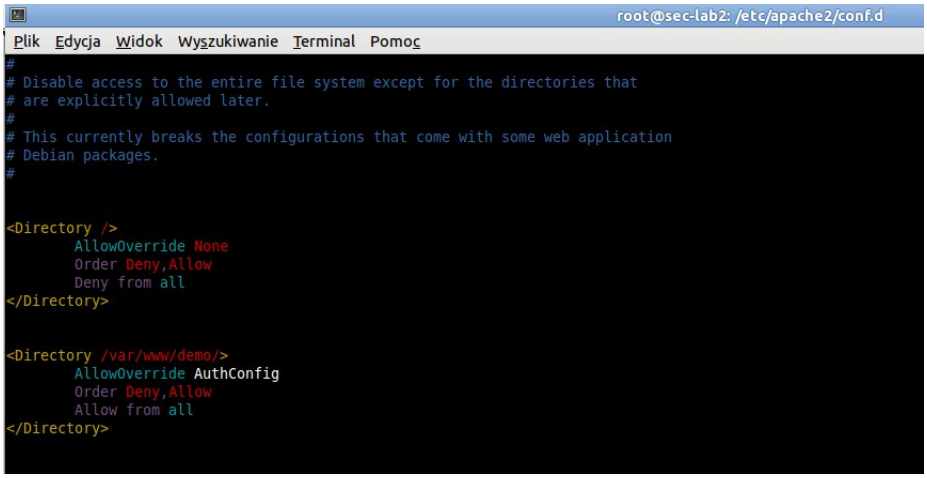
Rysunek 5.9. Plik konfiguracyjny .htaccess

Poszczególne zmienne określają:

— **AuthType** - rodzaj autoryzacji;
— **AuthName** - informacja, która będzie wyświetlana w oknie w którym należy podać nazwę użytkownika i hasło;
— **AuthUserFile** - nazwa pliku, w którym zawarte są dane dotyczące autoryzacji, czyli nazwa użytkownika oraz wynik funkcji jednokierunkowej z hasła;
— **require** - wybór użytkowników, którzy mogą uzyskać dostęp do zdefiniowanego zasobu.

Poprawna konfiguracja dostępu do zasobów kontrolowanych przez plik `.htaccess` wymaga modyfikacji jeszcze jednego pliku konfiguracyjnego - `conf.d/security`. W pliku tym należy określić konkretną politykę dostępu, co będzie wyjątkiem w skali całego systemu. Jeżeli polityka globalna jest zgodna z polityką, jaka wymagana jest w przypadku pliku `.htaccess`, wówczas nie trzeba definiować wyjątku. Na Rys. 5.10 przedstawiona jest przykładowa konfiguracja a poniżej opisane są poszczególne zmienne.

— **AllowOverride** - kiedy serwer znajdzie plik .htaccess (jak określono w AccessFileName) to musi wiedzieć, które dyrektywy zadeklarowane w tym pliku mogą unieważnić wcześniejsze dyrektywy konfiguracyjne;
— **Order** - kolejność interpretacji logicznej systemu kontroli dostępu;
— **Allow from** - parametr określający, które hosty mogą uzyskać dostęp do obszaru serwera; dostęp może być kontrolowany przez hosta, adres IP, zakres adresów IP lub innych cech charakterystycznych przechowywanych w zmiennych środowiskowych systemu.

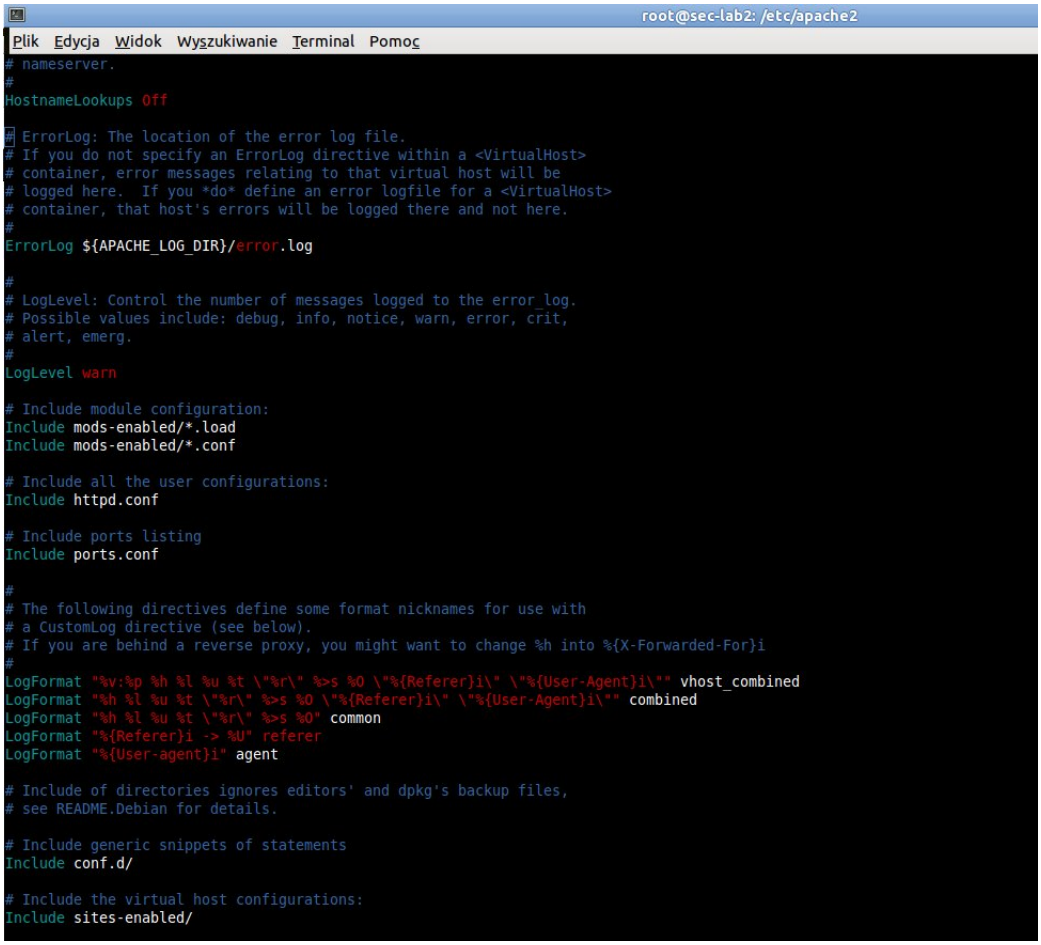


Rysunek 5.10. Plik konfiguracyjny conf.d/security

5.2.5. Zapisy kontrolne

Poprawne działanie serwera WWW jest monitorowane dzięki wykonywanym zapisom kontrolnym. Poziom szczegółowości jest określany w parametrze `LogLevel` (Rys.5.11). Parametr ten może przyjmować następujące wartości (uszeregowane od najmniej szczegółowego): `debug, info, notice, warn, error, crit, alert, emerg`.

Miejsce przechowywania dziennika zapisów kontrolnych określona jest w parametrze `ErrorLog`. W przedstawionym na Rys.5.11 przykładzie ścieżka określona jest w zmiennej systemowej `${APACHE_LOG_DIR}`, która określona jest w pliku `envvar` (Rys.5.8).



Rysunek 5.11. Plik konfiguracyjny apache2 - zapisy kontrolne, dyrektywa include

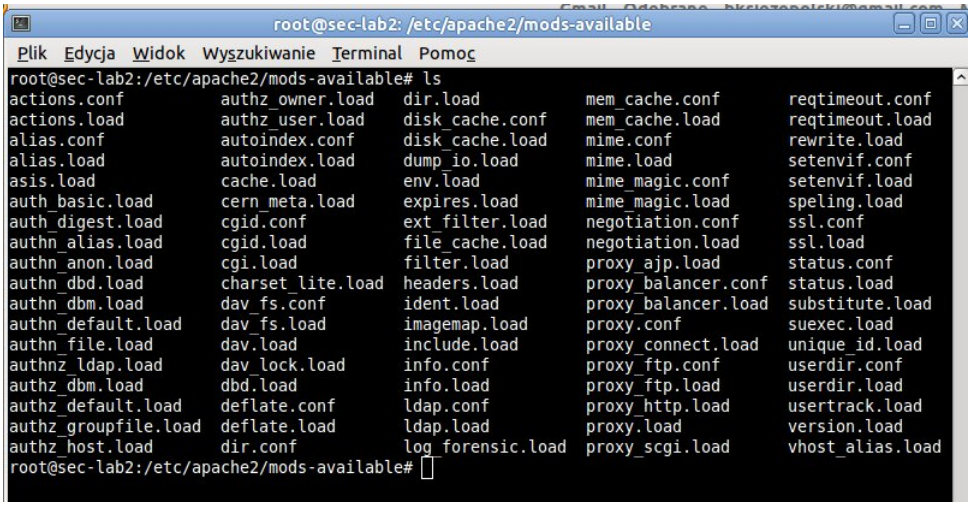
5.2.6. Dyrektywa `Include`

Głównym plikiem konfiguracyjnym serwera apache2 jest plik `apache2.conf`. Wszelka konfiguracja może być określona w tym pliku, ale dla przejrzystości zostały utworzone osobne pliki konfiguracyjne, scharakteryzowane ze względu na definiowane tam funkcjonalności. Ich charakterystyka podana jest w tabeli 5.1 a przykład ich definicji na Rys.5.11.

5.2.7. Zarządzanie modułami

Lista dostępnych modułów serwera apache2 określona jest w katalogu `mods-available` (Rys.5.12).

Lista aktualnie włączonych modułów zawarta jest w katalogu `mods-enabled`, zawiera ona linki symboliczne do modułów określonych w katalogu `mods-available`.



Rysunek 5.12. Lista dostępnych modułów

Przykładowe wpisy wyglądają następująco:

```
alias.conf -> ../mods-available/alias.conf
alias.load -> ../mods-available/alias.load
auth_basic.load -> ../mods-available/auth_basic.load
authn_file.load -> ../mods-available/authn_file.load
authz_default.load -> ../mods-available/authz_default.load
authz_groupfile.load -> ../mods-available/authz_groupfile.load
authz_host.load -> ../mods-available/authz_host.load
authz_user.load -> ../mods-available/authz_user.load
autoindex.conf -> ../mods-available/autoindex.conf
autoindex.load -> ../mods-available/autoindex.load
```

Rysunek 5.13. Lista dostępnych modułów

Można ją również wyświetlić przy pomocy komendy `apache2ctl -M`.

Dodatkowe moduły do pracującego serwera apache2 można włączyć przy pomocy polecenia `a2enmod`. Polecenie to w rzeczywistości tworzy dowiązanie symboliczne do plików znajdujących się w katalogu `mods-available`. Nazwy te są identyczne jak nazwy prawdziwych modułów. W rzeczywistości dopiero pliki te zawiera ścieżkę do modułów znajdujących się w systemie. Budowa tych plików przedstawiona jest poniżej.

```
root@sec-lab2:/etc/apache2/mods-available# apache2ctl -M
apache2: Could not reliably determine the server's fully \\
qualified domain name, using 127.0.1.1 for ServerName
Loaded Modules:
 core_module (static)
 log_config_module (static)
 logio_module (static)
 mpm_worker_module (static)
 http_module (static)
 so_module (static)
 alias_module (shared)
 auth_basic_module (shared)
 authn_file_module (shared)
 authz_default_module (shared)
 authz_groupfile_module (shared)
 authz_host_module (shared)
 authz_user_module (shared)
 autoindex_module (shared)
 Syntax OK
```

Rysunek 5.14. Inna metoda wyświetlania modułów apacha2

```
root@sec-lab2:/etc/apache2/mods-available# cat proxy.load
LoadModule proxy_module /usr/lib/apache2/modules/mod_proxy.so
```

Rysunek 5.15. Zawartość pliku wskazującego ścieżkę modułu apacha2

5.3. Zaawansowana konfiguracja

W dalszej części książki zostaną przedstawione zaawansowane zagadnienia związane z serwerem apache2. Z listy wielu ciekawych zagadnień zostały wybrane dwa: bezpieczeństwo oraz wirtualne hosty. Zagadnienie bezpieczeństwa serwera apache2 jest bardzo złożone i jego opis może być przedmiotem osobnej książki. Z tego powodu w ramach tej książki zostanie omówione jedno zagadnienie, mianowicie wsparcie protokołu TLS przez serwer apache2.

5.3.1. Wirtualne hosty

Wirtualne hosty są niezwykle przydane w przypadku serwerów WWW. Dzięki nim można np. udostępniać na jednym serwerze, o konkretnym adresie IP, kilka witryn WWW, gdzie każda z nich będzie dostępna pod inną domeną. Plikiem odpowiedzialnym za konfigurację wirtualnych hostów jest plik `/etc/apache2/sites-available/default`. Poniżej przedstawiona jest zawartość tego pliku.

```
root@sec-lab2:/etc/apache2/sites-available# cat default
<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        DocumentRoot /var/www
        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>
        <Directory /var/www/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>

        ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
        <Directory "/usr/lib/cgi-bin">
                AllowOverride None
                Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log

        # Possible values include: debug, info, notice, warn
        # alert, emerg.
        LogLevel warn

        CustomLog ${APACHE_LOG_DIR}/access.log combined

    Alias /doc/ "/usr/share/doc/"
    <Directory "/usr/share/doc/">
        Options Indexes MultiViews FollowSymLinks
        AllowOverride None
        Order deny,allow
        Deny from all
        Allow from 127.0.0.0/255.0.0.0 ::1/128
    </Directory>

</VirtualHost>
```

Rysunek 5.16. Plik konfiguracyjny wirtualnego hosta

Podstawowa konfiguracja wirtualnego hosta jest bardzo prosta, wystarczy dodać do pliku `/etc/apache2/sites-available/default` wpisy podane niżej.

```
<VirtualHost www.tls-demo.pl:80>
DocumentRoot /var/www/tls-demo/
ServerName www.tls-demo.pl
</VirtualHost>
```

Rysunek 5.17. Konfiguracja wirtualnego hosta

W tym przypadku serwer WWW będzie kierował wszelki ruch kierowany na port 80 i o adresie `www.tls-demo.pl` do katalogu głównego tego wirtualnego hosta. Katalog główny wirtualnego hosta jest określany jak wartość parametru `/var/www/tls-demo/`.

5.3.2. Bezpieczeństwo WWW - protokół TLS

Tematem tego podrozdziału jest konfiguracja serwera apache tak, żeby wspierał połączenia realizowane przy pomocy protokołu TLS. W pierwszym kroku należy włączyć moduł ssl, który jest odpowiedzialny za realizację połączeń (`a2enmod ssl`). Następnie należy zrestartować serwer apache2 (`/etc/init.d/apache2 restart`). Po tych czynnościach serwer apache zaczął nasłuchiwać na połączenia nie tylko na porcie 80, ale również na porcie 443 (Przykład poniżej).

```
root@sec-lab2:/etc/apache2# a2enmod ssl
Enabling module ssl.
See /usr/share/doc/apache2.2-common/README.Debian.gz on how to \\
configure SSL and create self-signed certificates.
Run '/etc/init.d/apache2 restart' to activate new configuration!

root@sec-lab2:/etc/apache2# /etc/init.d/apache2 restart
 * Restarting web server apache2

root@sec-lab2:/etc/apache2# netstat -ltn
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address Foreign Address State
tcp        0      0 0.0.0.0:80    0.0.0.0:*       LISTEN
tcp        0      0 0.0.0.0:443   0.0.0.0:*       LISTEN
```

Rysunek 5.18. Włączanie obsługi protokołu TLS

Prezentowany przykład będzie dotyczył konfiguracji obsługi protokołu TLS dla wirtualnego hosta. W przypadku konfiguracji tylko głównego serwera należy konfigurować wyłącznie standardowe pliki.

W pierwszym kroku zostanie utworzony wirtualny host `www.tls-demo.pl`, który będzie odnosił się do witryny umieszczonej w katalogu `/var/www/tls-demo`. W tym celu warto użyć wzoru standardowej konfiguracji hosta wirtualnego wspierającego protokół TLS. Standardowo, serwer apache2 przechowuje ten wzór w pliku `/etc/apache2/sites-available/default-ssl` i dotyczy on standardowej strony obsługiwanej przez serwer. Dla wirtualnego hosta tworzymy kopię pliku `/etc/apache2/sites-available/default-ssl` i zapisujemy ją np. jako `/etc/apache2/sites-available/tls-demo-ssl`. W pliku tym należy ustawić adres/nazwę wirtualnego hosta, który będzie nasłuchiwał na porcie 443 (`<VirtualHost www.tls-demo.pl:443>`). Kolejnym kluczowym parametrem jest określenie katalogu głównego witryny przypisanej do zdefiniowanego wirtualnego hosta (`DocumentRoot /var/www/tls-demo`). W podstawowej konfiguracji należy zmienić jeszcze dwa parametry (`ServerAdmin` oraz `ServerName`). W dalszej części rozdziału przedstawiona została konfiguracja dla wirtualnego hosta `www.tls-demo.pl` (Rys. 5.19, 5.20 i 5.21).

```
<IfModule mod_ssl.c>
<VirtualHost www.tls-demo.pl:443>
        ServerAdmin webmaster@tls-demo.pl
        ServerName www.tls-demo.pl

        DocumentRoot /var/www/tls-demo

        <Directory />
                Options FollowSymLinks
                AllowOverride None
        </Directory>

        <Directory /var/www/tls-demo/>
                Options Indexes FollowSymLinks MultiViews
                AllowOverride None
                Order allow,deny
                allow from all
        </Directory>

        ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
```

Rysunek 5.19. Konfiguracja wirtualnego hosta www.tls-demo.pl - część 1

Następnie należy wyłączyć obsługę protokołu TLS dla ustawień domyślnych. Domyśle ustawienia znajdują się w pliku `default-ssl`. Wyłączyć obsługę danej strony lub wirtualnego hosta zdefiniowanej w plikach konfiguracyjnych można przy pomocy polecenia `a2dissite`. Poniżej przedstawione jest wyłączenie domyślnej obsługi protokołu TLS (Rys. 5.22).

W następnym kroku należy włączyć obsługę protokołu TLS dla wirtualnego

```
        <Directory "/usr/lib/cgi-bin">
                AllowOverride None
                Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Order allow,deny
                Allow from all
        </Directory>

        ErrorLog ${APACHE_LOG_DIR}/error.log

        LogLevel warn

        CustomLog ${APACHE_LOG_DIR}/ssl_access.log combined

        Alias /doc/ "/usr/share/doc/"
        <Directory "/usr/share/doc/">
                Options Indexes MultiViews FollowSymLinks
                AllowOverride None
                Order deny,allow
                Deny from all
                Allow from 127.0.0.0/255.0.0.0 ::1/128
        </Directory>
```

Rysunek 5.20. Konfiguracja wirtualnego hosta www.tls-demo.pl - część 2

```
SSLEngine on

SSLCertificateFile    /etc/ssl/certs/ssl-cert-snakeoil.pem
SSLCertificateKeyFile /etc/ssl/private/ssl-cert-snakeoil.key

BrowserMatch "MSIE [2-6]" \
        nokeepalive ssl-unclean-shutdown \
        downgrade-1.0 force-response-1.0

BrowserMatch "MSIE [17-9]" ssl-unclean-shutdown

</VirtualHost>
<IfModule>
```

Rysunek 5.21. Konfiguracja wirtualnego hosta www.tls-demo.pl - część 3

hosta zdefiniowanego wcześniej. W tym celu posłużymy się poleceniem `a2ensite`. Po włączeniu domy należy aktywować nową konfigurację (Rys. 5.23).

Jeżeli konfiguracja wykonywana jest lokalnie, wówczas należy przypisać domenie

```
root@sec-lab2:/etc/apache2/sites-available# a2dissite default-ssl
Site default-ssl already disabled
```

Rysunek 5.22. Wyłącznie domyślnej obsługi protokołu TLS

```
root@sec-lab2:/etc/apache2/sites-available# a2ensite tls-demo-ssl
Enabling site tls-demo-ssl.
Run '/etc/init.d/apache2 reload' to activate new configuration!
```

Rysunek 5.23. Włączenie obsługi domeny tls-demo-ssl

adres IP, komputera na którym przeprowadzamy konfigurację. Wpis ten należy dodać do pliku konfiguracyjnego (Rys. 5.24).

```
192.168.0.121        www.tls-demo.pl
```

Rysunek 5.24. Wpis w pliku hosts

Jeżeli konfiguracja została przeprowadzano poprawnie, to serwer WWW będzie realizował protokół TLS w roli serwera. Istotnym elementem protokołu TLS jest możliwość weryfikacji tożsamości witryny, z którą się łączymy. Dzieje się to przy pomocy certyfikatów kluczy publicznych używanych w infrastrukturze klucza publicznego. Domyślnie podczas instalacji systemu Linux często tworzone są testowe certyfikaty, które zawierają pole określone przez daną dystrybucję linuxa. W przypadku realizacji protokołu TLS dla serwera WWW istotnym elementem jest utworzenie certyfikatu, który będzie zawierał dane identyfikująca konkretną domenę. W tym celu należy wykorzystać bibliotekę `openssl` do utworzenia stosownych certyfikatów. Wszelkie parametry, które może zawierać certyfikat klucza publicznego, można podać z ręcznie utworzonego pliku konfiguracyjnego. Niestety, ręczna konfiguracja jest złożonym procesem. Istnieje możliwość wygenerowania klucza prywatnego wraz z samopodpisanym certyfikatem w sposób automatyczny. W tym celu należy zainstalować pakiet `ssl-cert`. Dalej zaprezentowane zostało sprawdzenie, czy już jest zainstalowany oraz polecenie, które pobierze pakiet z repozytorium (Rys. 5.25).

```
root@sec-lab2:# dpkg --list | grep ssl-cert
ii  ssl-cert                                          1.0.28

apt-get install ssl-cert
```

Rysunek 5.25. Pobieranie pakietu ssl-cert

Pakiet ten zawiera plik konfiguracyjny /usr/share/ssl-cert/ssleay.cnf, w którym można określić podstawowe pola, które będzie zawierał certyfikat. Dalej zaprezentowana została zawartość tego pliku, szczegóły można znaleźć w dokumentacji [3] (Rys. 5.26).

```
root@sec-lab2:# cat /usr/share/ssl-cert/ssleay.cnf
#
# SSLeay example configuration file.
#

RANDFILE                = /dev/urandom

[ req ]
default_bits            = 2048
default_keyfile         = privkey.pem
distinguished_name      = req_distinguished_name
prompt                  = no
policy                        = policy_anything

[ req_distinguished_name ]
commonName                      = @HostName@
```

Rysunek 5.26. Zawartość pliku ssleay.cnf

W dalszej części konfiguracji zostanie utworzona para: klucz prywatny - samopodpisany certyfikat klucz publicznego. W tym celu należy wykonać polecenie make-ssl-cert wraz z dwoma parametrami. Pierwszy wskazuje plik konfiguracyjnym, na podstawie, którego zostanie wygenerowana para kluczy (/usr/share/ssl-cert/ssleay.cnf) oraz miejsce, gdzie będzie utworzony plik z kluczami (/etc/ssl/private/www.tls-demo.crt) (Rys. 5.27).

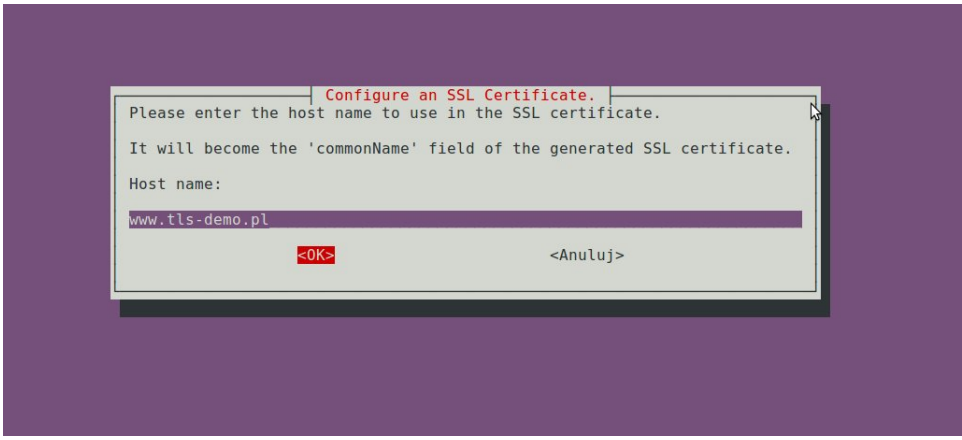
```
root@sec-lab2:# make-ssl-cert /usr/share/ssl-cert/ssleay.cnf \\
/etc/ssl/private/www.tls-demo.crt
```

Rysunek 5.27. Tworzenie certyfikatu dla witryny tls-demo.pl

Wykonanie tego polecenia spowoduje wyświetlenie się okna (Rys.5.28), w którym mamy wpisać adres witryny, dla której tworzony jest właśnie certyfikat.

Tak utworzona para kluczy, będzie znajdowała się w jednym pliku, który został podany. Wygenerowane klucze są według standardu PEM, poniżej przedstawiony jest przykład (Rys. 5.29).

Nie jest zalecane, żeby klucze te były przechowywane w jednym pliku, dlatego należy je rozdzielić. W tym celu można skopiować oraz przenieść plik z kluczami a następnie przy pomocy edytora utworzyć pliki wyłącznie z pojedynczymi kluczami. Czynności te zaprezentowane są poniżej (Rys. 5.30).



Rysunek 5.28. Okno programu make-ssl-cert

```
root@sec-lab2:# cat /etc/ssl/private/www.tls-demo.crt
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAvr9V6dTZUi2cyxs1KU1WkvwDBTDQRDkefaB8ahV6ty5ZwqCh
z0Yd5JjgvmrSSYtAhBr9aERtGcSgwsW798pWaxBzBCH6j9RiPEy+SEZ8g6eeFuwK
....
....
Rk4QcOsAxZYVa6S9bl3Hg0RMot0gcdplh9sJ57hIL8SUcZKt3wjYmaB1sMdAhJVJ
J9JYZFKIN0b7YBDjQhisrLw2eaWXzg5DvpFKfqTLd/9AOylnMkdC
-----END RSA PRIVATE KEY-----
-----BEGIN CERTIFICATE-----
MIICsDCCAZgCCQCxvsmeOD742TANBgkqhkiG9w0BAQUFADAaMRgwFgYDVQQDEw93
d3cudGxzLWRlbW8ucGwwHhcNMTIwMzEzMTQ0MDAyWhcNMjIwMzExMTQ0MDAyWjAa
....
....
MjyhHIeRobWRRmkdg5H48KFDm4qOfNK7YwCsyzV3c5TY7sehnBefc/ceI5Va/mKE
afGl38Q/cSXgGWDF8gkHzvQ/+Ao=
-----END CERTIFICATE-----
```

Rysunek 5.29. Zawartość wygenerowane pliku z kluczami

Plik z kluczem prywatnym powinien być zabezpieczony przed niepowołaną modyfikacją. Należy ustawić prawa wyłącznie dla właściciela pliku, którym jest `root` (Rys. 5.31).

Ostatnią czynnością związaną z kluczami, jest modyfikacja pliku `/etc/apache2/sites-available/tls-demo-ssl` w którym znajdują się parametry określające ich umiejscowienie w systemie. Parametry te wraz z ich wartościami dla omawianego przykładu, przedstawione są poniżej (Rys. 5.32).

Po dokonaniu zmian w konfiguracji, apache2 musi ją ponownie ją wczytać (Rys. 5.33).

```
root@sec-lab2:# cp /etc/ssl/private/www.tls-demo.crt \\
/etc/ssl/private/www.tls-demo.key
root@sec-lab2:# mv /etc/ssl/private/www.tls-demo.crt \\
/etc/ssl/certs/www.tls-demo.pem

root@sec-lab2:# vim /etc/ssl/private/www.tls-demo.key
root@sec-lab2:# vim /etc/ssl/certs/www.tls-demo.pem
```

Rysunek 5.30. Przenoszenie kluczy do osobnych plików

```
root@sec-lab2:# chmod 600 /etc/ssl/private/www.tls-demo.key
```

Rysunek 5.31. Zmiana prawa dostępu do klucza

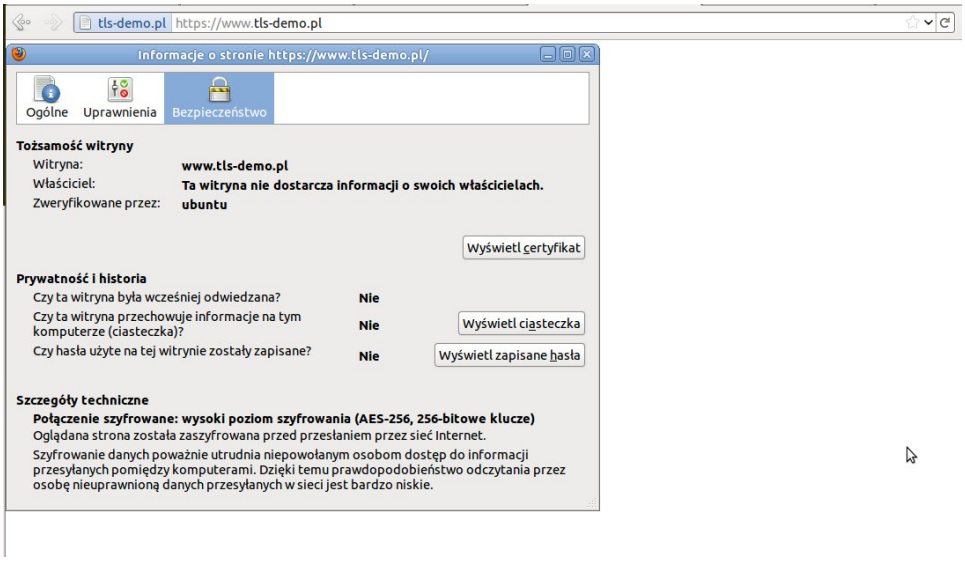
```
root@sec-lab2:# vim /etc/apache2/sites-available/tls-demo-ssl

SSLCertificateFile    /etc/ssl/certs/www.tls-demo.pem
SSLCertificateKeyFile /etc/ssl/private/www.tls-demo.key
```

Rysunek 5.32. Określenie lokalizacji kluczy w systemie

```
root@sec-lab2:# /etc/init.d/apache2 reload
```

Rysunek 5.33. Ponowne wczytanie konfiguracji przez serwer apache2

Tak skonfigurowany serwer WWW będzie obsługiwał protokół TLS w roli serwera. Jeżeli w przeglądarce internetowej zostanie wpisany adres domeny przypisanej do rozważanego wirtualnego hosta, wówczas będzie można zweryfikować tożsamość witryny, z którą zostało nawiązane połączenie 5.34. Warto dodać, że w przypadku samopodpisanych certyfikatów, tożsamość witryny jest stwierdzana przez stronę, która sama podpisała certyfikat. Żeby zagwarantować wysoki poziom zaufania, należy tworzyć certyfikaty klucz publicznych weryfikowanych przez zaufane Centra Autoryzacji [4].



Rysunek 5.34. Certyfikat witryny www.tls-demo.pl

Rozdział 6

Zapora ogniowa oraz translacja adresów sieciowych - iptables

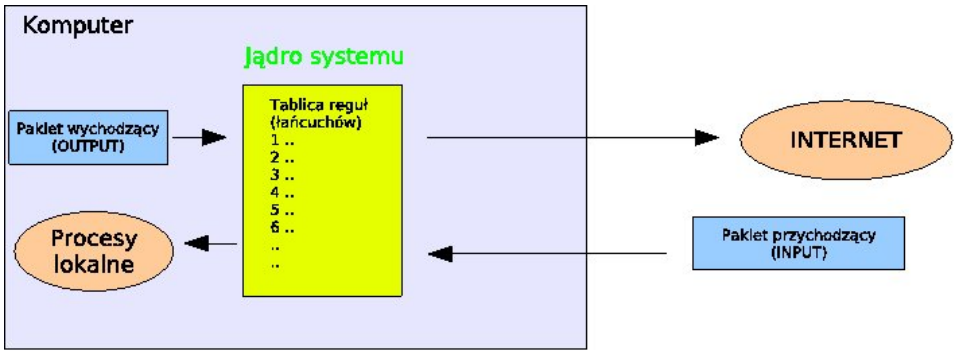
Zagadnienia związane z zapora ogniową oraz translacją adresów sieciowych są kluczowe w przypadku chęci zagwarantowania odpowiedniego poziomu bezpieczeństwa serwerów sieciowych. W literaturze jest wiele książek omawiających podstawy budowy reguł w oparciu o `iptables` [5]. Wszelkie szczegóły można znaleźć również w podręcznikach systemowych (manualach), opisujących zagadnienia filtracji pakietów oraz translacji adresów sieciowych [5]. W poniższym rozdziale zostaną omówione zasady budowania reguł w oparciu o `iptables` stosowane do filtrowania pakietów oraz translacji adresów sieciowych (NAT). W dalszej części rozdziału znajdują się zadania wraz z rozwiązaniami dotyczące problematyki filtrowania pakietów.

6.1. Iptables - podstawy

Iptables jest programem, za pomocą którego można utworzyć reguły filtrowania pakietów, które zapisywane są w tablicy znajdującej się w jądrze systemu. Filtrowanie IP jest mechanizmem decydującym o tym, które pakiety IP należy przyjąć do systemu, a które należy odrzucić lub porzucić. Przy pomocy reguł iptables można definiować wiele kryteriów określających filtrowanie pakietów. Podstawowe to np.:

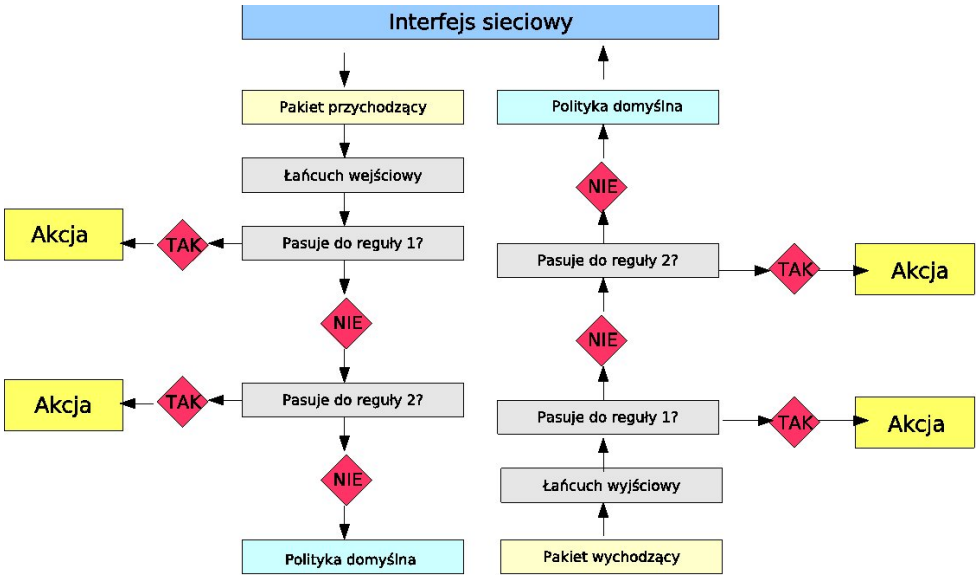
— typ protokołu: TCP,UDP, ICMP;
— numer portu (dla TCP/UDP);
— typ pakietu: SYN/ACK, ICMP Echo request;
— adres źródłowy pakietu;
— adres docelowy pakietu.

W przypadku, gdy w systemie włączone jest filtrowanie pakietów, wówczas pakiety przychodzące/wychodzące za pośrednictwem dowolnego interfejs sieciowego są wcześniej sprawdzane. Polega to na tym, że na poziomie jądra systemu znajdują się reguły, które tworzą pewien wzorzec mający na celu wychwycić konkretne rodzaje pakietów. Następnie wykonywaną są przewidziane dla konkretnych pakietów akcje. Na Rys. 6.1 przedstawione jest schematyczne umiejscowienie tworzonych reguł przy pomocy iptables. Wszystkie pakiety przechodzące przez dowolny interfejs sieciowy są wcześniej filtrowane.



Rysunek 6.1. Tablica reguł

Zasada filtrowania przedstawiona jest na Rys. 6.2. Reguły filtrujące, znajdujące się w jądrze systemu, umieszczone są w kolejności ich wykonania/uruchomienia. Tak utworzona struktura tworzy kolejność reguł. Dla przykładu, pakiet przychodzący do systemu jest sprawdzany pod kątem zgodności z regułami znajdującymi się w jądrze w kolejności ich wykonania. To jest bardzo istotne, ponieważ w przypadku gdy pakiet będzie pasował do reguły o numerze 1, wówczas zostanie wykonana przewidziana w regule akcja i następne reguły nie będą już sprawdzane (wyjątkiem są reguły polityki domyślnej oraz zapisów kontrolnych). Zasada ta przedstawiona jest na Rys.6.2.



Rysunek 6.2. Zasada filtrowania pakietów

6.1.1. Składnia iptables - tablica filter

Iptables posiada dwie podstawowe tablice reguł. Pierwsza **filter** (domyślna) zawiera wbudowane łańcuchy INPUT (pakiety przychodzące) i OUTPUT (pakiety wychodzące) oraz FORWARD (dla pakietów przekazywanych przez hosta). Druga tablica **nat** dotyczy reguł stosowanych dla pakietów, dla których dokonywana jest translacja adresów IP (ang. Network Address Translation - NAT). Tablica **nat** zawiera łańcuchy PREROUTING (interpretowane przed decyzją routingu), POSTROUTING (interpretowane po decyzji routingu) oraz OUTPUT (dla pakietów wychodzących).

Reguły filtrowania budowane są przy pomocy schematu, który przedstawiony jest poniżej. W pierwszej kolejność zostanie przedstawiony schemat dla tablicy **filter**.

Przykładowa reguła filtrowania pakietów polegająca na akceptowaniu połączeń ze zdalnymi serwerami **ssh** przedstawiona jest poniżej.

```
iptables -A/-I ... -i/-o ... -p ... (--syn / ! -syn ) \\
         -s ... --sport ...  -d ... --dport ... -j ...

-A/-I   : dodanie łańcucha na koniec tabeli (A) lub na początek (I)
Rodzaj łańcuchów:INPUT,OUTPUT,FORWARD
-i/-o   : wskazujemy na rodzaj interfejsu, jeżeli -i
to pakiet przychodzący, jeżeli -o to pakiet wychodzący
-p      : rodzaj protokołu (TCP/UDP/ICMP)
(- -syn / !--syn ) : opcje opisujące stan połączenia
-s      : adres IP źródłowy (nadawcy)
--sport : port źródłowy (nadawcy)
-d      : adres IP odbiorcy
--dport : port odbiorcy
-j      : rodzaj akcji (ACCEPT, DROP, REJECT, LOG)

ACCEPT  : pakiet zostanie zaakceptowany
DROP    : pakiet zostanie porzucony
REJECT  : pakiet zostanie odrzucony
LOG     : na podstawie pakietu zostanie wykonany zapis kontrolny
```

Rysunek 6.3. Budowa reguł - tablica filter

```
iptables -A OUTPUT -o eth0 -p tcp -s 192.168.1.1 \\
         --sport 1024:65535 -d 0/0 --dport 22 -j ACCEPT

iptables -A INPUT -i eth0 -p tcp !--syn -s 0/0 \\
         --sport 22 -d 192.168.1.1 --dport 1024:65535 -j ACCEPT
```

Rysunek 6.4. Reguła iptables - Akceptowanie połączeń w roli klienta z serwerami ssh

Opisywany do tej pory sposób filtrowania połączeń nazywany jest często jako filtrowanie statyczne. W takim podejściu twórca zapory ogniowej musi przewidzieć wszelkie pakiety, które będą wymieniane w przypadku udostępnienia usługi sieciowej. Istnieje możliwość zastosowania tzw. filtrowania dynamicznego, które polega na tym, że jądro systemu będzie samodzielnie śledziło stan połączenia, które zostało zaakceptowane. W tym celu należy wykorzystać dodatkowy moduł dostępny dla programu iptables, który pozwoli śledzić stan poszczególnych połączeń. Jest to moduł **state**, który pozwala analizować stan połączenia śledzony przy pomocy modułu systemowego **ip_conntrack**. Podanie w regule opcji **-m state** udostępnia dodatkową opcję **--state**, której parametrem są możliwe stany połączenia. Stany połączeń mogą być następujące:
— **NEW (NOWY)** - pakiet, który tworzy nowe połączenie;

— **ESTABLISHED (NAWIĄZANY)** - pakiet który należy do istniejącego połączenia (np. pakiet odpowiedzi, lub pakiet wychodzący w połączeniu, które otrzymało już odpowiedź);
— **RELATED (ZWIĄZANY)** - pakiet, który jest związany z istniejącym połączeniem, ale nie jest jego częścią; połączenie ftp dla danych;
— **INVALID (BŁĘDNY)** - pakiet, który nie może być zidentyfikowany (mogą to być wyczerpanie się pamięci, lub błędy ICMP, które nie należą do żadnego znanego połączenia).

Reguła filtrowania pakietów polegająca na akceptowaniu połączeń ze zdalnymi serwerami **ssh** ale realizowana przy pomocy śledzenia stanu połączenia przedstawiona jest poniżej.

```
iptables -A OUTPUT -o eth0 -p tcp --dport 22 -j ACCEPT \\
-m state --state NEW

iptables -A INPUT -p tcp -j ACCEPT -m state --state ESTABLISHED
iptables -A OUTPUT -p tcp -j ACCEPT -m state --state ESTABLISHED
```

Rysunek 6.5. Reguła iptables (stan połączenia) - Akceptowanie połączeń w roli klienta z serwerami ssh

6.1.2. Składnia iptables - tablica nat

Tablica **nat** odpowiedzialna jest za przechowywanie reguł dotyczących translacji adresów IP. Rozróżniamy dwa rodzaje translacji NAT: **źródłowy NAT (SNAT)** i **docelowy NAT (DNAT)**.

SNAT ma miejsce wtedy, gdy zostaje zmieniany **adres źródłowy** pierwszego pakietu, czyli kiedy zmieniany jest adres maszyny, z której inicjowane jest połączenie. **SNAT** wykonywany jest zawsze **po routingu** (ang. post-routing), tuż przed tym, gdy pakiet opuści maszynę. **Masquerading** jest specjalizowaną formą SNAT.

DNAT ma miejsce wtedy, gdy zostaje zmieniany **adres docelowy** pierwszego pakietu, czyli kiedy zmieniany jest adres maszyny, do której ma dotrzeć połączenie. **DNAT** wykonywany jest zawsze **przed routingiem** (ang. pre-routing), w momencie gdy, pakiet zostaje odebrany z łącza.

Przykłady zastosowania translacji NAT przedstawione są w dalszej części rozdziału.

```
Adres źródłowy zostanie zmieniony na 1.2.3.4
iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.3.4

Adres źródłowy zostanie zmieniony na 1.2.3.4 dla portów 100-123
iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT \\
--to 1.2.3.4:100-123

Wykonanie maskowanie połączeń na interfejsie eth0
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Rysunek 6.6. Przykładowe translacje SNAT

```
Zmiana adresów docelowych na 10.0.0.2
iptables -t nat -A PREROUTING -i eth0 -j DNAT --to 10.0.0.2

Zmiana adresów docelowe ruchu WWW na 10.0.0.2 oraz port 8080
iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT \\
--to 10.0.0.2:8080

Dostęp przez  sieć wewnętrzną do publicznego serwera WWW, który
przesłonięty jest przez DNAT z publicznego adresu (212.182.2.1)
na adres sieci wewnętrznej (192.168.1.10)
iptables -t nat -A PREROUTING -d 212.182.2.1 -p tcp --dport 80 \\
-j DNAT --to 192.168.1.10
```

Rysunek 6.7. Przykładowe translacje DNAT

6.2. Zadania

6.2.1. Zadanie 1

Napisz ścianę ognia tzw. **statyczną** za pomocą programu iptables [5] na komputer osobisty, który będzie pozwalał łączyć się z dowolnymi stronami WWW oraz serwerami SSH. Twój adres IP=192.168.1.100.

Rozwiązanie

```
#!/bin/bash
#eth1 - interfejs zew
iz=eth1

echo "USUNIECIE STARYCH Regul"
iptables -F

echo "BLOKUJEMY CALY RUCH SIECIOWY"
```

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

echo "wpuszczam lo"
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT


echo "Łączność za pomocą protokołu ssh"
iptables -A OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 22 -j ACCEPT
iptables -A INPUT -i $iz -p tcp ! --syn -s 0/0 --sport \
22 -d 192.168.1.100 --dport 1024:65535  -j ACCEPT

echo "Łączność za pomocą protokołu http"
iptables -A OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
 1024:65535 -d 0/0 --dport 80 -j ACCEPT
iptables -A INPUT -i $iz -p tcp ! --syn -s 0/0 --sport \
80 -d 192.168.1.100 --dport 1024:65535  -j ACCEPT

echo "Włączenie zapytań DNS"
iptables -A OUTPUT -o $iz -p udp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 53 -j ACCEPT
iptables -A INPUT -i $iz -p udp  -s 0/0 --sport \
53 -d 192.168.1.100 --dport 1024:65535  -j ACCEPT
```

6.2.2. Zadanie 2

Napisz ścianę ognia tzw. statyczną za pomocą programu iptables na komputer pełniący następujące funkcje:
Klient: WWW, https, skype, ssh
Serwer: ssh, www

Rozwiązanie

```
#!/bin/bash
#eth1 - interfejs zew
iz=eth1

echo "USUNIECIE STARYCH Regul"
iptables -F

echo "BLOKUJEMY CALY RUCH SIECIOWY"
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

```
echo "wpuszczam lo"
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

##### Klient #####

echo "Łączność za pomocą protokołu ssh"
iptables -A OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 22 -j ACCEPT
iptables -A INPUT -i $iz -p tcp ! --syn -s 0/0 --sport \
22 -d 192.168.1.100 --dport 1024:65535  -j ACCEPT

echo "Łączność za pomocą protokołu http"
iptables -A OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 80 -j ACCEPT
iptables -A INPUT -i $iz -p tcp ! --syn -s 0/0 --sport 80 \
-d 192.168.1.100 --dport 1024:65535  -j ACCEPT

echo "Łączność za pomocą protokołu https i skype \
(ustawione połączenia wychodzące na 443)"
iptables -A OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 443 -j ACCEPT
iptables -A INPUT -i $iz -p tcp ! --syn -s 0/0 --sport \
443 -d 192.168.1.100 --dport 1024:65535  -j ACCEPT

echo "Włączenie zapytań DNS"
iptables -A OUTPUT -o $iz -p udp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 53 -j ACCEPT
iptables -A INPUT -i $iz -p udp  -s 0/0 --sport \
53 -d 192.168.1.100 --dport 1024:65535  -j ACCEPT

##### Serwer #####

echo "Łączność z naszym serwerem ssh"
iptables -A INPUT -i $iz -p tcp -s 0/0 --sport \
1024:65535 -d 192.168.1.100 --dport 22 -j ACCEPT
iptables -A OUTPUT -o $iz -p tcp ! --syn -s 192.168.1.100 \
--sport 22 -d 0/0 --dport 1024:65535  -j ACCEPT

echo "Łączność z naszym serwerem www"
iptables -A INPUT -i $iz -p tcp -s 0/0 --sport \
1024:65535 -d 192.168.1.100 --dport 80 -j ACCEPT
iptables -A OUTPUT -o $iz -p tcp ! --syn -s 192.168.1.100 \
--sport 80 -d 0/0 --dport 1024:65535  -j ACCEPT
```

6.2.3. Zadanie 3

Do ściany ognia utworzonej w zadaniu 2 dodaj logowanie połączeń ze zdalnymi serwerami ssh.

Rozwiązanie

Przed regułami dotyczącymi *Łączności za pomocą protokołu ssh* należy dodać regułę:

```
iptables -A OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 22 -j LOG
```

lub w dowolnym miejscu regułę z parametrem *-I* co powoduje wstawienie reguły na początek listy reguł typu *filter*:

```
iptables -I OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 22 -j LOG
```

6.2.4. Zadanie 4

Napisz ścianę ognia tzw. 'dynamiczną' (filtrującą po stanie protokołu) za pomocą programu iptables na komputer pełniący funkcje opisane w zadaniu 2.

Rozwiązanie

```
#!/bin/bash
#eth1 - interfejs zew

modprobe ip_conntrack
modprobe ip_conntrack_ftp

iz=eth1

echo "USUNIECIE STARYCH Reguł"
iptables -F

echo "BLOKUJEMY CALY RUCH SIECIOWY"
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
ISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o $iz -p tcp -s 192.168.1.100 --sport \
1024:65535 -d 0/0 --dport 20 -m state --state ESTABLISHED -j ACCEPT


#### Pasywny - serwer ######

iptables -A INPUT -i $iz -p tcp -s 0/0 --sport 1024:65535 \
-d 192.168.1.100 --dport 1024:65535 -m state --state \
ESTABLISHED,RELATED -j ACCEPT
```

```
iptables -A OUTPUT -o $oz -p tcp -s 192.168.1.100 --sport 1024:65535 \
-d 0/0 --dport 1024:65535 -m state --state ESTABLISHED -j ACCEPT
```

6.2.5. Zadanie 5

Przykładowa architektura sieci komputerowej zaprezentowana jest na rysunku 6.8. Utwórz regułę na komputerze bramie (IPX,IP4), która będzie zmieniała adresy sieci wewnętrznej (IP1,IP2,IP3) na adresy komputera pełniącego rolę bramy.

Rozwiązanie

Reguła zmieniająca adresy sieci wewnętrznej na stały adres publiczny IPX:

```
iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to IPX
```

Reguła zmieniająca adresy sieci wewnętrznej na zmienny adres publiczny taki jak w danym momencie przypisany jest do interfejsu *eth0*:

```
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

6.2.6. Zadanie 6

Napisz regułę, która dla architektury przedstawionej na rysunku 6.8 będzie kierowała pakiety z sieci Internet na serwer WWW, który będzie zainstalowany na komputerze w sieci wewnętrznej o IP: IP1.

Rozwiązanie

Reguła przekazująca porty dla połączeń WWW (port 80) dla pakietów kierowanych na interfejs zewnętrzny na komputer w sieci wewnętrznej:

```
iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth0 -j DNAT \
--to IP1:80
```

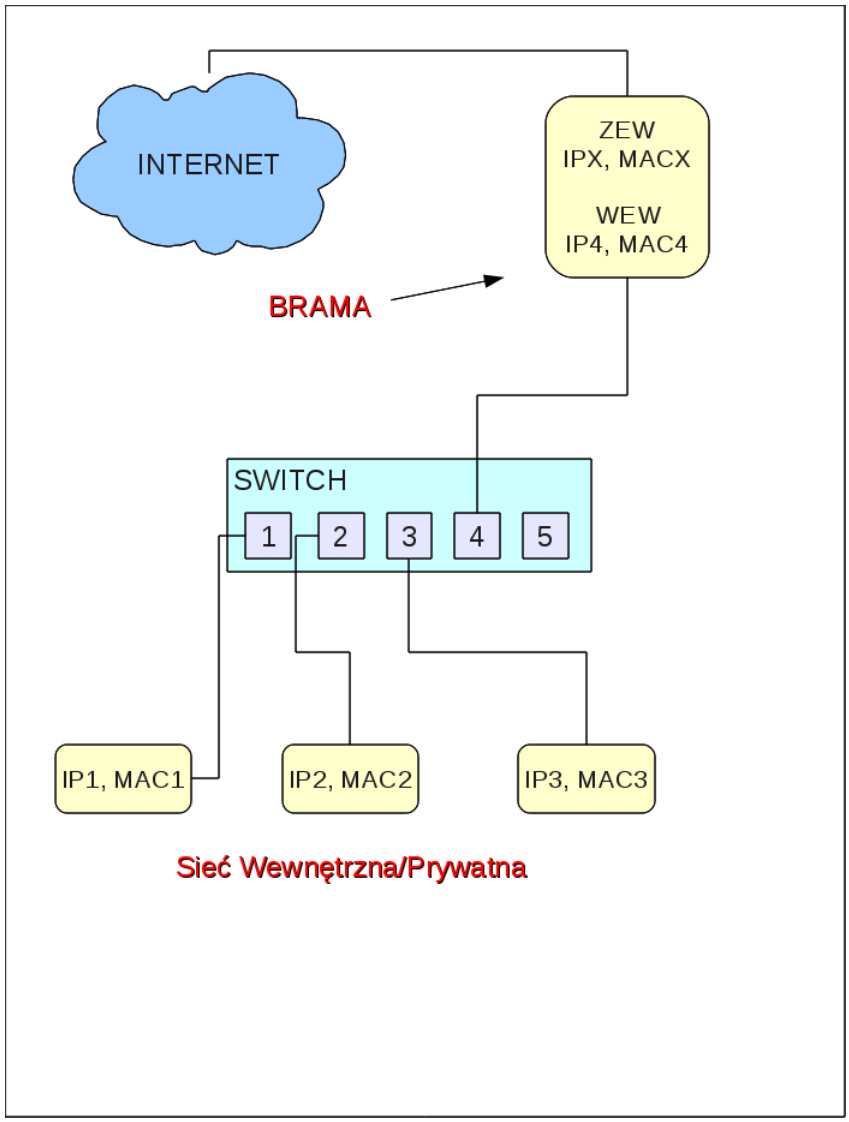
6.2.7. Zadanie 7

Ustaw regułę, która spowoduje, że wychodzące połączenia ssh będą zawsze realizowane na znany Ci serwer ssh.

Rozwiązanie

Reguła powodująca, że wszelkie połączenia wykonywane na port 22 (ssh) będą kierowane na znany adres IP:

```
iptables -t nat -A OUTPUT -p tcp --syn --dport 22 -j DNAT \
--to Znane_IP
```



Rysunek 6.8. Architektura sieci wewnętrznej wraz z bramą

Rozdział 7

Skanowanie - sposób kontroli serwerów sieciowych

Ważnym krokiem kontroli serwerów sieciowych jest wykonywania ścieżki technicznej audytu systemów IT, a w szczególności procesu skanowania. Skanowanie jest to proces, który ma na celu zebranie szczegółowych informacji na temat oprogramowania znajdującego się na hostach. Wiedza na ten temat jest kluczowa, ponieważ właśnie błędy w oprogramowaniu pozwalają na przeprowadzenie ataku. Każda udostępniona usługa to linie kodu napisane w konkretnym języku oprogramowania. Niestety, programiści pisząc aplikacje popełniają błędy, których wykorzystanie umożliwia wykonanie w systemie operacji nieprzewidzianej przez twórcę oprogramowania. Taką operacją może być na przykład zdalny dostęp do systemu bez potrzeby wcześniejszej autoryzacji. Inną możliwością wykonania ataku jest wykorzystanie błędnej konfiguracji danej usługi sieciowej, ale i w tym przypadku jest to związane z konkretnym oprogramowaniem. W systemach operacyjnych usługi sieciowe reprezentowane są przez numery portów, czyli w przypadku chęci połączenia z konkretną usługą udostępnianą przez serwer, komputer kliencki komunikuje się z odpowiednim numerem portu. Na przykład serwer WWW standardowo nasłuchuje połączeń na porcie 80 (jest możliwość zmiany tego numeru portu). Jeżeli kierowany jest do niego pakiet na porcie docelowym o numerze 80, wówczas system będzie wiedział, że pakiety te kierowane są do serwera WWW.

Proces skanowania składa się z dwóch etapów. Pierwszy polega na sprawdzeniu, jakie usługi udostępniane są przez konkretne hosty w sieci. W drugim etapie określane są wersje oprogramowania realizujące dane usługi. W kolejnych podrozdziałach zostaną opisane techniki wykonania procesu skanowania hostów.

Istotnym zadaniem, jakie musi zostać wykonane przed rozpoczęciem skanowania, jest uzyskanie pisemnej zgodny na wykonywane testy penetracyjne. Żadna operacja nie może zostać wykonana, dopóki szczegółowy plan testów nie zostanie zatwierdzony przez kierownictwo danej organizacji. Program testów powinien zawierać następujące informacje:

1. Listę obiektów (hostów) przewidzianych do testów wraz z ich adresami IP.
2. Listę osób, które wykonują testy, wraz z przyporządkowaniem ich do konkretnych skanowanych obiektów. Dla wszystkich skanowanych obiektów lub grupy obiektów musi być wskazana osoba odpowiedzialna. Do tej osoby muszą być określone dane kontaktowe.
3. Czas przeprowadzania testów powinien być szczegółowo określony. Do skanowanych obiektów trzeba przyporządkować okres dnia, kiedy będą przeprowadzane testy wraz z określeniem ich czasu trwania.

Po uzyskaniu zgody kierownictwa audytowanej organizacji na wykonywane skanowanie można zacząć przeprowadzać testy. Skanowanie można wykonać przy użyciu technik, które wykorzystują różne protokoły modelu TCP/IP. Do tego celu można wykorzystać protokoły: ICMP, TCP, UDP oraz protokoły warstwy aplikacji. Proces skanowania można wykonać przy użyciu różnych aplikacji, które pozwalają wysyłać specjalnie spreparowane pakiety, wykorzystując wspomniane protokoły modelu TCP/IP, w naszym przypadku zostanie użyte głównie narzędzie *nmap* [6]. W dalszej części zostaną opisane najważniejsze rodzaje skanowania. Zostaną one zaprezentowane na przykładzie narzędzia *nmap*.

7.1. NMAP

W systemach operacyjnych aplikacje sieciowe reprezentowane są przez numery portów. Jeżeli na porcie, który odpowiada konkretnej usłudze, system nasłuchuje połączeń, to znaczy, że dana usługa jest uruchomiona na tym hoście. Aplikacja *nmap* określa ten stan portu jako **otwarty**. Precyzując można powiedzieć, że wówczas na danym porcie host przyjmuje połączenia TCP oraz pakiety UDP. Kolejnym podstawowym stanem, w jakim może znaleźć się port jest **zamknięty**. To oznacza, że dany port jest dostępny (odpowiada na wysłane pakiety), ale nie ma tam nasłuchującej na połączenia aplikacji. Następnym stanem jest stan **filtrowany**, który informuje, że połączania na tym porcie są filtrowane przez ścianę ognia samego hosta lub np. router. W taki wypadku nie wiadomo, czy port jest otwarty. Wspomniane 3 stany określane przez aplikacje *nmap* są podstawowymi stanami portów, ale *nmap* posiada jeszcze 3 kolejne, dodatkowe stany. Może to być stan **niefiltrowany**. W takim przypadku *nmap* nie jest w stanie określić, czy port jest w stanie otwartym czy zamkniętym. Kolejnym stanem jest stan **otwarty|filtrowany**, czyli nie wiadomo, czy port jest otwarty, czy filtrowany. Ostatnim stanem dodatkowym jest stan **zamknięty|filtrowany**, czyli nie można wskazać, czy jest to stan zamknięty, czy filtrowany. W dalszej części zostaną przedstawione najistotniejsze wersje skanowania wykonywane za pomocą aplikacji *nmap*.

7.1.1. TCP SYN : flaga -sS

Skanowanie typu TCP SYN polega na wysłaniu pakietów SYN do danego hosta, czyli jest to chęć nawiązania połączania za pomocą protokołu TCP. Na taki typ skanowania host może odpowiedzieć pakietem SYN/ACK, co oznacza, że na danym porcie nasłuchuje aplikacja i *nmap* oznacza ten port jako otwarty. Tak rozpoczęte połączenie nie zostanie zestawione, ponieważ dla pakietu SYN/ACK nie zostanie wysłany pakiet potwierdzający ACK. Taki stan połączenia nazywany jest w połowie nawiązanym (ang. half-open). W przypadku, gdy port jest zamknięty, wysyłany jest do strony skanującej pakiet RST, który kończy proces nawiązywania połączenia. Jeżeli nie ma żadnej odpowiedzi na taki pakiet, wówczas takie skanowania ponawiane są kilkakrotnie i jeżeli na wszystkie zapytania nie ma odpowiedzi, wówczas port określany jest jako **filtrowany**.

7.1.2. TCP CONNECT : flaga -sT

Ten typ skanowania polega na próbie pełnego zestawienia połączenia TCP. Połączenie to jest zestawiane za pomocą funkcji systemowej *connect()*, a nie jak w przypadku innych skanowań, za pomocą pakietów typu *RAW*. Zestawiając połączenia typy *RAW*, aplikacja *nmap* będzie otrzymywała odpowiedzi w postaci pakietów wraz z nagłówkami, a nie jak w przypadku standardowych połączeń, otrzyma tylko dane bez nagłówków. W przypadku, gdy usługa nasłuchuje na tym porcie, wówczas nawiązywane jest połączenie. Niestety, taka próba zostanie odnotowana w logach systemowych skanowanego hosta, co dla administratora tego hosta będzie oznaczało, że host był skanowany. Podobnie jak w przypadku skanowania TCP SYN, w sytuacji, gdy na danym porcie nie nasłuchuje żadna usługa, ale host jest aktywny w sieci, wówczas zostanie zgłoszony stan zamknięty.

7.1.3. TCP NULL : flaga -sN

Operując na pakietach typu TCP mamy możliwość niewskazania żadnej flagi. System, który otrzyma pakiet TCP bez flagi typu SYN,RST lub ACK, spowoduje wysłanie flagi RST w przypadku portu zamkniętego. W przypadku portu otwartego zostanie zgłoszony błąd przekroczenia czasu oczekiwania na połączenie. Jeżeli dany port jest filtrowany, wówczas *nmap* otrzyma komunikat *ICMP unreachable*. Ten typ skanowania pozwala na przejście przez bezstanową ścianę ognia lub router. Niestety, niektóre systemy operacyjne bezwarunkowo po odebraniu takiego pakietu wysyłają pakiet z flagą RST.

7.1.4. TCP ACK : flaga -sA

Skanowanie TCP ACK polega na wysyłaniu pakietów z flagą ACK. W wyniku takiego skanowania porty bez filtrowania zawsze odpowiedzą flagą RST. W takim przypadku otrzymamy odpowiedź **niefiltrowany**. Porty, które nie odpowiedzą lub odpowiedzą komunikatem *ICMP unreachable*, zostaną określone jako **filtrowane**. Tego typu skanowanie wykorzystywane jest do testowania reguł ściany ognia.

7.1.5. TCP ZOMBI SCAN : flaga -sI

Skanowanie typu *Zombi* umożliwia skanowanie systemu przez wysyłanie pakietów, w których adres IP nadawcy będzie sfałszowany. Adres nadawcy będzie adresem hosta *Zombi*, co spowoduje, że systemy wykrywania włamań na badanym hoście będą myślały, że skanowanie jest wykonywane przez hosta *Zombi*. Wykonanie takiego skanowania jest możliwe w przypadku, gdy stos TCP/IP hosta zombi działa standardowo, a w szczególności ważne jest żeby ID wysyłanych pakietów IP było zwiększane zawsze o 1. Sprawdzenie, czy port jest otwarty można wykonać w 3 krokach.

1. Początkowo wysyłamy pakiet SYN/ACK do wybranego przez nas hosta, który będzie pełnił rolę *Zombi*. Ważne, żeby był to host, który często nie wysyła pakietów IP, może to być np. drukarka. W odpowiedzi na ten pakiet otrzymujemy obecną wartość ID pakietu IP.
2. W kolejnym kroku wysyłamy pakiet SYN do badanego hosta, ale jako adres IP nadawcy wpisujemy adres hosta *Zombi*. W przypadku, gdy port jest otwarty, badany host wyśle na adres IP nadawcy pakiet SYN/ACK, co spowoduje, że host *Zombi* odpowie pakietem z flagą RST.
3. W ostatnim kroku przesyłamy ponownie pakiet SYN/ACK do hosta *Zombi* i sprawdzamy wartość ID otrzymanego pakietu IP. W przypadku, gdy wartość ta została zwiększona o 2, wówczas w kroku 2 badany host wysłał pakiet SYN/ACK czyli port jest otwarty. Jeżeli wartość ta została zwiększona tylko o 1, wówczas w kroku 2 badany host nie wysłał pakietu SYN/ACK tylko pakiet RST, co świadczy o tym, że port jest zamknięty. Warto dodać, że w przypadku, gdy badany port jest filtrowany, również wartość ID pakietu IP zostanie zwiększona o 1, ponieważ w kroku 2 badany host nie wyśle żadnego pakietu do hosta *Zombi*.

7.1.6. UDP SCAN : flaga -sU

Oprócz skanowania portów metodami wykorzystującymi protokół TCP istnieje możliwość skanowania za pomocą protokołu UDP. Warto skanować również te porty, ponieważ duża część usług sieciowych wykorzystuje właśnie protokół UDP. W przypadku takiego skanowania wysyłane są pakiety DNS bez danych i w przypadku, gdy badany host zwróci komunikat ICMP port unreachable kod 3, wówczas port będzie zamknięty. W przypadku, gdy zgłoszony jest błąd ICMP port unreachable kod 1,2,9,10,13, wówczas port jest **filtrowany**. Jeżeli zostanie zwrócony pakiet UDP, wtedy wiadomo, że port jest **otwarty**.

7.1.7. PROTOCOL SCAN : flaga -sO

Interesującym typem skanowania jest sprawdzenie, jakie protokoły IP obsługiwane są przez badanego hosta. W tym przypadku nie są skanowane porty, ale numery protokołów. W tym celu zwiększany jest w nagłówkach numer odpowiedzialny za wersję protokołu. Jeżeli aplikacja *nmap* otrzyma odpowiedź, wówczas będzie wiadomo, że dany protokół jest wspierany.

7.1.8. ICMP SCAN : flaga -sP

Za pomocą tej opcji można sprawdzić, które hosty z podanej puli adresowej są w danym momencie aktywne. W tym celu wysyłane jest zapytanie *icmp request* oraz pakiet TCP SYN na port 80 badanego hosta. Oczekiwana jest odpowiedź, czyli *icmp reply*, oraz pakiet TCP SYN/ACK. Jeżeli skanowanie nie jest wykonywane z konta uprzywilejowanego, wówczas wysyłany jest wyłącznie pakiet TCP SYN na port 80 przy użyciu funkcji connect() opisanej wyżej. W przypadku gdy testy wykonywane są w obrębie sieci lokalnej Ethernet, wtedy wykonywane są dodatkowo zapytania ARP.

7.1.9. NMAP : Opcje Skanowania

Polecenie skanowania wykonywane za pomocą aplikacji *nmap* wydawane jest z linii poleceń terminala. Oprócz zdefiniowania podstawowej wersji testów należy uzupełnić zapytanie o dodatkowe opcje precyzujące testy. W dalszej części podrozdziału zostaną opisane dodatkowe opcje narzędzia *nmap*.

7.1.9.1. Zakres portów : flaga -p

Za pomocą tej opcji można określić konkretne porty, które mają być przeskanowane. Można podać pojedynczy port lub cały zakres (np. 1-10).

7.1.9.2. Szybkie skanowanie : flaga -F

Jeżeli chcemy, żeby zostały przetestowane tylko porty standardowych usług sieciowych, wówczas można włączyć opcję szybkiego skanowania.

7.1.9.3. Liczba prób skanowania: flaga –max-retries

Wykonując skanowanie dowolnego portu, często nie uzyskamy żadnej odpowiedzi. Taka sytuacja może być spowodowana tym, że testowany ruch jest filtrowany.

Innym powodem może być fakt, że pakiet zaginął w sieci, dlatego aplikacja *nmap* ponawia testy wiele razy. Za pomocą tej flagi można określić, ile razy mają być ponawiane zapytania. Ta opcja jest istotna w przypadku, gdy wydajność jest ważnym czynnikiem oraz gdy nie chcemy generować nadmiarowego ruchu sieciowego.

7.1.9.4. Opóźnienia testów : –scan-delay

Opcja ta pozwala na określenie opóźnień pomiędzy wykonywanymi testami. Jako parametr podajemy czas, który standardowo liczony jest w milisekundach, ale można go podać również w sekundach (s), minutach (m) lub w godzinach (h). Parametr ten ma oczywisty wpływ na wydajność komputera, który przeprowadza testy, ale jeszcze ważniejszą sprawą jest oszukanie systemów wykrywania włamań. Jest to możliwe dlatego, że spowalniając testy wtapiamy się w naturalny ruch sieciowy, co utrudni systemom wykrycie przeprowadzonych testów.

7.1.9.5. Fragmentacja : flaga -f

Za pomocą fragmentacji można wysyłać skanujące pakiety w sposób pofragmentowany, czyli pakiet TCP dzielony jest na klika części. Takie działanie utrudni analizę przychodzących pakietów przez systemy wykrywania włamań strony badanej. Maksymalna wielkość pakietów została ustalona przez aplikację *nmap* na 8 bajtów. Oczywiście każdy fragment nagłówka TCP dostanie własny nagłówek IP, dzięki temu pakiet będzie mógł dotrzeć do adresata.

7.1.9.6. Decoy : flaga -D

Skanowanie typu Decoy polega na tym, że konkretne skanowanie wykonywane jest równolegle z różnych adresów IP nadawcy, co powoduje, że systemy wykrywania włamań nie będą widziały, który ze skanów pochodzi z autentycznego IP. Oczywiście, po analizie przebytej drogi na routerach można dotrzeć do autentycznego IP, z którego przeprowadzany był test, ale maskowanie testów w ten sposób powoduje kolejne utrudnienia dla systemów wykrywania włamań.

7.1.9.7. IPspoofing : flaga -S

Za pomocą tej opcji można ustawić źródłowy adres IP strony skanującej na dowolny. W ten sposób można wykonać atak IPspoofingu, co powoduje, że strona skanowana nie będzie wiedziała, kto wykonuje testy. W tym przypadku ewentualne pakiety zwrotne będą kierowane na podmieniony adres IP, czyli nie uzyskamy informacji na temat skanowanych portów. Tego typu skany można wykonywać, żeby wprowadzić w błąd stronę audytowaną, że jest ona skanowana przez konkretny adres IP danej organizacji, co może wprowadzić dodatkową dezorientację.

7.1.9.8. Zmiana portu źródłowego : flaga -g

Bardzo istotną funkcją jest możliwość zmiany adresu źródłowego nadawcy. Ten parametr jest istotny, ponieważ bardzo często ściany ognia filtrują ruch wyłącznie po porcie źródłowym nadawcy. Strona skanująca może wykonać test z portu źródłowego, reprezentującego port powszechnie używany (np. DNS:53) na dowolny port, który jest celem testów.

7.1.9.9. Wielkość danych : flaga –data-length

Standardowe testy wykonywane przez wysyłanie pakietów nie zawierają żadnych danych, czyli wysyłane są same nagłówki. Takie zachowanie aplikacji *nmap* jest wykonywane ze względu na wydajność przeprowadzanych skanowań. Natomiast, gdy chcemy, żeby przeprowadzane testy były podobne do normalnego ruchu sieciowego, wówczas należy określić dane, które będą dodawane do pakietów.

7.1.9.10. Zmiana adresu MAC : flaga –spoof-mac

Podobnie jak w przypadku fałszowania adresu IP nadawcy, możemy również zmienić adres MAC strony wykonującej skanowanie. Jako argument podawany jest zarówno adres MAC, jak i dowolny adres producenta karty sieciowej.

7.1.9.11. Zła suma kontrolna : flaga –badsum

Ciekawą opcją jest ustawianie błędnej sumy kontrolnej wysyłanego pakietu. System operacyjny, przyjmujący pakiety, automatycznie odrzuci te których suma kontrolna nie będzie się zgadzała. Inne zachowanie mają systemy wykrywania włamań, które często nie analizują tego pola. Dzięki temu, jeżeli wyślemy pakiety z błędną sumą i otrzymamy odpowiedź, będziemy wiedzieli, że tę odpowiedź wygenerował nie system operacyjny hosta, tylko system wykrywania włamań.

7.1.10. Wersje oprogramowania oraz system operacyjny

Celem pierwszego kroku skanowania jest określenie, jakie usługi są uruchomione na badanym hoście. Aplikacja *nmap* wskaże, jakie porty są otwarte i przypisze im nazwę zgodnie ze standardowym ich przyporządkowaniem. W kroku drugim należy dowiedzieć się czegoś więcej na temat potencjalnie uruchomionych usług na badanym hoście. W tym celu *nmap* łącząc się z konkretnymi usługami, zbiera banery skanowanych usług sieciowych i następnie przyporządkowuje je do banerów zgromadzonych w swojej bazie danych. Dzięki takiej analizie *nmap* określa prawdziwą nazwę protokołu warstwy aplikacji (np. http), nazwę aplikacji (np. Apache), wersję aplikacji, typ urządzenia (np. drukarka) oraz rodzaj systemu operacyjnego wraz z jego wersją. Poniżej zostaną przedstawione opcje skanowania, które umożliwiają uzyskanie wspomnianych informacji.

7.1.10.1. Wersja usługi : flaga -sV

Za pomocą tej flagi włączana jest detekcja wersji skanowanych usług. W tym przypadku będą analizowane dane wysyłane przez badanego hosta w odpowiedzi na skany.

7.1.10.2. Skanowanie RPC : flaga -sR

Usługi typu RPC (ang. Remote Procedure Call) są źródłem wielu luk bezpieczeństwa, dlatego ich wykrycie jest istotne. Za pomocą tej opcji będą wykonywane dodatkowe skanowania określone jako NULL SunRPC dla wszystkich wykrytych wcześniej otwartych portów. Jeżeli do danego portu przyporządkowane są konkretne usługi, wówczas wyświetlana jest ich nazwa wraz z wersją.

7.1.10.3. System operacyjny : flaga -O

System operacyjny wykrywany jest dzięki analizie budowy pakietów wysyłanych jako odpowiedzi na przeprowadzane testy skanujące. Systemy operacyjne posiadają swoje indywidualne metody budowy i sekwencji wysyłania pakietów w ramach realizacji stosu TCP/IP. Aplikacja *nmap* posiada bazę danych tych informacji i zebrane dane są analizowane, w wyniku czego określany jest system operacyjny. Dodatkowo, podczas przeprowadzanych testów systemu operacyjnego, skanowany host może dostarczać innych informacji, np. czas od ostatniego restartu lub zainstalowany service pack w przypadku systemu Windows.

7.1.10.4. Zgadywanie systemu operacyjnego : flaga –osscan-guess

Istnieje możliwość, że przeprowadzane testy systemu operacyjnego nie pozwolą na określenie, jaki system operacyjny jest skanowany. Dzieje się tak, gdy administrator zadbał o to, żeby system wraz z uruchomionymi usługami nie wysyłał standardowych odpowiedzi lub wysyłał je specjalnie spreparowane, tak żeby wprowadzać w błąd strony skanujące. W takim przypadku można określić opcję, która postara się zgadnąć system operacyjny, bazując na niepełnych lub sprzecznych danych.

7.2. Optymalizacja skanowania

7.2.1. Sztuczny ruch sieciowy: hping

`Hping` [7] jest narzędziem o wielkich możliwościach służącym do generowania oraz analizowania pakietów TCP/IP. Obecnie najnowszą wersją programu jest wersja 3 i program oznaczany jest jako `hping3`. Aplikacja obsługuje protokoły TCP, UDP, ICMP oraz RAW-IP i potrafi wygenerować i wysłać prawie dowolny pakiet w obrębie tych protokołów. Ponadto posiada wiele ciekawych opcji, takich jak tryb traceroute czy możliwość przesyłania plików kanałem 'ukrytym'. `hping` najczęściej wykorzystywany jest do testowania sieci, firewalli, systemów fragmentacji, protokołów, skanowania hostów, wykrywania i identyfikowania systemów oraz innych czynności związanych z audytem sieciowym. W tym podrozdziale zaprezentowano podstawowe możliwości pakietu `hping` w zakresie generowania sztucznego ruchu sieciowego.

Jedną z podstawowych opcji jest wysłanie pakietu TCP z zadanymi flagami. Flagi można ustawić za pomocą opcji:

```
-F -fin ustaw flagę FIN
-S -syn ustaw flagę SYN
-R -rst ustaw flagę RST
-P -push ustaw flagę PUSH
-A -ack ustaw flagę ACK
-U -urg ustaw flagę URG
-X -xmas ustaw flagę X (nieużywana, kod: 0x40)
-Y -ymas ustaw flagę Y (nieużywana, kod: 0x80)
```

Port ustawiamy za pomocą opcji `-p numer_portu`. W przypadku gdy nie ustawiono portu, pakiet zostanie wysłany na port 0. Chcąc wysłać pakiet przez dany interfejs sieciowy, należy użyć flagi `-I nazwa_interfejsu`.

Większość opcji `hping`a wymaga praw użytkownika root do manipulowania pakietami. Przykładowe wysłanie pakietu z flagą SYN na port 0 hosta 192.168.0.1 wygląda następująco:

```
#  hping3 -S 192.168.0.1
HPING 192.168.0.1 (wlan0 192.168.0.1): S set, 40 headers + 0 data bytes
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=RA seq=0 win=0 rtt=1.9 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=RA seq=1 win=0 rtt=1.6 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=RA seq=2 win=0 rtt=6.5 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=RA seq=3 win=0 rtt=1.7 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=RA seq=4 win=0 rtt=1.7 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=RA seq=5 win=0 rtt=1.5 ms
...
```

Polecenie wysyła w odstępie czasowym pakiety z ustawioną flagą SYN. Dodatkowo otrzymano informacje odnośnie do pakietów zwrotnych, takie jak: długość pakietu, adres źródłowy, czas życia pakietu, port źródłowy, opóźnienie, pozostałe flagi pakietu. Transmisja między komputerami wygląda następująco:

```
14:06:37.207568 IP linux.local.2944 > 192.168.0.1.0: Flags [S], seq 583281012, win 512,
length 0
14:06:37.209353 IP 192.168.0.1.0 > linux.local.2944: Flags [R.], seq 0, ack 583281013,
win 0, length 0
14:06:38.207757 IP linux.local.2945 > 192.168.0.1.0: Flags [S], seq 2020272793, win 512,
length 0
14:06:38.209319 IP 192.168.0.1.0 > linux.local.2945: Flags [R.], seq 0, ack 2020272794,
win 0, length 0
14:06:39.207865 IP linux.local.2946 > 192.168.0.1.0: Flags [S], seq 38001498, win 512,
length 0
14:06:39.214354 IP 192.168.0.1.0 > linux.local.2946: Flags [R.], seq 0, ack 38001499, win
0, length 0
```

Dzięki powyższemu listingowi można poznać dodatkowe szczegóły wymiany pakietów, takie jak: dokładny czas wysłania i otrzymania pakietu, port źródłowy naszego komputera, numery sekwencyjne poszczególnych pakietów.

Wysłanie pakietów z flagą RST lub FIN skutkuje brakiem odpowiedzi z drugiej strony. Przedstawia to poniższy listing:

```
# hping3 -R 192.168.0.1 -p 80
HPING 192.168.0.1 (wlan0 192.168.0.1): F set, 40 headers +
0 data bytes

- 192.168.0.1 hping statistic -
```

```
8 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Po wysłaniu ośmiu pakietów na port 80 nie otrzymano żadnej odpowiedzi. Badany system właśnie tak reaguje na pakiety z flagą FIN lub RST. Potwierdza to listing komunikacji:

```
14:39:51.596433 IP linux.local.2496 > 192.168.0.1.www:
Flags [R], seq 1513188863, win 512, length 0
14:39:52.596569 IP linux.local.2497 > 192.168.0.1.www:
Flags [R], seq 968123380, win 512, length 0
14:39:53.596699 IP linux.local.2498 > 192.168.0.1.www:
Flags [R], seq 1983996553, win 512, length 0
14:39:54.596820 IP linux.local.2499 > 192.168.0.1.www:
Flags [R], seq 1250340851, win 512, length 0
```

Jak widać pakiety wysyłane do komputera 192.168.0.1 na port 80 (www) zostają bez odpowiedzi, ponieważ mają ustawioną flagę RST.

Jak wspomniano wcześniej, za pomocą `hping`a możemy korzystać z innych protokołów. Przykładem jest wysyłanie pakietów UDP; należy w tym celu użyć opcji `-2`. Oto przykład badania hosta datagramami UDP:

```
# hping3 -2 192.168.0.1
HPING 192.168.0.1 (wlan0 192.168.0.1): udp mode set,
28 headers + 0 data bytes
ICMP Port Unreachable from ip=192.168.0.1 name=UNKNOWN
status=0 port=2117 seq=0
ICMP Port Unreachable from ip=192.168.0.1 name=UNKNOWN
status=0 port=2118 seq=1
ICMP Port Unreachable from ip=192.168.0.1 name=UNKNOWN
status=0 port=2119 seq=2
ICMP Port Unreachable from ip=192.168.0.1 name=UNKNOWN
status=0 port=2120 seq=3
ICMP Port Unreachable from ip=192.168.0.1 name=UNKNOWN
status=0 port=2121 seq=4
```

UDP jest protokołem bezpołączeniowym, na porcie 0 hosta 192.168.0.1 nie działa żadna usługa; uzyskano jedynie informację o niedostępności bez datagramów zwrotnych, co potwierdza listing ze 'śledzenia' powyższej komunikacji:

```
14:59:52.469808 IP linux.local.2117 > 192.168.0.1.0: UDP, length 0
14:59:53.469998 IP linux.local.2118 > 192.168.0.1.0: UDP, length 0
14:59:54.470125 IP linux.local.gsigatekeeper > 192.168.0.1.0: UDP, length 0
14:59:55.470253 IP linux.local.2120 > 192.168.0.1.0: UDP, length 0
```

Analogicznie do flagi `-1` można uzyskać pakiety ICMP. Poniższy listing prezentuje odpowiedzi hosta na pakiety ICMP:

```
# hping3 -1 192.168.0.1
HPING 192.168.0.1 (wlan0 192.168.0.1): icmp mode set, 28 headers + 0 data bytes
len=28 ip=192.168.0.1 ttl=64 id=43118 icmp_seq=0 rtt=2.0 ms
len=28 ip=192.168.0.1 ttl=64 id=43119 icmp_seq=1 rtt=1.6 ms
len=28 ip=192.168.0.1 ttl=64 id=43120 icmp_seq=2 rtt=1.7 ms
```

Aplikacja pozwala również na skanowanie portów komputera. Używamy do tego prefiksu ++ przed numerem portu. Oznacza to wysyłanie poszczególnych pakietów na kolejne porty, począwszy od ustalonego. W sytuacji, gdy ustalono dodatkowo flagę pakietu na SYN, będzie można wykryć obecność otwartego portu. Prezentuje to poniższy listing:

```
# hping3 -S 192.168.0.1 -p ++78
HPING 192.168.0.1 (wlan0 192.168.0.1): S set, 40 headers + 0 data bytes
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=78 flags=RA seq=0 win=0 rtt=1.9 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=79 flags=RA seq=1 win=0 rtt=1.6 ms
len=44 ip=192.168.0.1 ttl=64 DF id=0 sport=80 flags=SA seq=2 win=5840 rtt=1.8
ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=81 flags=RA seq=3 win=0 rtt=1.7 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=82 flags=RA seq=4 win=0 rtt=1.6 ms
```

Polecenie wysyła na kolejne porty, zaczynając od portu o numerze 78, pakiety SYN. Odpowiedzi z flagami RA oznaczają, iż pakiet został odrzucony i port najprawdopodobniej jest zamknięty. Odpowiedź SA oznacza port otwarty. Jak widać, od hosta 192.168.0.1 otrzymano odpowiedź z flagami SA z portu źródłowego 80. Oznacza to, że najpewniej komputer o adresie 192.168.0.1 ma uruchomiony serwer www.

Opcja `-a IP` służy do ustalania źródłowego adresu IP dla wysyłanych pakietów. Dodatkowo za pomocą opcji `-i` można ustalić liczbę wysyłanych pakietów w czasie jednej sekundy. Poniżej wygenerowano dużą liczbę fałszywych połączeń na port 80 badanego hosta:

```
# hping3 -a 192.168.0.155 -S 192.168.0.1 -i u10000 -p 80
HPING 192.168.0.1 (wlan0 192.168.0.1): S set, 40 headers + 0 data bytes

- 192.168.0.1 hping statistic -
561 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Podając się za 192.168.0.155, wysłano 561 pakietów z flagą SYN na port 80 komputera 192.168.0.1. Oto jak wygląda fragment takiego wysyłania fałszywych pakietów:

```
16:23:19.617380 IP 192.168.0.155.1138 > 192.168.0.1.www: Flags [S], seq 478053686, win
512, length 0
16:23:19.627438 IP 192.168.0.155.1139 > 192.168.0.1.www: Flags [S], seq 328202492, win
```

```
512, length 0
16:23:19.637495 IP 192.168.0.155.1140 > 192.168.0.1.www: Flags [S], seq 1930392010, win
512, length 0
16:23:19.647551 IP 192.168.0.155.1141 > 192.168.0.1.www: Flags [S], seq 1194871468, win
512, length 0
16:23:19.657613 IP 192.168.0.155.1142 > 192.168.0.1.www: Flags [S], seq 982934497, win
512, length 0
```

Do manipulowania pakietami `hping` służy jeszcze wielu opcji. Jednymi z ciekawszych mogą być: `-t ttl` (ustawienie czasu życia pakietu), `-m mtu` (ustawienie maksymalnej jednostki transferu), `-s port` (ustawienie portu źródłowego dla pakietu) czy `-d rozmiar` (ustawienie rozmiaru pakiet). Przykładem zastosowania może być testowanie odpowiedzi hosta na różne kombinacje tych opcji, np:

```
# hping3 -A 192.168.0.1 -s 11111 -d 1007 -t 22 -m 1024
HPING 192.168.0.1 (wlan0 192.168.0.1): A set, 40 headers + 1007 data bytes
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=R seq=0 win=0 rtt=2.4 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=R seq=1 win=0 rtt=2.7 ms
len=40 ip=192.168.0.1 ttl=64 DF id=0 sport=0 flags=R seq=2 win=0 rtt=2.8 ms
```

Wysyłano kolejne pakiety ACK zaczynając od portu źródłowego 1111, z wielkością danych pakietu 1007 bajtów, z czasem życia pakietu równym 22 oraz z MTU o wartości 1024. Badany host odpowiada pakietem RST, a szczegółowy fragment prezentuje następujący listing:

```
16:39:57.409427 IP linux.local.11116 > 192.168.0.1.0: Flags [.], seq 566196180:566197184,
ack 1991169248, win 512, length 1004
16:39:57.411866 IP 192.168.0.1.0 > linux.local.11116: Flags [R], seq 1991169248, win 0,
length 0
16:39:58.409625 IP linux.local.11117 > 192.168.0.1.0: Flags [.], seq 4018932636:4018933640,
ack 1942238656, win 512, length 1004
16:39:58.412365 IP 192.168.0.1.0 > linux.local.11117: Flags [R], seq 1942238656, win 0,
length 0
16:40:05.900055 IP linux.local.11111 > 192.168.0.1.0: Flags [.], seq 766813950:766814954,
ack 1875029729, win 512, length 1004
16:40:05.902383 IP 192.168.0.1.0 > linux.local.11111: Flags [R], seq 1875029729, win 0,
length 0
16:40:06.900595 IP linux.local.11112 > 192.168.0.1.0: Flags [.], seq 941150028:941151032,
ack 103787603, win 512, length 1004
16:40:06.903195 IP 192.168.0.1.0 > linux.local.11112: Flags [R], seq 103787603, win 0,
length 0
```

Widać odpowiedzi hosta poszczególnymi pakietami RST na odpowiednie porty.

7.3. Praktyczne zastosowania

Pierwszą czynnością, którą warto wykonać, jest sprawdzenie obecności hostów w sieci lokalnej. Skorzystano tutaj z metody ICMP SCAN opisanej wyżej.

```
# nmap -sP 192.168.0.1-255

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 14:59 CEST
Host 192.168.0.1 appears to be up.
MAC Address: 00:15:E9:02:71:B0 (D-Link)
Host 192.168.0.105 appears to be up.
Host 192.168.0.122 appears to be up.
MAC Address: 00:16:CE:87:88:3A (Hon Hai Precision Ind. Co.)
Nmap finished: 255 IP addresses (3 hosts up) scanned in 24.930 seconds
```

Jak widać, program zwrócił adresy trzech komputerów, z których jeden (192.168.0.105) jest naszym adresem. Poza informacjami o aktywnych hostach otrzymujemy informacje o adresach MAC. Dodatkowo skaner identyfikuje firmy, które są uprawnione do nadawania swoim produktom tych adresów. W przypadku np. routerów może stanowić to cenną informację. Adresy, które odpowiedziały na skanowanie ICMP, to adres bramy 192.168.0.1 oraz zdalny host 192.168.0.122.

Następnie przetestowano host 192.168.0.122. W celu określenia, jakie protokoły obsługuje host, użyto poniższego polecenia:

```
# nmap -sO 192.168.0.122

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 15:54 CEST
Interesting protocols on 192.168.0.122:
Not shown: 251 closed protocols
PROTOCOL STATE SERVICE
1 open icmp
2 open|filtered igmp
6 open tcp
17 filtered udp
136 open|filtered udplite
```

Dane zwrócone przez program mówią o dosyć standardowym zestawie protokołów, obsługiwanych przez komputer. Kolejnym krokiem może być przeskanowanie interesujących portów na badanej maszynie. W tym celu można użyć np. skanowania TCP NULL.

```
# nmap -sN 192.168.0.122 -p 21,22,23,25,80
Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 15:45 CEST
Interesting ports on 192.168.0.122:
PORT STATE SERVICE
21/tcp open|filtered ftp
22/tcp open|filtered ssh
```

```
23/tcp closed telnet
25/tcp closed smtp
80/tcp open|filtered http
MAC Address: 00:16:CE:87:88:3A (Hon Hai Precision Ind. Co.)

Nmap finished: 1 IP address (1 host up) scanned in 7.928 seconds
```

Ważnym elementem powyższego polecenia jest użycie opcji *-p*, za pomocą której definiujemy porty sprawdzone w procesie skanowania. Pozostałe porty nie zostaną zbadane. W przedstawionym przypadku interesującymi portami są: 21,22,23,25,80. Wynik skanera informuje nas o stanie portów. Porty 23 oraz 25 są zamknięte, natomiast pozostałe mogą być otwarte bądź filtrowane. W celu weryfikacji tych wyników można użyć skanowania TCP SYN:

```
# nmap -sS 192.168.0.122 -p 21,22,23,25,80

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 15:45 CEST
Interesting ports on 192.168.0.122:
PORT STATE SERVICE
21/tcp open ftp
22/tcp open ssh
23/tcp closed telnet
25/tcp closed smtp
80/tcp open http
MAC Address: 00:16:CE:87:88:3A (Hon Hai Precision Ind. Co.)

Nmap finished: 1 IP address (1 host up) scanned in 6.736 seconds
```

Powyższe skanowanie rozwiało wątpliwości co do stanów danych portów. Otwartymi portami (z naszego zbioru) są 21, 22 oraz 80.

W powyższym przykładzie skan obejmował zdefiniowane przez użytkownika porty, jednak w sytuacji, gdy nie wiadomo jakich usług spodziewać się na sprawdzanym hoście, warto użyć flagi `-F` programu, która zbada podstawowe numery portów. To przedstawi między innymi zaprezentowany niżej przykład:

```
# nmap -F -vv -scan-delay 10s -max-retries 1 192.168.0.122

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 17:07 CEST
Initiating ARP Ping Scan at 17:07
Scanning 192.168.0.122 [1 port]
Completed ARP Ping Scan at 17:07, 10.01s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 17:07
Completed Parallel DNS resolution of 1 host. at 17:07, 6.52s elapsed
Initiating SYN Stealth Scan at 17:07
Scanning 192.168.0.122 [1256 ports]
SYN Stealth Scan Timing: About 0.32% done
Discovered open port 21/tcp on 192.168.0.122
```

```
Discovered open port 80/tcp on 192.168.0.122
Discovered open port 443/tcp on 192.168.0.122
Discovered open port 22/tcp on 192.168.0.122
...
```

Polecenie to sprawdza standardowe porty. Do opcji skanera dodatkowo dodano flagę `-vv`, która odpowiada za dodatkowe informacje dla osoby skanującej, oraz opcję `-scan-delay 10s`, która wymusza 10 sekund przerwy między badaniem kolejnych portów. Opcja ta przydaje się np. do ominięcia systemów Firewall/IDS. Ograniczyliśmy również odpytywania tych samych portów kilkakrotnie. Program w trybie rzeczywistym informuje nas o wykrytych otwartych portach (21,80,443,22). Oto, co może pokazać program nasłuchujący przychodzący ruch sieciowy na skanowanej maszynie:

```
17:34:39.773893 IP 192.168.0.105.45065 > laptop.local.ftp:
S 2163566778:2163566778(0) win 2048 <mss 1460>
17:34:39.776600 IP 192.168.0.105.45065 > laptop.local.ftp:
R 2163566779:2163566779(0) win 0
17:34:49.777686 IP 192.168.0.105.45065 > laptop.local.ldaps: S 2163566778:2163566778(0)
win 4096 <mss 1460>
17:35:09.785391 IP 192.168.0.105.45065 > laptop.local.domain: S 2163566778:2163566778(0)
win 3072 <mss 1460>
17:35:19.787463 IP 192.168.0.105.45065 > laptop.local.telnet: S 2163566778:2163566778(0)
win 1024 <mss 1460>
17:35:29.791095 IP 192.168.0.105.45065 > laptop.local.smtp: S 2163566778:2163566778(0)
win 3072 <mss 1460>
17:35:39.795158 IP 192.168.0.105.45065 > laptop.local.www:
S 2163566778:2163566778(0) win 4096 <mss 1460>
17:35:39.797813 IP 192.168.0.105.45065 > laptop.local.www:
R 2163566779:2163566779(0) win 0
17:35:49.798841 IP 192.168.0.105.45065 > laptop.local.1723: S 2163566778:2163566778(0)
win 1024 <mss 1460>
17:35:59.801715 IP 192.168.0.105.45065 > laptop.local.ldap: S 2163566778:2163566778(0)
win 3072 <mss 1460>
17:36:09.805507 IP 192.168.0.105.45065 > laptop.local.auth: S 2163566778:2163566778(0)
win 2048 <mss 1460>
17:36:19.807497 IP 192.168.0.105.45067 > laptop.local.ldaps: S 2180344250:2180344250(0)
win 1024 <mss 1460>
17:36:29.811213 IP 192.168.0.105.45065 > laptop.local.256:
S 2163566778:2163566778(0) win 3072 <mss 1460>
17:36:39.814108 IP 192.168.0.105.45065 > laptop.local.rtsp: S 2163566778:2163566778(0)
win 4096 <mss 1460>
17:36:49.817953 IP 192.168.0.105.45065 > laptop.local.3389: S 2163566778:2163566778(0)
win 1024 <mss 1460>
17:36:59.821811 IP 192.168.0.105.45065 > laptop.local.https: S 2163566778:2163566778(0)
win 3072 <mss 1460>
17:36:59.826447 IP 192.168.0.105.45065 > laptop.local.https: R 2163566779:2163566779(0)
win 0
17:37:09.825592 IP 192.168.0.105.45065 > laptop.local.1492: S 2163566778:2163566778(0)
win 1024 <mss 1460>
17:37:19.829515 IP 192.168.0.105.45065 > laptop.local.1547: S 2163566778:2163566778(0)
win 3072 <mss 1460>
17:37:29.835329 IP 192.168.0.105.45065 > laptop.local.1669: S 2163566778:2163566778(0)
win 1024 <mss 1460>
17:37:39.837240 IP 192.168.0.105.45065 > laptop.local.198:
```

```
S 2163566778:2163566778(0) win 3072 <mss 1460>
17:37:49.841095 IP 192.168.0.105.45065 > laptop.local.6103: S 2163566778:2163566778(0)
win 3072 <mss 1460>
17:37:59.843964 IP 192.168.0.105.45065 > laptop.local.898:
S 2163566778:2163566778(0) win 3072 <mss 1460>
```

Komputerem skanującym jest 192.168.0.105. Jak widać kolejne pakiety wysyłane są w odstępach co dziesięć sekund. Oczywiście wydłuża to znacznie skanowanie i sensowne jest użycie tej opcji wraz z niedużym zakresem portów, ale technika ta posiada pewne zalety. Przy sporym, dodatkowym ruchu sieciowym pakiety te mogą zostać przeoczone nie tylko przez człowieka, ale również przez system Firewall/IDS. Opcja `-max-retries 1` wyklucza kilkukrotne skanowanie jednego portu. Przy odpowiednim dobraniu parametrów `-scan-delay`, `-max-retries` oraz niewielkim zakresie portów `-p` skanowanie może okazać się bardzo "ciche" oraz skuteczne.

Sama informacja o stanie danego portu nie jest jeszcze pełna. Usługi mogą działać na innych portach niż standardowe (np. 21 ftp może działać na porcie 80 http). Także w tym momencie nie mamy pewności, jaka usługa faktycznie komunikuje się za pomocą jakiego portu. Poza tym w większości powyższych skanowań nie brano pod uwagę innych portów niż wyświetlone za pomocą flagi `-p` programu `nmap`. Mając już listę niektórych usług dostępnych na serwerze, należy określić wersję oprogramowania. Dobrze byłoby połączyć te dwa zadania, czyli przeskanowanie standardowych portów bez konieczności ich listowania oraz wstępne sprawdzenie, co tak naprawdę pod danym portem się kryje. W tym celu skorzystamy z flagi `-sV`. Dodatkowo użyjemy opcji *-f* odpowiedzialnej za fragmentowanie pakietów.

```
# nmap -sV -f 192.168.0.122

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 16:48 CEST
Interesting ports on 192.168.0.122:
Not shown: 1692 closed ports
PORT STATE SERVICE VERSION
21/tcp open ftp?
22/tcp open ssh OpenSSH 4.6p1 Debian 5ubuntu0.6 (protocol 2.0)
80/tcp open http Apache httpd 2.2.8 Unix DAV/2 mod_ssl/2.2.8
OpenSSL/0.9.8e PHP/5.2.5 mod_apreq2-20051231/2.6.0 mod_perl/2.0.2 Perl/v5.10.0
443/tcp open ssl/http Apache httpd 2.2.8 ((Unix) DAV/2
mod_ssl/2.2.8 OpenSSL/0.9.8e PHP/5.2.5 mod_apreq2-20051231/2.6.0 mod_perl/2.0.2
Perl/v5.10.0)
3306/tcp open mysql MySQL (unauthorized)
MAC Address: 00:16:CE:87:88:3A (Hon Hai Precision Ind. Co.)
Servic e Info: OS: Linux
```

Skanowanie przyniosło wiele informacji o używanym na komputerze oprogramowaniu. Widać wersje oprogramowania, dowiadujemy się również, że skanowany system jest pod kontrolą Linuksa (można przypuszczać, iż jest to Ubuntu). Przy okazji wykryto dostępną kolejną usługę w postaci serwera MySQL. Wiele szczegółowych informacji podał nam serwer http. Jest nim Apache w wersji 2.2.8, z dostępnymi

modułami mod_apreq2, mod_perl, mod_dav oraz z kryptografią opartą na bibliotece OpenSSL w wersji 0.9.8e (mod_ssl). Znamy wersje modułów oraz wersję języka Perl, zainstalowanego na serwerze.
Opcja `-f` opóźnia skanowanie, ale pozwala w pewnych sytuacjach ominąć systemy Firewall oraz IDS. Po stronie skanowanej fragment ruchu przychodzącego, podczas tego skanowania, wygląda następująco:

```
...
17:47:38.409942 IP 192.168.0.105.53590 > laptop.local.ftp:
R 3004592011:3004592011(0) win 0
17:47:38.411182 IP 192.168.0.105.53590 > laptop.local.ftp:
R 3004592011:3004592011(0) win 0
17:47:43.391709 IP 192.168.0.105.53593 > laptop.local.ftp:
F 24:24(0) ack 1 win 183 <nop,nop,timestamp 23773821 9034748>
17:47:43.392028 IP 192.168.0.105.53594 > laptop.local.ftp:
S 3304269998:3304269998(0) win 5840 <mss 1460,sackOK,timestamp
23773821 0,nop,wscale 5>
17:47:43.394759 IP 192.168.0.105.53594 > laptop.local.ftp:
. ack 1 win 183 <nop,nop,timestamp 23773824 9035998>
17:47:43.395888 IP 192.168.0.105.53594 > laptop.local.ftp:
P 1:61(60) ack 1 win 183 <nop,nop,timestamp 23773824 9035998>
17:47:43.410984 IP 192.168.0.105.53591 > laptop.local.ftp:
R 3082401686:3082401686(0) win 0
17:47:43.412244 IP 192.168.0.105.53591 > laptop.local.ftp:
R 3082401686:3082401686(0) win 0
17:47:48.396503 IP 192.168.0.105.53594 > laptop.local.ftp:
F 61:61(0) ack 1 win 183 <nop,nop,timestamp 23778825 9035999>
17:47:48.398645 IP 192.168.0.105.53595 > laptop.local.ftp:
S 3388790076:3388790076(0) win 5840 <mss 1460,sackOK,timestamp
23778825 0,nop,wscale 5>
17:47:48.409696 IP 192.168.0.105.53595 > laptop.local.ftp:
. ack 1 win 183 <nop,nop,timestamp 23778832 9037250>
17:47:48.414004 IP 192.168.0.105.53595 > laptop.local.ftp:
P 1:13(12) ack 1 win 183 <nop,nop,timestamp 23778837 9037250>
17:47:48.418264 IP 192.168.0.105.53592 > laptop.local.ftp:
R 3158979442:3158979442(0) win 0
17:47:48.419288 IP 192.168.0.105.53592 > laptop.local.ftp:
R 3158979442:3158979442(0) win 0
17:47:53.409143 IP 192.168.0.105.53595 > laptop.local.ftp:
F 13:13(0) ack 1 win 183 <nop,nop,timestamp 23783837 9037253>
...
```

Jak widać opcja `-f` odpowiedzialna jest za zupełnie inny przebieg skanowania. Komputer 192.168.0.105 potrzebuje wysłania wielu fragmentów pakietu, by sprawdzić jeden port (ftp).

Po przeprowadzeniu powyższych testów można być prawie pewnym co do systemu operacyjnego hosta 192.168.0.122 (laptop.local). Jest nim Linux Ubuntu. Jak

pamiętamy z pierwszego skanowania, oprócz tego hosta w sieci lokalnej dostępna jest również brama sieciowa o adresie IP 192.168.0.1. Teraz należy przyjrzeć się temu systemowi.
Będziemy chcieli określić usługi bramy sieciowej oraz jej system operacyjny. Z poprzednich skanowań można przypuszczać, iż jest to urządzenie firmy D-Link (adres MAC na to wskazywał).

```
# nmap -O 192.168.0.1

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 16:05 CEST
Interesting ports on 192.168.0.1:
Not shown: 1695 closed ports
PORT STATE SERVICE
80/tcp open http
515/tcp open printer
MAC Address: 00:15:E9:02:71:B0 (D-Link)
No exact OS matches for host (If you know what OS is running on it,
see http://insecure.org/nmap/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=4.20%D=4/13%OT=80%CT=1%CU=36847%PV=Y%DS=1%G=Y%
M=0015E9%TM=4BC47A2
OS:A%P=i686-redhat-linux-gnu)SEQ(SP=0%GCD=64%ISR=50%TI=BI%II=
BI%SS=S%TS=U)O
OS:PS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)WIN(W1=
16D0%W2=16D0%W
OS:3=16D0%W4=16D0%W5=16D0%W6=16D0)ECN(R=Y%DF=N%T=40%W=16D0%
O=M5B4%CC=N%Q=)T
OS:1(R=Y%DF=N%T=40%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=N%T=40%
W=0%S=Z%A=S%F=AR%
OS:O=%RD=0%Q=)T3(R=Y%DF=N%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%
Q=)T4(R=Y%DF=N%T=4
OS:0%W=0%S=A%A=S%F=AR%O=%RD=0%Q=)T5(R=Y%DF=N%T=40%W=0%S=
Z%A=S+%F=AR%O=%RD=0
OS:%Q=)T6(R=Y%DF=N%T=40%W=0%S=A%A=S%F=AR%O=%RD=0%Q=)T7(R=
Y%DF=N%T=40%W=0%S=
OS:Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=40%TOS=0%IPL=38%U
N=0%RIPL=G%RID=G%R
OS:IPCK=G%RUCK=G%RUL=G%RUD=G)IE(R=Y%DFI=N%T=40%TOSI=Z%CD=S%
SI=S%DLI=S)


Network Distance: 1 hop

OS detection performed. Please report any incorrect results
at http://insecure.org/nmap/submit/ .
Nmap finished: 1 IP address (1 host up) scanned in 19.200
seconds
```

Skanowanie za pomocą omówionej wcześniej opcji `-O` ustaliło otwarte porty bramy, adres MAC i odległość od naszego systemu. Otwartymi portami bramy są serwer http oraz serwer wydruku, a odległość od niej do naszego hosta to 1 przeskok. Niestety, jak widać *nmap* nie był w stanie zidentyfikować systemu operacyjnego hosta. Otrzymaliśmy jedynie odcisk TCP/IP, który w przyszłości może pomóc deweloperom programu w dokładnym ustaleniu systemu. W tej sytuacji można użyć opcji `-osscan-guess`, która spróbuje na podstawie tych danych określić najbliższy podobny system do badanego. Oto wyniki skanowania z próbą odgadnięcia systemu na skanowanym hoście:

```
# nmap -O -osscan-guess 192.168.0.1

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 16:44 CEST
Interesting ports on 192.168.0.1:
Not shown: 1695 closed ports
PORT STATE SERVICE
80/tcp open http
515/tcp open printer
MAC Address: 00:15:E9:02:71:B0 (D-Link)
Device type: WAP|firewall|broadband router
Running (JUST GUESSING) : D-Link embedded (92%),
WatchGuard embedded (88%), Linksys embedded (85%)
Aggressive OS guesses: D-Link DI-824VUP Wireless VPN Router (92%),
Watchguard Firebox X5w firewall/WAP (88%), Linksys BEFSR41 WAP (85%)
No exact OS matches for host (If you know what OS is running on it,
see http://insecure.org/nmap/submit/ ).
TCP/IP fingerprint:
OS:SCAN(V=4.20%D=4/13%OT=80%CT=1%CU=35111%PV=Y%DS=1%G=Y%
M=0015E9%TM=4BC4837
OS:4%P=i686-redhat-linux-gnu)SEQ(SP=0%GCD=64%ISR=50%TI=BI%II
=BI%SS=S%TS=U)O
OS:PS(O1=M5B4%O2=M5B4%O3=M5B4%O4=M5B4%O5=M5B4%O6=M5B4)WIN(W1
=16D0%W2=16D0%W
OS:3=16D0%W4=16D0%W5=16D0%W6=16D0)ECN(R=Y%DF=N%T=40%W=16D0%
O=M5B4%CC=N%Q=)T
OS:1(R=Y%DF=N%T=40%S=O%A=S+%F=AS%RD=0%Q=)T2(R=Y%DF=N%T=40
%W=0%S=Z%A=S%F=AR%
OS:O=%RD=0%Q=)T3(R=Y%DF=N%T=40%W=0%S=Z%A=S+%F=AR%O=%RD=0
%Q=)T4(R=Y%DF=N%T=4
OS:0%W=0%S=A%A=S%F=AR%O=%RD=0%Q=)T5(R=Y%DF=N%T=40%W=0%
S=Z%A=S+%F=AR%O=%RD=0
OS:%Q=)T6(R=Y%DF=N%T=40%W=0%S=A%A=S%F=AR%O=%RD=0%Q=)T7(
R=Y%DF=N%T=40%W=0%S=
OS:Z%A=S+%F=AR%O=%RD=0%Q=)U1(R=Y%DF=N%T=40%TOS=0%IPL=38%
UN=0%RIPL=G%RID=G%R
OS:IPCK=G%RUCK=G%RUL=G%RUD=G)IE(R=Y%DFI=N%T=40%TOSI=Z%CD=S%
```

```
SI=S%DLI=S)


Network Distance: 1 hop

OS detection performed. Please report any incorrect results at
http://insecure.org/nmap/submit/ .
Nmap finished: 1 IP address (1 host up) scanned in 20.179 seconds
```

Aplikacja `nmap` w tym skanowaniu podała procentową pewność obecności na badanym komputerze jednego z trzech systemów: D-Link DI-824VUP Wireless VPN Router (92%), Watchguard Firebox X5w firewall/WAP (88%), Linksys BEFSR41 WAP (85%). Biorąc pod uwagę wynik tego skanowania oraz adres MAC, wszystko wskazuje na to, iż badanym hostem jest urządzenie firmy D-Link.

Kolejnym ciekawym zastosowaniem *nmap*a jest skanowanie, w którym generowany jest dodatkowy ruch sieciowy, by ukryć swoje działanie. Skanowanie z opcją `-D` pozwoli nam wysyłać zreplikowane skanowanie do hosta, podając się za kilka hostów. W poniższym przykładzie użyto komendy `nmap -D 192.168.0.111 192.168.0.122` , tak by skanowany host miał problem z odróżnieniem faktycznie skanującego go systemu:

```
# nmap -D 192.168.0.111 192.168.0.122

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 18:55 CEST
Interesting ports on 192.168.0.122:
Not shown: 1692 closed ports
PORT STATE SERVICE
21/tcp open ftp
22/tcp open ssh
80/tcp open http
443/tcp open https
3306/tcp open mysql
```

Flaga `-D 192.168.0.111` nakazuje skanerowi zreplikować pakiety i zmienić w nich adres źródłowy na 192.168.0.111. Wówczas skanowany host, (192.168.0.122) śledząc ruch wejściowy, ujrzy:

```
18:57:19.948267 IP 192.168.0.105.43726 > laptop.local.canna:
S 1713102994:1713102994(0) win 1024 <mss 1460>
18:57:19.948944 IP 192.168.0.111.43726 > laptop.local.canna:
S 1713102994:1713102994(0) win 1024 <mss 1460>
18:57:19.953877 IP 192.168.0.105.43726 > laptop.local.850:
S 1713102994:1713102994(0) win 1024 <mss 1460>
18:57:19.954563 IP 192.168.0.111.43726 > laptop.local.850:
S 1713102994:1713102994(0) win 2048 <mss 1460>
18:57:19.960205 IP 192.168.0.105.43726 > laptop.local.5530:
S 1713102994:1713102994(0) win 4096 <mss 1460>
```

```
18:57:19.962211 IP 192.168.0.111.43726 > laptop.local.5530:
S 1713102994:1713102994(0) win 3072 <mss 1460>
18:57:19.965671 IP 192.168.0.105.43726 > laptop.local.522:
S 1713102994:1713102994(0) win 2048 <mss 1460>
18:57:19.966336 IP 192.168.0.111.43726 > laptop.local.522:
S 1713102994:1713102994(0) win 2048 <mss 1460>
18:57:19.972353 IP 192.168.0.105.43726 > laptop.local.1373:
S 1713102994:1713102994(0) win 1024 <mss 1460>
18:57:19.974366 IP 192.168.0.111.43726 > laptop.local.1373:
S 1713102994:1713102994(0) win 2048 <mss 1460>
18:57:19.992671 IP 192.168.0.105.43726 > laptop.local.1349:
S 1713102994:1713102994(0) win 3072 <mss 1460>
18:57:19.994945 IP 192.168.0.111.43726 > laptop.local.1349:
S 1713102994:1713102994(0) win 3072 <mss 1460>
18:57:19.999714 IP 192.168.0.105.43726 > laptop.local.816:
S 1713102994:1713102994(0) win 3072 <mss 1460>
18:57:20.000403 IP 192.168.0.111.43726 > laptop.local.816:
S 1713102994:1713102994(0) win 2048 <mss 1460>
18:57:20.004793 IP 192.168.0.105.43726 > laptop.local.311:
S 1713102994:1713102994(0) win 3072 <mss 1460>
18:57:20.005467 IP 192.168.0.111.43726 > laptop.local.311:
S 1713102994:1713102994(0) win 4096 <mss 1460>
...
```

Skanowany host będzie miał problem z jednoznacznym ustaleniem, która maszyna go skanuje, ponieważ pakiety przychodzą naprzemiennie z dwóch adresów IP. Oczywiście, w celu zmylenia komputera docelowego można podać więcej adresów IP z opcją `-D`.

Ostatnim przykładem użycia skanera *nmap* jest sfałszowanie próby skanowania. Pozwala nam na to opcja `-S`. Dzięki niej można ustawić adres źródłowy pakietów skanujących. Po zmianie tego adresu na inny (nie nasz), nie będziemy w stanie sprawdzić, które porty są dostępne. Naszym celem będzie imitacja skanowania ze strony bramy hosta 192.168.0.122

```
# nmap -e eth1 -S 192.168.0.1 -badsum 192.168.0.122

WARNING: If -S is being used to fake your source address, you may
also have to use -e <interface> and -P0 . If you are using it to
specify your real source address, you can ignore this
warning.

Starting Nmap 4.20 ( http://insecure.org ) at 2010-04-13 19:11 CEST
```

Używając tej opcji, najczęściej będziemy musieli zdefiniować użyty do transmisji interfejs sieciowy. Realizuje to flaga `-e`. Dodatkowo użyto opcję `-badsum` tak, by zwiększyć szansę odnotowania tego skanowania po stronie skanowanej. Analizując ruch wejściowy, host 192.168.0.122 jest przekonany, że skanuje go brama sieciowa

(adres 192.168.0.1):

```
19:14:01.302510 IP 192.168.0.1.33124 > laptop.local.2023:
S 2016873625:2016873625(0) win 1024 <mss 1460>
19:14:01.304616 IP 192.168.0.1.33124 > laptop.local.bprd:
S 2016873625:2016873625(0) win 4096 <mss 1460>
19:14:01.305664 IP 192.168.0.1.33124 > laptop.local.891:
S 2016873625:2016873625(0) win 4096 <mss 1460>
19:14:01.314524 IP 192.168.0.1.33124 > laptop.local.156:
S 2016873625:2016873625(0) win 3072 <mss 1460>
19:14:01.320583 IP 192.168.0.1.33124 > laptop.local.82:
S 2016873625:2016873625(0) win 1024 <mss 1460>
19:14:01.321600 IP 192.168.0.1.33124 > laptop.local.568:
S 2016873625:2016873625(0) win 3072 <mss 1460>
19:14:01.322824 IP 192.168.0.1.33124 > laptop.local.953:
S 2016873625:2016873625(0) win 4096 <mss 1460>
19:14:01.393653 IP 192.168.0.1.33125 > laptop.local.1007:
S 2016808088:2016808088(0) win 4096 <mss 1460>
19:14:01.394729 IP 192.168.0.1.33125 > laptop.local.9992:
S 2016808088:2016808088(0) win 4096 <mss 1460>
19:14:01.402529 IP 192.168.0.1.33125 > laptop.local.779:
S 2016808088:2016808088(0) win 4096 <mss 1460>
19:14:01.404497 IP 192.168.0.1.33125 > laptop.local.2023:
S 2016808088:2016808088(0) win 2048 <mss 1460>
19:14:01.406612 IP 192.168.0.1.33125 > laptop.local.891:
S 2016808088:2016808088(0) win 4096 <mss 1460>
19:14:01.407818 IP 192.168.0.1.33125 > laptop.local.bprd:
S 2016808088:2016808088(0) win 2048 <mss 1460>
19:14:01.416507 IP 192.168.0.1.33125 > laptop.local.156:
S 2016808088:2016808088(0) win 3072 <mss 1460>
```

Spis rysunków

Spis skryptów

Bibliografia

[1] M. Stutz: *Linux - Najlepsze przepisy*, Helion, Gliwice, 2005

[2] A. Robbins, N. H. F. Beebe: *Programowanie skryptów powłoki*, Helion, Gliwice, 2006

[3] Openssl Project: http://www.openssl.org/

[4] B.Księżopolski, Z.Kotulski - Bezpieczeństwo e-urzędu - Centrum Certyfikacji, Współczesne Problemy Systemów Czasu Rzeczywistego, Rozdział 31, str. 349-359, Wydawnictwa Naukowo-Techniczne, Warszawa 2004

[5] Iptables: http://www.netfilter.org/

[6] Nmap Home Page: nmap.org/

[7] Hping - Active Network Security Tool: www.hping.org/