



PSIR

Laboratorium

Ćwiczenie 1 2024.10.25
Temat: Podstawy tworzenia i uruchamiania oprogramowania sieciowego bazującego na gniazdach w systemie Linux.

Warunki wstępne:

Zapoznać się z:
Materiałami wykładowymi.

1. Przygotowanie środowiska do pracy

Dla celów ćwiczeń laboratoryjnych przygotowano obraz maszyny wirtualnej działającej pod wirtualizatorem VirtualBOX[1].

Dla poprawnego działania maszyn wirtualnych konieczne jest zainstalowanie programu: VirtualBox w wersji przynajmniej 6.1 lub nowszej, można taki program pobrać ze strony:

<https://www.virtualbox.org/wiki/Downloads>

Dla systemu Windows, np.:

<https://download.virtualbox.org/virtualbox/7.0.10/VirtualBox-7.0.10-158379-Win.exe>

dla systemów zgodnych z Linux metoda zależy od tzw. dystrybucji, informacje podano na stronie:

https://www.virtualbox.org/wiki/Linux_Downloads

Pliki z obrazami maszyn wirtualnych (rozszerzenie OVA) dla wirtualizatora VirtualBox, dostępne są pod adresem:

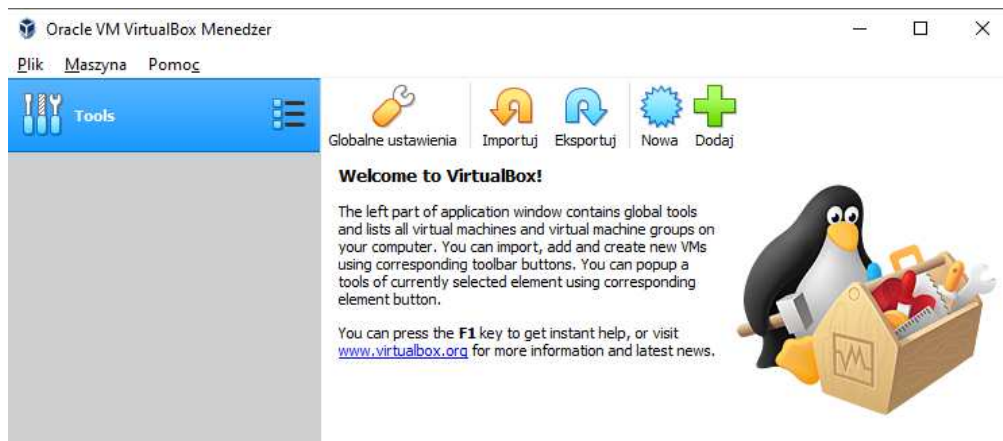
https://secure.tele.pw.edu.pl/~apruszko/psir/psir23z_20230908_1052.ova

Uwaga!

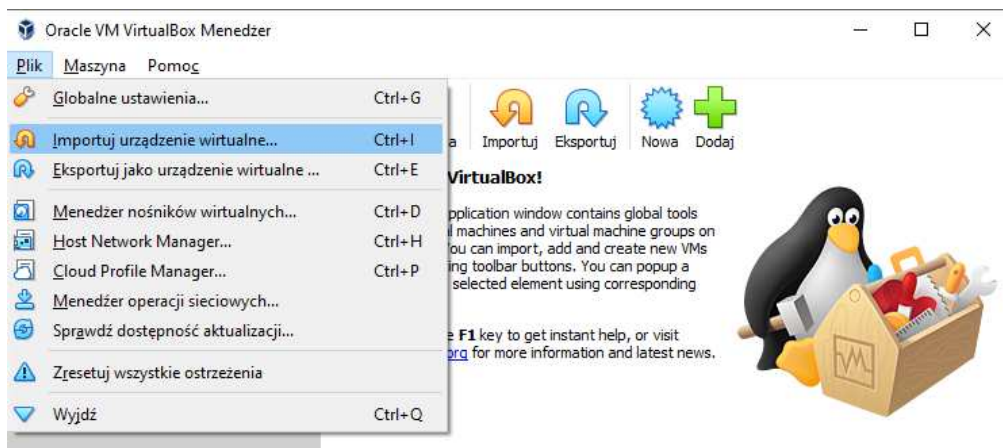
Plik `psir23z_20230908_1052.ova` „waży” około 1,2GB, więc jego pobieranie może zająć pewien czas a miejsce do którego zapisujemy ten plik musi posiadać odpowiednią wolną przestrzeń. Po zaimportowaniu maszyna wirtualna zajmie prawie 3GB. Dodatkowo dla sprawniejszego i pewniejszego tworzenia i testowania kodów tworzonych podczas laboratorium wartym rozważenia jest użycie dwóch sklonowanych maszyn wirtualnych (opis ich tworzenia w dalszej części dokumentu) co może wymagać dwukrotnie większej przestrzeni dyskowej.

2.Import i ustawienie maszyny wirtualnej

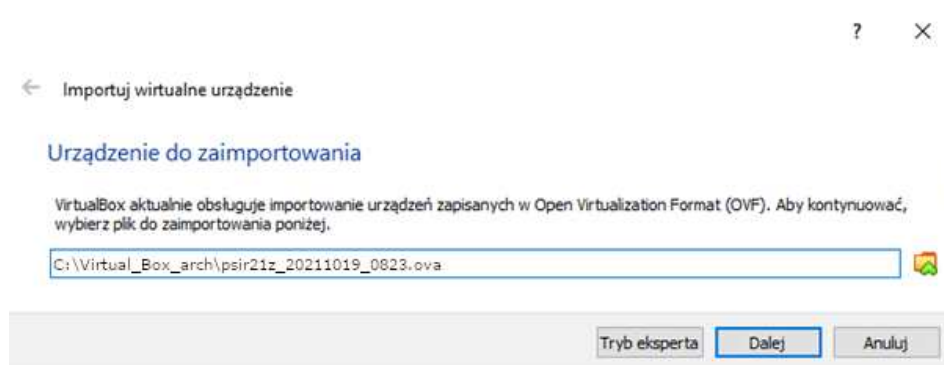
Po pobraniu na dysk lokalny swojego komputera pliku z obrazem maszyny wirtualnej (`psir23z_20230908_1052.ova`), możliwe jest ich zaimportowanie. Dla zaimportowania maszyny wirtualnej należy uruchomić menadżer VirtualBox którego główne okno wyglądać może tak:



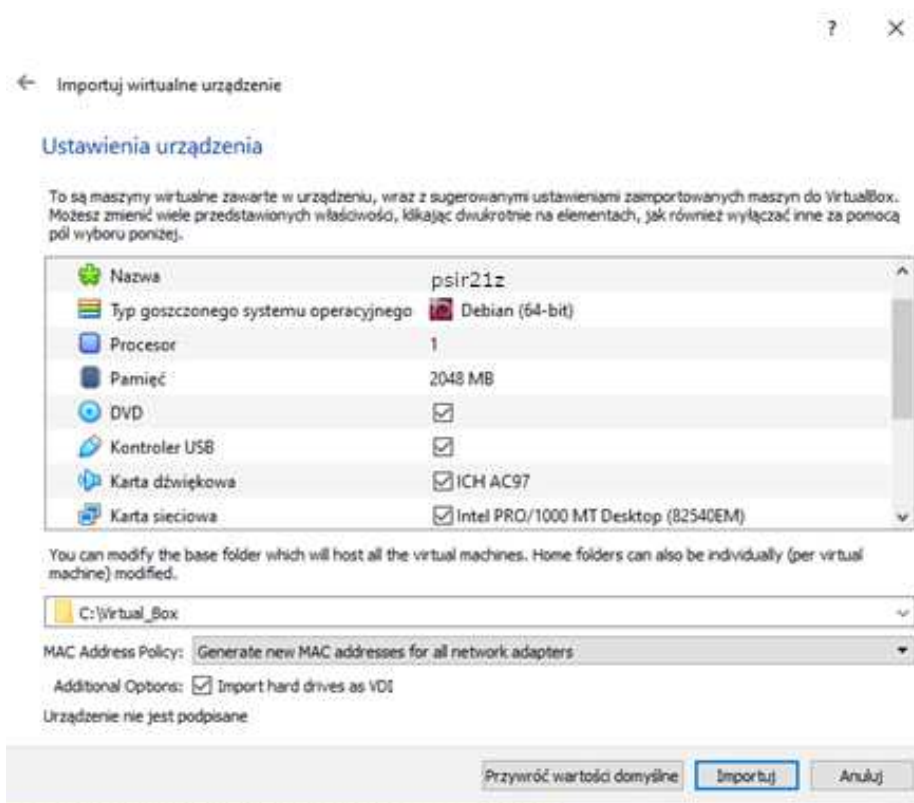
A następnie wybieramy „File” → ”Importuj urządzenie wirtualne”:



po czym wybieramy nazwę pliku z obrazem do zaimportowania (tutaj: psir23z_20230908_1052.ova):



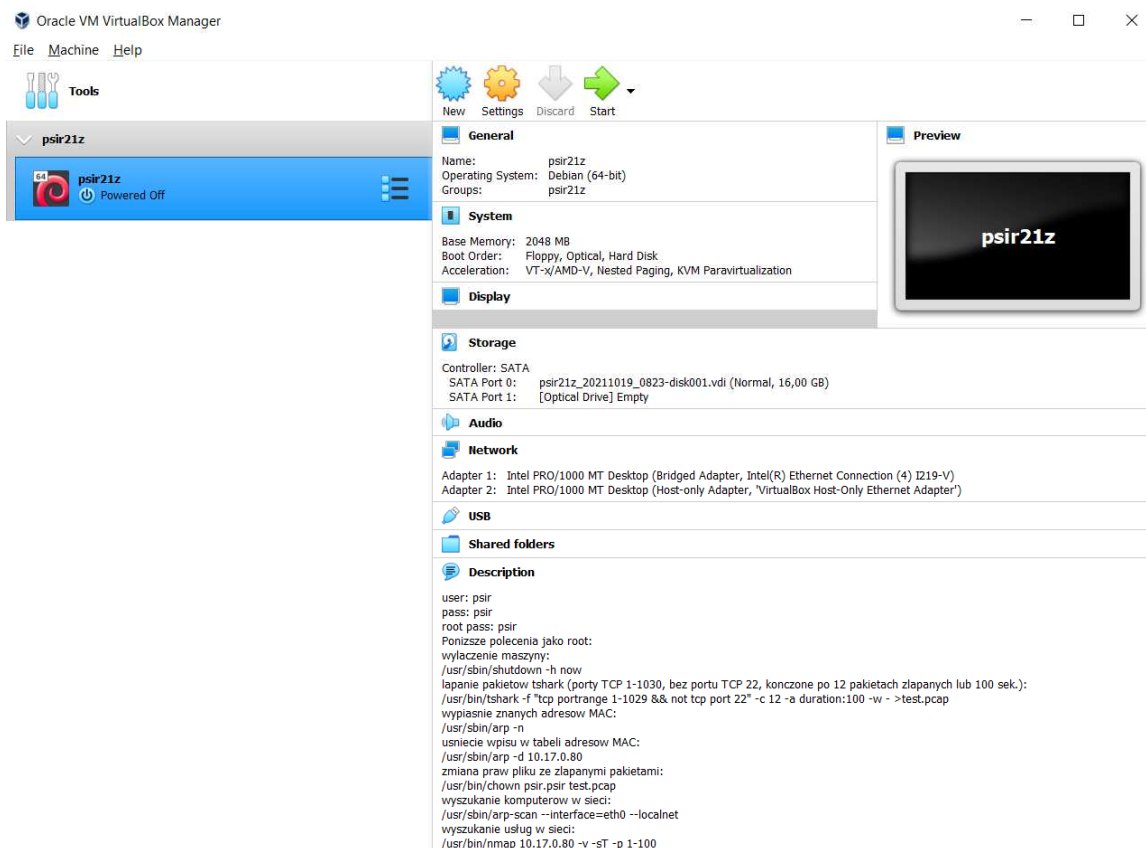
Po zatwierdzeniu ukaże się nam okienko z informacjami o importowanej maszynie wirtualnej:



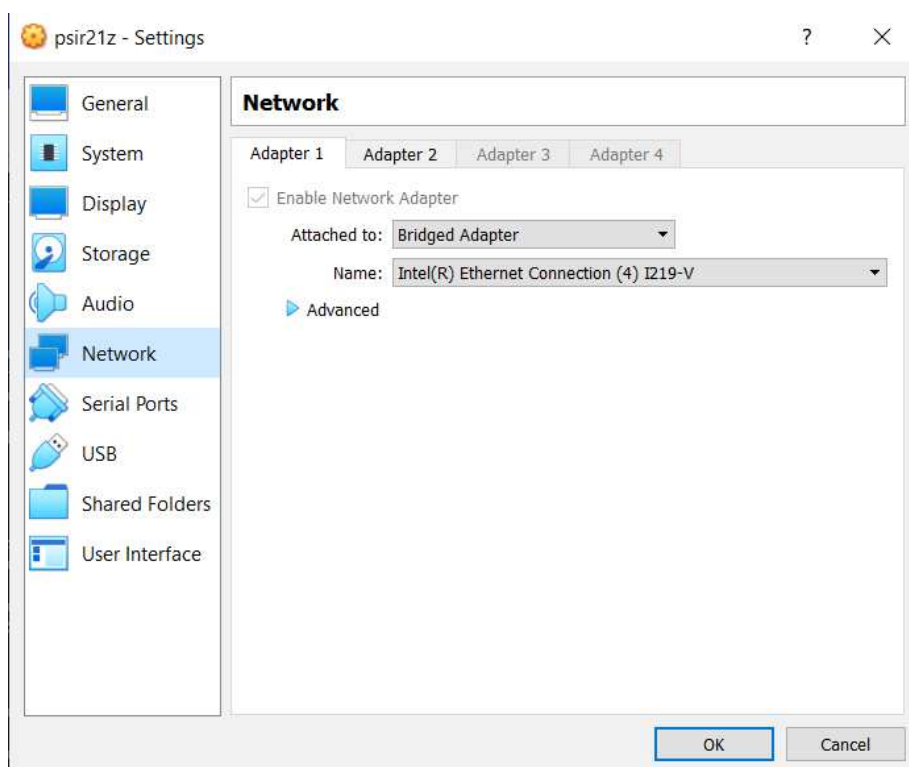
Należy zwrócić uwagę na niektóre parametry, w szczególności – na wielkość pamięci oddanej do dyspozycji systemowi wirtualnemu. Generalnie powinno się ją ustawić na wartość jak najniższą, lecz jeszcze wystarczającą (tutaj: 2048MB), tak, żeby pozostałej pamięci komputera wystarczyło dla innych maszyn wirtualnych i dla macierzystego systemu operacyjnego. Dla pokazanej wyżej wartości 2048MB, komputer, na którym wykonuje się VirtualBox, musi być wyposażony w pamięć RAM o wielkości co najmniej 3GB (a najlepiej ponad 4GB).

Podczas używania sklonowanej maszyny (druga identyczna maszyna wirtualna) należy pamiętać, iż obie potrzebują pamięci operacyjnej RAM do swojego działania. W przypadku stosowania komputerów o małej ilości pamięci RAM używanie dwóch maszyn wirtualnych może być bardzo trudne lub wręcz nie możliwe.

Po dokonaniu ewentualnych modyfikacji możemy przejść do dalszych kroków – poprzez kliknięcie „Importuj”, system przez dłuższą chwilę będzie kopiował dane, po czym w menadżerze programu VirtualBox pojawi się nowa maszyna wirtualna, co pokazuje obrazek:

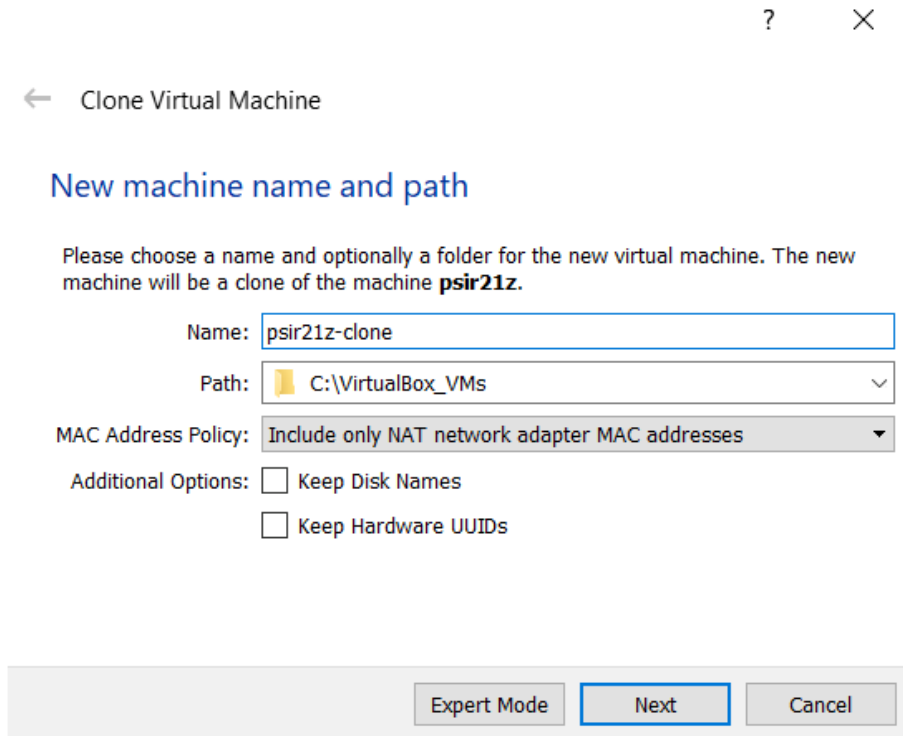


Przed uruchomieniem tej maszyny wirtualnej, dokładniejszej uwagi wymaga ustawienie jej wirtualizowanych kart sieciowych. Maszyna podstawowo musi korzystać z jednej karty wirtualnej w trybie pracy mostkowej (bridged), tak aby maszyna wirtualna pracowała jako następne i niezależne urządzenie w naszej sieci LAN. Ustawienie takie pokazano na rysunku:



Pole „Name” może zawierać inne opcje – zależą one od dostępnych w maszynie goszczącej kart sieciowych, zakładane jest, że maszyna goszcząca zawiera przynajmniej jedną taką kartę sieciową.

Zadanie laboratoryjne można także wykonywać z użyciem dwóch maszyn wirtualnych. Jak już wspomniano, można sklonować właśnie zaimportowaną maszynę wirtualną - o nazwie „psir21z”. Aby tego dokonać należy wybrać z listy zaimportowanych, dostępnych i jeszcze nie uruchomionych maszyn wirtualnych maszynę wzorcową – tu : „psir21z”, a następnie klikając na nią prawym klawiszem myszki wybrać z menu opcję „klonowania”. Po krótkiej chwili pojawi się następujące okienko dialogowe:

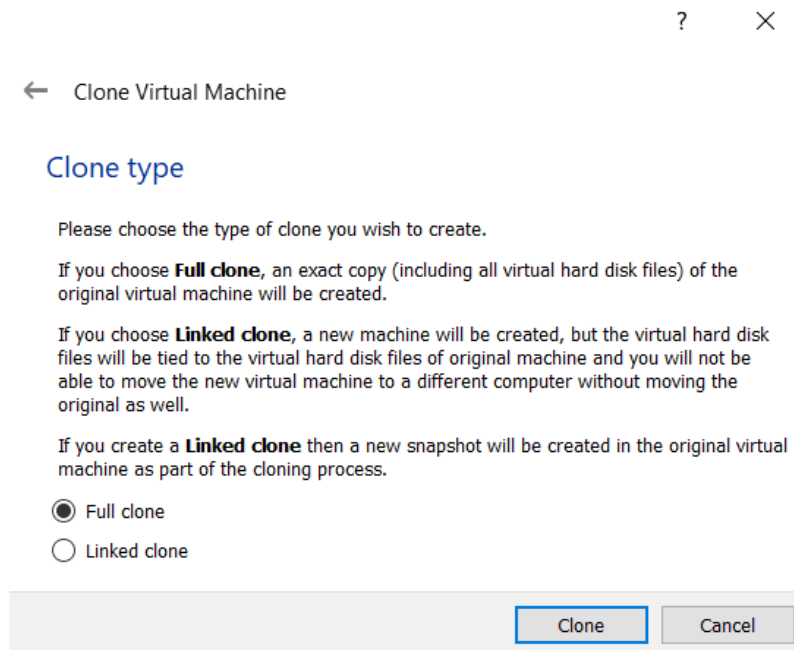


The screenshot shows a window titled "Clone Virtual Machine" with a back arrow and help/cancel icons. The main heading is "New machine name and path". Below it, a text instruction says: "Please choose a name and optionally a folder for the new virtual machine. The new machine will be a clone of the machine **psir21z**." The form contains the following fields and options:

- Name:** A text box containing "psir21z-clone".
- Path:** A dropdown menu showing "C:\VirtualBox_VMs".
- MAC Address Policy:** A dropdown menu showing "Include only NAT network adapter MAC addresses".
- Additional Options:** Two checkboxes: "Keep Disk Names" (unchecked) and "Keep Hardware UUIDs" (unchecked).

At the bottom, there are three buttons: "Expert Mode", "Next" (highlighted with a blue border), and "Cancel".

Na powyższym rysunku nazwa „psir21z-clone” została wpisana ręcznie. Zasadniczo opcje klonowania na tym oknie dialogowy powinny zostać jak pokazano. Po kliknięciu na „Next” ukaże nam się kolejne okno dialogowe:



The screenshot shows a window titled "Clone Virtual Machine" with a back arrow and help/cancel icons. The main heading is "Clone type". Below it, a text instruction says: "Please choose the type of clone you wish to create." The form contains the following text and options:

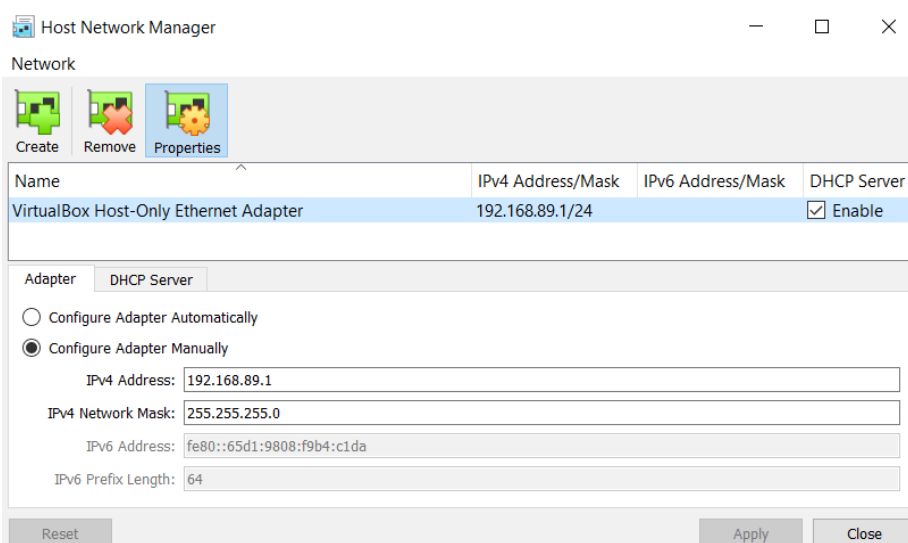
- Full clone:** "If you choose **Full clone**, an exact copy (including all virtual hard disk files) of the original virtual machine will be created."
- Linked clone:** "If you choose **Linked clone**, a new machine will be created, but the virtual hard disk files will be tied to the virtual hard disk files of original machine and you will not be able to move the new virtual machine to a different computer without moving the original as well."
- Snapshot:** "If you create a **Linked clone** then a new snapshot will be created in the original virtual machine as part of the cloning process."

At the bottom, there are two radio buttons: "Full clone" (selected) and "Linked clone" (unselected). Below the radio buttons are two buttons: "Clone" (highlighted with a blue border) and "Cancel".

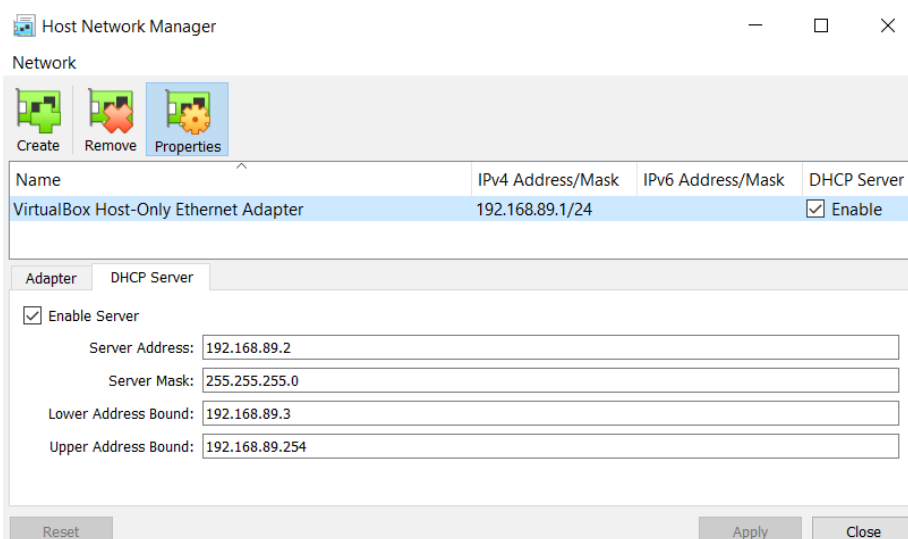
Po wybraniu „Full clone” i naciśnięciu klawisza „Clone” wirtualizator wykona cały proces (trwa on parę długich chwil) i w liście maszyn wirtualnych pojawi się „psir21z-clone”.

Głównym celem utworzenia klonu maszyny wirtualnej „psir21z”, jest możliwość tworzenia aplikacji typu klient-serwer działających na niezależnych maszynach. Zabieg ten ma upewnić, że podczas testów ten sam kod zadziała w 100% także w docelowym środowisku pracy aplikacji nawet gdy maszyny na których pracują klient i serwer są oddalone od siebie o duże odległości.

Przed uruchomieniem obu maszyn wirtualnych konieczne jest dokonanie sprawdzenia konfiguracji ich drugich kart sieciowych. Karty sieciowe „Adapter 1” powinny być ustawione tak samo - co pokazano na wcześniejszym rysunku zatytułowanym: „psir21z – Settings”. Przed przystąpieniem do konfigurowania drugiej karty sieciowej, należy skonfigurować tzw. „Host Network Manager”. Jest on odpowiedzialny za kreowanie sieci wewnętrznych. Sieci takie nie mają łączności ze światem zewnętrznym ale z punktu widzenia maszyn goszczonych i aplikacji na nich uruchamianym w ramach laboratorium nie stanowi to większego problemu. Po wybraniu w menu wirtualizatora „File->Host Network Manager” dla dalszych prac należy nacisnąć na ikonkę karty sieciowej z podpisem „Create”. Po tej operacji (może ona trwać dłuższa chwilę) powinien pojawić się następujący stan:



Ustawienia pod wpisem „Configure Adapter Manually” ukażą proponowaną konfigurację sieci wewnętrznej. W ramach zajęć wygodniej jednak jest pozostawić powyższą konfigurację i skorzystać z serwera DHCP i tylko zweryfikować jego ustawienia. Aby to zrobić klikamy na zakładkę „DHCP Server” a następnie uaktywniamy opcję „Enable Server”. Można także zmienić ustawienia tego serwera, choć w zajęciach laboratoryjnych zostawimy je bez zmian:



Także tutaj warto zaakceptować proponowane ustawienia i przejść dalej - klikając na klawisz „Apply”. Takie zachowanie zapewni, że goszczone maszyny wirtualne („psir21z” i „psir21-clone”) będą się „widziały” poprzez tę wewnętrzną sieć IP.

Na tym etapie warto sprawdzić czy aby druga karta sieciowa każdej z maszyn jest podłączona do właśnie utworzonej nowej sieci – sprawdzamy to w ustawieniach maszyny w opcjach „sieci”, w zakładce „Adapter 2” powinno być ustawienie:

Attached to: Host-only Adapter

Name: VirtualBox Host-Only Ethernet Adapter”.

Po uruchomieniu każdej z maszyn (klawisz „Start”) można się do nich zalogować poprzez konsolę używając uwierzytelnień (stan z 2023.11.07):

użytkownik: student

hasło: student77

Hasło dla użytkownika root jest identyczne. Użytkownik student może korzystać z polecenia sudo używając swoich uwierzytelnień.

Proszę nie aktualizować systemów operacyjnych maszyn wirtualnych – może to spowodować nie przewidziane problemy z ich działaniem.

Przed przystąpieniem do prac, dobrze jest sprawdzić jakie numery IP zostały przydzielone każdej z nich. Aby zbadać numer IP należy wydać polecenie „ip a”, co pokazuje rysunek poniżej:

```
Debian GNU/Linux 10 psir21z tty1
psir21z login: root
Password:
Last login: Fri Oct 22 15:08:10 CEST 2021 on tty1
Linux psir21z 4.19.0-17-amd64 #1 SMP Debian 4.19.194-3 (2021-07-18) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@psir21z:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:3d:f5:ab brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.145/24 brd 10.0.1.255 scope global dynamic enp0s3
        valid_lft 1758sec preferred_lft 1758sec
    inet6 fe80::a00:27ff:fe3d:f5ab/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:48:97:f6 brd ff:ff:ff:ff:ff:ff
    inet 192.168.89.3/24 brd 192.168.89.255 scope global dynamic enp0s8
        valid_lft 559sec preferred_lft 559sec
    inet6 fe80::a00:27ff:fe48:97f6/64 scope link
        valid_lft forever preferred_lft forever
root@psir21z:~#
```

Aby pokazać powyższy stan głównej maszyny wirtualnej („psir21z”) na powyższym rysunku zalogowano się na konto jako użytkownik z uprawnieniami użytkownika „root”. Proszę pamiętać aby nie stosować tego konta do ciągłej pracy z tymi maszynami.

Na powyższym rysunku widzimy że mamy dwie karty sieciowe: enp0s3 o numerze IP: 10.0.1.145, enp0s8 o numerze IP: 192.168.89.3. Oznaczenia kart oraz przypisane im numery IP mogą się w docelowym środowisku różnić, ważne jest, że pierwsza karta jest w tym przykładzie podpięta do sieci LAN a druga do sieci wewnętrznej. Podobnie ma się sytuacja z maszyną podrzędną („psir21z-clone”) co pokazuje rysunek:

```
Debian GNU/Linux 10 psir21z tty1
psir21z login: root
Password:
Last login: Fri Oct 22 15:08:22 CEST 2021 on tty1
Linux psir21z 4.19.0-17-amd64 #1 SMP Debian 4.19.194-3 (2021-07-18) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@psir21z:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:f9:17:a1 brd ff:ff:ff:ff:ff:ff
    inet 10.0.1.149/24 brd 10.0.1.255 scope global dynamic enp0s3
        valid_lft 1662sec preferred_lft 1662sec
    inet6 fe80::a00:27ff:fef9:17a1/64 scope link
        valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:03:d6:7b brd ff:ff:ff:ff:ff:ff
    inet 192.168.89.4/24 brd 192.168.89.255 scope global dynamic enp0s8
        valid_lft 461sec preferred_lft 461sec
    inet6 fe80::a00:27ff:fe03:d67b/64 scope link
        valid_lft forever preferred_lft forever
root@psir21z:~# _
```

Tu widać, że karta sieciowa enp0s3 ma numer IP: 10.0.1.149, enp0s8 o numerze IP: 192.168.89.4.

W przypadku obu maszyn wirtualnych ich interfejsy o oznaczeniu enp0s3 będą służyły do komunikacji ze światem zewnętrznym a interfejsy enp0s8 do komunikacji między sobą. Aby mieć pewność, że obie maszyny wirtualne będą mogły komunikować się między sobą bez pośrednictwa routera czy innej infrastruktury, należy w maszynie głównej wydać polecenie:

```
ping 192.168.89.4
```

a w maszynie podrzędnej:

```
ping 192.168.89.3
```

Jeżeli polecenia „ping” będą mogły komunikować się z zadanymi maszynami oznaczać to będzie że konfiguracje zostały wykonane poprawnie.

W niektórych przypadkach określona maszyna wirtualna może nie otrzymywać numeru IP. Czasami jest to wyłącznie kwestia odczekania paru chwil. Jeżeli takie podejście nie pomaga trzeba skontrolować zawartość pliku: /etc/network/interfaces. W każdej z maszyn powinien ten plik wyglądać następująco:

```
# This file describes the network interfaces available on your system

# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*
```



```
# The loopback network interface

auto lo

iface lo inet loopback

# The primary network interface

allow-hotplug enp0s3

iface enp0s3 inet dhcp

allow-hotplug enp0s8

iface enp0s8 inet dhcp
```

Jeżeli tak nie jest proszę plik ten poddać edycji wydając polecenie:

```
su -c "joe /etc/network/interfaces"
```

Po jego zamknięciu (CTRL-K oraz X) maszynę na której wykonano edycję tego pliku najlepiej zrestartować. Jeżeli mimo to nie pomogło – proszę o kontakt z prowadzącym.

Wartym wspomnienia jest to, że zastosowanie połączenia poprzez tzw. Host Network obu maszyn może rozwiązać problem krótkotrwałego braku dostępu do Internetu przez maszyny wirtualne. Przypadek taki może mieć miejsce m.in. gdy używamy dostępu do Internetu poprzez modem sieci komórkowej. Wtedy to prace nad oprogramowaniem możemy wykonywać we wnętrzach obu maszyn wirtualnych, a tylko na krótki czas synchronizacji ze zdalnym repozytorium GIT zapewnić łączność z Internetem.

4. Użytkowanie systemu GIT na przydzielonych maszynach

Przed rozpoczęciem pracy z powyżej opisanymi maszynami wirtualnymi, należy po zalogowaniu się do każdej z nich - wykorzystując protokół SSH i uruchamiając na swoim prywatnym komputerze program SSH (pod Linux) lub Putty (pod Windows), dokonać klonowania indywidualnego zdalnego repozytorium GIT[2]. Dane dostępowe są przesyłane osobnymi listami elektronicznymi.

Pierwsze klonowanie wykonujemy w dowolnym miejscu, ale w obrębie katalogu domowego przydzielonej maszyny wirtualnej, np.: wykonując kolejno:

```
cd ~
mkdir moje_prace
cd moje_prace
```

Przed rozpoczęciem klonowania należy wyjaśnić, iż program GIT przed rozpoczęciem klonowania zapyta o nasze dane uwierzytelniające (dane te przesłane zostaną studentom listem elektronicznym). Dla celów dalszego opisu będą to dane hipotetycznego i nie istniejącego użytkownika – tj. login: psir. Aby dokonać klonowania wydajemy polecenie:

```
git clone https://psir@equ.tele.pw.edu.pl/psir2023Z/psir
```

Podany powyżej adres (tutaj jest to: <https://psir@equ.tele.pw.edu.pl/psir2023Z/psir>) jest specyficzny dla każdego indywidualnego repozytorium GIT – proszę nie modyfikować go i użyć zgodnie z powyższym zmieniając miejsca, w których występuje login (tu jest on użyty dwa razy – co jest poprawne!). Proszę pamiętać, że w systemie Linux (czyli na maszynach wirtualnych) pytanie o dane uwierzytelniające mogą być podawane w linii poleceń po wywołaniu narzędzia GIT. Dodatkowo proszę pamiętać, aby nie używać podczas podawania tych danych metody kopiuj-wklej, z pewnych przyczyn proces taki może zakończyć się porażką autoryzacji.

Aby mieć pewność, że zawartość naszego indywidualnego zdalnego repozytorium GIT jest obecna w najświeższej wersji na każdej z maszyn wirtualnych, możemy w ich wnętrzu wykonać polecenie:

```
git pull
```

Po każdej zmianie kodu źródłowego, można zsynchronizować zawartość lokalnej kopii repozytorium z tą zdalną. Aby to uczynić trzeba wykonać poniższe operacje, dla powyższego przykładu we wewnątrz katalogu: ~/moje_prace:

a) Dodanie wszystkich swoich poprawek do lokalnego repozytorium (domyślnie po sklonowaniu danych ze zdalnego repozytorium pracujemy wyłącznie na własnej kopii), kropka użyta w poniższym poleceniu oznacza – dodaj wszystkie zmodyfikowane pliki w aktualnym katalogu lub w jego którymś z podkatalogów:

```
git add .
```

Po wydaniu tego polecenia warto zobaczyć czy wszystkie nasze poprawione pliki zostały uwzględnione, wydając polecenie:

```
git status
```

b) Zatwierdzanie zbioru poprawek/treści (niezbędne, aby system poprawnie rozpoznawał kiedy i co zostało uznane za zrobione):

```
git commit -a -m "Praca nad lab1"
```

Zaleca się unikanie stosowania polskich znaków diakrytycznych w sekcji objętej cudzysłowem a będących zdawkowym opisem czego dotyczy zbiór zatwierdzonych poprawek/treści.

c) Po wydaniu powyższego polecenia możemy swoje poprawki i treści (nowe pliki) „wypchnąć” do zdalnego repozytorium (tak aby np.: prowadzący mogli je sprawdzić i wystawić ocenę):

```
git push
```

Dodatkowo dla zapewniania sobie największej pewności, możemy na swoim prywatnym komputerze (np.: maszynie gdzie uruchomiono wirtualizator) zweryfikować czy wszystkie wytworzone przez siebie pliki znalazły się na dedykowanym zalanym repozytorium GIT – wykonując ponowne klonowanie repozytorium albo gdy to klonowanie wykonano już wcześniej – przez polecenie „git pull”.

Więcej na temat systemu GIT można znaleźć w Internecie, w tym na stronie projektu[2]. Dodatkowo przed laboratorium w katalogu lab1 prowadzący ćwiczenia umieszcza plik PDF o nazwie niemal identycznej jak nazwisko i imiona osoby, dla której utworzono zdalnej repozytorium.

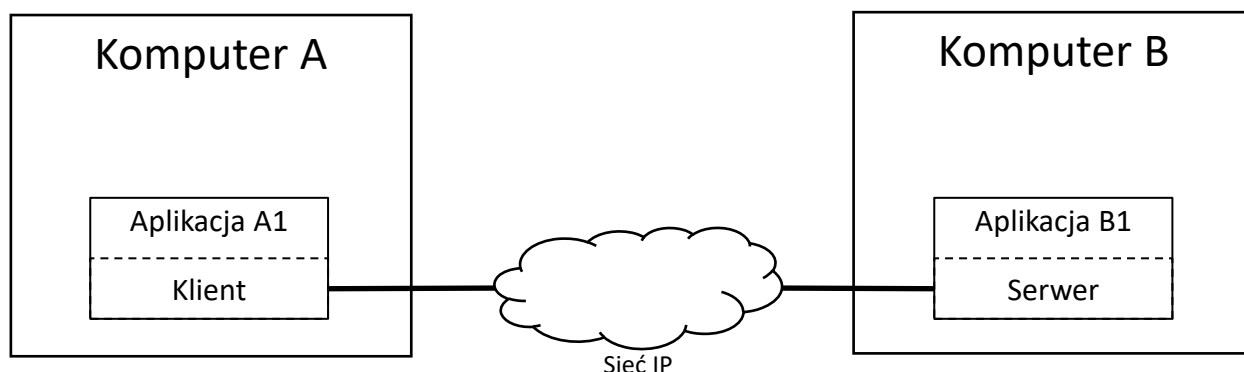
5. Sprawdzenie narzędzi na przydzielonej maszynie

Podobnie jak podczas pracy w innych laboratoriach przed przystąpieniem do pracy na maszynie wirtualnej należy oprócz opisanych wcześniej procedur, sprawdzić wersje używanych narzędzi, aby później tę informację można było zamieścić w raporcie (przydatne w kwestiach spornych). W systemie Linux, aby sprawdzić większość narzędzi, należy je uruchomić dodając opcję wywołania „--version” (proszę zauważyć, że argument wywołania jest czytelny dla człowieka tekstem i przed takimi argumentami wstawia się dwa znaki ”-”), dla sprawdzenia kompilatora języka C jest to:

```
gcc --version
```

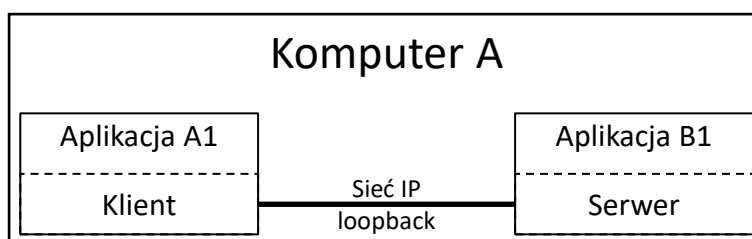
6. Architektury aplikacji sieciowych

Istnieje wiele możliwych scenariuszy współdziałania aplikacji komunikujących się poprzez sieć IP. Oprócz wielu aspektów jako pierwszy ustalmy, że komunikacja będzie realizowana za pomocą protokołu TCP. Możemy założyć też, że istnieją dwa komputery (A o IP: 192.168.89.3 i B o IP: 192.168.89.4) a na nich działają dwie aplikacje (A1, B1), co pokazuje poniższy rysunek (scenariusz I):



W tym scenariuszu aplikacja A1 działać będzie jako klient TCP a aplikacja B1 jako serwer TCP, natomiast „Sieć IP” jest siecią wirtualizowaną przez VirtualBox. Tutaj należy pamiętać, że z oczywistych powodów kod klienta i serwera zgodnie z informacjami podanymi na zajęciach wykładowych będą się w odpowiedni sposób różniły.

Jednak aby móc tworzyć oprogramowanie także w przypadku gdy na maszynie goszczącej (Host) nie mamy odpowiednich zasobów (przestrzeń dyskowa, pamięci operacyjnej, moc obliczeniowa procesora) można skorzystać ze scenariusza jedno komputerowego. W scenariusz takim gdy klient chce się komunikować z serwerem poprzez dowolny protokół IP a oba procesy są uruchomione na tym samym komputerze, muszą używać tzw. interfejs pętli zwrotnej (ang. loopback)[3]. W takiej sytuacji aplikacja A1 będzie łączyć się poprzez interfejs pętli zwrotnej, czyli praktycznie z adresem IP numerycznym: 127.0.0.1. Takie działanie prowadzi do następującej architektury:



Mimo pozornej trywialności takiego podejścia, ten typ łączności jest dość popularny, pomagając testować tworzone oprogramowanie gdyż obie aplikacje mogą działać na tym samym komputerze. Podejście takie mimo, iż poprawne i dozwolone podczas zajęć laboratoryjnych, ma pewną wadę: brak pewności czy proces komunikacji jest poprawnie zaimplementowany.

7.Kompilacja i uruchomienie aplikacji

Po utworzeniu oprogramowania, aplikacje sieciowe w przytoczonym środowisku kompiluje się w typowy sposób – zakładając, że kod jest zapisany w pliku main.c będzie to:

```
gcc -g main.c -o obraz_aplikacji
```

Jeżeli kod został skompilowany poprawnie, aby go uruchomić należy wydać polecenie:

```
./obraz_aplikacji
```

Powyższe uruchomienie aplikacji może ujawnić błędy w implementacji. Jest jednak możliwość użycia debugera GDB[4] za pomocą którego można znaleźć wiele z błędów. Program GDB to element wchodzący skład pakietu GCC[5]. Aby program ten mógł poprawnie działać należy podczas kompilacji kodu źródłowego użyć opcji ‘-g’. Opcja ta nakaze kompilatorowi i linkerowi dodanie do pliku wynikowego odpowiednich informacji dla debugera. Aby uruchomić z użyciem programu GDB tak właśnie skompilowaną aplikację, należy w linii poleceń napisać:

```
gdb ./obraz_aplikacji
```

Program GDB zgłosi się komunikatem (lub podobnym):

```
GNU gdb (Debian 8.2.1-2+b3) 8.2.1
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./obraz_aplikacji...done.
```

Po czym GDB czeka na polecenia wydawane przez użytkownika. Jednym z poleceń jest listowanie kodu, wydając polecenie ('l' od 'list'):

```
1 90
```

program GDB wyświetli nam zawartość kodu źródłowego w okolicach linii 90:

```
85 unsigned char my_str[MAX_BUF];
86 unsigned char my_str2[MAX_BUF];
87 struct sockaddr_in cliAddr;
88 int cliLen=sizeof(cliAddr);
90 int pos=recvfrom(s, my_str, MAX_BUF,0,(struct sockaddr *)&cliAddr, &cliLen);
91 if(pos==0){
92     printf("Peer was disconeted\n");break;
93 }else if(pos<0){
94     printf("ERROR: %s (%s-%d)\n", strerror(errno), __FILE__, __LINE__);
```

Program GDB umożliwia też pracę linia po linii, czyli tzw. pracę krokową – uruchomienie wykonania jednej linii czyli jednego korku, uruchamia polecenie 's' (step), można także zlecić wykonanie wielu linii przez tzw. pracę wielokrokową 's 10' (10 kolejnych linii). Wykonanie polecenia 's' w pewnych przypadkach jest niekorzystne, gdyż GDB będzie wchodzić także w implementacje funkcji w tym także tych systemowych. Aby przejść przez kod implementacji bez wchodzenia w funkcje, w GDB można użyć polecenia 'n' (next).

Możliwa jest także praca ciągła – polecenie 'c' (continue). Najwygodniej jednak korzystać z polecenia 'c' oraz ustawienia tzw. pułapki (ang. break point). Aby ustawić taką pułapkę, określamy numer linii w której chcemy ją ustawić poprzez wprowadzenie polecenia:

```
b 91
```

Prześle ono programowi GDB aby ustawił pułapkę w linii 91. Uruchamiając aplikację z pełną prędkością, przez wydanie polecenia 'c', program będzie się wykonywał aż do napotkania którejkolwiek pułapki. Po dotarciu przez aplikację do linii z taką pułapką, program GDB wyświetli coś takiego:

```
Breakpoint 1, main () at main.c:91
91  if(pos==0){
```

Po czym zatrzyma się w oczekiwaniu na następne polecenia. Tutaj oprócz ponownego uruchomienia wykonywania programu (polecenie 'c'), można poddać tzw. inspekcji lub modyfikacji określone zmienne czy wręcz rejestry procesora. Najbardziej interesujące i podstawowe

jest wypisanie stanu zmiennych, służy do tego polecenie 'p' (print). Polecenie to należy wydać z argumentem określającym obiekt do inspekcji np.:

```
p my_str
```

po czym pojawi się (lub coś podobnego):

```
$3 = "test\n\000\230\000"
```

Informacja ta została tu podana w formacie mieszanym (tekstowym i oktalnym gdy określone znaki były z klasy niedrukowalnych), gdzie '\$3' jest numerem rezultatu polecenia 'p my_str'- tutaj można to pominąć.

W pewnych przypadkach wymagane jest zaprezentowanie wartości w określonej notacji. Dla przykładu w formacie szesnastkowym należy wydać polecenie:

```
p/x my_str
```

```
$4 = {0x74, 0x65, 0x73, 0x74, 0xa, 0x0, 0x98, 0x0}
```

lub w dziesiętnym:

```
p/d my_str
```

```
$5 = {116, 101, 115, 116, 10, 0, -104, 0}
```

Po więcej informacji jak używać GDB proszę sięgnąć do dokumentacji tego programu.

Uruchamianie (czyli usuwanie błędów) kodu aplikacji według powyższej metody nie rozwiązuje poważniejszego problemu jakim jest analiza ruchu sieciowego. Jeżeli kod aplikacji będzie komunikował się poprzez sieć IP, to dla celów diagnostycznych można rozważyć uruchomienie programu tcpdump[6]. Program ten pomoże analizować jakie pakiety i do kogo zostały wysłane tak aby umożliwić zgrubną analizę ruchu sieciowego. Dla przykładu na maszynie wirtualnej głównej można ten program uruchomić w następujący sposób:

```
su -c "/usr/sbin/tcpdump -n -i enp0s8 \"not tcp port 22\" -vvv"
```

Wywołanie to musi być wykonane z prawami administratora i dzięki poleceniu „su -c” staje się to możliwe. Należy jednak pamiętać, że polecenie to będzie pytało o hasło użytkownika „root”. Opcja „-n” nakazuje nie tłumaczyć nazw tylko przytaczać je w surowej postaci. Kolejną opcją jest „-i enp0s8”. Informuje ona tcpdump aby łapał pakiety przesyłane tylko poprzez interfejs „enp0s8”. Zapis \"not tcp port 22\" oznacza tutaj konieczność pominięcia wszelkich pakietów które związane są z protokołem TCP i portem 22, dzięki czemu ewentualny ruch związany z protokołem SSH nie będzie łapany.

Ostatnie opcje „-vvv” włączają tryb szerszego cytowania złapanych pakietów. Po pojawieniu się ruchu sieciowego wykreowanego między maszyną główną a podrzędną narzędzie mogło by złapać następujące pakiety (tu ruch był wymieniany między maszynami z IP: 192.168.89.3 i IP: 192.168.89.5):

```
tcpdump: listening on enp0s8, link-type EN10MB (Ethernet), capture size 262144 bytes
14:15:35.276302 IP (tos 0x0, ttl 64, id 7211, offset 0, flags [DF], proto TCP (6), length 60)
  192.168.89.3.34244 > 192.168.89.5.12345: Flags [S], cksum 0x3388 (incorrect -> 0x657d),
  seq 3692844898, win 64240, options [mss 1460,sackOK,TS val 2546938604 ecr 0,nop,wscale 7],
  length 0
14:15:35.276770 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto TCP (6), length 60)
  192.168.89.5.12345 > 192.168.89.3.34244: Flags [S.], cksum 0xd9ad (correct), seq 3514462896,
  ack 3692844899, win 65160, options [mss 1460,sackOK,TS val 2078263323 ecr 2546938604,nop,
  wscale 7], length 0
14:15:35.276787 IP (tos 0x0, ttl 64, id 7212, offset 0, flags [DF], proto TCP (6), length 52)
  192.168.89.3.34244 > 192.168.89.5.12345: Flags [.], cksum 0x3380 (incorrect -> 0x050c),
  seq 1, ack 1, win 502, options [nop,nop,TS val 2546938605 ecr 2078263323], length 0
14:15:39.004003 IP (tos 0x0, ttl 64, id 7213, offset 0, flags [DF], proto TCP (6), length 57)
  192.168.89.3.34244 > 192.168.89.5.12345: Flags [P.], cksum 0x3385 (incorrect -> 0x2496),
```

```

seq 1:6, ack 1, win 502, options [nop,nop,TS val 2546942332 ecr 2078263323], length 5
14:15:39.004621 IP (tos 0x0, ttl 64, id 65432, offset 0, flags [DF], proto TCP (6), length 52)
192.168.89.5.12345 > 192.168.89.3.34244: Flags [.], cksum 0xe7df (correct), seq 1, ack 6,
win 510, options [nop,nop,TS val 2078267051 ecr 2546942332], length 0
14:15:40.405550 IP (tos 0x0, ttl 64, id 7214, offset 0, flags [DF], proto TCP (6), length 52)
192.168.89.3.34244 > 192.168.89.5.12345: Flags [F.], cksum 0x3380 (incorrect -> 0xe26d), seq 6,
ack 1, win 502, options [nop,nop,TS val 2546943733 ecr 2078267051], length 0
14:15:40.406611 IP (tos 0x0, ttl 64, id 65433, offset 0, flags [DF], proto TCP (6), length 52)
192.168.89.5.12345 > 192.168.89.3.34244: Flags [F.], cksum 0xdcea (correct), seq 1, ack 7,
win 510, options [nop,nop,TS val 2078268453 ecr 2546943733], length 0
14:15:40.406664 IP (tos 0x0, ttl 64, id 7215, offset 0, flags [DF], proto TCP (6), length 52)
192.168.89.3.34244 > 192.168.89.5.12345: Flags [.], cksum 0x3380 (incorrect -> 0xdcf0),
seq 7, ack 2, win 502, options [nop,nop,TS val 2546943735 ecr 2078268453], length 0

```

Narzędzie tcpdump choć przydatne może być nie wystarczające, w takim przypadku można skorzystać z narzędzia TSHARK[7]. Pozwala ono podobnie jak WireShark[8] łapać pakiety i zachowywać je do przyszłej analizy. TSHARK ma jednak przewagę nad WireShark polegającą na możliwości łapania pakietów na lekkich maszynach nie posiadających tzw. kary graficznej. Wywołanie programu TSHARK wymaga także praw administratora, tutaj jednak zalecane jest zalogowanie się jako „root” (pierwsza z linii podanych w poniższej ramce) i dopiero wtedy uruchomienie programu TSHARK (po poleceniu z pierwszej linii zostaniemy zapytani o hasło dla użytkownika root):

```

su -
/usr/bin/tshark -n -i enp0s8 -w pakiety.pcap "not tcp port 22"

```

Po każdym złapanym pakiecie program raportuje (w miejscu jego działania) ile ich złapał. Aby przerwać proces łapania można w dowolnym momencie nacisnąć kombinację klawiszy CTRL i C. Program po zakończeniu swojego działania utworzy plik pakiety.pcap ze złapanymi pakietami. Aby pakiety te można było w wygodny sposób wyciągnąć z maszyny wirtualnej i potem analizować należy – zmienić prawa dostępu do pliku pakiety.pcap:

```

chown student.student pakiety.pcap

```

a następnie przesunąć ten plik do katalogu domowego użytkownika student:

```

mv pakiety.pcap /home/student/

```

aby na końcu z poziomu systemu maszyny goszczącej (np.: systemu Windows) pobrać ten plik, wywołując polecenie PSCP - będącego jednym z elementów zestawu PuTTY - do lokalnego systemu plików (zakładając że proces łapania pakietów wykonano z maszyny nadrzędnej):

```

pscp student@192.168.89.3:pakiety.pcap .

```

w przypadku systemu Linux, polecenie to wydawane także z linii poleceń, wyglądało by bardzo podobnie:

```

scp student@192.168.89.3:pakiety.pcap .

```

Po czym plik pakiety.pcap, można już otworzyć za pomocą programu WireShark[8] i dokładnie analizować zarówno przepływ pakietów jak i ich treści.

8.Literatura

- [1] VirtualBOX, <https://www.virtualbox.org>, ostatnia wizyta 2023.10.02
- [2] GIT, <https://git-scm.com>, ostatnia wizyta 2023.10.02
- [3] Loopback, <https://en.wikipedia.org/wiki/Loopback>, ostatnia wizyta 2023.10.02
- [4] GDB, <https://www.gnu.org/software/gdb>, ostatnia wizyta 2023.10.02
- [5] GCC, <https://gcc.gnu.org>, ostatnia wizyta 2023.10.02
- [6] TCPDUMP, <https://en.wikipedia.org/wiki/Tcpdump>, ostatnia wizyta 2023.10.02
- [7] TSHARK, <https://www.wireshark.org/docs/man-pages/tshark.html>, ostatnia wizyta 2023.10.02
- [8] WireShark, <https://www.wireshark.org/docs>, ostatnia wizyta 2023.10.02