Sistemas de Software Seguros
Segurança de Software
2024/2025
Class Project: Setup VM & Initial Exercises

NOTE: You do not have to deliver a report for this project!

Setup for the class project:

All class projects will be based on an Ubuntu VirtualBox image created for the course. The password for users `ss` and `root` is the same and is `ss`. The image can be obtained from:

https://cloud.admin.di.fc.ul.pt/index.php/s/AEfK8g8oQDiiyPJ

Notice that the image can take some time to download because it is around 4.5 Gbytes. Therefore, it is advisable that either:

- you use a pen to have a local copy of the image
- alternatively, try to run the project on your own computer

You should avoid making a copy of the image on the local disk of the lab's computers because there is usually limited space.

Do the following to install the image: a) run VirtualBox on Windows (in a lab machine or your computer); 2) Use `File->Import appliance` to set up the image. Find further instructions in the following sections.

NOTE: If you install the image in one of the labs, it becomes visible to other students who enter the same computer (after you log out). Therefore, **do not forget to remove the virtual machine** after using it.

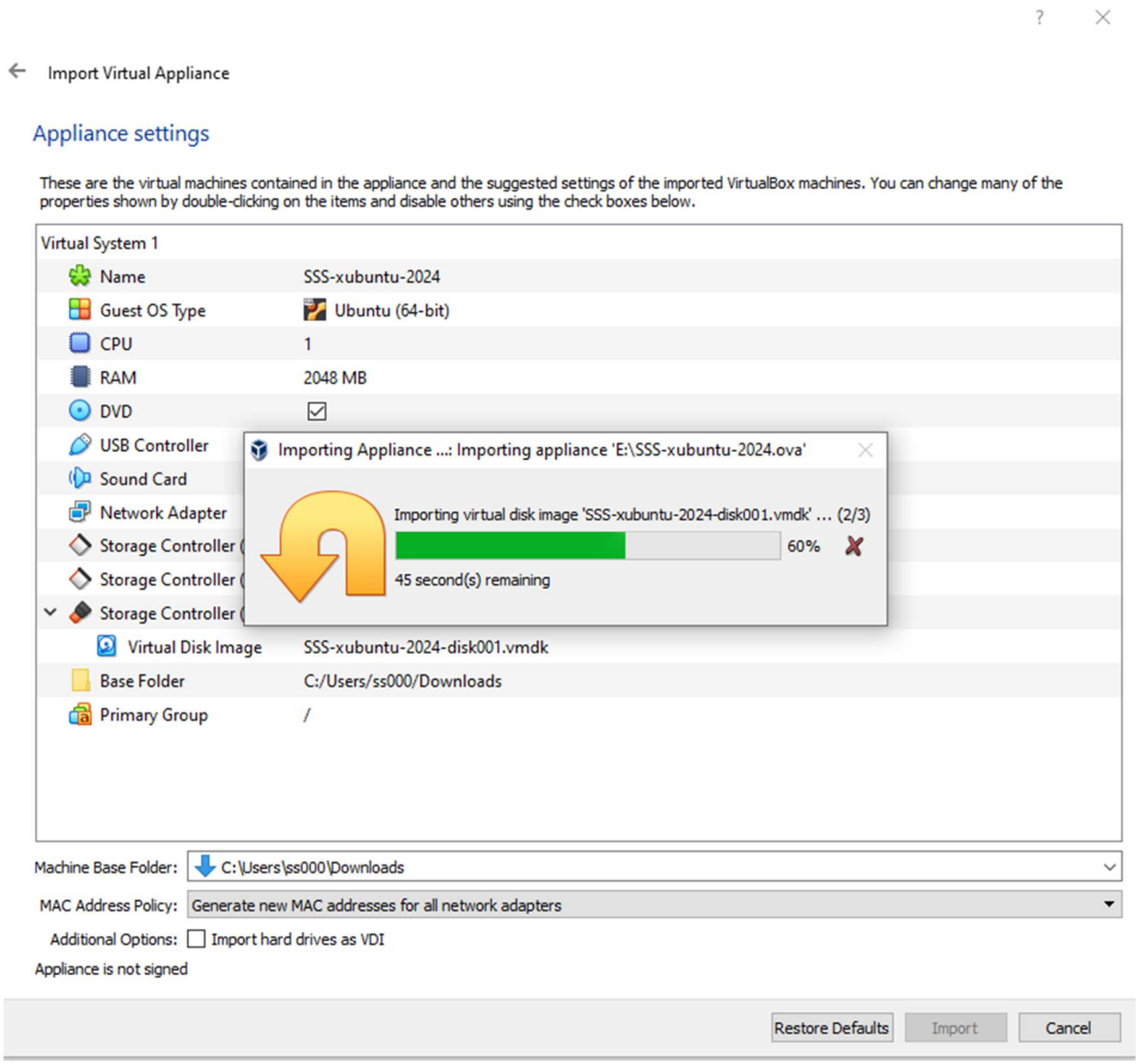
1. Setup the VirtualBox VM

Notice all procedures indicated below were tested in VirtualBox version 7.1.0. There might be slight differences in the procedure for other versions.

- Start VirtualBox, and then

  Select `File->Import appliance`

Indicate the location of the `.ova` file you have downloaded. Modify the `Machine Base Folder` to the Downloads directory, choose the MAC address policy to `Generate new MAC addresses for all network adapters`, and unselect the `Import hard drives as VDI`.



Now, you should have a new VM image available in VirtualBox. A new window will pop up when clicking twice in the VM, making the course image available.

In the FCUL lab, removing the file you downloaded is a good idea since there is often limited disk space. Do not forget also to **empty the recycle bin**!

- Depending on your display, you may need to adjust the scaling settings (for example, because the icons are too small). You must select from the menus on top of the VM `<Machine> | Settings`. Next, on the pop-up window, you need to choose `Display`. By setting the `Scale Factor` to different values, you can adjust the size of the VM window.

You can change the OS configuration if the font size is too small. Select the button at the top left corner, and then choose `Settings -> Appearance`. Then, select the `Fonts` tab. You can then choose the size of the fonts to the appropriate value.

- VirtualBox lets you set up a few extensions that facilitate the interactions with the virtual machine. Notice that there are slight differences in how these extensions are set depending on the version of the software that you are using.

<u>You need to do the following to set up or update the Guest Additions</u>:

Start the virtual machine
Select `<Devices> | Insert Guest Additions CD images ...`
On the CD, select `autorun.sh` and create a link to it on the desktop (`right button | Send To | Desktop`), and then run the file by pressing it on the desktop
Provide your `ss` password
Press return
Shutdown (not restart) the virtual machine for the Guest Additions to take effect, and then start the virtual machine again

NOTE: depending on the Xubuntu version that you have installed, it might happen that it does not bring some of the tools pre-installed (see if there are error messages while running the `autorun.sh`). In this case, you will need first to install some of these tools, like:

```
sudo apt install gcc
sudo apt install make
sudo apt install make-guile
```

Ultimately, you might want to remove `autorun.sh` from the desktop and then remove the Guest Additions CD by doing `<Devices> | Optical Devices | Remove disk from virtual drive ...`

<u>After the guest extensions are installed, you can do several things</u>:

To have a shared clipboard between the host OS and the guest OS
`Select <Devices> | Shared Clipboard | Bidirectional`

You can also do similar things, like “Drag and Drop.”

To have a shared directory between the host OS and the guest OS
Select `<Devices> | Shared folders Settings …`
Select the plus `(+)` sign on the right

Indicate the `Folder Path` in the host OS. Call the `Folder Name` for example "`SharedDir`", select `Auto-mount` and `Make Permanent`.

Start a window in the virtual machine
In the case of Xubuntu, the directory appears in
`/media/sf_SharedDir/`
but you need to have root privileges to access it
(do "`sudo su - root`" with password "`ss`")

Do not upgrade the Linux Base! Some tools and vulnerable applications that we will use in the course will not work in a new version of Linux

2. Notes on OS protection

In some of the labs during the semester, we have to turn off two of the OS/compiler protections. First, by employing flag `-fno-stack-protector` while compiling, we prevent `gcc` from adding information to the stack that would detect various kinds of buffer overflows. In addition, we disable Address Space Layout Randomization (ASLR) from the OS.

1) To prevent this until the next reboot, do the following in the "root" account:

```
more /proc/sys/kernel/randomize_va_space
```

If it is larger than 0, then randomization is on.

2) Run the command to turn off this protection temporarily (you need to run it every time you reboot the machine).

```
echo 0 >  /proc/sys/kernel/randomize_va_space
```

3) To turn off this protection permanently, you need to add a file `/etc/sysctl.d/01-disable-aslr.conf` containing:

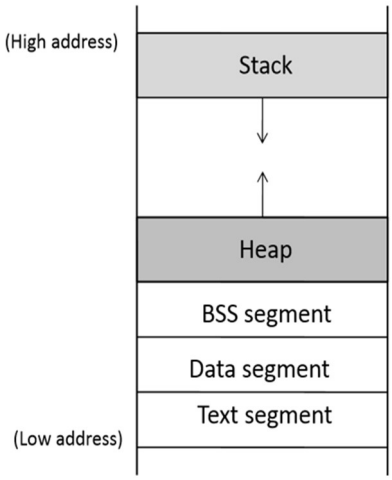
```
kernel.randomize_va_space = 0
```

3. Run a test program

Recall that C programs store data in the following places:

- **Stack** – local variables
- **Heap** – dynamic memory (malloc, new)
- **BSS segment** - uninitialized global variables
- **Data segment**– initialized global variables

The **text segment** is where the code program is stored.



Enter the virtual machine as explained above. Start by writing the following C code and save it in a file with the name `exerc1.c`:

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

int gi1 = 123;  // global variable initialized

int g1;  // global variable uninitialized

int main (int argc, char **argv)  // Command line arguments
{
    int l1;   // local variable

    printf("l1   [%p]\n", &l1);
    printf("argc [%p], &argv [%p], argv [%p], *argv [%p] \n",
              &argc, &argv, argv, *argv);  // *argv = program name

    printf("g1   [%p]\n", &g1);
    printf("gi1  [%p]\n", &gi1);

    return 0;
}
```

a) Understand what the code does, and then compile the program in the following way:

```
gcc -g -o exerc1 exerc1.c
```

b) Run the program in the following way:

```
./exerc1 "DI-FCUL is the best!"
```

c) Try to understand the output from the above execution and relate it with the code you wrote. Apparently, information is stored in memory in different locations.

- Can you determine which variables are stored in lower memory locations?

- Which of the following are stored in higher memory addresses: local variables or inputs to the program as command line arguments?

d) Now, you should modify the program so that 16 bytes are allocated dynamically with `malloc()`, and then print the address where they were stored. Is dynamically allocated memory stored above or below global variables?

e) Now experiment compiling with the following command and the run `./exerc1_1`:

```
gcc -fno-stack-protector -fno-asynchronous-unwind-tables -o exerc1_1 exerc1.c
```

Does anything change? What?