SYS2

Wild Linux Kernel Object Module

2025

Version 1

Jules AUBERT <jules1.aubert@epita.fr>

Copyright

Contents

1 General rules

The following informations are very important and should be read carefully :

Failure to comply with any of the following instructions will result in penalties up to and including the multiplication of the final mark by 0.

These instructions are clear, unambiguous and have a precise objective. Moreover, they are non-negotiable.

Do not fear asking if you cannot understand one of those rules.

General rules 0 : You must read the subject.
General rules 1 : You must follow the rules.
General rules 2 : You must not handout late

General rules 3 : The work must be handed as described in Hand out format.
General rules 4 : The rendered work must not contain binary, temporary, or error files (**`*~`**, **`*.o`**, **`*.a`**, **`*.so`**, **`*#*`**, **`*core`**, **`*.log`** , **`*.exe`**, binaries, ...).
General rules 5 : However, if a binary file folder is present, it must be known from an external source (example: a Download section of an open source project). You **must** then explain in the file `README` its usefulness and its source.
General rules 6 : Throughout this document, case (uppercase and lowercase characters) is very important. You must strictly respect the upper and lower case imposed in the subject's messages and filenames.
General rules 7 : Throughout this document, **`login.x`** corresponds to your login.
General rules 8 : Any delay, even one second, results in the non-negotiable score of 0.
General rules 9 : Cheating (sharing code, copying code or text, ...) implies **at best** the non-negotiable grade of 0.
General rules 10 : If there are any problems with the project, you should contact the people responsible for the subject as soon as possible at the indicated email addresses.

Advice : Do not wait too much before beginning, even if something looks easy, you will never be protected from a stubborn bug.

2 Hand out format

Project managers :	**Jules AUBERT** `<jules1.aubert@epita.fr>`
Tags :	**`[SYS2][WLKOM]`**
Team size :	4 max
Submission procedure :	git repository
Directory name :	`epita-apprentissage-wlkom-apping-202X-login.x`
Repository url :	
`login.x@git.forge.epita.fr:p/epita-apprentissage/wlkom-apping-202X/epita-app`	
Date of submission :	Look on the Forge intra
Project duration :	Look on the Forge intra
Architecture/OS :	Linux amd64
Language(s) :	c
Compiler :	**`cc`**
Linux version:	**`EPITA's laptop (Ubuntu 24.10)`**

The following files are required :

`AUTHORS`	A list of the students **logins** working on the subject. One if alone, several for a proposed project.
`README`	Contain a project description as well as ways to build and run it, and a list of features. It can be a simple texte file, a markdown file (README.md), a pdf file (README.pdf) or even an HTML doc (README/ is a directory containing HTML files).
`TODO`	List what still needs to be implemented. It must be updated regularly.
`rootkit/...`	The rootkit directory with the rootkit source code.
`rootkit/Makefile`	The Makefile to compile the kernel object.
`attacking_program/...`	The directory with the attacking program source code.

Your code will be tested and inspected by a human, you should therefore respect the given specifications in order to get points. Document your design decisions and your failed attempts. Those things will impact your grade positively.

The expected tree structure for the project is as follows :

```
epita-apprentissage-wlkom-apping-202X-login.x/
epita-apprentissage-wlkom-apping-202X-login.x/AUTHORS
epita-apprentissage-wlkom-apping-202X-login.x/README*
epita-apprentissage-wlkom-apping-202X-login.x/TODO
epita-apprentissage-wlkom-apping-202X-login.x/rootkit/Makefile
epita-apprentissage-wlkom-apping-202X-login.x/rootkit/...
epita-apprentissage-wlkom-apping-202X-login.x/attacking_program/...
epita-apprentissage-wlkom-apping-202X-login.x/...
```

You will never lose points because of some source file (source code/headers/config) or Makefile as long as their presence can be justified. Empty files can lead to penalties.

3 Introduction

About this document

This document is the project part of the APPING SYS2 class. It contains a project to develop a Linux kernel object.
You will approaches multiple notions which relate to kernel development and special file manipulation.
Have fun!
All work will be reviewed and taken into account for the final mark. Show me your best!

4 Submission

4.1 Goal

In this exercise, you are expected to document your work in every way and architecture it to produce a **pedagogical** rootkit.

Pedagogical means I will allow some *bad practices* to ease the use of your rootkit deployment and usage in exchange of documentation.

4.2 AUTHORS

Your AUTHORS file must contain your login(s).

Example:

```
julesh$ cat -e AUTHORS
aubert_o$
claire.leroux$
junior$
pedro.miranda$
julesh$
```

AUTHORS example

4.3 README

Your **README** file(s) must explain several things, such as:

- How your project works (no need to technically explain the code unless you want to document it with **Doxygen**)
- How to use your project
- ... Whatever else needs to be documented

You can document in a markdown file, in a PDF, in HTML files, whatever easy to use.

Your submission will be tested on an EPITA's laptop with Ubuntu 24.10.

You need to submit documentations to deploy two virtual machines. One for the attacking program and one for the rootkit (the victim). You are free on the distributions to use as long as both of them are using Linux.

4.4 Environment

You work firstly on the EPITA's laptop using Ubuntu 24.10. You must use **free to use** and **open source** softwares on the Ubuntu 24.10 and inside the virtual machines. In this case **free** means non-pricing. Most of the time, **free** means *libre*. I don't ask for libre softwares. Although in your everyday life, using libre softwares is a daily win :).

4.5 Virtualization

You will need virtualization tools using hypervisor. As some students will use VirtualBox for rookie reasons, some may use QEMU/KVM. Or even Proxmox? You can use whatever you want, but now that you've passed the first ING1 semester, you should consider leveling up and use better adapted softwares.

4.6 Documentation and README

You must document everything **Claire Leroux** needs to

- Install the virtualization softwares
- Create the virtual machines
- Configure the virtual machines
- Download the Linux distros and ISO files
- Install the distros inside the virtual machines
- Configure the installations of these virtual machines
- Configure the distros post install
- Deploy your files
- Compile the kernel module
- Load the kernel module
- Install your attacking software
- Use your attacking software
- Why do you use this specific distro?
- And why at this specific version?
- Why this specific kernel version?
- ... everything more she needs to become the best hacker using your project I haven't think about

Write a step-by-step documentation. You can use screenshots.

Re-read and rethink about all the steps. The ones I've wrote just above and the ones you may need to add to have a full documentation. Don't hesitate to add more if you think it needs more information.
You are expected to document **E V E R Y T H I N G**.

4.7 Attacking virtual machine and attacking program

The use of the attacking virtual machine and attacking program is to be a *computer* you already control. In a real-world situation, it's a server with a public IP (and sometimes with a dumb domain name) waiting for the connection of the rootkit. In this case, it's a virtual machine.

Once the rootkit sends a connection, you can use your software to send commands to control the victim machine. It means your rootkit needs to know the IP to contact.

How do I do this? Do I need to put a static IP from the virtual machine configuration using the hypervisor? This is the closest to a real world scenario. Or do I ask the user to look at the IP of the attacking machine given by the DHCP first and put it as a parameter when loading the rootkit inside the kernel?

You will earn the same amount of points doing one or the other. Once again this is a **pedagogical** rootkit. I just want you to think about the possibilities and have fun doing it.

4.8 Victim virtual machine and rootkit

The victim machine will load your rootkit inside the kernel and you will be able to control the victim machine from your attacking machine.

The victim machine can be the distro you want at the version you want but using a Linux kernel at least version 4.0.0.

You don't have to use the most recent Linux kernel. Because it has security inside making it more difficult to use a rootkit, unless you know how to deactivate them from your rootkit going unnoticed. You will earn the same amount of points if you use X or Y version of a kernel. Once again this is a **pedagogical** rootkit. I just want you to think about the possibilities and have fun doing it. But the most recent you can use, the better for you. All I need from you is to technically explain why do I have to use the version you're asking me to use and not a more recent one. Don't just write "Because then it doesn't work". I want technically to know what part won't work if I use a more recent kernel.

The victim machine can be configured the way you want. The more security you keep, the better for you. But you can ask me to deactivate specific securities (and you have to document the reasons, technically). You will earn the same amount of points if you decide to install it with full security or decide to deactivate some securities. Once again this is a **pedagogical** rootkit. I just want you to think about the possibilities and have fun doing it. But I need you to document technically why I have to deactivate certain securities if you decide to deactivate them.

4.9 I've almost forget

Once again... You are expected to document **E V E R Y T H I N G**.

5 Grading

5.1 Grading

You have features to implement. As you know, I am very hungry for documentation.

For the grading, You will be grade on each feature **if and only if** is it accompanied by documentation.

5.2 10/20

Some features are non-negotiable to implement. They will be marked as **mandatory**. If you implement all mandatory features, you will get the grade to validate the project.

The mandatory features will give you some amount of points:

- Compile: 0.5 point
- Connection: 3 points
- Persistence: 1.5 points
- Executing commands: 5 points

5.3 The other ones

The next features will give you some amount of points:

- Password: 1 point
- Upload: 1.5 points
- Download: 1.5 points
- Hide the rootkit from the module list: 1 point
- Hide a line from a file in userland: 2 point
- Hide a directory / files from userland: 2 point
- Cipher the data on the network: 1 point

6 Attacking VM and the attacking program

6.1 Goal

In this exercise, you are expected to produce a virtual machine and a program to communicate with the rootkit.

6.2 Virtual machine

The virtual machine executing the attacking program must be a Linux distribution. It can be whatever distribution you want.
It must run on the EPITA's laptop.

6.3 Technology

You are free to use the technology you want to code the attacking program. As long as it run on the attacking virtual machine.

Make it fun.

You like C? Make it in C!
You like Python? Make it in Python!
You like assembly? Forget this idea!
You like Java? Make it in Java! You prefer to script Java instead of compiling it? Make it in Javascript! (How many students are raging right now?)

You like another tech? Use this tech!

You want to make it as a CLI program? Make a great CLI program!
And you're not even obligated to do everything with text input:

- You want to have a web UI? Make a cool web UI!
- You want to have a graphical application? Make a nice GUI!

Don't hesitate to make buttons and graphics stuff to command and control from your UI if you want to develop more than a CLI program. Have some fun!

Be sure to document everything Claire Leroux needs to deploy and use it without having to guess from the beginning to the end.

7 Victim VM and EpiRootkit

7.1 Goal

In this exercise, you are expected to produce a kernel object serving as a rootkit.

7.2 Virtual machine

The virtual machine executing the rootkit must be a Linux distribution. It can be whatever distribution you want.
It must run on the EPITA's laptop.

7.3 Module kernel creation

Your Makefile must be able to create the **epirootkit** module kernel and clean the repo.

You are free about your Makefile, so make whatever target you want to make whatever you want. Just document it.

7.4 Rootkit features

The following features can be made in the order you want. I list them in the order I think is the easiest, but make yourself confortable if you want to make them in the reverse order.

7.5 Compiling

This is **mandatory**.

Please make your code compile...

7.6 Connection

This is **mandatory**.

I want a visual alert about the state of the connection between the rootkit and the attacking program. Once loaded in the kernel, your rootkit must contact the attacking program installed on the attacking server. Make your attacking program visually alert (stdout, stderr, in a log file displayed on real time, whatever ...) the connection.

If your rootkit can't contact the attacking program or get disconnected, it must continue to try to contact the attacking program until the connection is successful. The attacking program must then change the state of the connection.

It is for you to think about how long it goes without trying a new connection? No waiting? 1 minute? 5 minutes? You will earn the same amount of point, anyway, but make it works (and don't make it too long for my own test please...).

The connection **must be** persistent. Sending one socket to alert the attacking program and getting disconnected just after that is not good enough.

7.7 Persistence

This is **mandatory**.

Your rootkit must stay loaded after a reboot.

7.8 Password

This is **mandatory**.

Secure the access to the rootkit with a password. Don't make it a hardcoded value in the rootkit.

7.9 Executing commands

This is **mandatory**.

From your attacking program, make it possible to execute programs or scripts and getting the **stdout**, **stderr** and **exit status** back to the attacking program.

7.10 Upload and download

Make commands in your attacking program to upload and download files.

It is at your own choice to upload the files wherever you want on the filesystem (at the root? in a directory only for the rootkit?).

7.11 Cardboard box

Your rootkit may needs files in the filesystem. What if something from the userland tries to access it? Make a directory specially for your rootkit and forbid its access from userland. Hide your files from users!

If you're using a system file to be persistent, be able to hide the content you added from the user. Only the content you added, not the entirely of the file.

Maybe hooking on some syscalls would be useful?
(read(2), write(2), getdents64(2) are a great start, ***kof kof***).

Last but not least, hide your module from the modules list. Be indetectable, like a ninja.

7.12 Crypto

All your commands and connection are in plaintext on the network, network admins will see your attack! Encrypt the connection.

7.13 Free porn

Do you have new ideas? Make them happen!

You can ask me if you want to know if your idea is worth implementing.

The idea here is to have fun. Make a super cool rootkit with awesome features!

Here are some ideas for bonuses, but you can propose something else:

- Make a command to upgrade the rootkit at a newer version
- Hide your own network packets
- Intercept keystrokes and log them to the attacking program
- Redirect network traffic somewhere else
- Make a reverse shell for the attackers
- Control your rootkit from a Discord channel (we call this a CL0Pinette/LaGelee dans le milieu)

8 Documentation

8.1 Official

Official documentation for the Linux kernel development and Linux kernel objects development: https://www.kernel.org/doc/html/latest

8.2 Books

- **Professional Linux Kernel Architecture** ; ISBN: 978-0-470-34343-2
- **Linux Kernel Debugging** ; ISBN: 978-1-80107-503-9
- **Linux Kernel Programming - Part 1** ; ISBN: 978-1-78995-343-5
- **Linux Kernel Programming - Part 2** ; ISBN: 978-1-80107-951-8
- **Rootkits and Bootkits** ; ISBN: 978-1-59327-716-1

8.3 Useful links

- https://www.kernel.org/doc/html/latest/core-api/printk-basics.html
- https://www.kernel.org/doc/html/latest/networking/kapi.html
- https://archive.kernel.org/oldlinux/htmldocs/kernel-API/API-call-usermode
  html
- https://archive.kernel.org/oldlinux/htmldocs/kernel-API/API-call-usermode
  html
- https://archive.kernel.org/oldlinux/htmldocs/kernel-API/API-call-usermode
  html
- https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules
  html#exercises
- https://linux-kernel-labs.github.io/refs/heads/master/labs/kernel_modules
  html#extra-exercises
- https://tldp.org/LDP/lkmpg/2.6/html/index.html

- https://xcellerator.github.io/posts/linux_rootkits_01/
- https://x86sec.com/posts/2020/03/08/linux-kernel-rootkit/
- https://github.com/rootfoo/rootkit
- https://github.com/SourceCodeDeleted/rootkitdev-linux

Please do not copy paste snippets of code like des petits sagouins. :)