Malware

Laboratory 1: Rootkit creation

http://personals.ac.upc.edu/enricm/Docencia/Malware/Lab1.pdf

Enric Morancho
(enric.morancho@upc.edu)

Departament d'Arquitectura de Computadors
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

2026-27

Content is licensed under a Creative Commons Attribution 4.0 International license

Index

- Rootkit concept
- Linux case
- Windows case

Rootkit

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- A rootkit is a collection of computer software, typically malicious, designed to enable access to a computer or an area of its software that is not otherwise allowed (for example, to an unauthorized user) and often masks its existence or the existence of other software [1].
- This lab focuses on **masking the existence of a process and faking the contents of a file**.
  - Linux case:
    - Hide a process to `ps` command
    - Fake the contents of a file
  - Windows case:
    - Hide a process to Task Manager
  - We will devote three lab sessions to this lab unit

Index

- Rootkit concept
- Linux case
  - Hide a process to ps command
  - Fake the contents of an ordinary file
- Windows case

Software environment

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- We must access a Linux system
  - Physical or through a virtual machine
  - All Linux distributions are OK
  - Standard tools (gcc tool chain)
  - Unprivileged access

Index

- Rootkit concept
- Linux case
  - Hide a process to `ps` command
  - Fake the contents of an ordinary file
- Windows case

How does ps command work?

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- `ps` displays information about a selection of the active processes
- `ps` command traverses the `/proc` pseudo file system
  - `/proc` is an interface to kernel data structures
    - Contents created *on the fly* by the kernel when the directory is accessed
  - `/proc` contains a directory for every process
    - Directory names are the process identifiers (*pid*)
    - Each directory is populated with many files that give information about the process (process status, memory mappings, open files, network connections, ...)
- In Linux, directory traversing is performed by `openat()` and `getdents()` system calls
  - However, `libc` library wraps these system calls by library functions `opendir()` and `readdir()`
    - `ps` source code calls these library functions
    - Other commands (`top`, `ls`, `find`, ...) also call them to scan directories

Example: opendir() and readdir()

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

```c
#include <dirent.h>

DIR *dirp;                              // Directory stream
struct dirent *direntp;                 // Directory entry

dirp = opendir(directory_name);         // Opens directory stream

while ((direntp = readdir(dirp)) != NULL) {   // Next file
    // direntp points to a directory entry struct
    // that describes a file with fields like:
    // - d_ino: inode number
    // - d_name: null-terminated filename
    // ...
    ...
}

closedir(dirp);                         // Closes directory stream
```

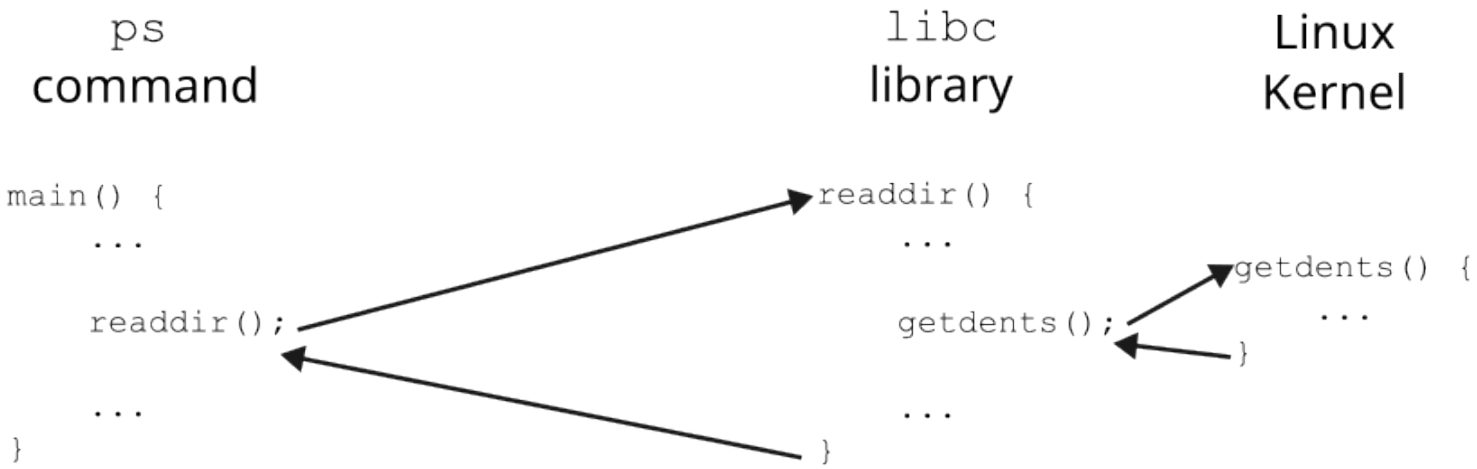
Linux `libc` interception (1/2)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- We will exploit that `libc` library is **dynamically linked** to `ps` exec file
  - `readdir()` calls are resolved at run time
- We will **intercept** `readdir()` library calls
  - By preloading a new dynamic library that also implements `readdir()`
  - With `dlsym()` function we can resolve the memory location for the real `readdir()` function
  - The intercepting `readdir()` calls the real `readdir()` and, if the retrieved data corresponds to the process we want to hide, just iterates again to the next directory entry
- The new dynamic library must be compiled with flags `-fPIC -shared`
  - `PIC`: Position Independent Code
  - `so`: shared object
  - Advice: use `make` tool and `Makefile` to automate compilation
- Library preloading by running `LD_PRELOAD=./library_name.so ps`
  - Or modifying the file `/etc/ld.so.preload` and running just `ps`

Linux libc interception (2/2)

Original Execution



Intercepted Execution



Expected result (exercise 1)

- If `lib1.so` hides information about process with pid=11009

```
$ ps
  PID TTY          TIME CMD
11009 pts/0    00:00:00 bash
14223 pts/0    00:00:00 ps
$
$ LD_PRELOAD=./lib1.so ps
  PID TTY          TIME CMD
14255 pts/0    00:00:00 ps
$
```

- Process with pid=11009 is hidden to command `ps`

Suggested roadbook (exercise 1)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

1. Implement a basic function that intercepts `readdir()`:
   - At library load time:
     - Get the address of the original `readdir()` function
     - Use `__attribute__((constructor))` to execute code at load time
   - Implement your `readdir()` function
     - Just forwards the call to the original one
   - Reference implementation: lib1.c, Makefile
     - In some systems, you must intercept `readdir64()`
2. Implement a PID-filtering mechanism that retrieves the target *pid* from three possible sources (from lowest to highest priority, where higher priority overrides lower):
   - Use a hardcoded *pid* in the library code
   - Retrieve the *pid* from an environment variable (e.g., `FILTER_PID`)
   - Read the *pid* from a configuration file (e.g., `/tmp/lib.cfg`)
3. Be careful with false positives
   - Think about lateral effects with other commands and ordinary files with the same name

Index

- Rootkit concept
- Linux case
  - Hide a process to ps command
  - Fake the contents of an ordinary file
- Windows case

Fake the contents of an ordinary file

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- Our goal is to fake the content of an ordinary file (e.g.: `/etc/passwd`)
  - `cat /etc/passwd` (and any other command reading this file) must get an alternative content
- We consider two approaches to store the fake content:
  - Another file (e.g.: `/tmp/passwd`)
  - The data section of the library object file
- Our intercepting library will provide alternative file content
- You can assume...
  - Read-Only mode
  - Absolute filenames
    - Otherwise, you should construct them or track i-node numbers

Expected result (exercise 2)

- `lib2.so` fakes contents of `/tmp/aaa` with `/tmp/aaa.fake`

```
$ cat /tmp/aaa
original content
$ cat /tmp/aaa.fake
fake content
$ LD_PRELOAD=./lib2.so cat /tmp/aaa
fake content
$ LD_PRELOAD=./lib2.so grep t /tmp/aaa
fake content
$ LD_PRELOAD=./lib2.so more /tmp/aaa
fake content
$
```

Suggested roadbook (exercise 2)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

1. Intercept `open()` library function:
   - Interface to `open()` system call
   - Typically used by `cat`, `wc`, ... commands
   - As `open()` is a variadic function, you will need `stdarg.h`'s macros to retrieve optional parameters [2]
   - If the file to be opened is the target file, open the alternate file instead
2. Intercept `openat()` library function:
   - Interface to `openat()` system call
   - Typically used by `grep`, ... commands
   - Also a variadic function
   - Proceed analogously
3. Intercept `fopen()` library function:
   - Corresponds to `fopen()` stdio library function
   - Typically used by `more`, `vim`, ... commands
   - Proceed analogously
4. Depending on the Linux distribution, you might need to intercept `open64()`, `openat64()` and `fopen64()`.
   - You may use `gdb` or other tools to figure it out

Expected result (exercise 3)

- `lib3.so` fakes contents of `/tmp/aaa` with the contents of a buffer initialized in `lib3.c`
  - Useful in read-only filesystems
  - It is invisible to tools that scan the filesystem looking for new files

```
$ cat /tmp/aaa
original content
$ grep fake_content lib3.c
...
fake_content[] = "This is the fake content of the file";
...
$ LD_PRELOAD=./lib3.so cat /tmp/aaa
This is the fake content of the file
$ LD_PRELOAD=./lib3.so grep fake /tmp/aaa
This is the fake content of the file
$ LD_PRELOAD=./lib3.so more /tmp/aaa
This is the fake content of the file
$
```

Suggested roadbook (exercise 3)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

1. Using `memfd_create()`, create an anonymous file stored in RAM
2. Initialize its content
3. Intercept `open()` and `openat()` library functions
4. Use `fdopen()` to associate a `FILE *` structure to the file descriptor
5. Intercept `fopen()` library function

Exercise 4

- The proposed solutions to exercises 2 and 3 are not good enough
  - There is a trace of the trick in the `/proc` directory
    - The `fd` directory of the `/proc` entry of each process shows the file descriptors currently opened by the process and their corresponding files

```
$ ls -l /proc/pid of the process/fd # Exercise 3
total 0
lrwx------ 1 ... 0 -> /dev/pts/0
lrwx------ 1 ... 1 -> /dev/pts/0
lrwx------ 1 ... 2 -> /dev/pts/0
lrwx------ 1 ... 3 -> '/memfd:...'
$
```

- Try to solve the issue....
  - On the `cat` scenario (`open()` syscall), implement a solution where `/proc/`*pid*`/fd` shows that the process opened the real filename
  - Hint: To show the real filename in `/proc`, the application must open the real file. Then, you should intercept the `read()` operations too.
    - On newer systems, `cat` may process files using the `splice()` syscall or direct syscalls rather than the classic `read()/write()` loop.
      In this case, you can force `cat` into the `read()/write()` loop to intercept `read()` calls by using the `-A` flag (`cat -A /tmp/aaa`).
  - Think if you should intercept other calls like `close()`, `stat()`, ...

Final remarks on library preloading (1/2)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- In some cases, Linux disables library preloading
  - Execution of `setuid` commands, ...
    - Imagine executing `passwd` command with a fake `/etc/passwd` file
- Limitations
  - Statically-linked binaries
    - Compiled with `-static` flag
  - Direct syscall invocations
    - Assembly code

Final remarks on library preloading (2/2)

- Possible extensions for the exercises:
  - Filter-out processes by executable name
  - Context-aware spoofing: vary behavior based on caller process name
    - Different file contents for `cat` versus `shasum`
  - Fake the contents of a whole directory structure
  - Hide network connections
    - `netstat` command also traverses `/proc` to get network information
- More information:
  - `man ld.so`
    - `/etc/ld.so.preload` allows implicit library preloading
    - `LD_DEBUG=bindings` show symbol bindings
    - ...
  - `nm -u -D object_file` lists unresolved dynamic symbols
  - "Linkers and Loaders" [3]
- Library preloading in macOS
  - Uses the environment variable `DYLD_INSERT_LIBRARIES`

Index

- Rootkit concept
- Linux case
- Windows case

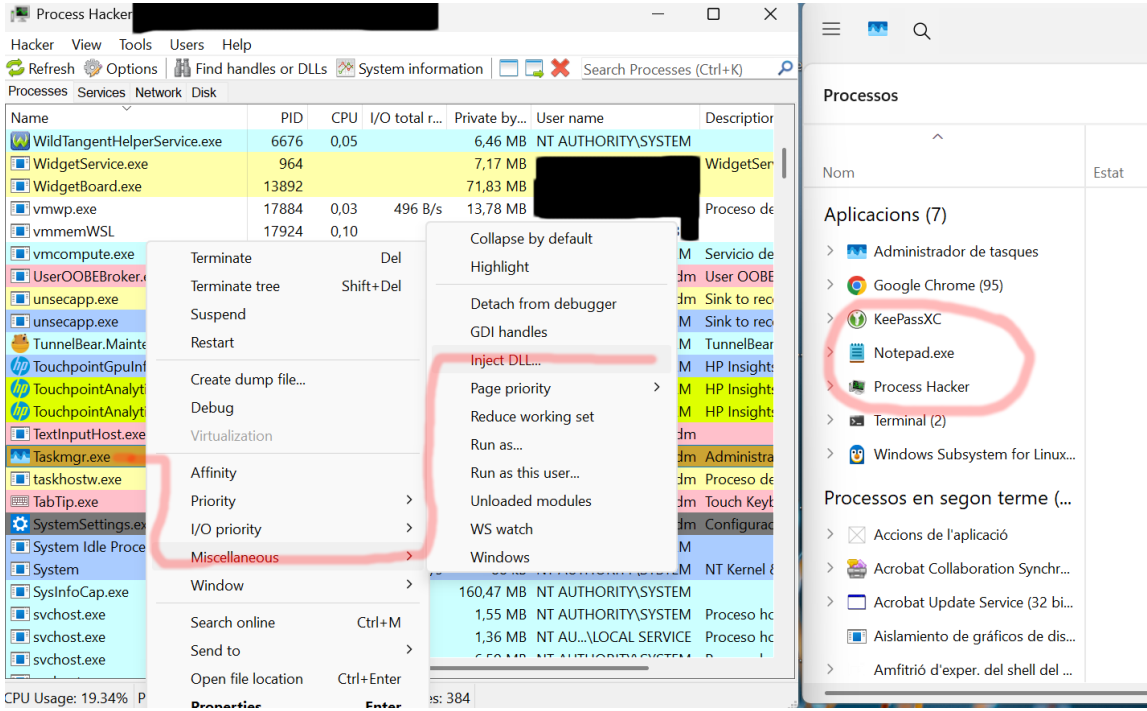
Roadmap

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

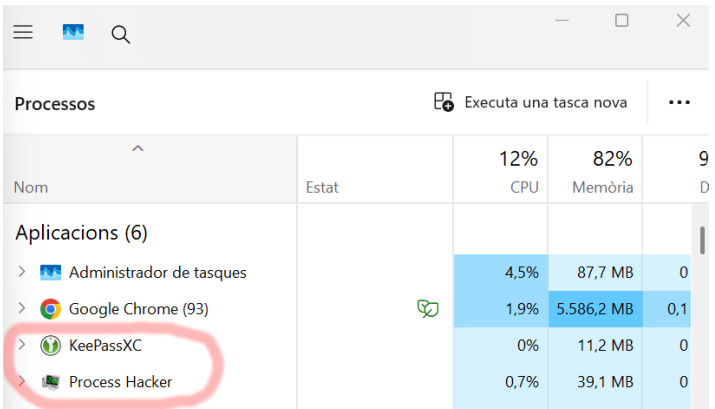
- You will analyze a code that hides a process to the *Task Manager*
- Software environment
  - Windows 11 with
    - Visual Studio [4]
    - Microsoft Research Detours Package [5]
    - Process Hacker (aka System Informer) [6]
- Follow the steps in [7].
  - This code appears to be harmless, but run it at your own responsibility.
  - I tried it on an old laptop but couldn't get it to work...
    - The previous course, some students were able to "correct" it.
      Link: `dllmain_corrected.cpp`
    - This version works correctly on my laptop
  - Note that Windows Defender and other antivirus may affect the outcome.

Example: DLL that hides Notepad.exe

DLL injection
in Process Hacker
(in System Informer,
Properties → Modules
→ Options → Load Module



Result



Deliverable for Lab1

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- Each group must deliver a `zip` file with:
  - A `pdf` file explaining the difficulties you encountered, how you solved them, screenshots showing your codes in action and AI usage declaration. (10%)
  - Your source-code files:
    - Linux intercepting library `lib1.c` and `Makefile` (Slide 12) (25%)
    - Linux intercepting library `lib2.c` and `Makefile` (Slide 16) (15%)
    - Linux intercepting library `lib3.c` and `Makefile` (Slide 18) (20%)
    - Linux intercepting library `lib4.c` and `Makefile` (Slide 19) (20%)
    - Windows DLL (Slide 23) (10%)
- Upload your zip file to Atenea [8].
- Deadline: October 20th, 23:59.

Index

- Appendix
  - GOT (Global Offset Table) Hooking

Index

- Appendix
  - GOT (Global Offset Table) Hooking

GOT (Global Offset Table) Hooking (1/2)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- **Linux (ELF binaries)**: Uses the GOT for dynamic linking:
  - Stores the resolved runtime addresses of shared library functions.
  - Resides in the virtual memory space of the process.
  - Initialized at load time or on first call (*lazy binding*).
- **Function Interception**: Overwriting GOT entries redirects calls:
  - Locating target entries requires parsing the ELF dynamic section.
  - Protected by **RELRO** (*Relocation Read-Only*):
    - *Partial RELRO*: Keeps `.got.plt` writable for *lazy binding*.
    - *Full RELRO*: Makes the whole GOT read-only at startup.
- **Windows Equivalent**: Uses the **IAT** (*Import Address Table*):
  - The same hooking concept applies by modifying PE import entries.

GOT (Global Offset Table) Hooking (2/2)

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

- Example: `gotHK.c`

```
$ cat gotHK.c
...
int
fake_puts (const char *s)
{
  write (2, "FAKE puts\n", 10);
}
...
int
main (int argc, char *argv[])
{
  puts ("First call to puts()\n");   // Works OK

  // Parses ELF structure and redirects puts() to fake_puts()
  dl_iterate_phdr (replace_got_symbol, NULL);

  puts ("Second call to puts()\n");  // Redirected to fake_puts()

  return 0;
}
$ make gotHK
cc gotHK.c -o gotHK
$ ./gotHK
First call to puts
FAKE puts
$
```

References

[1] *Rootkit*, [Online]. Available: https://en.wikipedia.org/wiki/Rootkit.

[2] Geeks for Geeks, *Variadic Functions in C*, [Online]. Available: https://www.geeksforgeeks.org/c/variadic-functions-in-c/.

[3] J. R. Levine, *Linkers and Loaders*, 1st. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999, ISBN: 1558604960.

[4] Microsoft, *Download Visual Studio*, [Online]. Available: https://visualstudio.microsoft.com/es/downloads/.

[5] Microsoft, *Microsoft Research Detours Package*, [Online]. Available: https://github.com/microsoft/Detours.

[6] *Process Hacker*, [Online]. Available: http://processhacker.sourceforge.net/.

[7] S12 - 0x12Dark Development, *Hide Processes in Task Manager*, [Online]. Available: https://medium.com/@s12deff/hide-processes-in-task-manager-64043c7c2c4b.

[8] *MALWARE (Curs 10) Moodle*, [Online]. Available: https://atenea.upc.edu/course/view.php?id=107508.

Malware

Laboratory 1: Rootkit creation

http://personals.ac.upc.edu/enricm/Docencia/Malware/Lab1.pdf

Enric Morancho
(enric.morancho@upc.edu)

Departament d'Arquitectura de Computadors
Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya



2026-27

Content is licensed under a Creative Commons Attribution 4.0 International license