Université de Liège

INFO0940-1 Operating Systems

Project 1 - Tracer

Authors : BAKIJA Asad
POLET Quentin

Teachers : L. MATHY
G. GAIN
J. FRANCART

Second semester 2020-2021

Master in engineering/computer science

1 Implementation

1.1 Profiler mode

The idea is quite simple. The tracer will follow the execution of the tracee with the help of ptrace. As soon as a call instruction is detected it pushes the instruction into a stack keeping in memory its name and return address. The return address is simply the current address + 5, the relative call operation being coded on 5 bytes. For the call address (used to find the name which is present in nm result), the next for bytes of the instruction are read. These bytes contain the offset of the called function. When a return is detected, the tracer checks that it is the address of the function at the top of the stack. If this is the case, the function is popped out of the stack and the parent function (the one that called the instruction that has just been popped) has its instruction counter incremented by the number of instructions counted by the popped function.

At each instruction, the current function, the one at the top of the stack, sees its instruction count incremented by one. In addition to that, the different functions are structured in a tree, which allows to easily display the call tree at the end of the execution.

1.2 Syscall mode

Using *ptrace* and *PTRACE_SYSCALL*, the tracer stops the tracee each time it enters or leaves a syscall. We simply call it twice (to stop at each syscall entry) and, between these two calls, we look at the value of *orig_eax*, which contains the code of the current syscall. We retrieve its name using an array containing the names of all the provided syscalls.

2 Questions

2.1 What are the opcodes that you have used to detect the call and ret instructions ?

Since we only had to consider relative calls, we have used the following opcodes :

- call : E8
- ret : C2 and C3

2.2 Why are there many functions called before the ”real” program and what is/are their purpose(s) ?

The purpose of the functions called before the main is to make sure that everything is in a coherent state, call the main function and handle the return.
The first function called is *__libc_start_main*. As stated in its specification : “The __libc_start_main() function shall perform any necessary initialization of the execution environment, call the main function with appropriate arguments, and handle the return from main().” [1]

1. `https://refspecs.linuxbase.org/LSB_3.1.0/LSB-generic/LSB-generic/baselib---libc-start-main-.html`

2.3 According to you, what is the reason for statically compiling the ”tracee” program ?

The reason we statically compiled the tracee is because it is easier to analyse afterwards. Once statically compiled, its function names can be retrieved by looking at the ELF file using a tool such as nm.
When dynamically compiled, the program needs to know the addresses of the functions it will call in order to be able to use them. To do so, a mechanism called relocation is used. This operation is done by the dynamic linker, which is called when the program is run.
Relocations depend on 4 sections of the ELF file which are .got (Global Offset Table), .plt (Procedure Linkage Table), .got.plt and .plt.got. Without going into too much detail, these sections are filled and loaded into the program's memory at startup or updated during the execution by the dynamic linker. Using these sections, the relocation mechanism allows the process to get the addresses of functions defined in dynamically linked libraries during execution.
Relocation is still used even if the program is statically compiled. This can be observed by analysing the ELF file of the tracee [2], which contains .got and .plt sections. It is done for performance reasons [3] and was seen during the project since some function names could not be retrieved by simply analysing the ELF file.

3 Time spent and main difficulties

The work was interesting and not very complicated once we understood what to do and how to do it, where to look, ... We would have liked more details from the beginning. Searching for something specific is interesting but searching in total vagueness is quite frustrating and wastes a lot of time.

2. readelf -S tracee
3. `https://reverseengineering.stackexchange.com/a/6397`