MCS: Secure Execution Environments 2022-23

Practical Exercises: Supervision of a Linux process execution using `ptrace`

March 24, 2023 Due date: no date

Changelog

- v1.0 - Initial version.

1 Introduction

The goal of this laboratory guide is to learn the fundamentals behind the tracing of processes with the system call `ptrace`. Is system call is the cornerstone of several operations, such as debugging and tracing the execution flow of an application. Check the applications `strace` and `ltrace` for further details.

These exercises must be executed in Linux systems.

2 The `ptrace` system call

In Linux a process can control, or trace, the execution of another process by using the system call `ptrace`. However, to do so, the tracing process must have some privileges over the tracee; usually the tracee is a child of the tracer. Processes with the same effective UID can trace each other.

A processes being traced usually enters the STOPPED state upon initiating a system call, and the same happens when leaving that same system call. This allows a tracer to wait for tracee process to enter such state (with the system call `wait` or any other similar one) and then to analyse the system call being requested, or the values being returned by a system call.

When `ptrace` is used by a debugger, it is is able to set breakpoints at specific locations on binary executables by replacing the instructions in the beginning of a breakpoint location by the sequence of CPU instructions that are used in all system call: a trap. Upon this trap, the program being debugged enters the STOPPED stated, and the debugger gets control over it, being then able to restore the original instructions that where replaced by the trap in order to resume the execution of the debugged program.

For more details, execute:

```
man ptrace
```

3 The `ptrace`-based system call follower

The following program is a highly-simplified twin of the `strace` command: it launches an arbitrary program that gets its system calls followed by a tracer. The tracer currently highlights the calling of 2 system calls, `open` and `close`, but others could be highlighted as well.

Empirically you can notice that this program is not able to catch and follow system calls that create new processes (`fork`, `vfork`, `clone`, etc.). Explain why. As a homework, change the program to do so. Note: you need, at least, to use extra options, you need to change the way the tracer uses to wait for child processes and you need to change the following of system call entering/existing sequences.

```c
#include <sys/types.h>
#include <sys/ptrace.h>
#include <sys/syscall.h>
#include <sys/wait.h>
#include <sys/reg.h>
#include <sys/user.h>
#include <stdio.h>
#include <unistd.h>

int
main ( int argc, char ** argv )
{
    pid_t child;

    if (child = fork()) { // father
        int status;
        int syscall_entering = 1;
        struct user_regs_struct user_regs;
        int options_set = 0;

        for (;;) {
            wait( &status );
            // printf( "Status = %d\n", status );
            if (WIFEXITED(status)) return 0;

            if (options_set == 0) {
                // ptrace( PTRACE_SETOPTIONS, child, 0, 0 );
                options_set = 1;
            }

            if (syscall_entering == 1) {
                syscall_entering = 0;

                ptrace( PTRACE_GETREGS, child, 0, &user_regs );

                // printf( "[%d] Entering syscall %lld\n", child, user_regs.orig_rax );
                if (user_regs.orig_rax == SYS_openat) {
                    printf( "[%d] Called open\n", child );
                }
                else if (user_regs.orig_rax == SYS_close) {
                    printf( "[%d] Called close\n", child );
                }
            }
            else {
                syscall_entering = 1;
                // printf( "[%d] exiting syscall\n", child );
            }

            ptrace( PTRACE_SYSCALL, child, 0, 0 );
        }
    }
    else { // child
        ptrace( PTRACE_TRACEME, 0, 0, 0 );
        execvp( argv[1], argv + 1 );
    }

    return 0;
}
```