CSC369 Assignment 1 - Hijacking System Calls and Monitoring Process

May 18, 2020

1 Overview

- In this assignment, you will achieve the goal of hijacking (intercepting) system calls by writing and installing a very basic kernel module to the Linux kernel.

  Here is what "hijacking (intercepting) a system call" means. You will implement a new system call named my_syscall, which will allow you to send commands from userspace, to intercept another pre-existing system call (like read, write, o pen, etc.). After a system call is intercepted, the intercepted system call would log a message first before continuing performing what it was supposed to do.

  For example, if we call my_syscall with command REQUEST_SYSCALL_INTERCEPT and target system call number __NR_mkdir (which is the macro representing the system call mkdir) as parameters, then the mkdir system call would be intercepted; then, when another process calls mkdir, mkdir would log some message (e.g., "muhahaha") first, then perform what it was supposed to do (i.e., make a directory).

  But wait, that's not the whole story yet. Actually we don't want mkdir to log a message whenever any process calls it. Instead, we only want mkdir to log a message when a certain set of processes (PIDs) are calling mkdir. In other words, we want to monitor a set of PIDs for the system call mkdir. Therefore, you will need to keep track, for each intercepted system call, of the list of monitored PIDs. Our new system call will support two additional commands to add/remove PIDs to/from the list.

  When we want to stop hijacking a system call (let's say mkdir but it can be any of the previously hijacked system calls), we can invoke the interceptor (my_syscall), with a REQUEST_SYSCALL_RELEASE command as an argument and the system call number that we want to release. This will stop intercepting the target system call mkdir, and the behaviour of mkdir should go back to normal like nothing happened.

2 Checklist

-

3 Goal

4 Requirements

5 Error Conditions

6 General Information