ITP30002 Operating System

Homework 1

LKM Rootkit

Date assigned:
Apr 7 (Tue)

Due date:
9 PM, Apr 15 (Tue)

Summary

- You are asked to create a Loadable Kernel Module (LKM) that works as an agent in kernel space for your commands
  1. Block a specific user from opening a specific file
  2. Prevent a killing of a processes created by a specific user

- Prerequisuite
  - You have to study three examples to build up backgrounds on LKM
  - You have to have your own system running Unbuntu 16.04.6 because you need to have superuser (root) permission in doing this homework
    - recommend to set a virtual machine on your computer, or on a cloud

- Submission
  - Due date & time: 9:00 PM, Apr 15 (Thur)
  - Deliverables: (1) write-up, (2) demo video (up to 5 min), (3) source code
  - Late submission: will be accepted within 24 hours at 30% penalty

System Requirement

- Use Ubuntu 16.04.6 with Linux Kernel 4.15.0
  - A LKM largely depends on a specific kernel version. It is very likely that your LKM is not compatible if it is developed under another version
  - You can prepare a machine as a virtual one or phicial one.
  - Recommend to use virtual instances
    - VMware: Ubuntu 16.04.6 LTS Desktop image http://releases.ubuntu.com/16.04/
    - Amazon EC2: Ubuntu Server 16.04 LTS (HVM), SSD Volume Type
      - the update instruction is given as `EC2.sh` under the `hw1` branch
- Use GCC 5.4.0 or a higher version

Background: Loadable Kernel Module

- A Loadable Kernel Module (LKM) is a suite of functions in a file (i.e., module) that can be dynamically loaded to kernel space
  - many replaceable sub-components of the kernel (e.g., device drivers) are written as LKM
  - usually built as a .ko file
  - operations for LKM
    - `insmod` : install a LKM
    - `rmmod` : remove a LKM from the kernel
    - `lsmod` : list up currently loaded LKMs

- Example 1. `examples/bareminimum`

- c.f., Writing a Linux Kernel Module – Part 1. Introduction
  http://derekmolloy.ie/writing-a-linux-kernel-module-part-1-introduction/

Backgrond: Proc as LKM Interface

- *Proc* is a virtual file system where each file works as an agent for a kernel component to interact with user programs
  - placed under `/proc`
  - write to a file to send data/command to the kernel, and read a file to receive the information from the kernel

- A LKM can create a proc file with the file operations customized for making communication between kernel and user programs

- Example 2. `examples/hellokernelworld`

Background: Intercept System Call

- A LKM can access to system data structures by a symbol name via the kernel symbol table
  - `(void *) kallsyms_lookup_name(char * name)`

- You can intercept a system call by replacing the handler routine with a function of your own
  - a list of syscall handler types can be found at `include/linux/syscalls.h` of Linux kernel https://elixir.bootlin.com/linux/v4.15/source

- Example 3. `examples/openhook`
  - count how many times a certain file has been opened

Useful Links

- Linux kernel 4.15 source code
  https://elixir.bootlin.com/linux/v4.15/source

- Kernelnewbies.org
  https://kernelnewbies.org/Documents

- Linux kernel programming tutorial
  https://linux-kernel-labs.github.io/refs/heads/master/

Your Assignment

- Create a toy rootkit `mousehole.ko`
  - two features
    - block a specific user from opening a specific file
    - prevent a killing of a processes created by a specific user
  - create a command-line (text) interface `/proc/mousehole`
- Create a user-level program `jerry.c`, a user interface of the mousehole module
  - transfer a user command to the mousehole module via `/proc/mousehole`
  - display the information given from mousehole module

Function 1. Block File Opening of User

- The user gives a filename *fname* and a specific username *uname* to `jerry`
- Then, `jerry` commands `mousehole` to make every opening of a file whose name contains *fname* by a user *uname* fail
  - Still the specifieid user *uname* can access other files, and other users can access the specified files
- `jerry` restores the file opening behavior back to normal if the user gives another command to an earlier set up
- Hint
  - Replace the handler routine for `sys_open()`
  - The uid of a user can be obtained by calling the `id` command

Function 2. Prevent Killing of Processes

- The user passes a username (e.g., `guest`) to `jerry`
- Then, `mousehole` makes no other process kill a process created by the user, until the user commands to release this immortality
- Hint
  - Replace the handler routine for `sys_kill()`
  - It is possible to iterate the list of all processes in the system by using the `for_each_process` macro
    - see `examples/listprocesses`
  - A process is represented as a `task_struct` object in kernel
    - https://elixir.bootlin.com/linux/v4.15/source/include/linux/sched.h#L520

Jerry

- You must construct and use a client application *jerry* to communicate with the mousehole module
  - not directly accessing `/proc/mousehole`

- You must exercise proper system APIs in constructing `jerry.c`

Write Up & Demo

- Write-up
  - up to 2 pages in the given template
  - describe how you accomplish implementing functionalities
  - discuss issues or/and ideas as you had for the homework
  - submit a PDF file

- Demo
  - find scenarios to show that your program fulfills the given requirements
  - videorecord the execution of the scenarios with narration, upto 5 min
    - narration must be given in English
  - upload the video to a streaming service (e.g., YouTube) and submit the URL

Get Some Help from TAs

- Teaching assistants
  - Mr. Jeewoong Kim jeewoong@handong.edu
  - Ms. Juyoung Jeon juyoungjeon@handong.edu
  - Mr. Hanyoung Yoo hanyoungyoo@handong.edu

- Services
  - Help equip Ubuntu systems for experiments
  - Repeat what's explained in this homework description

- How to contact
  - Ask a question on Piazza, or on Slack
  - TAs will offer online help desk sessions. The schedule is to be announced.

Submission

- Your submission must include the followings
  - write-up (upto 2 pages)
    - your write-up will be open for peer evaluation
  - video demo (e.g., YouTube)
    - put the URL in your write-up
  - all related source code files
    - `mousehole.c`
    - `jerry.c`

- How to submit
  - upload your files to a homework repository in **Hisnet**

Evaluation

- Criteria
  - Fulfillment of requirements 40%
  - Soundness of demonstration 20%
  - Clarity in technical description 20%
  - Novelty in discussion 20%

- Notes
  - Evaluation will be primary based on your write-up and video demo
  - TAs will rehearse the demo with your submitted files on Ubuntu 16.04 Kernel 4.15.0
  - Your write-up and video may be open to the classmates for review