CSCI 5573: Advanced Operating Systems
Fall 2024

Programming Assignment Two

Due Date: 11/08/2024

1. Inserting a new system call
   Since system call manipulation is a common technique utilized by rootkits, one useful tool would be to identify every system call that is being intercepted. Continuing from the rootkit you wrote in Programming Assignment One, your next task is to introduce a new system call *rtkit_chk* in the kernel that prints out a list of all system calls that are currently being intercepted as well as all new system calls introduced in the kernel after the introduction of *rtkit_chk*. If there are no such system calls, *rtkit_chk* prints a message "No changes have occurred in the system call table".

2. Scheduling events from user space and in kernel
   The *rtkit_chk* that you have developed identifies system call interference only when it is explicitly invoked. Instead, we would like to check periodically for any changes in the system call table, i.e. without invoking *rtkit_chk* explicitly.

   (a) Write a Cron job to invoke *rtkit_chk* periodically.

   (b) Cron jobs are scheduled at the user space and can easily be interfered with by an adversary, so your next task is to move this periodic scheduling inside the kernel using work queues.

3. A simple sandbox to run untrusted programs using chroot and seccomp
   Write a simple C program that opens an existing text file, prints out the file content on the terminal and then adds a short sentence at the end of the file. Compile this program and create a binary. Next create a jailed environment to run this binary as well as a shell and the ls command. Your jailed environment should provide minimal filesystem access and system call exposure. When happens if you try to run a different program that tries to (1) create a new file, (2) open a file not included in your jailed environment, or (3) invokes an unauthorized system call?

4. Isolated computing environment
   Implement an isolated computing environment called a *capsule*. Input to a capsule is a zipfile provided by the user. When launched, a capsule creates an isolated virtualized computing environment containing the directory zipped in the input zipfile and all files/libraries needed to run a shell (e.g. *bash*) and the *ls* command, and starts the shell with in this computing environment. The zipfile provided by the user contains one or more executable binaries along with all the libraries needed to run those binaries. A capsule enforces the following resource limits on all process that are run with in this isolated environment:

   (1) Processes cannot access any files or directories other than the ones in the input zipfile and the ls and bash commands.
   (2) The total CPU utilization of all processes in a capsule cannot exceed 5% at any point in time.
   (3) The total memory usage of all processes in a capsule cannot exceed 2 MB.
   (4) The maximum disk read/write rate of processes in a capsule is 1 MB/sec.

(5) All processes in a capsule belong to a single isolated process namespace with process ids starting at 1.
(6) Processes in a capsule have their own private mount namespace.
(7) None of the processes in a capsule have access to any network interface.
(8) Processes in a capsule have their own IPC resources.

To test the system, create a set of test programs to check the limits of a capsule. Your test programs should demonstrate what happens if a process tries to access resources it is not authorized to or exceed its resource limits. Include executable binaries of all these test programs in the input zipfile along with all libraries needed, so that you can run them with in a capsule.

5. Extra Credit
   Write an exploit to show that it is possible to escape out of the chroot jail you created in Question 3. NOTE: This exploit is available on the Internet. Make sure you understand very clearly how it works before using it.