Principles of Operating Systems

CMSC 421 - Spring 2018

Project 2

Initial design due by 11:59 PM EDT on April 15, 2018

Final design and implementation due by 11:59PM EDT on May 9, 2018

Introduction/Objectives

In this project, you will create a new version of the Linux kernel that adds functionality to support a simple intrusion detection system. This system will operate by logging the system calls made by a process in the kernel, while analysis and intrusion detection will be done in userspace. This assignment is designed to teach a simple method of intrusion detection, as well as to reinforce the idea of how userland and kernel-space interact through the use of system calls.

Before you begin, be sure to create your new GitHub repository for Project 2 by using the link posted on the course Piazza page. Then, follow the same steps you did in Project 0 to clone this new repository (obviously, substitute `project2` for `project0` from the earlier instructions). You may, at this time, remove the `/usr/src/vanilla-project1` directory you created for Project 1, as you will be cloning your new project 2 repository into a new directory named `/usr/src/vanilla-project2`.

As a first step, change the version string of the new kernel to reflect that it is for Project 2 for this course. That is, make the version string read `4.15.0-cmsc421project2-USERNAME` (substituting your UMBC username where appropriate).

For this project, we request that any new code/documentation be placed in your kernel source tree as follows:

```
/usr/src/project2/                    - Root of your kernel source tree
|- p2ids/                             - All new code/documentation will be in this subdirectory
   |- Makefile                        - Kernel-space Makefile (only referring to the kernel/ subdirectory)
   |- README                          - Basic README explaining what's in the directory, how to use it, etc.
   |- design/                         - Directory containing design documentation
      |- initial.pdf                  - Initial design document (due April 15th)
      |- final.pdf                    - Final design document (due with final submission)
   |- userland/                       - Directory containing user-space code
      |- Makefile                     - Used to build the user-space code
      |- *.c, *.h, *.py, *.cpp, etc.  - User-space code for your project
   |- kernel/                         - Directory containing kernel-space code
      |- Makefile                     - Used to build the kernel-space code
      |- source files and directories- Your kernel-space code, possibly in subdirectories as needed
   |- idsdata/                        - Directory containing sampled "good" data for the operation of the IDS
      |- README                       - Document describing how to use data files with the IDS
      |- * (files and directories)    - As needed
   |- misc/                           - Optional directory containing any other miscelaneous information/data
      |- README                       - Document describing what any data/documentation here is.
      |- * (files and directories)    - As needed
```

Incremental Development

One of the nice things about using GitHub for submitting assignments is that it lends itself nicely to an incremental development process. As they say, Rome wasn't built overnight — nor is most software. Part of our goal in using GitHub for assignment submission is to give all of the students in the class experience with using an source control system for incremental development.

You are encouraged in this project to plan out an incremental development process for yourself — one that works for you. There is no one-size-fits-all approach here. For instance, you could easily contrive fake logging sequences to prototype/debug the user-space component of this project without even touching the kernel-space portion. If in doubt, please contact your TAs to determine if a given incremental development approach seems feasible.

You should not attempt to complete this entire project in one sitting. Also, we don't want you all waiting until the last minute to even start on the assignment. Students doing either of these in this course tend to end up with poor grades on assignments. To this end, we are requiring you to make at least **five** non-trivial commits to your GitHub repository for the assignment. These five commits must be made on different dates and at least **two** must be done during the initial design phase of the project (that is to say that two commits must be done by 11:59 PM EDT on April 15 for full credit).

A non-trivial commit is defined for this assignment as one that meets all of these requirements:

- Does not contain only documentation (i.e, just committing a `README` file does not count), with the exception of significant work on a design document. That is to say that your required design document **does** count toward your required commits.
- Does not contain only `Makefile` modifications or creation.
- Modifies/creates at least 10 lines of code in a combination of existing or newly created .c or .h files. That is to say, creating a new file with 10 lines of **code** counts, but creating a new file with a 10 line comment does not.
- Code modifications/creation must be relevant to the project. Creating a bunch of useless files/functions that are unrelated or otherwise superfluous to the assignment does not count. It is ok to reorganize your code after you have started and remove pieces of code, of course, but if you are obviously only adding code to the repository early on that you completely delete later (or that has nothing to do with the assignment), then that commit will not count toward the requirements herein.

Failure to adhere to these requirments will result in a significant deduction in your score for the assignment. This deduction will be applied after the rest of your score is calculated, much like a deduction for turning in the assigment with a late penalty.

Project Design

Much of your grade on this assignment (35%) is based on the design of your system (so, only 65% of the grade is based on your actual code). A system of this scope should warrant careful design consideration, especially considering the portions of the kernel source code that you will be required to modify in order to make it work. To that end, we are requiring you to submit a design document early on in the project period to ensure that you are on the right track with your thought process on the assignment.

For a project of this scope, we are expecting a 3-5 page initial design document explaining what code you will write, both in userland and in kernel-space. In particular you should document where you intend to make changes to the kernel source code (including files and approximate line numbers). You should strive to make your design a simple to follow as possible, and to minimize any changes to the kernel source code where possible. That is not to say that you should make your changes so small that the system cannot work, but it also means that you should not be writing several thousands of lines of code in the kernel either. Your initial design document should show that you have taken the time to put careful consideration into the proposed implementation of the system. The more detailed your initial design, the more likely that we will be able to evaluate the feasibility of your design (and provide guidance to potentially correct any design issues early on).

Your final design document should be detailed enough that a skilled kernel programmer could follow the document and re-implement your design without consulting your code. We expect you to revise your document to keep up with any changes you make in your design as you move forward with the implementation of the project — that is to say that your final design document should be more detailed than your initial one, and may even contain segments of your code to help explain the system, if necessary. As this document should be more detailed than your initial design, we are also expecting it to probably be of a greater length than the initial design document. To this end, we are expecting the final design document to be at least 5 pages long.

Intrusion Detection Systems

An intrusion detection system (IDS) is a computer program that attempts to identify (and thwart) attacks that might be performed on the system by various "bad guys". Attacks on computer systems, especially those that are parts of corporate and government networks are extremely common, and there are many different approaches to preventing such attacks. In this project, you will be implementing a relatively simple IDS that could be used on a Linux-based computer to help identify (and prevent) an attacker from gaining access to the system's resources.

There are several time-tested approaches to the development of an IDS, but you will be focusing on a single interesting approach to the problem: keeping track of the system calls made by a monitored process and checking for abnormalities in the sequences of system calls made. When an attacker breaks into a process (through a buffer-overflow or some other means), he or she will need to make system calls in order to attempt to access the resources of the system that are under attack. As the system calls that the attacker will perform will likely be different than those performed by a process that is not under attack, it follows that by monitoring both healthy and broken processes, we can develop a scheme to identify those that might be under attack for further action to be taken.

A naïve approach to developing an IDS would be to simply monitor a process and see if it were to make any system calls that it would normally not make. While this may identify some attacks, it is certainly not the most effective way to do so while monitoring system calls made by a process. Instead, you will be implementing an IDS which compares sequences of system calls made by a monitored process to known good sequences (which you will also have to build as a part of your IDS). By this, we mean that you should make reference logs of system calls made by processes and use them as a part of your system to determine if a process might be under attack.

Consider a process that makes the following system calls in the following order (where O = `open`, R = `read`, M = `mmap`, W = `write`, and C = `close`):
O R M M W R C
If we were to examine this stream of system calls with a window size of 5, we would see the following three sequences:
O R M M W
R M M W R
M M W R C

If we were to construct a reference log of all of the system calls made by the process, we could examine it with varying window sizes to see if any sequences seem abnormal for the process, and flag that for further review or for termination by the IDS. Experimental evidence suggests that even short sequences of system calls (of length six or more) provide an adequate baseline signature of normal functioning processes which has a high probability of being perturbed during an attack.

In this project, you will be required to implement a somewhat simplified version of the window-based system described above (with extra credit available to those who go further). You will be required to instrument the system call dispatcher of the Linux kernel with code that logs each time a system call is made. The log entries should contain the process ID and the system call number at a minimum (you may include any other information that you deem appropriate, such as user/group IDs, timestamps, etc.). You will then make a separate user-space process that runs in the backgorund and monitors these logs, checking for abnormalities in them. At a minimum, your user-space process should construct a bit array for each process under monitoring showing which system calls have been run in a window of the last `k` system calls (where `k` is a constant of your choosing — which should be chosen to demonstrate a working system). If a particular system call is made in the window, then you should set the bit for that system call to 1. Any system calls not made in that window would then have values of 0.

From this point, you must develop an idea of what is a healthy process versus one that is under attack. One useful metric would the hamming distance between a given bit string and information contained within the reference logs. For instance, referring back to the earlier logs given, if we order the bits in the order `OCRWM`, then we can see that the process would have exhibited the bit string `11111` in the window with size 7 that was given. The hamming distance between that bit string and one with the value `10110` would be 2.

It is important to note that a healthy process may not always exhibit the exact same behavior from run to run. For instance, consider a web server. If the web server has cached some of the pages it is serving in an in-memory cache of some sort, then it will likely make many less filesystem related calls when a request for a cached page comes in. Therefor, it is important to consider that just because there is a small perturbation in the logs versus the state that has been seen before, the process may not be under attack. Thus, you must carefully consider both the window size you will be checking and the maximum hamming distance that would be allowed before considering a process under attack — both of which should be documented in your design documents, along with your rationale for selecting the values that you choose.

Design Hints

It will be very helpful to you if you provide some way to turn on/off the entire logging/IDS system. You may do this through additional system calls added to the kernel, or by some other means (such as sysfs or procfs). Logging itself can be done to real files (such as a text file stored on your filesystem), through virtual files like what procfs gives you, or through something like a character device. However, the approach you take may limit your options in other ways.

There are two basic design approaches you can take to this assignment: either you can log all system calls made by every process, or you can log only selected processes (obviously providing some way to specify which process or processes to log, through a system call or some other means). Which of these you choose to do will likely affect some of the other aspects of your design, so it would be very helpful to figure out which approach you will take early on. For instance, if you make your logs available by way of a character device, logging all processes and using a user-space process to sort out the logs into each process will not work very well (think carefully about why).

In any case, you should not hard-code PIDs or filenames (if you choose to log to real filesystem files) in your kernel code. Provide some method to get this information into the kernel if it is necessary for your design.

In the kernel itself, look in the `arch/x86/entry/` directory in your kernel source tree for the code that handles entry into the kernel by way of system calls. Remember that you are working with a 64-bit kernel, so you don't want to look at files that have `_32` in the filename, most likely. Specifically, the `entry_64.S` file contains the actual entry point in the kernel where system call software interrupts end up going.

If you end up modifying any assembly code, remember that you must save and restore any registers that you modify to ensure that you do not break the normal operation of the code. That is not to say that you have to modify any assembly to complete this assignment, as it is possible to do so without modifying any assembly.

Finally, there is a system called ptrace in Linux that is able to trace the usage of system calls. **You are not allowed to use ptrace to implement your assignment — you must write the code yourself.**

Extra Credit

On this project, you will have a few opportunities for extra credit. Remember, at an absolute minimum, you must develop the bitstring hamming distance based approach that was described earlier in this document. If you do attempt to gain extra credit on this assignment, you still must develop the hamming distance based approach as well! No extra credit will be considered unless you have made a reasonable attempt at the rest of the project (so, you cannot use extra credit to make up for not writing a design document, for instance).

For 20% extra credit: Develop an IDS based on the paper Intrusion Detection using Sequences of System Calls by Hofmeyr, et al.

You may also be given extra credit for finding and implementing other similar schemes at the discretion of your instructor. If you wish to implement another scheme not described here or in the Hofmeyr paper, please clear it first with your instructor and TA.

Kernel Development

Remember when developing your kernel-space components that code in the kernel is often being accessed by multiple processes at a time. As there may be many processes running that you will be logging system calls from, you must ensure that you lock properly where needed to ensure concurrent access does not corrupt the state of your system.

As this system will involve adding code into the path of potentially every system call made, you should also ensure that your code is as efficient as possible. System calls are made **extremely** often, so even a small unnecessary delay in your code will cause your system to slow down signficantly as it snowballs across all of the processes that are running on your system.

As this code will be part of the kernel itself, correctness and efficiency should be of primary concern to you in the implementation. Particularly inefficient (memory-wise, algorithmic, or other poor coding choices) solutions to the problem at hand may be penalized in grading.

Finally, you are to implement this system on your own -- no group work is allowed on this assignment. If you use any external resources, you must cite them in your `README` file in the `p2ids/` directory.

Submission Instructions

You should follow the same basic set of instructions for submitting Project 2 that you did in previous projects. That is to say, you should do a `git status` to ensure that any files you modified are detected as such, then do a `git add` and a `git commit` to add each modified/newly created file or directory to the local git repository. Then do a `git push origin master` to push the changes up to your GitHub account.

Be sure that you have included all of your modified/newly created files, including the required documentation and that it all is in the correct directory layout, as specified in the introduction to the project.

You should also verify that your changes are reflected in the GitHub repository by viewing your repository in your web browser.

References

Below is a list of references that you may find useful in your quest to complete this project:

- The Linux Kernel API — documentation of the internal API for programming in the Linux kernel (please note that in the User Space Memory Access chapter, only certain functions are covered (which do "less checking"), the versions that do "more checking" are named the same, but without leading underscore characters)
- The Linux Cross-Reference (for version 4.15 of the kernel) — a cross-referenced copy of the Linux kernel source code for relatively easy searching
- The Open Group Base Specifications Issue 7/IEEE Std. 1003.1 - 2008, 2016 Edition/POSIX.1-2008
- The Unreliable Guide to Locking [in the Linux Kernel]

If in doubt, the Kernel API and Linux Cross Reference should be your ultimate guides.

What to do if you want to get a 0 on this project

Any of the following will cause a significant point loss on this project. Items marked with a * will result in a 0.

- Not pushing changes to your GitHub repository by the project due date. *
- Excessive unnecessary changes made to the kernel sources.
- Extraneous files are included.
- Files are missing that needed to be modified.
- Failure to follow the requirements in the "Incremental Development" section of the assignment.
- IDS implementation causes a significant slowdown of the system, especially if the slowdown affects processes not being monitored.

Please do not make us take off points for any of these things!