Name: Trevor Philip
Student Number: NL10252
Course: CSMC 421, Spring 2018
Date: 4/15/2018

Draft: Design Documentation for Project 2

Disclaimer: The information in this document is simply an initial idea of implementation. Some user-space implementation was done in the first commit, but it is subject to change and redesign.

Introduction

The purpose of this project is to develop an intrusion detection system for the Linux kernel. An intrusion detection system, or IDS for short, is a system by which the sequence of system calls, recorded and logged by the kernel, are analyzed and abnormalities in the sequences of these calls are detected. If an abnormality is detected, then a process can be declared as being under attack. For the IDS, two components are required: a logger and a log-analyzer.

The system call logger will require additional code to be added to the Linux kernel (specifically in the system call dispatcher) to log system calls in kernel space, and the analyzer for the logs of the system calls will be done as a separate user space program running in the background.

The choice of programming language for the kernel space program will be C, using the C90 standard, as this is the language and standard used by the Linux kernel as a whole and will need to be compiled with the kernel. The user space program will be written in C#, and will require the Mono Runtime to function.

Statement of Goals

- Create a kernel-space component to be added to the Linux kernel that monitors all processes and creates logs for them. The logging system should be toggled and controlled via system calls
- Create a multi-threaded user-space program that monitors the defined logging directory and parses any logs that are created there, applying internal intrusion detection logic to determine if the process in question is under attack.
- Provide examples of good and bad logs to show what would pass intrusion detection and what would not.

- Provide the appropriate makefiles for all parts of the system to ensure easy building and execution of the IDS system.

Kernel Space

In order for the intrusion detection system to work, system calls will have to be logged. Since system calls are kernel space components, the code for that will have to be written in kernel space.

In general, the goal for the kernel-space code is to keep the logging as simple as possible and with minimal changes needed to the existing kernel sources, so as to minimize the potential for kernel panics. The approach that will be followed for the kernel space component of the IDS will be as follows:

- Each time a system call is made, the system call is logged in the /var/logs directory, which will be created if it does not exist. This directory will be made accessible and readable to user space in order to allow the logs to be parsed, but the user space will not be allowed to delete or modify logs. The expectation here is that when the user space program sees the log, the log file is copied for parsing.
- Each process will have its own log file stored, using the following format: [process_id]_[timestamp]_[user_id]_syscall.log, where process_id is replaced with the process id, timestamp is replaced with the time the log started in Unix time, user_id denotes the ID of the user who owns the process, and the _syscall.log part is used for the user space program to be able to identify which logs to grab..
    - Note: Since the design hints for the project advised us not to hardcode the log entry file names in the kernel, an additional system call will be added to define the format for the log entry as a string which can be accessed as a root user. This string must contain [process_id],[timestamp], and [user_id] for replacement by the kernel when creating the logs.
        - sys_ids_log_format(format_string)
- As for the content of the logs, they will simply contain a comma-separated sequence of system call IDs. For example: “2,0,1,8,3” which corresponds to “sys_open,sys_read,sys_write,sys_lseek,sys_close”
- When the process ends, the log gets deleted as the process in question will no longer need to be monitored (this may require some clarification if we are monitoring closed processes as well). If the log is being read while the user space program is busy parsing it, the user space program will have to handle that situation, not the kernel.
- Some attempt will be made to ensure that the only process that will not be monitored is the IDS itself (the user space program). This may be possible to do

through some special flagging mechanism, but additional research may be needed for this.

- In order for the system calls to be logged, the system call dispatcher itself needs to have some additional lines of code added for logging. To avoid modifying existing Linux kernel sources too extraneously, the code for logging will be added in separate files and a simple call to the entry point of the logging system will be added to the system call dispatcher. This is a proposed way for it to be done:
    - In the assembly code for entry_64.S, there are labels called “ENTRY”. This appears to be where the entry of a system call happens. In the most basic ENTRY label, I could possibly call my system call dispatcher monitor and pass it the system call ID (which according to the comments, can be found in the RAX CPU register for the 64-bit CPU). I hope to, after line 166, call the logger function which is written in C. I can use the “asm { }” construct in the logger function to access whatever was put in the RAX register, or perhaps I could copy whatever is in RAX to r14 (and by using push and subsequent pop, I can preserve whatever was in that register before).
    - Note I may need feedback about this. I hope to do VERY LITTLE modification to assembly code. My goal here is to just call an external function written in C and let it try to read what it wants from the desired CPU register. If there is a way to do this without touching assembly code, I would change to that method.
    - The logger function itself will call code from files that are placed in the kernel/ directory.
- Finally, the logging system can be toggled from a privileged user-space program that is running as root. To facilitate this, two additional system calls will be added to the kernel:
    - sys_ids_log_on (will do nothing if already on, and will output an error if the log entry format is not defined in sys_ids_log_format)
    - sys_ids_log_off (will do nothing if already off)
- Note that any system calls that were added for the IDS logger will be ignored in intrusion detection.

User Space

The user space program will be a program written in C# under the Mono Runtime. Note that although C# is traditionally used in Windows environments (and compiled assemblies may even be misunderstood to be Windows executables due to the .exe and .dll files), it was always designed to be multi-platform due to the Common Language Infrastructure, and Mono makes it possible to execute programs written in C# natively on Linux and other operating systems.

The actual intrusion detection will be done here, and this program will be responsible for reading and parsing the logs created by the kernel-space system call logger. The approach followed by the user space program will be as follows:

- To facilitate easier grading, an install script (based on the one provided by the Mono website) will be provided as a Bash script named as install_mono.sh. This bash script downloads and installs Mono sources, which can take 20-30 minutes to complete. No reboot will be required.
- Once the install script is run, the provided Makefile should successfully compile the IDS without errors, and "make run" will be able to run the IDS after compilation.
- Once the IDS is launched, code that monitors the file system in /var/logs will execute. Each time a file is created or changed, an event gets added to the event queue, and once handled, the IDS will determine if the file follows the naming pattern expected for the IDS logger from the kernel.
- If the file does follow that naming pattern, its information (such as the process ID and user ID) is recorded in a local JSON database and the file itself copied to a local directory and provided a unique name. It then gets added to the log processing queue.
- Since this program is multi-threaded, adding to the queue will likely be done by more than one thread. The queue itself will be defined as a static variable and the Queue<> data structure will be used.
- A different thread will be responsible for popping from the queue. Each item that is popped from the queue represents a class with information about the log file that was created by the IDS logger component in the kernel. After the item is popped, a method is called that processes the log. This is where the important IDS logic lies, and whatever is contained in the log file will determine if the process is under attack.
- If the process is under attack, the process ID will be added to a separate database of suspected intrusions (also a JSON file), and an alert is given to the user.

Conclusion

Note that these are all high-level descriptions of how the IDS system will work. If possible and if time allows, the "extra credit" method will be used in user space for intrusion detection. However, for now, implementation will involve a simple detection of normal and abnormal system call sequences.

Additional References

- System call dispatcher information: https://linux-kernel-labs.github.io/master/lectures/syscalls.html
- Convenient Build Script for building and installing Mono: http://www.mono-project.com/docs/compiling-mono/linux/
- The Common Language Infastructure: https://en.wikipedia.org/wiki/Common_Language_Infrastructure