CSE 509 Assignment 2: System-call and Library Interception

1 Background

System-call interception provides a versatile mechanism for a number of security-related applications. The most obvious one is policy enforcement: one can define policies that govern which system calls are permissible for a process and which ones aren't. Alternatively, one can use system-call interception for transparently extending application functionality. For instance, you can extend an arbitrary application so that it can access remote files using the HTTP protocol. This can be done by intercepting open system calls, identifying file names that correspond to remote URLs, making an HTTP request to the remote site to fetch the content. Subsequent reads on the same file descriptor should be intercepted, and this remote data returned to the process.

There are several approach for doing system-call interception. One of the mechanisms available for this purpose on Linux is the `ptrace` mechanism. You can find out more by doing `man ptrace` on a Linux system. There are also lecture notes on system call interception that describe this system-call interception mechanism. To learn more about the ptrace interface, look at the following references:

- `http://www.linuxjournal.com/article/6100` is a link to a good article in the Linux journal that explains ptrace.
- `man ptrace` on Linux systems
- `strace` is a handy command-line tool that uses ptrace to intercept system calls, and prints them. You can understand how ptrace works by running `strace` on various programs. You are also welcome to read its source code, but don't copy it.

While ptrace provides a secure mechanism, it introduces significant performance overheads. An alternative approach that has low overheads is the library interception approach. This approach is based on the fact that system calls are usually low-level mechanisms that are not directly used by applications. Instead, applications call system-call wrappers in standard libraries (such as `glibc` on Linux). If all calls to these wrappers can be intercepted, then we will have a much more efficient approach. However, there are serious drawbacks as well — in particular, a malicious program can bypass these wrappers and make a direct system call (using a software trap instruction). So, this approach can be used only with benign applications. To use this approach, you can develop a library that provides a routine for each system call that you want to intercept, and using `LD_LIBRARY_PATH` mechanisms to ensure that your library comes before the standard system libraries. Here are a couple of references on the topic:

- `https://blog.netspi.com/function-hooking-part-i-hooking-shared-library-function-calls-in-linux/` illustrates library hooking using `LD_LIBRARY_PATH` technique, and illustrates it with examples.
- `http://samanbarghi.com/blog/2014/09/05/how-to-wrap-a-system-call-libc-function-in-linux/` describes the `LD_LIBRARY_PATH` technique, as well as a second, finer-granularity approach for wrapping individual library calls.
- `ltrace` is a command-line tool very similar to `strace` but can report calls to dynamic libraries instead of system calls.

Note that if you want to use library-based interception with untrusted code, then you need to use additional mechanisms to guard against bypass.

2 Assignment Description

The warmup part is intended to get you started right away. Although it has 40% of the credit for this assignment, I expect that it will take perhaps 20% of the time you spend on this assignment.

2.1 Warmup (40 points)

1. (9 points) Get familiar with `strace` by reading its man page and by using it. What are the system calls made by `ls` when you run it with no options? What about `ls -l`? Try the command on directories large and small to have confidence that you have exercised `ls` well. Can you explain the reasons for differences in system calls made?
2. (11 points) Get familiar with `ltrace`. Use it to study `ls` in the same was as the problem above. Use appropriate options to `ltrace` (or process the output using tools such as `grep` or `awk`) so as to leave out calls to utility functions such as malloc, free, getenv, getopt, strcmp, and so on, while only retaining calls that look like system calls. (But **do not** use the `-S` option to `ltrace`.) Comment on the output you get this way, as compared to `strace` output.
3. (20 points) Use `strace` to count the number of files accessed by applications when they are started up. Answer this question for `ls`, `nano`, Open office and Firefox or Chromium. Note that complex applications may create multiple child processes. Use appropriate options to `strace` so that it captures the system calls made by all of these processes. Note that applications such as Firefox open thousands of files on start up; if you don't see this, then you are making some mistake.
4. (Bonus: 5 points) Identify how many files are loaded by each of the programs mentioned in the previous part, as opposed to being opened for reading and/or writing.

Your submission for this part will be a PDF document that reports on your findings. Include all the relevant details, but don't go overboard. Ideally, each report will be one to two pages using normal document setting (such as the one used in this assignment description).

2.2 Transparent Application Functionality Extension (60 points)

An important use of system call (or library call) interception is to extend the functionality of applications *without requiring modifications to applications.* This extension may or may not be security-related. An example of a security extension is one that limits access to files, e.g., prevents files in your home directory from being overwritten. You will explore such extensions in this assignment.

It is silly to extend ***toy*** *applications this way.* So, don't test your implementation on toy applications. For grading, we will test against arbitrary applications, including complex applications such as gedit. But you are permitted to make one simplifying assumption, namely, that you can ignore any child processes created by an application. (On UNIX, creating a child process requires the use of `fork` or `clone` system call. Note that `execve` and related calls do not create a new (child) process, but simply change the program being executed.)

Choose **one** of the following extensions for your assignment. Make sure that you read and understand each part in detail so that you can make an informed decision that maximizes the points you get on this assignment.

- Write a ptrace-based extension that enables applications to access remote URLs as if they were local files. For instance, using your `urlextend` tool, I should be able to run a command such as

  ```
  urlextend wc http://www.cs.stonybrook.edu
  ```

  to count the number of words on a web page. Before you start writing code, use `strace` to identify which system calls you need to handle. For instance, many applications use `stat` before attempting to `open` a file, so you need to intercept `stat` in addition to `open`. When an application does open an URL, your extension should use a program such as `wget` to download the web page into a temporary local file. From this point, you should arrange it so that `read` operations on the "file" go to this temporary file.

  For bonus credit (10 points) you should get the extension to work with editors. Note that you cannot typically edit a web page because most web servers won't allow you to upload a web page. So, we will expect that the user will have to save edits in a new local file. Most editors are happy to open a read-only file, and handle it gracefully. So, your extension should make the application believe that it is opening a read-only file. (This again requires intercepting `stat` and returning appropriate information to the application.)

- Write a logger extension that will log all of the files and network connections opened by a program, together with the number of bytes read and written on each. Your extension should not affect the application behavior in any way. When the application terminates, your extension should print the information about files accessed. This information should include the file name, access mode, and number of bytes read and written. For network connections, the IP address and port information should be reported, together with the number of bytes written or read. For full credit, you should handle `mmap`, `dup` and `dup2` as well. (If a file is mmapped, you can assume that the entire contents are read or written, depending on the access mode with which the file was opened.) If you application does not handle these three system calls, then you will lose 10 out of 60 points for this part of the assignment.

- Write an extension that automatically creates backup copies of files before they are overwritten. Such a tool may be used to defend against ransomware. Your extension should detect any attempt to overwrite an existing file, e.g., opening a file for write access, opening a file with truncate flags, truncating a file, renaming a file, etc. In all of those cases, your extension should copy the existing file into a backup directory, say, `.backup` within the user's home directory. For simplicity, you can assume that an unlimited number of backups can be created. (You should not overwrite backups since an attacker can cause files to be deleted by overwriting a file twice.) Assume that an attacker cannot directly access the backup directory. (In reality, you will likely backup to the cloud, so this assumption is reasonable.)

  Your tool should not add any unnecessary overheads to operations that only read file contents.

Your submission for this part will be a tgz file that we should be able to untar on Linux. We should be able to run make inside the extracted directory to build your program, and then execute that program.

If you run out of time and are not able to complete the assignment, make sure that you have a submission that still compiles and works partially. Typically, such a partial implementation will not be able to handle all the relevant system calls, but is still useful to extend some of the simpler applications. But if your program does not compile or run, then you leave us no option except to give you zero credit. It is your responsibility to ensure that your program works on the VM that was used for your first assignment.