Homework #2

Advanced Programming in the UNIX Environment

Due: Apr 18, 2022

Monitor File Activities of Dynamically Linked Programs

In this homework, we aim to practice library injection and API hijacking. Please implement a simple logger program that can show *file-access-related* activities of an arbitrary binary running on a Linux operating system. You have to implement your logger in two parts. One is a *logger* program that prepares the runtime environment to inject, load, and execute a monitored binary program. The other is a *shared object* that can be injected into a program by the *logger* using **LD_PRELOAD**. You have to dump the library calls as well as the passed parameters and the returned values. Please check the "Requirements" section for more details.

The output of your homework is required to follow the spec *strictly*. The TAs will use the **diff** tool to compare the output of your program against the correct answer. When comparing the outputs, continuous spaces and tabs in the output are merged into a single space character.

Homework Submission

We will compile your homework by typing 'make' in your homework directory. You have to ensure your Makefile produces the executable ***logger*** and the shared object ***logger.so***. Please make sure your Makefile works and the output executable name is correct before submitting your homework. The ***logger*** may be implemented as a shell script. In that case, you can simply pack the shell script in your submitted file.

Please pack your C/C++ code and Makefile into a **zip** archive. The directory structure should follow the below illustration. The *id* is your student id. Please note that you don't need to enclose your id with the braces.

```
{id}_hw2.zip
└── {id}_hw2/
        ├── Makefile
        ├── hw2.c (or hw2.cpp)
        └── (any other c/c++ files if needed)
```

You have to submit your homework via the E3 system. Scores will be graded based on the completeness of your implementation.

Requirements

Program Arguments

Your program should work with the following arguments:

```
usage: ./logger [-o file] [-p sopath] [--] cmd [cmd args ...]
    -p: set the path to logger.so, default = ./logger.so
    -o: print output to file, print to "stderr" if no file specified
    --: separate the arguments for logger and for the command
```

If an invalid argument is passed to the *logger*, the above message should be displayed.

Monitored file access activities

The list of monitored library calls is shown below. It covers several functions we have introduced in the class.

chmod	**chown**	**close**	**creat**	**fclose**	**fopen**	**fread**	**fwrite**
open	**read**	**remove**	**rename**	**tmpfile**	**write**		

Output

You have to dump the library calls as well as the corresponding parameters and the return value. We have several special rules for printing out function arguments and return values:

- If a passed argument is a filename string, print the *real absolute path* of the file by using `realpath(3)`. If `realpath(3)` cannot resolve the filename string, simply print out the string untouched.
- If a passed argument is a descriptor or a FILE * pointer, print the *absolute path* of the corresponding file. The filename for a corresponding descriptor can be found in `/proc/{pid}/fd` directory.
- If a passed argument is a mode or a flag, print out the value in octal.
- If a passed argument is an integer, simply print out the value in decimal.
- If a passed argument is a regular character buffer, print it out up to 32 bytes. Check each output character using `isprint(3)` function and output a dot '.' if a character is not printable.
- If a return value is an integer, simply print out the value in decimal.
- If a return value is a pointer, print out it using `%p` format conversion specifier.
- Output strings should be quoted with double quotes.

A sample output of the homework is given below. More examples can be found in the "Sample Output" section.

```
$ ./logger ./sample
[logger] creat("/home/ta/hw2/tmp/aaaa", 600) = 3
[logger] chmod("/home/ta/hw2/tmp/aaaa", 666) = 0
[logger] chown("/home/ta/hw2/tmp/aaaa", 65534, 65534) = -1
[logger] rename("/home/ta/hw2/tmp/aaaa", "/home/ta/hw2/tmp/bbbb") = 0
[logger] open("/home/ta/hw2/tmp/bbbb", 1101, 666) = 4
[logger] write("/home/ta/hw2/tmp/bbbb", "cccc.", 5) = 5
[logger] close("/home/ta/hw2/tmp/bbbb") = 0
[logger] open("/home/ta/hw2/tmp/bbbb", 000, 000) = 4
[logger] read("/home/ta/hw2/tmp/bbbb", "cccc.", 100) = 5
[logger] close("/home/ta/hw2/tmp/bbbb") = 0
[logger] tmpfile() = 0x55c418842670
[logger] fwrite("cccc.", 1, 5, "/tmp/#14027911 (deleted)") = 5
[logger] fclose("/tmp/#14027911 (deleted)") = 0
[logger] fopen("/home/ta/hw2/tmp/bbbb", "r") = 0x55c418842670
[logger] fread("cccc.", 1, 100, "/home/ta/hw2/tmp/bbbb") = 5
[logger] fclose("/home/ta/hw2/tmp/bbbb") = 0
[logger] remove("/home/ta/hw2/tmp/bbbb") = 0
sample done.
[logger] write("/dev/pts/19", "sample done...", 14) = 14
```

Sample Output

Some basic usage.

```
$ ./logger
no command given.

$ ./logger -h
./logger: invalid option -- 'h'
usage: ./logger [-o file] [-p sopath] [--] cmd [cmd args ...]
        -p: set the path to logger.so, default = ./logger.so
        -o: print output to file, print to "stderr" if no file specified
        --: separate the arguments for logger and for the command

$ ./logger ls
[logger] fopen("/proc/filesystems", "re") = 0x55aa12cd6ad0
[logger] fclose("/proc/filesystems") = 0
hw2.c  logger  logger.so  Makefile  sample  sample.c  sample.txt
[logger] fclose("/dev/pts/7") = 0
[logger] fclose("/dev/pts/7") = 0

$ ./logger ls 2>/dev/null
```

```
hw2.c  logger  logger.so  Makefile  sample  sample.c  sample.txt

$ ./logger -o ls_al.txt -- ls -al
... [output of `ls -al`] ...

$ cat ls_al.txt
[logger] fopen("/proc/filesystems", "re") = 0x558c872fcad0
[logger] fclose("/proc/filesystems") = 0
[logger] fopen("/etc/passwd", "rme") = 0x558c872fcd00
[logger] fclose("/etc/passwd") = 0
[logger] fopen("/etc/group", "rme") = 0x558c872fcf30
[logger] fclose("/etc/group") = 0
[logger] fclose("/dev/pts/7") = 0
[logger] fclose("/dev/pts/7") = 0
```

Sample output for `exec()` or `system()`.

```
$ ./logger -o bash.txt -- bash
ta@lab:~/hw2$ ls -al
... [output of `ls -al`] ...
ta@lab:~/hw2$ exit
exit

$ cat bash.txt
[logger] open("/dev/tty", 4002, 0) = 3
[logger] close("/dev/tty") = 0
[logger] fopen("/etc/passwd", "rme") = 0x56161c8b69f0
[logger] fclose("/etc/passwd") = 0
[logger] close("/dev/pts/19") = 0
[logger] open("/etc/bash.bashrc", 0, 0) = 3
[logger] read("/etc/bash.bashrc", "# System-wide .bashrc file for i", 2319) = 2319
[logger] close("/etc/bash.bashrc") = 0
...
[logger] read("/dev/pts/7", "l", 1) = 1
[logger] fwrite("l", 1, 1, "/dev/pts/7") = 1
[logger] read("/dev/pts/7", "s", 1) = 1
[logger] fwrite("s", 1, 1, "/dev/pts/7") = 1
...
[logger] fopen("/proc/filesystems", "re") = 0x55b89284af30
[logger] fclose("/proc/filesystems") = 0
[logger] fclose("/dev/pts/7") = 0
...
```

Hints

When implementing your homework, you may inspect symbols used by an executable. We have mentioned that you cannot see any symbols if they were symbol stripped (using **strip** command). However, you may consider working with the **readelf** command. For example, we can check the symbols that are unknown to the binary:

```
$ nm /usr/bin/wget
nm: /usr/bin/wget: no symbols
$ readelf --syms /usr/bin/wget | grep open
    72: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND freopen64@GLIBC_2.2.5 (2)
    73: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND iconv_open@GLIBC_2.2.5 (2)
   103: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND gzdopen
   107: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND fdopen@GLIBC_2.2.5 (2)
   119: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND open64@GLIBC_2.2.5 (2)
   201: 0000000000000000     0 FUNC    GLOBAL DEFAULT  UND fopen64@GLIBC_2.2.5 (2)
```

Alternatively, you may consider using **nm -D** to read symbols. Basically, we have two different symbol tables. One is the regular symbol table, and the other is the dynamic symbol table. The one removed by **strip** is the regular symbol table. So you will need to work with **nm -D** or **readelf --syms** to read the dynamic symbol table.

Grading

- The output of your homework is required to follow the spec strictly.
- The TAs will use the **diff** tool to compare the output of your program against the correct answer. When comparing the outputs, continuous spaces and tabs in the output are merged into a single space character.
- When running the same test case multiple times, you may not have to worry about slight differences in node number, address, and tmp filename.
- If your program crashes in a test case, you will only get part or no scores.
- If your files are not packed correctly, you will receive a grade reduction.
- We have provided a sample implementation with two test cases. Please access our online sandbox (https://up.zoolab.org/service/) (tested only with Chrome, Firefox, and Edge; Safari is not supported). You can find everything in the directory ~/ta/hw2. Note that you can only access the service within campus networks or via a valid VPN network.
    - You can use `-d` in testcase2.sh to look at the diff result, but we still suggest checking it carefully by yourself to avoid any mistakes ignored by the additional diff filter.

Remarks

- Please implement your homework in C or C++.
- Using any non-standard libraries and any external binaries (e.g., via system()) are not allowed.
- No copycats. Please do not use codes from others (even open-source projects).
- We will test your program in **Ubuntu 20.04 LTS** Linux with the default gcc version (9.3.0).
