UP24 HW1

Due Date: 2024-04-22

Monitor File Activities of Dynamically Linked Programs

In this homework, we are going 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.

Part 1: Logger program
The logger program's job is to prepare the environment to inject, load, and execute a monitored binary.

Part 2: Shared object
This shared object will be injected into the monitored binary using `LD_PRELOAD`. It will

intercept file access related library calls and log them along with their parameters and return values.

Please check the requirements section for more details.

Requirements

Program Usage

The usage of the `logger` program is as follows

- `-p` : set the path to your shared object, default is `./logger.so`
- `-o` : print the output to `file` if specified, else print it to `stderr`

```
Usage: ./logger config.txt [-o file] [-p sopath] command [arg1 arg2 ...]
```

Check the examples section for more example usages.

API Function List

All functions listed below should be logged to `stderr` or to the file with the filename (option `file`) passed by the program's argument (check the examples for the detailed format).

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

1. `fopen`
   Allow a user to set the file access blacklist so that files listed in the blacklist cannot be opened. Note that filtering is based on the file name here. If the filename is in the blacklist, return NULL and set `errno` to EACCES. Note that for handling symbolic linked files, your implementation has to follow the links before performing the checks.

2. `fread`
   Allow a user to filter the read content based on a keyword blacklist. The filter should be active for all read operations. If the filter detects a matched keyword in a read content, return 0 and set `errno` to EACCES. Your implementation must log all content into a file. The log file should be named in the format `{pid}-{filename}-read.log` and be created at read function on this filename be called first time. If a filename is used more than one time in a process, keep logging the content into the same log file.

3. `fwrite`

Allow a user to set the file access blacklist so that files listed in the blacklist cannot be opened. If the filename is in the blacklist, return 0 and set `errno` to EACCES. Your implementation must log all content into a file. The log file should be named in the format `{pid}-{filename}-write.log` and be created at write function on this filename be called first time. If a filename is used more than one time in a process, keep logging the content into the same log file.

4. `connect`
   Allow a user to block connection setup to specific IP addresses. If the IP is blocked, return -1 and set `errno` to ECONNREFUSED.

5. `getaddrinfo`
   Allow a user to block specific host name resolution requests. If a host is blocked, return
   EAI_NONAME.

6. `system`
   Commands invoked by system function should also be hijacked and monitored by your program.

Configuration File Format

The configuration file is a text file containing blocked content for each API function. For each API, the general form is as follows.

```
BEGIN <API>-blacklist
rule1
rule2
...
END <API>-blacklist
```

A sample configuration `config.txt` is given below.

```
BEGIN open-blacklist
/bin/*
END open-blacklist
BEGIN read-blacklist
PRIVATE_KEY
END read-blacklist
BEGIN write-blacklist
/etc/passwd
END write-blacklist
BEGIN connect-blacklist
172.217.160.100
END connect-blacklist
BEGIN getaddrinfo-blacklist
people.cs.nycu.edu.tw/~chuang
www.cs.nycu.edu.tw
END getaddrinfo-blacklist
```

Notes:

- `*` might appear in paths.
- There are multiple lines between `BEGIN` and `END` tags except the `read-blacklist`

Examples

Here we provide some running examples. Please notice that the results presented here could be different from your runtime environment. You may simply ensure that the behavior is expected and the output format is correct.

All the running examples use the same configuration: config.txt (https://up.zoolab.org/unixprog/hw01/config.txt)
Here are all the sample executable files for the example below. Please download the zip file from this link and extract them: link (version 3) (https://up.zoolab.org/unixprog/hw01/examples.zip)

ex1-1

- input: `./logger config.txt ./ex1-1`
- output:

```
[logger] fopen("/bin/grep", "r") = 0x0
fp = (nil)
```

ex1-2

- input: `./logger config.txt ./ex1-2`

- output:

```
[logger] fopen("/etc/passwd", "r") = 0x64cf93f052a0
fp = 0x64cf93f052a0
```

ex2

- input: `./logger config.txt ./ex2`
- output:

```
[logger] fopen("./file.txt" , "w") = 0x64cf93f052a0
fp = 0x64cf93f052a0
[logger] fwrite("Hello World!\n", 1, 10, 0x64cf93f052a0) = 10
write size = 10
```

ex3-1

- input: `./logger config.txt ./ex3-1`
- output:

```
addr of buffer = 0x7ffedd45dd60
[logger] fopen("file.txt" , "w+") = 0x64cf93f052a0
fp = 0x64cf93f052a0
[logger] fwrite("PRIVATE_KEY\nABC123\n", 1, 12, 0x64cf93f052a0) = 12
write size = 12
[logger] fread(0x7ffedd45dd60, 1, 11, 0x64cf93f052a0) = 0
read size = 0
```

ex3-2

- input: `./logger config.txt ./ex3-2`
- output:

```
addr of buffer = 0x7ffedd45dd60
[logger] fopen("file.txt" , "w+") = 0x64cf93f052a0
fp = 0x64cf93f052a0
[logger] fwrite("Hello World!\n", 1, 12, 0x64cf93f052a0) = 12
write size = 12
[logger] fread(0x7ffedd45dd60, 1, 11, 0x64cf93f052a0) = 11
read size = 11
Hello World
```

ex4

- input: `./logger config.txt ./ex4 www.cs.nycu.edu.tw`
- output:
    - notes: If the logger outputs -2, it is also an acceptable answer.

```
[logger] getaddrinfo("www.cs.nycu.edu.tw" , (nil), 0x7ffd88b905f0,0x7ffd88b90
getaddrinfo: Name or service not known
```

- input: `./logger config.txt ./ex4 www.google.com`
- output:

```
[logger] getaddrinfo("www.google.com" , (nil), 0x7ffd88b905f0,0x7ffd88b905b8)
IP addresses for www.google.com:
IPv4: 172.217.160.100
IPv6: 2404:6800:4012:2::2004
```

ex5

- input: `./logger config.txt ./ex5 172.217.160.100`
- output:

```
[logger] connect(3, "172.217.160.100", 16) = -1
Error with client connecting to server
```

- input: `./logger config.txt ./ex5 20.27.177.113`
- output:

```
[logger] connect(3, "20.27.177.113", 16) = 0
Connect success!
```

ex6

- input: `./logger config.txt ./ex6`
- output:

```
Hello World!
[logger] system("echo Hello World!") = 0
```

Hidden case demo

Please check the details from here (https://md.zoolab.org/s/8nu0-zgnp).

Homework Submission

Please provide a `Makefile` to compile your source code. Make sure your code can be compiled by `make`. You need to upload a zip file, and the format is shown below.

- Due Date: 2024-04-22 15:30
- Filename: `{studentID}_hw1.zip`
- Format:

```
+---{studentID}_hw1
|    Makefile
|    other files...
```

Grading

1. [60%] Your program has the correct output for all test cases listed in the examples section.
   - 5 pts for `ex1-1` `ex1-2` `ex3-1` `ex3-2`
   - 10 pts for the others
2. [40%] Hidden case. Hidden test cases will be revealed on the day of the demo.
   - 10 pts for each testcase