CS 419: Computer Security

Recitation: week of 2020-10-05

Project 2 Discussion

TA: Shuo Zhang
Paul Krzyzanowski

October 8, 2020

Assignment 6 (Project 2)

- **This assignment has two parts**
- **This is an individual assignment**
- **Goal: use function interposition**
  - Replace *readdir* and *time* functions in existing programs

Environment

- **You must do this assignment on a Linux platform**
- **It uses shared library preloading, which is not available on BSD, macOS, or Windows systems**
- **Your personal Linux system will probably be fine**
  - But you are responsible to make sure it works on the Rutgers iLab machines

Environment

Download `a-6.zip` (see assignment) and unzip it

You will see

- **`Makefile`** – you can use this to build the zip file for submitting your program
- **`random`** – this is a demo of using LD_PRELOAD to replace a function
- **`hidefile`** – this is for Part 1
- **`unexpire`** – this is for Part 2

Background

- **LD_PRELOAD is an environment variable that can define a shared library that will be loaded & searched before any other library**
- **If a program needs to call a library function, this library will be checked first**
- **It's set as any shell environment variable:**

```
export LD_PRELOAD=$PWD/mylib.so
```

Specifies to:

- **Load the shared library $PWD/mylib.so**
  - $PWD expands to the path of the current directory
- **Check this for any needed functions first**

Example

- We looked at this in class
- Here's a C program to print 10 random numbers

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
      int i;
      srand(time(NULL));  // seed the generator with the current time
      for (i=0; i < 10; i++)
            printf("%d\n", rand()%100);
      return 0;
}
```

random.c

Example

- If we compile and run it, we get:

```
$ gcc -o random random.c
$ ./random
90
36
89
26
3
31
87
71
79
10
```

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
      int i;
      srand(time(NULL));
      for (i=0; i < 10; i++)
            printf("%d\n", rand()%100);
      return 0;
}
```

random.c

Example

Let's create a file `myrand.c` that redefines the *rand* function

```
int rand() {
      return 42;
}
```

myrand.c

Now compile it to a shared library & preload it

```
$ gcc -shared -fPIC myrand.c -o myrand.so -ldl
$ export LD_PRELOAD=$PWD/myrand.so
```

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
      int i;
      srand(time(NULL));
      for (i=0; i < 10; i++)
            printf("%d\n", rand()%100);
      return 0;
}
```

random.c

Example

If we run the program again, it uses our function instead of the standard one

We did not have to recompile the program!

```
$ ./random
42
42
42
42
42
42
42
42
42
42
```

```
int rand() {
      return 42;
}
```

myrand.c

```
#include <time.h>
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char **argv)
{
      int i;
      srand(time(NULL));
      for (i=0; i < 10; i++)
            printf("%d\n", rand()%100);
      return 0;
}
```

random.c

Part 1: Goal

- **Attackers sometimes try to hide their files on a system**
  - The best way is by modifying the kernel but we usually do not have the ability to modify the kernel
- **Instead, we will modify the `readdir` library function**
  - This is used by most tools that need to read directory contents on Linux
  - Example: *ls, find, zsh, sh*
- **We will create a new version of `readdir` that checks for a file name stored in the environment variable HIDDEN**
  - If the file is in the directory, it will not be made visible to the program that's looking at files in the directory
  - If you know it exists, you can still run it or open it by specifying its path

Example

Preload our library, which replaces the *readdir* function

```
$ export LD_PRELOAD=$PWD/hidefile.so
```

Use the ls command to list all the files in a directory

```
$ ls -l
total 408
-rw------- 1 pxk allusers    115 Oct  6 12:26 present.pptx
-rw------- 1 pxk allusers    141 Oct  6 12:35 secretfile-zzz
-rw------- 1 pxk allusers  94698 Oct  6 12:33 status-report-1.txt
-rw------- 1 pxk allusers 166518 Oct  6 12:33 status-report-2.txt
-rw------- 1 pxk allusers  77166 Oct  6 12:33 status-report-3.txt
-rw------- 1 pxk allusers  48858 Oct  6 12:33 status-report-4.txt
-rw------- 1 pxk allusers     14 Oct  6 12:34 testfile.c
```

Example

Set the file name that we want to hide

```
$ HIDDEN=secretfile-zzz
```

Run the `ls` command again – `secretfile-zzz` is missing!

```
$ ls -l
total 404
-rw------- 1 pxk allusers    115 Oct  6 12:26 present.pptx
-rw------- 1 pxk allusers  94698 Oct  6 12:33 status-report-1.txt
-rw------- 1 pxk allusers 166518 Oct  6 12:33 status-report-2.txt
-rw------- 1 pxk allusers  77166 Oct  6 12:33 status-report-3.txt
-rw------- 1 pxk allusers  48858 Oct  6 12:33 status-report-4.txt
-rw------- 1 pxk allusers     14 Oct  6 12:34 testfile.c
```

Example

We can run another command, like *find*

`secretfile-zzz` is still missing!

```
$ find .
.
./status-report-1.txt
./status-report-4.txt
./status-report-2.txt
./status-report-3.txt
./present.pptx
./testfile.c
```

Example

If we change the file name that we want to hide

```
$ HIDDEN=status-report-1.txt
```

And run the `ls` command – `status-report-1.txt` is missing!

```
$ ls -l
total 308
-rw------- 1 pxk allusers    115 Oct  6 12:26 present.pptx
-rw------- 1 pxk allusers    141 Oct  6 12:35 secretfile-zzz
-rw------- 1 pxk allusers 166518 Oct  6 12:33 status-report-2.txt
-rw------- 1 pxk allusers  77166 Oct  6 12:33 status-report-3.txt
-rw------- 1 pxk allusers  48858 Oct  6 12:33 status-report-4.txt
-rw------- 1 pxk allusers     14 Oct  6 12:34 testfile.c
```

Example

If we remove HIDDEN:

```
$ unset HIDDEN
```

Then we can see all the files:

```
$ ls -l
total 408
-rw------- 1 pxk allusers    115 Oct  6 12:26 present.pptx
-rw------- 1 pxk allusers    141 Oct  6 12:35 secretfile-zzz
-rw------- 1 pxk allusers  94698 Oct  6 12:33 status-report-1.txt
-rw------- 1 pxk allusers 166518 Oct  6 12:33 status-report-2.txt
-rw------- 1 pxk allusers  77166 Oct  6 12:33 status-report-3.txt
-rw------- 1 pxk allusers  48858 Oct  6 12:33 status-report-4.txt
-rw------- 1 pxk allusers     14 Oct  6 12:34 testfile.c
```

How to do the assignment

- **Write a version of `readdir` in `hidefile.c`**
  - Same interface as the standard ***`readdir`*** – look at the manual page
  - Each call to returns ***`readdir`*** one file
  - Call the REAL ***`readdir`*** function
  - If the file is the hidden file then do not return
    - Instead, call the REAL ***`readdir`*** function again to get the next file
- **Run make to compile it (see assignment instructions)**
- **Set `LD_PRELOAD=$PWD/hidefile.so` and run a command like `ls`**
  - See instructions
  - You can run **`make test`**
    This will create some test files and set `HIDDEN`

Things to know

- **You still want to call the REAL *`readdir`* function inside yours**
  - To do this, use the ***`ldsym`*** function to load and access the real version of the function from your library
  - Read the references in the assignment for instructions on how to use ***`ldsym`***
- **You need to read the value of the HIDDEN environment variable**
  - You can get this with a call to **`getenv`**

This is a small project

- **The implementation of hidefile.c will likely be <10 statements**
- **As always, develop and test incrementally**
  - Make sure you understand and can use & run the *random* example
  - Put ***`printf`*** statements so you know that your ***`readdir`*** is being called
  - Version 0: don't test files – just print a message and call the real readdir
  - Version 1: compare against a hard-coded name, such as "secret"
  - Version 2: get the environment variable and compare against that
  - Version 3: test – make sure it works and works if HIDDEN is not set
  - Version 4: remove your ***`printf`*** statements

Part 2

- **You are given a Linux program called *unexpire***
  - Pretend this is an evaluation version of a program that has an expiration time coded into it
  - The program exits (expires) if the current date is after January 1, 2020
  - It also refuses to run with any date earlier than October 1, 2020.
- **GOAL:**
  You wish to continue using this program past this hard-coded expiration time and you want to defeat its check for the time

Part 2: unexpire

- **The program calls the C library function *time()* to get the current time**
- **You will create a file called `newtime.c` that:**
  - Implements a different version of the time() function that returns a date in the range Oct 1 2020 … Jan 1 2020 so the expiration check will pass
  - However, you want the program to report the correct time ***after*** the check takes place
    - Your *time()* function will pass future requests straight through to the standard library *time* function
- **`newtime.c` will be compiled into a shared library that you will preload via**

```
export LD_PRELOAD=$PWD/newtime.so
```

Example runs

If we run *unexpire*, it tells us that access has expired

```
$ ./unexpire
It is now Oct 06 2020 19:47:23
You cannot run this program before Wed Jan  1 00:00:00 2020
This software expires at Thu Oct  1 01:00:00 2020

ACCESS DENIED: It is now Oct 06 2020 19:47:23. Access expired at Thu Oct  1 01:00:00 2020
```

But if we preload our *time* library – `newtime.so` – and run *unexpire*:

```
$ export LD_PRELOAD=$PWD/newtime.so
$ ./unexpire
It is now Sep 01 2020 01:00:00
You cannot run this program before Wed Jan  1 00:00:00 2020
This software expires at Thu Oct  1 01:00:00 2020

Sep 01 2020 01:00:00: access granted!
The current time is: Oct 06 2020 19:50:19
PASSED! You reset the time successfully!
```

What you need to do

- **This is similar to Part 1**
  - Your library will load and call the real function … in some cases
- **You need to define a suitable time**
  - Pick a time in the range Jan 1 2020 … Oct 1 2020
  - Figure out how to encode it so *time* can return it
  - You can compute this outside of the program
    - Or you can use a combination of ***`strptime`*** and ***`mktime`*** to set the time
    - ***`strptime`***: converts a human-friendly time into a struct tm
    - ***`mktime`***: converts a **`struct tm`** into seconds count that *time* can return
    - Do a bit of research – read the man pages – it's not hard!
- **You need to keep state**
  - You want to return your custom time just the first time – then pass through to *time*
  - You can keep state in a static or global variable

What to submit

- **You must do this assignment on an iLab system**
- **Submit a zip file that contains**
  - hidefile/hidefile.c *your definition of readdir for Part 1*
  - unexpire/newtime.c *your definition of time for Part 2*

To prepare the zip file, you can go to the top-level directory of the download package and run

```
make zip
```

The End