



CS 419: Computer Security

Recitation

Project 3: Function interposition

TAs: Hailun Ding, Yuange Li

© 2024 Paul Krzyzanowski. No part of this content may be reproduced or reposted in whole or in part in any manner without the permission of the copyright owner.

Project 3

- 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 p3.zip (see assignment on Canvas) 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 defines shared libraries that will be loaded & searched before any other libraries

- 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
```

This will:

- Load the shared library `$PWD/mylib.so`
 - `$PWD` expands to the path of the current directory
- Check this library for any functions the program needs before checking any other libraries

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

Notes:

- `shared` compiles a shared library
- `fpic` creates position-independent code
- `o myrand.so` names the library `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 361
-rw----- 1 pxk pxk      115 Apr  2 00:49 present.pptx
-rw-r----- 1 pxk pxk       55 Apr  2 00:49 secretfile
-rw----- 1 pxk pxk      141 Apr  2 00:49 secretfile.docx
-rw-r----- 1 pxk pxk   18198 Feb  1 00:00 secretfile.txt
-rw----- 1 pxk pxk   94698 Jul  5  2023 status-report-1.txt
-rw----- 1 pxk pxk  166518 Jul  5  2023 status-report-2.txt
-rw----- 1 pxk pxk   77166 Jul  5  2023 status-report-3.txt
-rw----- 1 pxk pxk   48858 Jul  5  2023 status-report-4.txt
-rw----- 1 pxk pxk       14 Apr  2 00:49 testfile.c
```

Example

Set the file name that we want to hide

```
$ export HIDDEN=secretfile.txt
```

Run the **ls** command again: `secretfile.txt` is missing!

```
$ ls -l
total 334
-rw----- 1 pxk pxk      115 Apr  2 00:49 present.pptx
-rw-r----- 1 pxk pxk       55 Apr  2 00:49 secretfile
-rw----- 1 pxk pxk      141 Apr  2 00:49 secretfile.docx
-rw----- 1 pxk pxk   94698 Jul  5  2023 status-report-1.txt
-rw----- 1 pxk pxk  166518 Jul  5  2023 status-report-2.txt
-rw----- 1 pxk pxk   77166 Jul  5  2023 status-report-3.txt
-rw----- 1 pxk pxk   48858 Jul  5  2023 status-report-4.txt
-rw----- 1 pxk pxk       14 Apr  2 00:49 testfile.c
```

Example

We can run another command, like *find*
`secretfile.txt` is still missing!

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

Example

If we change the file name that we want to hide

```
$ HIDDEN=status-report-2.txt
```

And run the **ls** command: `status-report-2.txt` is missing!

```
$ ls -l
total 232
-rw----- 1 pxk pxk   115 Apr  2 00:49 present.pptx
-rw-r----- 1 pxk pxk    55 Apr  2 00:49 secretfile
-rw----- 1 pxk pxk   141 Apr  2 00:49 secretfile.docx
-rw-r----- 1 pxk pxk 18198 Feb  1 00:00 secretfile.txt
-rw----- 1 pxk pxk 94698 Jul  5  2023 status-report-1.txt
-rw----- 1 pxk pxk 77166 Jul  5  2023 status-report-3.txt
-rw----- 1 pxk pxk 48858 Jul  5  2023 status-report-4.txt
-rw----- 1 pxk pxk   14 Apr  2 00:49 testfile.c
```

Example

If we remove HIDDEN:

```
$ unset HIDDEN
```

Then we can see all the files:

```
$ ls -l
total 361
-rw----- 1 pxk pxk    115 Apr  2 00:49 present.pptx
-rw-r----- 1 pxk pxk     55 Apr  2 00:49 secretfile
-rw----- 1 pxk pxk    141 Apr  2 00:49 secretfile.docx
-rw-r----- 1 pxk pxk 18198 Feb  1 00:00 secretfile.txt
-rw----- 1 pxk pxk 94698 Jul  5  2023 status-report-1.txt
-rw----- 1 pxk pxk 166518 Jul  5  2023 status-report-2.txt
-rw----- 1 pxk pxk  77166 Jul  5  2023 status-report-3.txt
-rw----- 1 pxk pxk 48858 Jul  5  2023 status-report-4.txt
-rw----- 1 pxk pxk    14 Apr  2 00:49 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 a second time 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

For full credit

- For the final 10% of the credit, support hiding multiple files
- Just like the PATH variable, allow multiple filenames separated by :

```
$ HIDDEN=status-report-2.txt:status-report-3.txt:secretfile
$ ls -l
total 232
-rw----- 1 pxk pxk    115 Apr  2 00:49 present.pptx
-rw----- 1 pxk pxk    141 Apr  2 00:49 secretfile.docx
-rw-r----- 1 pxk pxk 18198 Feb  1 00:00 secretfile.txt
-rw----- 1 pxk pxk 94698 Jul  5 2023 status-report-1.txt
-rw----- 1 pxk pxk 48858 Jul  5 2023 status-report-4.txt
-rw----- 1 pxk pxk    14 Apr  2 00:49 testfile.c
```

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, 2023
 - It also refuses to run with any date earlier than January 1, 2022.
- **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 Jan 1 2022 ... Jan 1 2023 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 Apr 02 2024 01:02:11
You cannot run this program before Sat Jan  1 00:00:00 2022
This software expires at Sun Jan  1 00:00:00 2023

ACCESS DENIED: It is now Apr 02 2024 01:02:11. Access expired at Sun Jan  1 00:00:00 2023
```

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

```
$ LD_PRELOAD=$PWD/newtime.so ./unexpire
It is now Sep 01 2022 01:00:00
You cannot run this program before Sat Jan  1 00:00:00 2022
This software expires at Sun Jan  1 00:00:00 2023

Sep 01 2022 01:00:00: access granted!
The current time is: Apr 02 2024 01:03:43
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 2022 ... Jan 1 2023
 - 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 some 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