Assignment 6, Fall 2015
CS 4630, Defense Against the Dark Arts
Inserting a Tricky Jump
Jumpicus Captiosam Curse

Purpose

This assignment will explore what it takes to create a stealthy virus that employs a "tricky jump." A tricky jump is a form of hijacking in which a jump is inserted to call some virus code. The jump is inserted in such a way that after the virus code runs, the program continues normal execution, thereby maintaining stealth.

Due Date

This assignment is due on Wednesday, 11/04/2015, at 10:00am

Tricky Jump (Jumpicus Captiosam)

A tricky jump can be efficiently implemented (only six bytes) as:

```
    push <Address of Virus Function>
    ret
```

When this sequence is executed, control is diverted to the virus code. When the virus code returns, control returns to the function that called the function that contained the tricky jump. If the virus writer inserts the tricky jump at the end of an application function (i.e., the epilogue code), then the program will continue to run as if nothing happened. Here's an example (taken from the output of objdump with the use of the --disassemble flag).

```
                    <deleted lines>
8048440:      83 ec 18                sub    $0x18,%esp
8048443:      c7 04 24 37 85 04 08    movl   $0x8048537,(%esp)
804844a:      e8 81 fe ff ff          call   80482d0 <puts@plt>
804844f:      c9                      leave
8048450:      c3                      ret
8048451:      90                      nop
8048452:      90                      nop
8048453:      90                      nop
8048454:      90                      nop
```

Notice the padding with nop's at the end of the function. This is essentially a "cavity" that gives the virus writer some room to work. If we insert a tricky jump starting where the ret instruction is located (address 0x08048450) then the virus code will be invoked and when the virus code returns, control will be returned to the function that invoked this function.

For this assignment, you will write a C program that infects an Linux executable and causes some virus code to be executed. You will use the "tricky jump" method of infection. To make things simpler, the virus code (i.e., function VirusCode) to invoke will already be in the executable that you are to infect.

(With more effort, we could also inject the virus payload, but for now we want to concentrate on getting the tricky jump inserted.)

The Linux executable you want to infect is called *target_code.bin*. This executable will be provided to you on the Collab site. When you run an uninfected version of *target_code.bin*, it produces the following output.

```
$ ./target_code.bin
Initialize application.
Begin application execution.
Terminate application.
$
```

Now, we run the infect program which processes *target_code.bin* and produces *infected_target_code.bin*. When we run *infected_target_code.bin*, it produces the following output.

```
$ ./infected_target_code.bin
Initialize application.
You have been infected with a virus!
Begin application execution.
Terminate application.
$
```

Notice that after calling the initialization function, the virus payload has been called and it prints a message and then execution continues in the application as normal.

Assignment Details

1. Write a C program called *infect.c* that when compiled and executed reads a Linux executable and produces new infected executable where a tricky jump has been inserted so that when the infected executable is run, function VirusCode() is executed before the main application runs. See the previous example output to see the output the infected executable should produce.

2. Your C program will take two parameters. The first parameter is the filename of the to-be-infected Linux executable. The second parameter is the name of the infected Linux executable that your program generates. For example, in the following example, *target_code.bin* is infected and a new executable *infected_target_code.bin* is generated.

```
$ ls
target_code.bin
$ ./infect target_code.bin infected_target_code.bin
$ ls
target_code.bin infected_target_code.bin
```

3. Your program should not change the size of the executable. That is *target_code.bin* and *infected_target_code.bin* should have identical sizes. The size of file *target_code.bin* is 5516 bytes.

4. Your should infect the *target_code.bin* you download from the collab. **DO NOT compile your own *target_code.bin*.** We provide source code only to help you understand the target_code.bin.

CS4630 Assignment 6

Methodology and Hints

1. You should use the utility objdump to examine the executable *target_code.bin*. The option --disassemble is useful. In particular, you need to determine the starting address of the virus code. The dissasembly will also help you determine the opcodes of the instructions that you need to insert (i.e., a push instruction and a ret instruction).

2. Other useful tools are hexdump and readelf. h*exdump* prints the hexadecimal dump of a raw executable file. *readelf -S* tells you the beginning address of .text section in the executable file.

3. The trick is that you must map the address of the location in the executable to the offset of the proper byte in the file. You need to do this mapping because the file offset where you want to write is not the same as the address of the instruction when the program is loaded in memory (which is what objdump is showing you). Using the disassembly from step 1, the hexadecimal dump of the raw file and the readelf output, you can determine which bytes of the file must be overwritten.

4. A very useful program to examine the file is ghex . You can install it using apt-get . It is a visual binary file editor. Google around to see what is available.

5. After you produce infected_target_code.bin you will probably need to set the execute permissions on the file.

6. The hard part is figuring out what locations in the file need to be changed and what they should be changed to. The code to do the infection is small. My infect.c is 39 lines of C code. I wasn't trying to make it short so a solution could easily be about 25 lines of code.

7. Very important hint: We are reading and writing binary files—not textfiles. You need to open the files in binary mode so the OS does not interpret the input characters (e.g., treat ^D as an end-of-file).

Submission Guideline

1. **WARNING: YOUR SUBMISSION ***MUST*** FOLLOW THIS GUIDELINE. THERE WILL BE 40% PENALTY FOR THOSE WHO FAIL TO FOLLOW THIS GUIDELINE.**
2. Submit your *infect.c* to collab. **THE FILENAME MUST BE infect.c.**
3. Your program will be compiled with the following command:

   ```
   gcc -m32 -o infect infect.c
   ```

   You will receive 0 points if your program fails to compile with this command.
4. **Your program MUST take two parameters.** The first parameter is the filename of the to-be-infected executable. The second parameter is the filename of the infected executable that your program generates. More information can be found in Assignment Details.