Software and Web-Security
Assignment 4, Monday, March 10, 2014

Handing in your answers: Submission via Blackboard (http://blackboard.ru.nl)

Deadline: Tuesday, March 18, 24:00 (midnight)

1. There are two variants of this homework exercise: the "normal" variant and the "hard" variant. Only choose the hard variant if you want some extra challenge, otherwise pick the normal one. Download either the program http://www.cs.ru.nl/~erikpoll/sws1/exercises/pwd-normal (normal) or the program http://www.cs.ru.nl/~erikpoll/sws1/exercises/pwd-hard (hard).

   (a) Use `gdb` to find out what the program does. Describe in detail (for example, equivalent C code) what the program does; write your answer to a file called `exercise1a`.

   (b) Find an input (password) that makes the program print "You're root!". Explain why this input gives you "root access". Write your answer (both, input and explanation) to a file called `exercise1b`. **Note:** Choose the input such that the program does not crash after printing "You're root!".

2. In assignment 2 you had to figure out the amount of stack space a called function uses. In this exercise, you will manipulate the stack to influence the flow of a program. Consider the following code:

```
#include <stdio.h>

void function_b(void) {
        char buffer[4];

        // ... insert code here

        fprintf(stdout, "Executing function_b\n");
}

void function_a(void) {
        int beacon = 0xa0b1c2d3;
        fprintf(stdout, "Executing function_a\n");
        function_b();
        fprintf(stdout, "Executed function_b\n");
}

int main(void) {
        function_a();
        fprintf(stdout, "Finished!\n");
        return 0;
}
```

   Place this code in a file. Create a Makefile so that the program compiles with debug symbols turned on (`-g`), no optimizations enabled (omit any `-O` flags), and make sure the frame pointer is present (`-fno-omit-frame-pointer`): `gcc -g -fno-omit-frame-pointer exercise4.c`

   When run it should print

```
Executing function_a
Executing function_b
Executed function_b
Finished!
```

   (a) Copy the program to a file called `exercise2a.c`. Change the designated section of `function_b` so that it changes the stack in such a way that the program crashes *after* printing

```
Executing function_a
Executing function_b
```

Note that you can only insert code *before* the call to `fprintf`, which has to remain the last call of the function. You are not allowed to change `main` or `function_a`. You are also not allowed to use additional print statements to generate the desired output.

(b) For this exercise you must run the program in `gdb`. `gdb`'s TUI mode (`gdb --tui a.out`) might be helpful.

Run the original program (without the changes from exercise 2a) until somewhere in `function_b()`. Inspect the stack frames for both the current stack frame (for `function_b()`) and the stack frame of its caller (`function_a()`). You can switch frames with the command `frame X`, where X is the number of the frame you want. For each frame, write the following information, and the steps you took to get it, to a file called `exercise2b`:

- which frame it is,
- which function the frame belongs to,
- the location (address in memory) of the frame,
- the location of the frame it was called by,
- (if applicable) the location of the frame it called,
- the value of the instruction pointer (`%eip` on 32-bit, `%rip` on 64-bit systems),
- the location of the return address,
- the value of the return address,
- the value of the base pointer (`%ebp`, `%rbp`),
- the location of the saved base pointer,
- the value of the saved base pointer,
- the value of the stack pointer, and
- the address of the local variable (try e.g. `print/x &beacon`).

You can print memory from addresses in `gdb` using `x`. To print 8 bytes at memory address 0x12345678 you would enter `x/xg 0x12345678`. See `help x` from within `gdb` for more information.

NOTE: It is easy to get this information; if you find you are using more than a few commands for each frame, ask for a hint.

(c) This exercise also has two variants, "normal" and "hard". The normal variant assumes the program will be run within gdb (without TUI mode, which breaks printing), which results in a predictable address layout. For the hard variant, run the program on the command line as normal.

Copy the program to a file called `exercise2c.c`. Change the designated section of `function_b` so that it changes the stack in such a way that the program prints

```
Executing function_a
Executing function_b
Finished!
```

Note that `Executed function_b` is not in the output. Once again, you are only allowed to insert code *before* the call to `fprintf`, and are not allowed to change the other functions. You are also not allowed to use additional print statements to generate the desired output.

Hint: The location of the return addresses, base pointers, their values and their location relative to the local variables, from exercise 2b, should help you determine what to do. However, after changing the code you will need to repeat some steps from exercise 2b to get the correct values for the new program.

3. Place the files `exercise1a`, `exercise1b`, `exercise2a.c`, `exercise2b`, and `exercise2c.c` in a directory called `sws1-assignment4-STUDENTNUMBER1-STUDENTNUMBER2` (again, replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers). Make a `tar.gz` archive of the whole `sws1-assignment4-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Blackboard.