Hacking in C

Assignment 4, Tuesday, May 11, 2021

Handing in your answers: Submission via Brightspace (http://brightspace.ru.nl)

Deadline: Thursday, May 20, 23:59

Helper programs: We have provided some helper programs that should help you perform some menial stuff, like printing addresses in little-endian order. Download them from Brightspace. Build them using the provided `Makefile`. Follow the instructions in `README.md` to install them. For this assignment, you will likely only need `reverseaddr`.

1. Consider the following program:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[1]) {
    char command[10];
    char buffer[100];
    strcpy(command, "/bin/ls");
    if (argc < 2) {
        fprintf(stderr, "Usage: %s <string>\n", argv[0]);
        exit(1);
    }

    strcpy(buffer, argv[1]);

    printf("* [INFO] The value of command is: \"%s\"\n", command);

    system(command);

    return 0;
}
```

This program copies its first (and only) command line argument to the `char` array `buffer`. Then it runs the command stored in the `char` array `command`, similar to how it would run on the commandline. By default, it runs `/bin/ls`. This makes it print a directory listing.

Your goal is to make it run an interactive shell instead.

(a) Download `buffer.tar.gz` from Brightspace, and unpack it using `tar -xvf buffer.tar.gz`. It contains this program and its `Makefile`. The `Makefile` makes sure that you compile it with `-fno-stack-protector`.

Compile the program, then run it with a few different inputs to observe its behaviour if you overrun the size of `buffer`.

(b) Give an *input* to this exact program that makes it run `/bin/sh` instead of `/bin/ls`. Write your solution to a file called `exercise1`.

Assignment continues on the next page.

2. 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 `pwd-normal` (normal) or the program `pwd-hard` (hard) from Brightspace. You may need to run `chmod +x pwd-normal` to mark the downloaded file as executable.

   (a) Use `gdb` to find out what the program does. Describe in detail (for example, equivalent C or pseudocode) what the `main` function of the program does; write your answer to a file called `exercise2a`.

   While you do not have the source code, using the `disassemble` command will print the assembly code of the program (after starting the execution of the program using the `gdb` command `start`).

   You can step through the program using `si` (step instruction) and `ni` (next instruction).

   The *normal* version of the exercise is compiled with debugging symbols, so commands like `info locals` will work.

   `gdb` will give you some information about the functions being called, but you may want to look for comparisons and jumps to infer the control flow.

   It may be helpful to look at the following:

   - The local variables
   - The function names
   - The external functions being called

   You may also refer to the following resources:

   - gdb quick reference: https://users.ece.utexas.edu/~adnan/gdb-refcard.pdf
   - Quick intro to gdb (YouTube video): https://www.youtube.com/watch?v=xQ0ONbt-qPs

   (b) Find an input ("password") that makes the program print "You're root!"[1]. Explain why this input gives you "root access". Write your answer (both the input and an explanation) to a file called `exercise2b`.

   Note: Choose the input such that the program does not crash after printing "You're root!".

Assignment continues on the next page.

[1] Don't bother trying to find the *valid* password.

3. Consider the following program:

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

// colour code magic
#define RED "\033[0;31m"
#define GREEN  "\033[0;32m"
#define NC  "\033[0m"

int check_passphrase(char* passphrase) {
    char buffer[100];
    strcpy(buffer, passphrase);
    if(strcmp(passphrase, "the magic words are squeamish ossifrage") == 0) {
        return 1;
    }
    return 0;
}

void launch_shell() {
    printf(GREEN "Launching shell." NC "\n");
    system("/bin/bash");
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <passphrase>\n", argv[0]);
        exit(0);
    }

    printf("* [DEBUG] Your input: %s\n", argv[1]);
    printf("* [DEBUG] The function launch_shell is at %p\n", launch_shell);

    if (check_passphrase(argv[1])) {
        launch_shell();
        exit(0);
    } else {
        fprintf(stderr,
                RED "Wrong password. This incident will be reported. "
                "https://xkcd.com/838/" NC "\n");
    }
    return 1;
}
```

The developer of it left some debugging code in the program. It should make your life easier.

(a) Download `functions.tar.gz` from Brightspace, and unpack it using `tar -xvf functions.tar.gz`. Compile the program using the included `Makefile`, which sets the correct flags.

(b) You should figure out how to get the program to start the shell **without supplying the correct password**. Using `gdb` may be helpful in making this exercise easier, but you can do it without. If you use `gdb`, try to first figure out what information you will need. `break` and `info frame` should be good starting points. Write your answer into a text file called `exercise3`.

Important: If you want to run your attack without `gdb`, you **must** disable address randomization, a mechanism that makes these attacks harder. Run the command `setarch $(uname -m) -R` to start a shell where address randomization is turned off. Without this, the `launch_shell` function will be at a different address every time!

Assignment continues on the next page.

4. Place the files

   - `exercise1`,
   - `exercise2a`,
   - `exercise2b`, and
   - `exercise3`

   in a directory called `hic-assignment4-STUDENTNUMBER1-STUDENTNUMBER2` (again, replace `STUDENTNUMBER1` and `STUDENTNUMBER2` by your respective student numbers). Make a `tar.gz` archive of the whole `hic-assignment4-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Brightspace.