Hacking in C

Assignment 4, Tuesday, May 19, 2020

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

Deadline: Tuesday, **May 26**, 12:30

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;
}
```

   (a) Download this program and its `Makefile` from https://rded.nl/hic/buffer.tar.gz. The `Makefile` makes sure that you compile it with `-fno-stack-protector`.

   (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`.

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 https://rded.nl/hic/pwd-normal (normal) or the program https://rded.nl/hic/pwd-hard (hard). 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 pseudo-code) what the `main` function of the program does; write your answer to a file called `exercise2a`. 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 https://www.youtube.com/watch?v=xQ0ONbt-qPs

   While you do not have the source code, using the `disassemble` command will print the assembly code of the program. 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.

(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 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!".

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

```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;
}
```

(a) Obtain the program from `https://rded.nl/hic/functions.tar.gz`. Compile it 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!

4. Place the files

   - `exercise1`
   - `exercise2a`,
   - `exercise2b`,
   - `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 `hackingc-assignment4-STUDENTNUMBER1-STUDENTNUMBER2` directory and submit this archive in Brightspace.