Lab 7 Exercise 2 - Protecting the Return Address

Name: NOEL LIM XIAN
Student Number: A0211270Y
Email: E0493357@u.nus.edu
Date: 31 OCTOBER 2022

Exercise 2

```
// EXERCISE 2
// Only f is modified from lab7-ex1.c
// Two compiled binaries are compiled for Exercise 2 as below. Both are executed
with arg `-p` to flip my_protect bit to 1 and activate the detection mechanism.
// 1. using `$ gcc -fno-stack-protector lab7-ex2.c`
// 2. using `$ gcc -fno-stack-protector -O2 lab7-ex2.c`
int f()
{

// Define canary size as 8 [characters].
#define CANARY_SIZE 8
 /*
   1.  Open "/dev/urandom" in read mode and assign the new FILE pointer from
fopen() to fp.
       This should be the first memory allocation instruction in the stack to
prevent interjection of the order of variables in this exercise.
       We get data "/dev/urandom" for the canary value so that we have random
values in each execution (harder for attacker to predict runtime, arbitrary value)
 */
 FILE *fp = fopen("/dev/urandom", "r");

 /*

   2.  The variables in the stack should be in contiguous address in order of
declaration:

     +--------+
     |        |  canary_value (8 bytes)
     +--------+
     |        |  buf (8 bytes)
     +--------+
     |        |  canary (8 bytes)
     +--------+
     .        .  c (1 byte)
```

```
    +--------+


    canary_value  - random value as original "cookie".
    buf           - buf to get data from stdin.
    canary        - canary that is immediate after buf in the addressing space.
 */

 char canary_value[CANARY_SIZE];
 char buf[8];
 char canary[CANARY_SIZE];
 char c;

 /* 3.   my_protect reads 8 bytes from /dev/urandom into canary_value. canary_value
is then copied into canary.

 */
 if (
    my_protect == 1)
 {
   int p = fread(&canary_value, 1, CANARY_SIZE, fp);
   if (p != CANARY_SIZE)
   {
     fprintf(stderr, "size read from /dev/urandom deviates from CANARY_SIZE \n");
     exit(1);
   }
   while (--p >= 0)
     canary[p] = canary_value[p];
 }

 fclose(fp);

 // LINE1 - don't change code from LINE1 to LINE2
 printf("Enter a string: ");
 for (i = 0; (c = getchar()) != '\n'; i++)
   buf[i] = c;
 buf[i] = '\0';
 printf("string = [%s]\n", buf);
 // LINE2

 /*  4. If there is buffer overflow from the array buf, canary will change and
differ from canary_value (the original value).
        We will compare each character of canary_value and canary.
        Upon detection of change, cookie_error is raised.
 */
```

```
 if (my_protect == 1)
 {
   for (int p = 0; p < CANARY_SIZE; p++)
   {
     if (canary[p] != canary_value[p])
     {
       cookie_error();
     }
   }
 }

 return 0;
}
```

END