CMPT 295

Name: Junchen Li

Student ID: 301385486

Date: 2020/3/12

Assignment 6

Objectives:

- Experimenting with Stack Randomization
- Investigating the size of some x86-64 assembly instructions

---

Submission:

- Submit your document called **Assignment_6.pdf**, which must include your answers to all of the questions in Assignment 6.
    - Add your full name and student number at the top of the first page of your document **Assignment_6.pdf**.
- When creating your assignment, first include the question itself and its number then include your answer, keeping the questions in its original numerical order.
- **If you hand-write your answers (as opposed to using a computer application to write them):** When putting your assignment together, do not take photos (no .jpg) of your assignment sheets! Scan them instead! Better quality -> easier to read -> easier to mark!
- Submit your assignment **Assignment_6.pdf** electronically on CourSys.

Due:

- Thursday, March 12 at 3pm
- Late assignments will receive a grade of 0, but they will be marked in order to provide the student with feedback.

Marking scheme:

This assignment will be marked as follows:

- All questions will be marked for correctness.

The amount of marks for each question is indicated as part of the question.

A solution will be posted after the due date.

---

1. [8 marks] Experimenting with Stack Randomization

   We saw this week that one of the ways to counter buffer overflow attacks was to use compilers (like `gcc` for Linux) that implemented “safety” mechanisms such as Stack Randomization. On way the compiler does this stack randomization is by assigning a different start address to the stack every time a program execxutes. This changing start address to the stack makes it difficult for hackers to predict the location of return addresses on the stack.

   Before you proceed with this question, make sure you read Section 3.10.4 in our textbook and review our course notes on this topic.

   In this assignment you are asked to confirm or disprove the fact that gcc does implement Stack Randomization by running your own experiement as follows:

   - Download, from our course web site, the password program (main.c) we saw as part of Lecture 21 demo.
   - Add a statement that prints the memory address of the first byte of the array `password`. Hint: you cannot use %d nor %x to print this memory address.
   - Execute your program 10 times and record the addresses at which this local variable array is stored on the stack.
   - Then state your conclusion and support it with your experiental results.
   - Finally, calculate the range of variance in the memory addresses you obtained (if any) over the 10 executions of your program.
   - Include your modified main.c in your solution to this assignment (in Assignment_6.pdf).

```
junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

1

password is 1

 THe first byte of address of Password is 0x7ffff333d127

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x
```

```
Enter Password:

2

password is 2

 THe first byte of address of Password is 0x7fff02cccf37

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

3

password is 3

 THe first byte of address of Password is 0x7fff6f0cb8b7

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

4

password is 4

 THe first byte of address of Password is 0x7fff157ae7f7

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

5
```

```
password is 5

 THe first byte of address of Password is 0x7ffde6f5d4e7

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

6

password is 6

 THe first byte of address of Password is 0x7fff325772a7

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

7

password is 7

 THe first byte of address of Password is 0x7fff88788917

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

8

password is 8

 THe first byte of address of Password is 0x7ffc0c46bd57
```

```
Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

9

password is 9

 THe first byte of address of Password is 0x7ffed1ac2ae7

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$ ./x

Enter Password:

0

password is 0

 THe first byte of address of Password is 0x7ffe9cc180f7

Access denied


junchenl@asb9838nu-e13:~/sfuhome/password$
```

Because of Thwarting Buffer Overflow Attacks, stack will ues its special property called Stack Randomization to random the address of using stack. Looking through all data from lab, we obtained addresses ranging from a minimum of 0x7ffc0c46bd57 to a maximum of 0x7ffff333d127. The range is between [140720514448727~140737273647399]. The difference between these two addresses is 0x3E6ED13D0.

```
# main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

bool IsPasswordOK() {

   bool retResult = 0;

   unsigned int aSize = 9;

   char goodpwd[] = "goodpass";

   // Memory storage for pwd

   char password[aSize];


   // Get input from keyboard

   gets(password);


   // char * result = fgets(password, aSize, stdin);

   // if ( result == NULL ) {

   //     printf("fgets unsuccessful\n");

   //     return 0;

   // }

   // Echo the password

   printf("password is %s\n", password);

   printf(" THe first byte of address of Password is %p \n", password);
```

```
    // strcmp(str1,str2) returns a value < 0 if str1 < str2.
    // strcmp(str1,str2) returns a value > 0 if str1 > str2.
    // strcmp(str1,str2) returns a value = 0 if str1 is equal to str2.
    if ( !strcmp(password, goodpwd) )
      retResult = 1; // !(0) -> !(false) = true since password equal goodpwd
    else
      retResult = 0; // !(1) -> !(true) = false since Password not equal goodpwd


    return retResult;
}
int main() {
    bool pwStatus;   // Password Status
    int result = 0;
    puts("Enter Password:");
    pwStatus = IsPasswordOK(); // Get & check password
    if ( pwStatus )          // 1 -> true
      puts("Access granted");
    else {                   // 0 -> false
      puts("Access denied");
      result = 1;
    }
    return result;
}
```

2. [12 marks] Investigating the size of some x86-64 assembly instructions

Complete the following three tables:

X86-64 Instructions	**Their size (in byte)**
xorq **%rax**, %rax	3
xorl %eax, %eax	2
movq $0, **%rax**	7
movl $0, **%eax**	5

X86-64 Instructions	**Their size (in byte)**
addl **$1**, %eax	3
leal 1(%eax), **%eax**	4
incl **%eax**	2

X86-64 Instructions	**Their size (in byte)**
addl $8, **%eax**	3
leal 8(%eax), **%eax**	4

To complete the above tables, you must:

- Write an assembly program and include all x86-64 instructions in it. Of course, this program is going to be rather nonsensical. Not a problem!
- Compile your program.
- Use the `objdump` tool on the object file you obtained from the compilation process.
- Looking at the result will lead you to the answer.
- In each table, **bold** the instruction that is the most space efficient.

- Include your program (*.s) in your solution to this assignment (in Assignment_6.pdf).
- Include the result you obtained from the **`objdump`** tool in your solution to this assignment (in Assignment_6.pdf).

```
# lettry.s


        .globl    lettry
  lettry:
          xorq  %rax, %rax
          xorl          %eax, %eax
          movq            $0, %rax
          movl            $0, %eax
          addl            $1, %eax
          leal            1(%eax), %eax
          incl            %eax
          addl            $8, %eax
          leal            8(%eax), %eax
```

```
# lettry.objdump
lettry.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:    48 31 c0                xor    %rax,%rax
   3:    31 c0                   xor    %eax,%eax
   5:    48 c7 c0 00 00 00 00    mov    $0x0,%rax
   c:    b8 00 00 00 00          mov    $0x0,%eax
  11:    83 c0 01                add    $0x1,%eax
  14:    67 8d 40 01             lea    0x1(%eax),%eax
  18:    ff c0                   inc    %eax
  1a:    83 c0 08                add    $0x8,%eax
  1d:    67 8d 40 08             lea    0x8(%eax),%eax
```
