Lab6: Buffer Overflow Principles

Getting Started

$ gcc -g stack1.c -o stack1

$ ./stack1

```
root@kali:~# cd Desktop
root@kali:~/Desktop# gcc -g stack1.c -o stack1
root@kali:~/Desktop# ./stack1
Returned Properly
```

Exercise 1. Now, you can write some code. Your job is to print the address of the variable buffer, in the C program stack1.c, and compile the C program as above. Run it three times, observe and write down the output addresses in address.txt, are these 3 addresses the same or not?

```
char buffer[12];
/* Fill in code here to print the address of
 * the array "buffer".
 * Your code here:
 */
printf("address:%p\n",buffer);
strcpy(buffer, str);
```

```
root@kali:~/lab1-code# gcc -g stack1.c -o stack1
root@kali:~/lab1-code# ./stack1
address:0×7ffe73ea4134
Returned Properly
root@kali:~/lab1-code# ./stack1
address:0×7ffc71d216c4
Returned Properly
root@kali:~/lab1-code# ./stack1
address:0×7ffe5c721624
Returned Properly
```

No, the address are different for every run.

Exercise 2. Use gdb to debug the program, as the following. You may find the online gdb manual http://www.sourceware.org/gdb/current/onlinedocs/gdb/ useful.

```
root@kali:~/lab1-code# gdb stack1
GNU gdb (Debian 8.3.1-1) 8.3.1
Copyright (C) 2019 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htm
l>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from stack1...
(gdb) b func
Breakpoint 1 at 0x1161: file stack1.c, line 13.
(gdb) r
Starting program: /root/lab1-code/stack1

Breakpoint 1, func (str=0x555555556010 "hello\n") at stack1.c:13
13          printf("address:%p\n",buffer);
```

```
(gdb) info r
rax            0x555555556010      93824992239632
rbx            0x0                 0
rcx            0x7ffff7fb4718      140737353828120
rdx            0x7fffffffe2a8      140737488347816
rsi            0x7fffffffe298      140737488347800
rdi            0x555555556010      93824992239632
rbp            0x7fffffffe180      0x7fffffffe180
rsp            0x7fffffffe160      0x7fffffffe160
r8             0x7ffff7fb6a50      140737353837136
r9             0x7ffff7fe4780      140737354024832
r10            0x0                 0
r11            0x27                39
r12            0x555555555070      93824992235632
r13            0x7fffffffe290      140737488347792
r14            0x0                 0
r15            0x0                 0
rip            0x555555555161      0x555555555161 <func+12>
eflags         0x206               [ PF IF ]
cs             0x33                51
ss             0x2b                43
ds             0x0                 0
es             0x0                 0
fs             0x0                 0
```

```
gs             0x0                 0
(gdb) x/2s 0x555555556010
0x555555556010: "hello\n"
0x555555556017: "Returned Properly"
(gdb) p &buffer
$1 = (char (*)[12]) 0x7fffffffe174
(gdb) x/4wx 0x7fffffffe174
0x7fffffffe174: 0x00000000      0x55555225      0x00005555      0xffffe1b0
(gdb) x/8wx $rbp
0x7fffffffe180: 0xffffe1b0      0x00007fff      0x555551cb      0x00005555
0x7fffffffe190: 0xffffe298      0x00007fff      0x55555070      0x00000001
(gdb) x/2i 0x00007fff
   0x7fff:      Cannot access memory at address 0x7fff
(gdb) x/2i 0x0000e1b0
   0xe1b0:      Cannot access memory at address 0xe1b0
(gdb) x/8wx $ebp
0xffffffffffffe180:     Cannot access memory at address 0xffffffffffffe180
(gdb) x/2i 0x00007fff
   0x7fff:      Cannot access memory at address 0x7fff
(gdb) x/2i 0x08048443
   0x8048443:   Cannot access memory at address 0x8048443
(gdb) x/2i 0x555551cb
   0x555551cb:  Cannot access memory at address 0x555551cb
(gdb) x/2i 0x55555070
```

```
(gdb) x/2i 0x55555070
   0x55555070:  Cannot access memory at address 0x55555070
(gdb) x/2i 0xffffe298
   0xffffe298:  Cannot access memory at address 0xffffe298
(gdb) disass func
Dump of assembler code for function func:
   0x0000555555555155 <+0>:     push   %rbp
   0x0000555555555156 <+1>:     mov    %rsp,%rbp
   0x0000555555555159 <+4>:     sub    $0x20,%rsp
   0x000055555555515d <+8>:     mov    %rdi,-0x18(%rbp)
=> 0x0000555555555161 <+12>:    lea    -0xc(%rbp),%rax
   0x0000555555555165 <+16>:    mov    %rax,%rsi
   0x0000555555555168 <+19>:    lea    0xe95(%rip),%rdi        # 0x55555555
6004
   0x000055555555516f <+26>:    mov    $0x0,%eax
   0x0000555555555174 <+31>:    callq  0x555555555050 <printf@plt>
   0x0000555555555179 <+36>:    mov    -0x18(%rbp),%rdx
   0x000055555555517d <+40>:    lea    -0xc(%rbp),%rax
   0x0000555555555181 <+44>:    mov    %rdx,%rsi
   0x0000555555555184 <+47>:    mov    %rax,%rdi
   0x0000555555555187 <+50>:    callq  0x555555555030 <strcpy@plt>
   0x000055555555518c <+55>:    mov    $0x1,%eax
   0x0000555555555191 <+60>:    leaveq
   0x0000555555555192 <+61>:    retq
End of assembler dump.
```

Exercise 3. Turn off the address space layout randomization, and then do exercise 1 again, write down the three addresses in args.txt, are those three addresses same or not?

Yes, as in picture shows, three addresses are the same.

```
root@kali:~/lab1-code# sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
root@kali:~/lab1-code# gcc -z execstack test-shell.c -o test-shell
root@kali:~/lab1-code# ./stack1
address:0x7fffffffe194
Returned Properly
root@kali:~/lab1-code# ./stack1
address:0x7fffffffe194
Returned Properly
root@kali:~/lab1-code# ./stack1
address:0x7fffffffe194
Returned Properly
root@kali:~/lab1-code#
```

Exercise 4. Use gdb, to print the value of the register %eip when the program crashes. How does the program run to this address?

```
root@kali:~/lab1-code# ./stack1 aaaaaaaaaa
address:0x7fffffffe184
Returned Properly
root@kali:~/lab1-code# ./stack1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
address:0x7fffffffe164
Segmentation fault
```

```
(gdb) i r $rip
rip            0x555555555161      0x555555555161 <func+12>
```

I got the value of register %rip by using command i r $rip. The value is 0x555555555161

If the program runs normally, it will jump to 0x7fffffffe184. When input be aaaaaaaaaaaaaaa, the function crushes $rip's value is 0x555555555161

The input is too long to make the buffer overflow, buffer can not store input "aaaaaaaaaaaaaaa" and overflow it