



中国科学技术大学

《网络安全》

实验报告

姓名：吴佳明

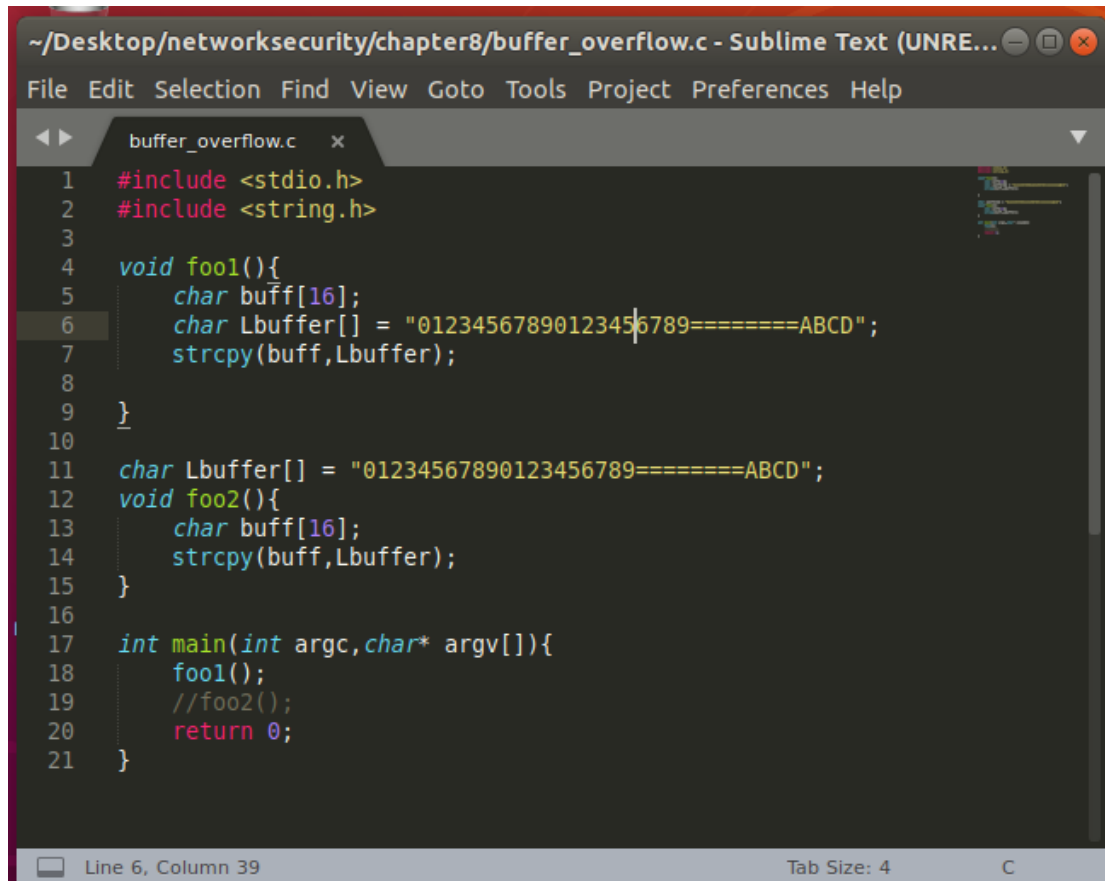
学号：SA19221027

年级：硕士研究生一年级

本实验基于 ubuntu18 64bit

首先对 foo1 函数进行分析。

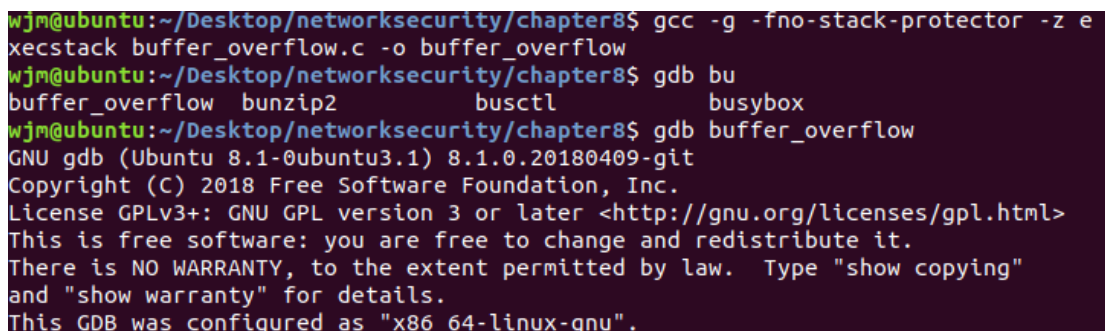
函数如下：



```
~/Desktop/networksecurity/chapter8/buffer_overflow.c - Sublime Text (UNRE...
File Edit Selection Find View Goto Tools Project Preferences Help

buffer_overflow.c x
1  #include <stdio.h>
2  #include <string.h>
3
4  void foo1(){
5      char buff[16];
6      char Lbuffer[] = "01234567890123456789=====ABCD";
7      strcpy(buff,Lbuffer);
8
9  }
10
11 char Lbuffer[] = "01234567890123456789=====ABCD";
12 void foo2(){
13     char buff[16];
14     strcpy(buff,Lbuffer);
15 }
16
17 int main(int argc,char* argv[]){
18     foo1();
19     //foo2();
20     return 0;
21 }
```

用 gcc 命令进行编译，使用 gdb 命令进行调试。



```
wjm@ubuntu:~/Desktop/networksecurity/chapter8$ gcc -g -fno-stack-protector -z e
xecstack buffer_overflow.c -o buffer_overflow
wjm@ubuntu:~/Desktop/networksecurity/chapter8$ gdb bu
buffer_overflow bunzip2 busctl busybox
wjm@ubuntu:~/Desktop/networksecurity/chapter8$ gdb buffer_overflow
GNU gdb (Ubuntu 8.1-0ubuntu3.1) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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".
```

使用 disas 命令查看汇编代码（先运行一次）。

```
(gdb) disas main
Dump of assembler code for function main:
   0x00005555555546c2 <+0>:    push    %rbp
   0x00005555555546c3 <+1>:    mov     %rsp,%rbp
   0x00005555555546c6 <+4>:    sub     $0x10,%rsp
   0x00005555555546ca <+8>:    mov     %edi,-0x4(%rbp)
   0x00005555555546cd <+11>:   mov     %rsi,-0x10(%rbp)
=>  0x00005555555546d1 <+15>:   mov     $0x0,%eax
   0x00005555555546d6 <+20>:   callq   0x55555555464a <foo1>
   0x00005555555546db <+25>:   mov     $0x0,%eax
   0x00005555555546e0 <+30>:   leaveq
   0x00005555555546e1 <+31>:   retq
End of assembler dump.
```

```
(gdb) disas foo1
Dump of assembler code for function foo1:
   0x000055555555464a <+0>:    push    %rbp
   0x000055555555464b <+1>:    mov     %rsp,%rbp
   0x000055555555464e <+4>:    sub     $0x40,%rsp
   0x0000555555554652 <+8>:    movabs  $0x3736353433323130,%rax
   0x000055555555465c <+18>:   movabs  $0x3534333231303938,%rdx
   0x0000555555554666 <+28>:   mov     %rax,-0x40(%rbp)
   0x000055555555466a <+32>:   mov     %rdx,-0x38(%rbp)
   0x000055555555466e <+36>:   movabs  $0x3d3d3d3d39383736,%rax
   0x0000555555554678 <+46>:   movabs  $0x444342413d3d3d3d,%rdx
   0x0000555555554682 <+56>:   mov     %rax,-0x30(%rbp)
   0x0000555555554686 <+60>:   mov     %rdx,-0x28(%rbp)
   0x000055555555468a <+64>:   movb    $0x0,-0x20(%rbp)
   0x000055555555468e <+68>:   lea     -0x40(%rbp),%rdx
   0x0000555555554692 <+72>:   lea     -0x10(%rbp),%rax
   0x0000555555554696 <+76>:   mov     %rdx,%rsi
   0x0000555555554699 <+79>:   mov     %rax,%rdi
   0x000055555555469c <+82>:   callq   0x555555554520 <strcpy@plt>
   0x00005555555546a1 <+87>:   nop
   0x00005555555546a2 <+88>:   leaveq
   0x00005555555546a3 <+89>:   retq
End of assembler dump.
(gdb)
```

设置断点并开始运行。

```
(gdb) b *(foo1+0)
Breakpoint 1 at 0x64a: file buffer_overflow.c, line 4.
(gdb) b *(foo1+82)
Breakpoint 2 at 0x69c: file buffer_overflow.c, line 7.
(gdb) b *(foo1+88)
Breakpoint 3 at 0x6a2: file buffer_overflow.c, line 9.
(gdb) b *(foo1+89(
A syntax error in expression, near `'.
(gdb) b *(foo1+89)
Breakpoint 4 at 0x6a3: file buffer_overflow.c, line 9.
(gdb) r
Starting program: /home/wjm/Desktop/networksecurity/chapter8/buffer_overflow

Breakpoint 1, foo1 () at buffer_overflow.c:4
4      void foo1(){
6      )
```

在第一个断点处查看寄存器 `rsp` 中的值以及该值作为内存地址时内存单元的内容。

```
(gdb) x/x $rsp
0x7fffffffdef8: 0x555546db
```

在第二个断点处查看 `buff` 的地址。

```
(gdb) c
Continuing.

Breakpoint 2, 0x000055555555469c in foo1 () at buffer_overflow.c:7
7          strcpy(buff,Lbuffer);
(gdb) p/x &buff
$1 = 0x7fffffffdee0
```

可以看到，`buff` 的地址和之前查看的寄存器 `esp` 的值差 `0x18`。

我们的 `Lbuffer` 字符串长度为 32 字节，值为 `01234567890123456789=====ABCD`，因此，在使用 `strcpy` 之后，从地址 `0x7fffffffdef8` 处开始，连续的 8 个内存地址的值为 `=====ABCD`。

下面我们看看是否正确。

在第三个断点处，查看接下来的汇编指令。

```
(gdb) c
Continuing.

Breakpoint 3, 0x00005555555546a2 in foo1 () at buffer_overflow.c:9
9      }
(gdb) x/x $rsp
0x7fffffffdeb0: 0x33323130
(gdb) x/i $pc
=> 0x5555555546a2 <foo1+88>:    leaveq
```

`Leaveq` 汇编指令，相当于

```
mov %rbp, %rsp
```

```
pop %rbp
```

因此，执行完 `leaveq` 的指令后，寄存器 `rsp` 指向的值为 `foo1` 函数的

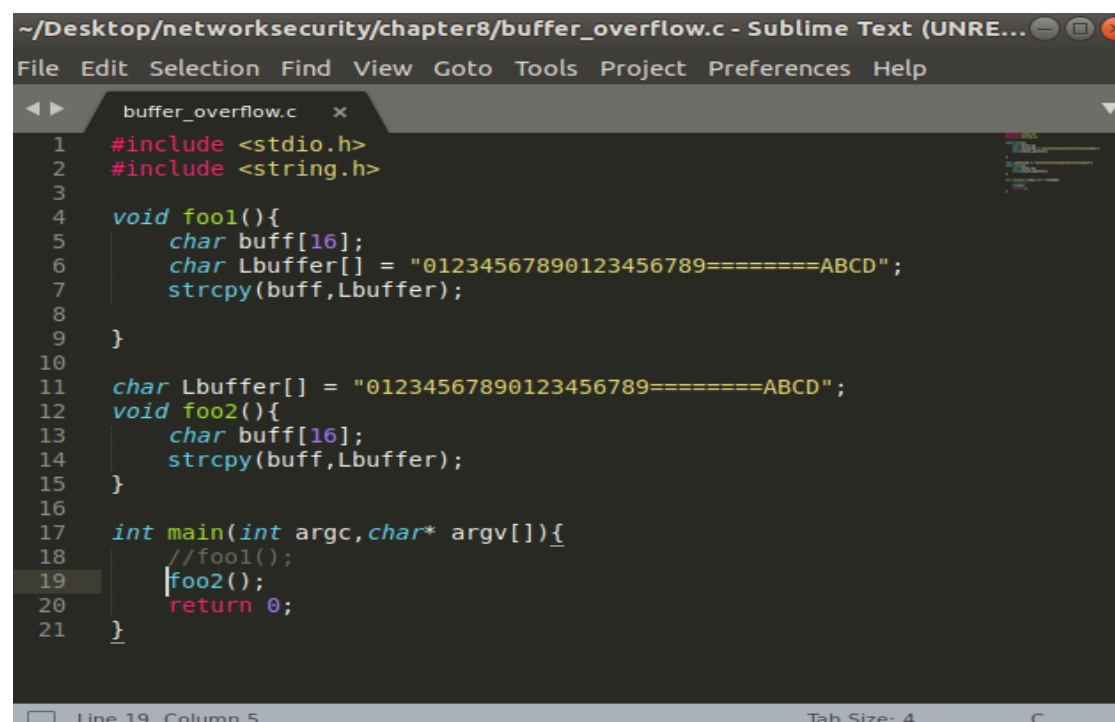
返回地址。

```
(gdb) ni
Breakpoint 4, 0x00005555555546a3 in foo1 () at buffer_overflow.c:9
9      }
(gdb) x/x $rsp
0x7fffffffdef8: 0x3d3d3d3d
(gdb) x/1gx $rsp
0x7fffffffdef8: 0x444342413d3d3d3d
(gdb) x/1gc $rsp
0x7fffffffdef8: 61 '='
(gdb) x/8c $rsp
0x7fffffffdef8: 61 '=' 61 '=' 61 '=' 61 '=' 65 'A' 66 'B' 67 'C' 68 'D'
(gdb) x/c $rsp+8
0x7fffffffdf00: 0 '\000'
(gdb) x/c $rsp+7
0x7fffffffdeff: 68 'D'
(gdb) x/c 0x7fffffffdeff
0x7fffffffdeff: 68 'D'
(gdb)
```

可以看到，现在地址 0x7fffffffdef8 到地址 0x7fffffffdeff 中的内容为
====ABCD。

下面对 foo2 进行分析。

函数如下：



```
~/Desktop/networksecurity/chapter8/buffer_overflow.c - Sublime Text (UNRE...
File Edit Selection Find View Goto Tools Project Preferences Help

buffer_overflow.c x
1  #include <stdio.h>
2  #include <string.h>
3
4  void foo1(){
5      char buff[16];
6      char Lbuffer[] = "01234567890123456789====ABCD";
7      strcpy(buff,Lbuffer);
8
9  }
10
11 char Lbuffer[] = "01234567890123456789====ABCD";
12 void foo2(){
13     char buff[16];
14     strcpy(buff,Lbuffer);
15 }
16
17 int main(int argc,char* argv[]){
18     //foo1();
19     foo2();
20     return 0;
21 }
```

用 gcc 命令进行编译，使用 gdb 命令进行调试。

```
wjm@ubuntu:~/Desktop/networksecurity/chapter8$ gcc -g -fno-stack-protector -z execstack buffer_overflow.c -o buffer_overflow_foo2
wjm@ubuntu:~/Desktop/networksecurity/chapter8$ gdb bu
buffer_overflow      bunzip2          busybox
buffer_overflow_foo2 busctl
wjm@ubuntu:~/Desktop/networksecurity/chapter8$ gdb buffer_overflow_foo2
GNU gdb (Ubuntu 8.1-0ubuntu3.1) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
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:
```

使用 `disas` 命令查看汇编代码（先运行一次）。

```
(gdb) disas main
Dump of assembler code for function main:
   0x00005555555546c2 <+0>:      push    %rbp
   0x00005555555546c3 <+1>:      mov     %rsp,%rbp
   0x00005555555546c6 <+4>:      sub     $0x10,%rsp
   0x00005555555546ca <+8>:      mov     %edi,-0x4(%rbp)
   0x00005555555546cd <+11>:     mov     %rsi,-0x10(%rbp)
   0x00005555555546d1 <+15>:     mov     $0x0,%eax
   0x00005555555546d6 <+20>:     callq   0x5555555546a4 <foo2>
   0x00005555555546db <+25>:     mov     $0x0,%eax
   0x00005555555546e0 <+30>:     leaveq
   0x00005555555546e1 <+31>:     retq
End of assembler dump.
(gdb) disas foo2
```

```
(gdb) disas foo2
Dump of assembler code for function foo2:
   0x00005555555546a4 <+0>:      push    %rbp
   0x00005555555546a5 <+1>:      mov     %rsp,%rbp
   0x00005555555546a8 <+4>:      sub     $0x10,%rsp
   0x00005555555546ac <+8>:      lea     -0x10(%rbp),%rax
   0x00005555555546b0 <+12>:     lea     0x200969(%rip),%rsi      # 0x555555755
020 <Lbuffer>
   0x00005555555546b7 <+19>:     mov     %rax,%rdi
   0x00005555555546ba <+22>:     callq   0x555555554520 <strcpy@plt>
   0x00005555555546bf <+27>:     nop
   0x00005555555546c0 <+28>:     leaveq
=> 0x00005555555546c1 <+29>:     retq
End of assembler dump.
```

设置断点并开始运行。

```
(gdb) b *(foo2+0)
Breakpoint 1 at 0x6a4: file buffer_overflow.c, line 12.
(gdb) b *(foo2+22)
Breakpoint 2 at 0x6ba: file buffer_overflow.c, line 14.
(gdb) b *(foo2+28)
Breakpoint 3 at 0x6c0: file buffer_overflow.c, line 15.
(gdb) b *(foo2+29)
Breakpoint 4 at 0x6c1: file buffer_overflow.c, line 15.
(gdb) r
```

在第一个断点处查看寄存器 `rsp` 中的值以及该值作为内存地址时内存

单元的内容。

```
(gdb) x/x $rsp
0x7fffffffdef8: 0x555546db
(gdb)
```

在第二个断点处查看 buff 的地址。

```
(gdb) c
Continuing.

Breakpoint 2, 0x00005555555546ba in foo2 () at buffer_overflow.c:14
14      strcpy(buff,Lbuffer);
1: x/i $pc
=> 0x5555555546ba <foo2+22>:    callq  0x555555554520 <strcpy@plt>
(gdb) x/x &buff
0x7fffffffdee0: 0xf7de59a0
(gdb)
```

可以看到，buff 的地址和之前查看的寄存器 esp 的值差 0x18。

我们的 Lbuffer 字符串长度为 32 字节，值为 01234567890123456789=====ABCD，因此，在使用 strcpy 之后，从地址 0x7fffffffdef8 处开始，连续的 8 个内存地址的值为 =====ABCD。

下面我们看看是否正确。

在第三个断点处，查看接下来的汇编指令。

```
(gdb) c
Continuing.

Breakpoint 3, 0x00005555555546c0 in foo2 () at buffer_overflow.c:15
15      }
1: x/i $pc
=> 0x5555555546c0 <foo2+28>:    leaveq
(gdb)
```

Leaveq 汇编指令，相当于

```
mov %rbp, %rsp
```

```
pop %rbp
```

因此，执行完 leaveq 的指令后，寄存器 rsp 指向的值为 foo2 函数的返回地址。

```
(gdb) c
Continuing.

Breakpoint 4, 0x000055555555546c1 in foo2 () at buffer_overflow.c:15
15      }
1: x/i $pc
=> 0x55555555546c1 <foo2+29>:    retq
(gdb) x/x $rsp
0x7fffffffdef8: 0x3d3d3d3d
(gdb) x/1gx $rsp
0x7fffffffdef8: 0x444342413d3d3d3d
(gdb) x/8c $rsp
0x7fffffffdef8: 61 '=' 61 '=' 61 '=' 61 '=' 65 'A' 66 'B' 67 'C' 68 'D'
69 '\n' 70 '\n' 71 '\n' 72 '\n'
```

可以看到，现在地址 0x7fffffffdef8 到地址 0x7fffffffdeff 中的内容为
====ABCD。