ICS BOMBLAB REPORT

胡译文 2021201719

PREPERATION

首先获取 objdump 等文件：可以看到一些 ~~没啥用的~~ 基本信息，如小端、UNIX - System V（因此mac跑不了）、主函数地址等。

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x21f0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          81048 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         13
  Size of section headers:           64 (bytes)
  Number of section headers:         37
  Section header string table index: 36
```

symbol table 里包含了一些函数名、变量的信息，但因为并没有按照地址或名称排序，阅读起来并不方便遂放弃。使用反汇编得到 objdump 文件，从主函数开始递归地阅读。首先思考如何防止炸，设置断点是一方面，但太麻烦，遂使用 hex 编辑器修改如下：

```
0000000000002be9 <explode_bomb>:
    2be9:       48 83 ec 08             sub    $0x8,%rsp
    2bed:       c3                      ret
    2bee:       c3                      ret
    2bef:       c3                      ret
    2bf0:       c3                      ret
    2bf1:       c3                      ret
    2bf2:       c3                      ret
    2bf3:       c3                      ret
    2bf4:       e8 77 f4 ff ff          call   2070 <puts@plt>
    2bf9:       48 8d 3d 9f 18 00 00    lea    0x189f(%rip),%rdi        # 449f <array.0+0x2bf>
    2c00:       e8 6b f4 ff ff          call   2070 <puts@plt>
    2c05:       48 8d 3d b0 18 00 00    lea    0x18b0(%rip),%rdi        # 44bc <array.0+0x2dc>
    2c0c:       e8 5f f4 ff ff          call   2070 <puts@plt>
```

为使得程序仍然能正常运行（间接跳转正确），需要保证修改后的指令长度不变。只是手欠 set 了两次寄存器最后还是炸了……（不知道程序究竟是通过什么获取的中间状态）（当试图去 phase_defused 里一探究竟时发现并看不懂，但是发现了 secret_phase 的入口）

PHASE 1

直接使用编辑器打开二进制程序，就可以看到第一题答案被明文编码在里面：

```
^@That's number 2.^@Halfway there!^@Good work!  On to the next...^@^@^@^@^@^@Welcome to my fiendish little bomb. You have 6 phases with^@^
@^@^@^@^@which to blow yourself up. Have a nice day!^@^@^@^@^@Phase 1 defused. (Interesting key, huh?)^@^@^@^@^@^@^@^@So you got that one.
  Try this one.^@^@^@^@^@Saiverd loclken a wethd, lafz fomdra Lay, Iliffzidra.^@^@^@Wow! You've defused the secret stage!^@^@^@^@^@^@^@^@^
@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@^@_ãÿÿ³ãÿÿºãÿÿÁãÿÿÈãÿÿÏãÿÿÖãÿÿÝãÿÿ^B^@^@
```

PHASE 2

将汇编翻译一下：

```
callq  2ca9 <read_six_numbers>
cmpl   $0x2,(%rsp)
jne    2487 <phase_2+0x2e>      # arg1 < 2: bomb
mov    %rsp,%rbx                # %rbx = arg1
lea    0x14(%rsp),%rbp          # %rbp = &arg1 + 0x14
jmp    2497 <phase_2+0x3e>`
callq  2be9 <explode_bomb>
jmp    247d <phase_2+0x24>
add    $0x4,%rbx                # do { arg0 += 4
cmp    %rbp,%rbx                # if (arg0 == %rbp) break
je     24ac <phase_2+0x53>
mov    0x4c83(%rip),%eax        # 7120 <mul.1>
imul   (%rbx),%eax              # M[%rbx] *= arg0
cmp    %eax,0x4(%rbx)
je     248e <phase_2+0x35>      # } while (ret == M[%rbx])
callq  2be9 <explode_bomb>
```

可以看到先乘 4，再乘 8，再乘 12，找规律可得答案。

PHASE 3

```
(gdb) x/s 93824992249345
0x555555558601: "%d %d"
```

首先可以看到输入是两个整数，且第一个整数（跳表）要求在 0~5 之间、第二个整数为负数。不断尝试第一个整数，可以发现仅有 5 不会必然导致炸弹。最后发现第二个整数在整个函数运行的过程中不会改变，直接观察最后比较时的答案，推定第二个整数的值。

```
-7327, -7243, -7236, -7229, -7232, -7215, -7208, -7201


['0x55555555651f',  # 0 bomb
 '0x555555556573',  # 1 bomb
 '0x55555555657a',  # 2 bomb
 '0x555555556581',  # 3 bomb
 '0x55555555657e',  # 4 bomb
 '0x55555555658f',  # 5 ok
 '0x555555556596',
 '0x55555555659d']
```

```python
import pipe as p

for hexnum in "0xffffe3d6   0xffffe3dd".split():
    x = int(''.join(['0' if i == '1' else '1' for i in bin(int(hexnum, 16))[2:]] | p.skip_while(lambda x: x == '0')), 2)
    print(-x)
```

PHASE 4

首先第一个和第二个整数是反着的…… 第二个整数是 4 在 phase_4 主体部分可以看出。递归部分调用 func4 ，不断调整寄存器：

```
(gdb) i r edi esi
edi            0x7  7
esi            0x4  4
(gdb) i r edi esi
edi            0x6  6
esi            0x5  5
(gdb) i r edi esi
edi            0x5  5
esi            0x6  6
(gdb) i r edi esi
edi            0x4  4
esi            0x7  7
……
```

可以计算出第一个整数是 246 。

PHASE 5

可以看到直接拿字符串里的值作为索引。有一个数字存了很多整数，在前 16 个里挑选 6 个使得求和等于 0x23 。最后分别是 0、1、2、3、6、9 ，顺序无所谓。不难。

PHASE 6

phase_6 好长，总之是读入一个 1~6 的排列，根据这个排列会取到一个值和一个指针，使得指针指向节点的值等于得到的值即可。发现每次输入的排列与得到的指针有一定规律，再加上有一个双层循环合理猜测是对六个节点的书进行排序。最终各个节点的序号是 3 5 1 4 6 2 。

```
(gdb) i r eax
eax            0x33b    827
(gdb) x/24d $rbx
0x555555565360 <node1>: 551     1       1431720816      21845
0x555555565370 <node2>: 827     2       1431720832      21845
0x555555565380 <node3>: 62      3       1431720848      21845
0x555555565390 <node4>: 654     4       1431720864      21845
0x5555555653a0 <node5>: 243     5       1431685792      21845
0x5555555653b0: 0       0       0       0
```

```
(gdb) i r eax
eax            0x33b    827
(gdb) x/24d $rbx
0x555555565370 <node2>: 827     2       0       0
0x555555565380 <node3>: 62      3       1431720864      21845
0x555555565390 <node4>: 654     4       1431685792      21845
0x5555555653a0 <node5>: 243     5       1431720800      21845
0x5555555653b0: 0       0       0       0
0x5555555653c0 <host_table>:    1431668316      21845   1431668321      21845
```

可以看到一个特殊的数字 21845 反复地出现，其实是 0x5555 ，与所有运行时地址前缀相同，也就是说与前一个 byte 拼成一个指针。

SECRET_PHASE

来到了 secret_phase，从 phase_defused 的入口里处分析，打印值以后发现是在第三阶段后读入了其他的字符串。依照国际惯例打印出来即可。~~看着好短~~。一通分析发现 fun7 调用的是一颗字典树。使用 gdb logging 功能输出（血的教训，一定要输出完整），从叶节点回到根节点即可。

```
0x55555555cb80 <t0+160>:    1431686080  21845   0   0  # a + 19 = t
0x55555555cbe0 <t69+32>:    0   0   1431686304  21845  # e
0x55555555cd00 <t70+96>:    0   0   1431686528  21845  # a + 12 = m
0x55555555ce00 <t71+128>:   1431686752  21845   0   0  # a + 15 = p
0x55555555ce80 <t72+32>:    0   0   1431687872  21845  # e
0x55555555d350 <t124+144>:  0   0   1431688096  21845  # a + 9*2 = s
0x55555555d440 <t125+160>:  1431679520  21845   0   0  # a + 19 = t
```

一些简单的小函数能显著提高效率。

```
def con(num):
    return num - 0x555500000000
```

RESULTS

汇总：

```
Saiverd loclken a wethd, lafz fomdra Lay, Iliffzidra.
2 8 32 128 512 2048
5 -717
246 4 Testify
012369
3 5 1 4 6 2
tempest
```