Lab2 报告

- 姓名：邵震哲
- 班级：1620204
- 学号：162020130
- 报告阶段：lab2
- 完成日期：2022.5.18
- 本次实验，隐藏关未完成，其他均完成。

目录

1. phase_1

- 思路

```
00001662 <phase_1>:
    1662:   f3 0f 1e fb             endbr32
    1666:   55                      push   %ebp
    1667:   89 e5                   mov    %esp,%ebp
    1669:   53                      push   %ebx
    166a:   83 ec 0c                sub    $0xc,%esp
    166d:   e8 5e fd ff ff          call   13d0 <__x86.get_pc_thunk.bx>
    1672:   81 c3 f2 38 00 00       add    $0x38f2,%ebx
    1678:   8d 83 e0 e1 ff ff       lea    -0x1e20(%ebx),%eax    //把这个地址传
给eax
    167e:   50                      push   %eax
    167f:   ff 75 08                pushl  0x8(%ebp)     //准备参数给
strings_not_equal
    1682:   e8 a5 05 00 00          call   1c2c <strings_not_equal>
    1687:   83 c4 10                add    $0x10,%esp
    168a:   85 c0                   test   %eax,%eax     //结果需要为1，即输入和
密码相同
    168c:   75 05                   jne    1693 <phase_1+0x31>
    168e:   8b 5d fc                mov    -0x4(%ebp),%ebx
    1691:   c9                      leave
    1692:   c3                      ret
    1693:   e8 19 08 00 00          call   1eb1 <explode_bomb>
    1698:   eb f4                   jmp    168e <phase_1+0x2c>
```

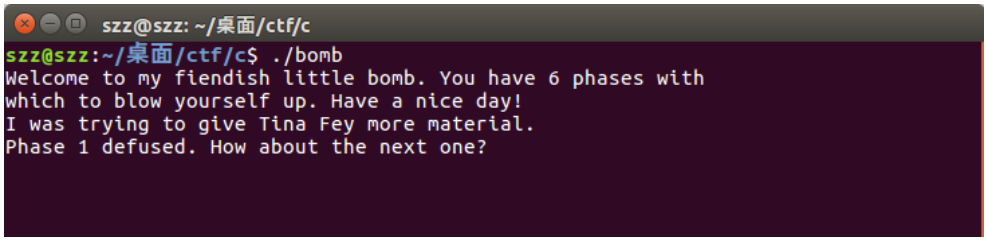
`1678:  lea    -0x1e20(%ebx),%eax`

把地址传给了eax，接下来把eax压栈给strings_not_equal准备参数，那么此时eax寄存器里的就是拆解密码的地址

在phase_1下断点，ni运行到`push %eax`这一步，p查看eax的值，再查看对应地址的内容即可

`I was trying to give Tina Fey more material.`

- 完成截图



2. phase_2

- 思路

```
0000169a <phase_2>:
    169a:   f3 0f 1e fb             endbr32
    169e:   55                      push   %ebp
    169f:   89 e5                   mov    %esp,%ebp
    16a1:   57                      push   %edi
    16a2:   56                      push   %esi
    16a3:   53                      push   %ebx
    16a4:   83 ec 34                sub    $0x34,%esp
    16a7:   e8 24 fd ff ff          call   13d0 <__x86.get_pc_thunk.bx>
    16ac:   81 c3 b8 38 00 00       add    $0x38b8,%ebx
    16b2:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
    16b8:   89 45 e4                mov    %eax,-0x1c(%ebp)
    16bb:   31 c0                   xor    %eax,%eax
    16bd:   8d 45 cc                lea    -0x34(%ebp),%eax
    16c0:   50                      push   %eax
    16c1:   ff 75 08                pushl  0x8(%ebp)
    16c4:   e8 3e 08 00 00          call   1f07 <read_six_numbers>
    16c9:   83 c4 10                add    $0x10,%esp
    16cc:   83 7d cc 01             cmpl   $0x1,-0x34(%ebp)
    16d0:   75 08                   jne    16da <phase_2+0x40>
```

```
    16d2:   8d 75 cc                lea    -0x34(%ebp),%esi
    16d5:   8d 7d e0                lea    -0x20(%ebp),%edi
    16d8:   eb 13                   jmp    16ed <phase_2+0x53>
    16da:   e8 d2 07 00 00          call   1eb1 <explode_bomb>
    16df:   eb f1                   jmp    16d2 <phase_2+0x38>
    16e1:   e8 cb 07 00 00          call   1eb1 <explode_bomb>
    16e6:   83 c6 04                add    $0x4,%esi
    16e9:   39 fe                   cmp    %edi,%esi
    16eb:   74 0b                   je     16f8 <phase_2+0x5e>
    16ed:   8b 06                   mov    (%esi),%eax
    16ef:   01 c0                   add    %eax,%eax
    16f1:   39 46 04                cmp    %eax,0x4(%esi)
    16f4:   74 f0                   je     16e6 <phase_2+0x4c>
    16f6:   eb e9                   jmp    16e1 <phase_2+0x47>
    16f8:   8b 45 e4                mov    -0x1c(%ebp),%eax
    16fb:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
    1702:   75 08                   jne    170c <phase_2+0x72>
    1704:   8d 65 f4                lea    -0xc(%ebp),%esp
    1707:   5b                      pop    %ebx
    1708:   5e                      pop    %esi
    1709:   5f                      pop    %edi
    170a:   5d                      pop    %ebp
    170b:   c3                      ret
    170c:   e8 3f 17 00 00          call   2e50 <__stack_chk_fail_local>
```

首先看到 `call   1f07 <read_six_numbers>` 读取了六个数字

```
    16cc:   83 7d cc 01             cmpl   $0x1,-0x34(%ebp)
    16d0:   75 08                   jne    16da <phase_2+0x40>
```

先和1比较，不相同就爆炸。

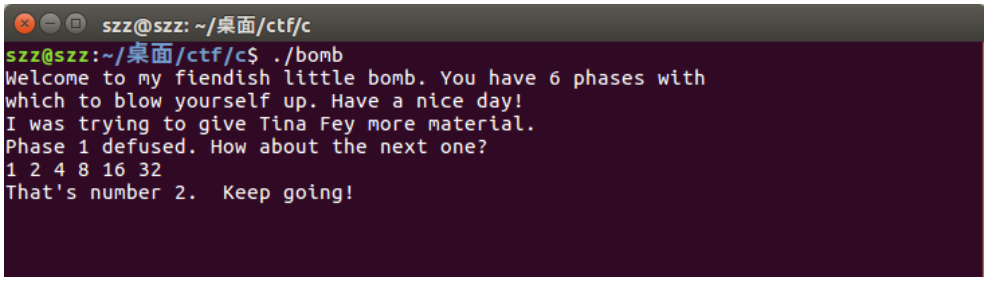
```
    16cc:   83 7d cc 01             cmpl   $0x1,-0x34(%ebp)
    16d0:   75 08                   jne    16da <phase_2+0x40>
    16d2:   8d 75 cc                lea    -0x34(%ebp),%esi
    16d5:   8d 7d e0                lea    -0x20(%ebp),%edi
    16d8:   eb 13                   jmp    16ed <phase_2+0x53>

    16e6:   83 c6 04                add    $0x4,%esi    //到下一个数
    16e9:   39 fe                   cmp    %edi,%esi

    16ed:   8b 06                   mov    (%esi),%eax
    16ef:   01 c0                   add    %eax,%eax    //自己加自己
    16f1:   39 46 04                cmp    %eax,0x4(%esi)   //和下一个数比较，
需要相同
    16f4:   74 f0                   je     16e6 <phase_2+0x4c>
    16f6:   eb e9                   jmp    16e1 <phase_2+0x47>
```

然后跳转到16ed，esi自己加自己，即乘2，和esi+4(即下一个数)比较，比完了esi+4，对后一个数进行操作。

也就是后一个数是前一个数的两倍

那么密码就是 `1 2 4 8 16 32`

- 完成截图



3. phase_3

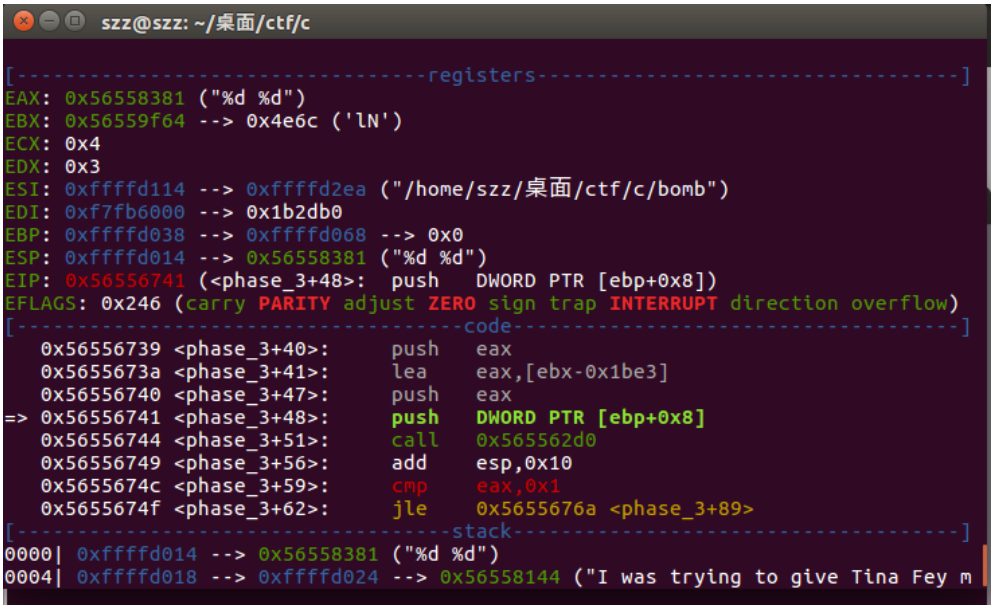
- 思路

```
00001711 <phase_3>:
    1711:   f3 0f 1e fb             endbr32
    1715:   55                      push   %ebp
    1716:   89 e5                   mov    %esp,%ebp
    1718:   53                      push   %ebx
    1719:   83 ec 14                sub    $0x14,%esp
    171c:   e8 af fc ff ff          call   13d0 <__x86.get_pc_thunk.bx>
    1721:   81 c3 43 38 00 00       add    $0x3843,%ebx
    1727:   65 a1 14 00 00 00       mov    %gs:0x14,%eax
    172d:   89 45 f4                mov    %eax,-0xc(%ebp)
    1730:   31 c0                   xor    %eax,%eax
    1732:   8d 45 f0                lea    -0x10(%ebp),%eax
    1735:   50                      push   %eax
    1736:   8d 45 ec                lea    -0x14(%ebp),%eax
    1739:   50                      push   %eax
    173a:   8d 83 1d e4 ff ff       lea    -0x1be3(%ebx),%eax
    1740:   50                      push   %eax
    1741:   ff 75 08                pushl  0x8(%ebp)
    1744:   e8 87 fb ff ff          call   12d0 <__isoc99_sscanf@plt>
    1749:   83 c4 10                add    $0x10,%esp
    174c:   83 f8 01                cmp    $0x1,%eax
    174f:   7e 19                   jle    176a <phase_3+0x59>
    1751:   83 7d ec 07             cmpl   $0x7,-0x14(%ebp)
    1755:   0f 87 90 00 00 00       ja     17eb <.L38+0x7>
    175b:   8b 45 ec                mov    -0x14(%ebp),%eax
    175e:   89 da                   mov    %ebx,%edx
    1760:   03 94 83 40 e2 ff ff    add    -0x1dc0(%ebx,%eax,4),%edx
    1767:   3e ff e2                notrack jmp *%edx
    176a:   e8 42 07 00 00          call   1eb1 <explode_bomb>
    176f:   eb e0                   jmp    1751 <phase_3+0x40>
```

看到读取函数

```
    1744:   e8 87 fb ff ff          call   12d0 <__isoc99_sscanf@plt>
```

运行到这一步时，gdb调试看到准备的参数是 `(%d %d)`，可知读取的是两个数字




```
    174c:   83 f8 01                cmp    $0x1,%eax
    174f:   7e 19                   jle    176a <phase_3+0x59>
    1751:   83 7d ec 07             cmpl   $0x7,-0x14(%ebp)
    1755:   0f 87 90 00 00 00       ja     17eb <.L38+0x7>

    17eb:   e8 c1 06 00 00          call   1eb1 <explode_bomb>
```

这一步eax存的是读取出来的数字个数，和1比较跳转，小于等于1就爆炸。再把第一个参数和7比较，如果大于7就爆炸

后面根据eax的值（即`-0x14(%ebp)`，输入的第一个数）来计算跳转位置，很明显是个switch

```
    175b:   8b 45 ec                mov    -0x14(%ebp),%eax
    175e:   89 da                   mov    %ebx,%edx
    1760:   03 94 83 40 e2 ff ff    add    -0x1dc0(%ebx,%eax,4),%edx
    1767:   3e ff e2                notrack jmp *%edx
```

然而无论怎么跳，最后都要到L27 ret

```
00001771 <.L27>:
    1771:   b8 b6 03 00 00          mov    $0x3b6,%eax
    1776:   2d ea 01 00 00          sub    $0x1ea,%eax
    177b:   05 1a 01 00 00          add    $0x11a,%eax
    1780:   2d 27 02 00 00          sub    $0x227,%eax
    1785:   05 27 02 00 00          add    $0x227,%eax
    178a:   2d 27 02 00 00          sub    $0x227,%eax
    178f:   05 27 02 00 00          add    $0x227,%eax
    1794:   2d 27 02 00 00          sub    $0x227,%eax   //一系列操作，不用管，最后看eax里的值即可

    1799:   83 7d ec 05             cmpl   $0x5,-0x14(%ebp)
    179d:   7f 05                   jg     17a4 <.L27+0x33>
    179f:   39 45 f0                cmp    %eax,-0x10(%ebp) //最后把eax和第二个参数比较，相同就成功
    17a2:   74 05                   je     17a9 <.L27+0x38>
    17a4:   e8 08 07 00 00          call   1eb1 <explode_bomb>

    17a9:   8b 45 f4                mov    -0xc(%ebp),%eax
    17ac:   65 33 05 14 00 00 00    xor    %gs:0x14,%eax
```

```
    17b3:   75 42                   jne    17f7 <.L38+0x13>
    17b5:   8b 5d fc                mov    -0x4(%ebp),%ebx
    17b8:   c9                      leave
    17b9:   c3                      ret
```

从L27看出来，把第一个数和0x5进行比较，如果大于5，就爆炸，因此第一个数不能大于5

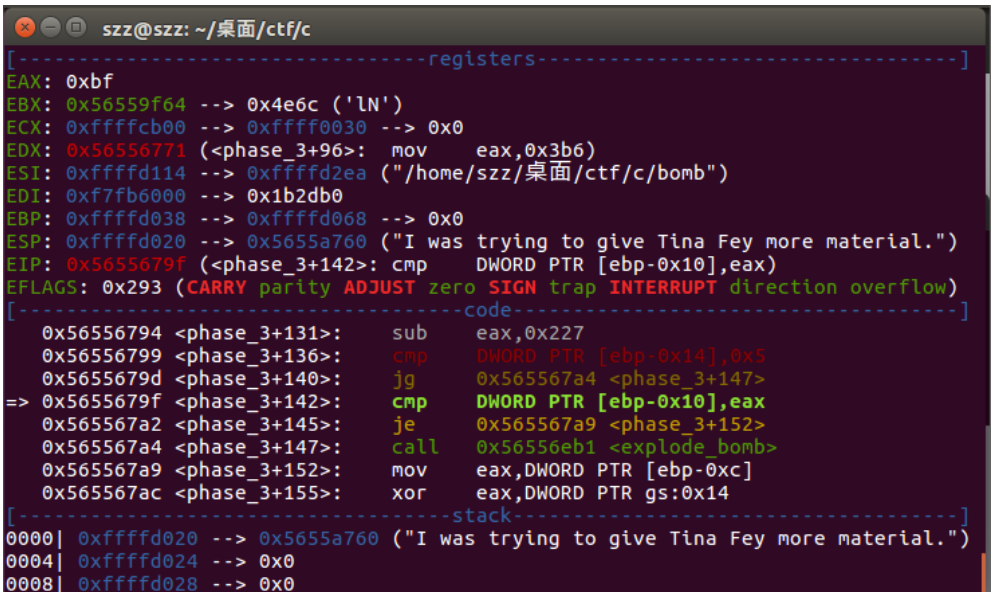
```
    1799:   83 7d ec 05             cmpl   $0x5,-0x14(%ebp)
    179d:   7f 05                   jg     17a4 <.L27+0x33>
    179f:   39 45 f0                cmp    %eax,-0x10(%ebp)
    17a2:   74 05                   je     17a9 <.L27+0x38>
    17a4:   e8 08 07 00 00          call   1eb1 <explode_bomb>
    17a9:   8b 45 f4                mov    -0xc(%ebp),%eax
```

可以先第一个数为0，则先跳转到L32

```
000017ba <.L32>:
    17ba:   b8 00 00 00 00          mov    $0x0,%eax
    17bf:   eb b5                   jmp    1776 <.L27+0x5>
```

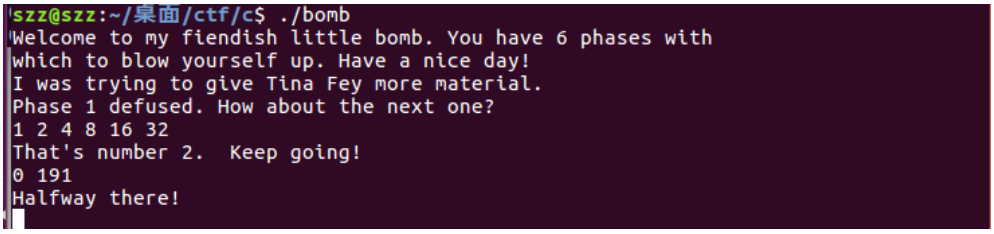
然后从L32跳到L27

经过一系列操作后，最后的`cmp %eax,-0x10(%ebp)`，`eax`的值为`0xbf`，因此输入的第二个数为191即可



拆解密码就是0 191

- 完成截图



4. phase_4

- 思路

```
    1885:   50                      push   %eax
    1886:   ff 75 08                pushl  0x8(%ebp)
    1889:   e8 42 fa ff ff          call   12d0 <__isoc99_sscanf@plt>
```

开头和`phase_3`一样，读取两个数字

```
    1896:   83 7d ec 0e             cmpl   $0xe,-0x14(%ebp)
    189a:   76 05                   jbe    18a1 <phase_4+0x4b>
    189c:   e8 10 06 00 00          call   1eb1 <explode_bomb>
```

首先第一个数要小于等于14

```
    18a4:   6a 0e                   push   $0xe
    18a6:   6a 00                   push   $0x0
    18a8:   ff 75 ec                pushl  -0x14(%ebp)
    18ab:   e8 4c ff ff ff          call   17fc <func4>
```

这里把（第一个数，0，14）作为参数调用`func4(-0x14(%ebp), 0, 0xe)`

```
    18ab:   e8 4c ff ff ff          call   17fc <func4>
    18b0:   83 c4 10                add    $0x10,%esp
    18b3:   83 f8 0f                cmp    $0xf,%eax
    18b6:   75 06                   jne    18be <phase_4+0x68>
    18b8:   83 7d f0 0f             cmpl   $0xf,-0x10(%ebp)
    18bc:   74 05                   je     18c3 <phase_4+0x6d>
    18be:   e8 ee 05 00 00          call   1eb1 <explode_bomb>
```

调用后得到的值要等于0xf，同时输入的第二个数也需要等于0xf，即15

由于第一个数小于14，可以爆破得到答案5

也可以通过分析func4函数来得到答案，把func4函数尝试反汇编得到

```
int func4(int a1, int a2, int a3)//刚开始 a1=-0x14(%ebp),a2=0,a3=14
{
  int v3; // ebx
  v3 = (a3 - a2)/2 + a2;
  if ( v3 > a1 )
  {
    v3 += func4(a1, a2, v3 - 1);
  }
  else if ( v3 < a1 )
  {
    v3 += func4(a1, v3 + 1, a3);
  }
  return v3;
}
```

写程序验证， 输入5时返回15

- 完成截图

```
szz@szz: ~/桌面/ctf/c
szz@szz:~/桌面/ctf/c$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
0 191
Halfway there!
5 15
So you got that one.  Try this one.
```

5. phase_5

- 思路

```
    18fe:   56                      push   %esi
    18ff:   e8 02 03 00 00          call   1c06 <string_length>
    1904:   83 c4 10                add    $0x10,%esp
    1907:   83 f8 06                cmp    $0x6,%eax
    190a:   75 53                   jne    195f <phase_5+0x86>
```

esi为输入的数组的地址，要满足string_length为6

```
    1917:   0f b6 14 06             movzbl (%esi,%eax,1),%edx   //从esi地址取出对应eax位置的字符放到edx
    191b:   83 e2 0f                and    $0xf,%edx            //与操作
    191e:   0f b6 14 11             movzbl (%ecx,%edx,1),%edx   //去ecx指向的地址中edx位置取字符
    1922:   88 54 05 ed             mov    %dl,-0x13(%ebp,%eax,1)   //保存到ebp-0x13开头的对应的eax地址
    1926:   83 c0 01                add    $0x1,%eax
    1929:   83 f8 06                cmp    $0x6,%eax
    192c:   75 e9                   jne    1917 <phase_5+0x3e>
```

这是循环体。

eax小于6时，从esi地址，即输入的字符串，取出对应eax位置的字符

和0xf与操作后，去ecx指向的地址中取字符，保存到ebp-0x13开头的对应的地址

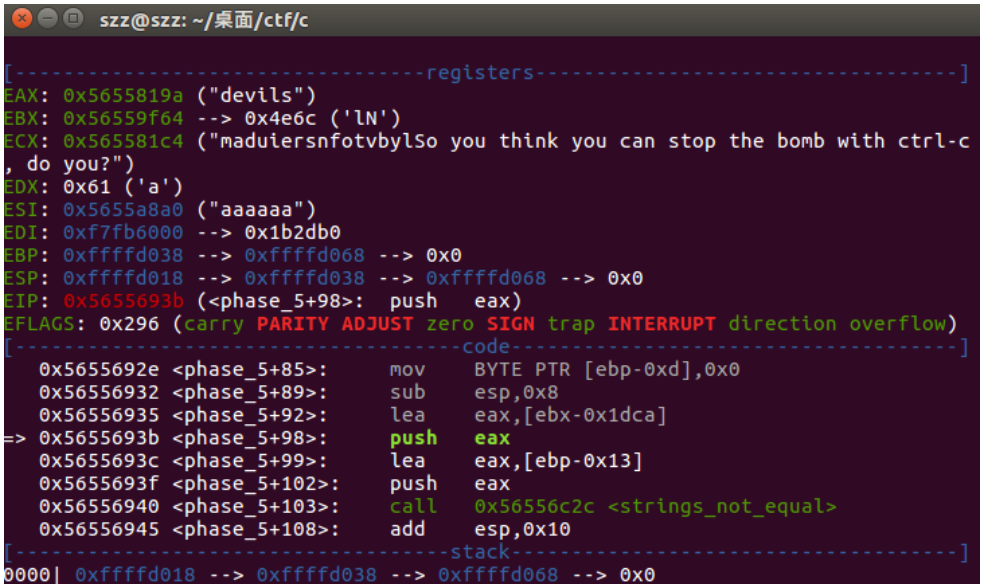
```
szz@szz: ~/桌面/ctf/c

[----------------------------------registers-----------------------------------]
EAX: 0x5655819a ("devils")
EBX: 0x56559f64 --> 0x4e6c ('lN')
ECX: 0x565581c4 ("maduiersnfotvbylSo you think you can stop the bomb with ctrl-c
, do you?")
EDX: 0x61 ('a')
ESI: 0x5655a8a0 ("aaaaaa")
EDI: 0xf7fb6000 --> 0x1b2db0
EBP: 0xffffd038 --> 0xffffd068 --> 0x0
ESP: 0xffffd018 --> 0xffffd038 --> 0xffffd068 --> 0x0
EIP: 0x5655693b (<phase_5+98>:  push   eax)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x5655692e <phase_5+85>:     mov    BYTE PTR [ebp-0xd],0x0
   0x56556932 <phase_5+89>:     sub    esp,0x8
   0x56556935 <phase_5+92>:     lea    eax,[ebx-0x1dca]
=> 0x5655693b <phase_5+98>:     push   eax
   0x5655693c <phase_5+99>:     lea    eax,[ebp-0x13]
   0x5655693f <phase_5+102>:    push   eax
   0x56556940 <phase_5+103>:    call   0x56556c2c <strings_not_equal>
   0x56556945 <phase_5+108>:    add    esp,0x10
[------------------------------------stack-------------------------------------]
0000| 0xffffd018 --> 0xffffd038 --> 0xffffd068 --> 0x0
```

ecx指向的：`maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?`

```
    1935:   8d 83 36 e2 ff ff       lea    -0x1dca(%ebx),%eax
    193b:   50                      push   %eax
    193c:   8d 45 ed                lea    -0x13(%ebp),%eax
    193f:   50                      push   %eax
    1940:   e8 e7 02 00 00          call   1c2c <strings_not_equal>
```

最后传入两个参数给strings_not_equal，判断相等

在动态调试中，到这一步时，去看eax的值即可知道要对比的字符串是什么



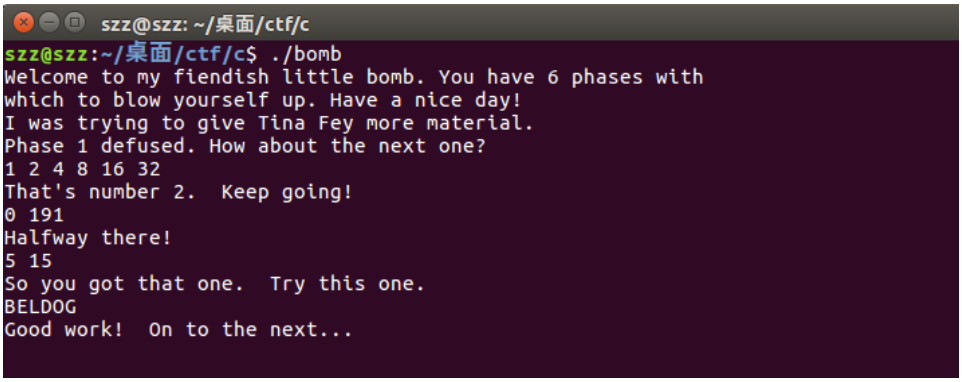
可知是`devils`

因此按表反向查找即可，最后给出对应末尾的可见字符即可

以下是python解题脚本

```python
tables = "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
answer = "devils"
flag = ""
for i in answer:
    index = tables.index(i)
  flag += chr(64 + index )  #64为 0100 0000
print(flag)
#BELDOG
```

- 完成截图



6. phase_6

- 思路

```
    1998:   50                      push   %eax
    1999:   ff 75 08                pushl  0x8(%ebp)
    199c:   e8 66 05 00 00          call   1f07 <read_six_numbers>
```

先看到读取6个数字

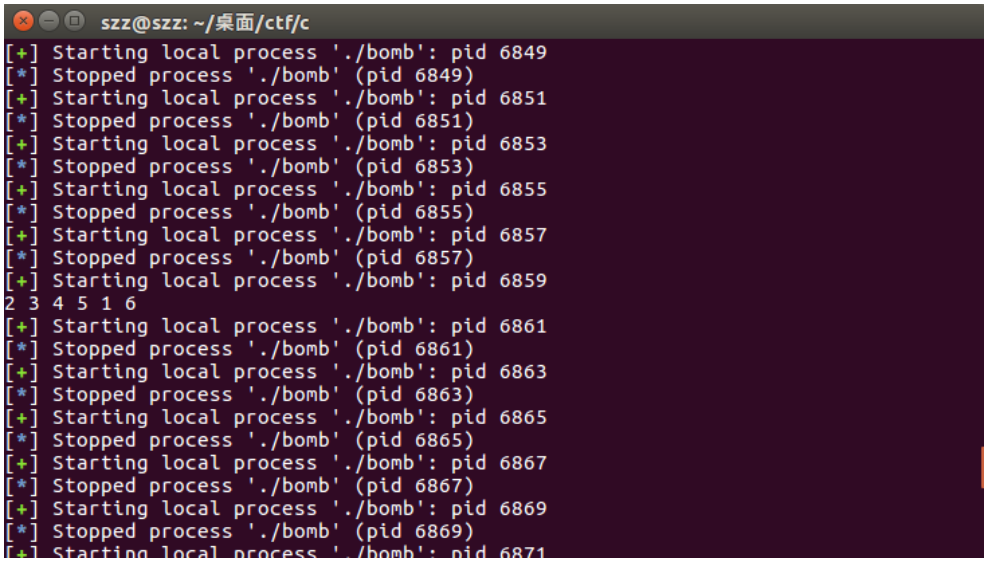
```
    19c0:   83 c6 04                add    $0x4,%esi    //esi存放的是循环中的当
前值，+4到循环的下一个数
    19c3:   39 75 a4                cmp    %esi,-0x5c(%ebp)
    19c6:   74 0e                   je     19d6 <phase_6+0x64>

    19c8:   8b 06                   mov    (%esi),%eax  //对每个数进行检查相同的
循环
    19ca:   39 47 fc                cmp    %eax,-0x4(%edi)
    19cd:   75 f1                   jne    19c0 <phase_6+0x4e>
    19cf:   e8 dd 04 00 00          call   1eb1 <explode_bomb>
    19d4:   eb ea                   jmp    19c0 <phase_6+0x4e>
    19d6:   83 45 9c 04             addl   $0x4,-0x64(%ebp)
    19da:   8b 45 9c                mov    -0x64(%ebp),%eax
    19dd:   89 c7                   mov    %eax,%edi
    19df:   8b 40 fc                mov    -0x4(%eax),%eax
    19e2:   89 45 98                mov    %eax,-0x68(%ebp)
    19e5:   83 e8 01                sub    $0x1,%eax
    19e8:   83 f8 05                cmp    $0x5,%eax    //减1后和5比较，大于5就
爆炸

    19eb:   77 cc                   ja     19b9 <phase_6+0x47>
    19ed:   83 45 a0 01             addl   $0x1,-0x60(%ebp) //循环的计数+1
    19f1:   8b 45 a0                mov    -0x60(%ebp),%eax
    19f4:   83 f8 05                cmp    $0x5,%eax        //和5比较，大于就退
出检验循环
    19f7:   7f 05                   jg     19fe <phase_6+0x8c>
    19f9:   8b 75 9c                mov    -0x64(%ebp),%esi
    19fc:   eb ca                   jmp    19c8 <phase_6+0x56>  //这里进入对每
个数进行检查相同的循环

    19fe:   be 00 00 00 00          mov    $0x0,%esi    //退出检验循环的地址
    1a03:   89 f7                   mov    %esi,%edi
    1a05:   8b 4c b5 b4             mov    -0x4c(%ebp,%esi,4),%ecx
    1a09:   b8 01 00 00 00          mov    $0x1,%eax
    1a0e:   8d 93 94 05 00 00       lea    0x594(%ebx),%edx
    1a14:   83 f9 01                cmp    $0x1,%ecx
    1a17:   7e 0a                   jle    1a23 <phase_6+0xb1>
    1a19:   8b 52 08                mov    0x8(%edx),%edx
    1a1c:   83 c0 01                add    $0x1,%eax
    1a1f:   39 c8                   cmp    %ecx,%eax
    1a21:   75 f6                   jne    1a19 <phase_6+0xa7>
    1a23:   89 54 bd cc             mov    %edx,-0x34(%ebp,%edi,4)
    1a27:   83 c6 01                add    $0x1,%esi
    1a2a:   83 fe 06                cmp    $0x6,%esi
    1a2d:   75 d4                   jne    1a03 <phase_6+0x91>
```

这一段循环检查输入的六个数字是否有相同的。同时要满足每个数字-1都不能大于5，即每个数字都小于等于6。

因此这里可以取巧，六个数 1 2 3 4 5 6 排列组合暴力穷举出答案

第一种解题方法，暴力破解，以下是python解题脚本

```
from pwn import *
from itertools import permutations

perm = permutations("123456")   # 123456的排列组合

for i in perm:
    r=process('./bomb')

    r.recvuntil('Have a nice day!\n')
    r.sendline('I was trying to give Tina Fey more material.')

    r.recvline()
    r.sendline('1 2 4 8 16 32')

    r.recvline()
    r.sendline('0 191')

    r.recvline()
    r.sendline('5 15')

    r.recvline()
    r.sendline('BELDOG')

    r.recvline()
    s = ' '.join(i)
    r.sendline(s)
    answer = r.recvline()
    if(answer=='\n'):   #如果炸了，就关闭
        r.close()
    else:               #没炸说明是正确答案
        print(s)

#2 3 4 5 1 6
```



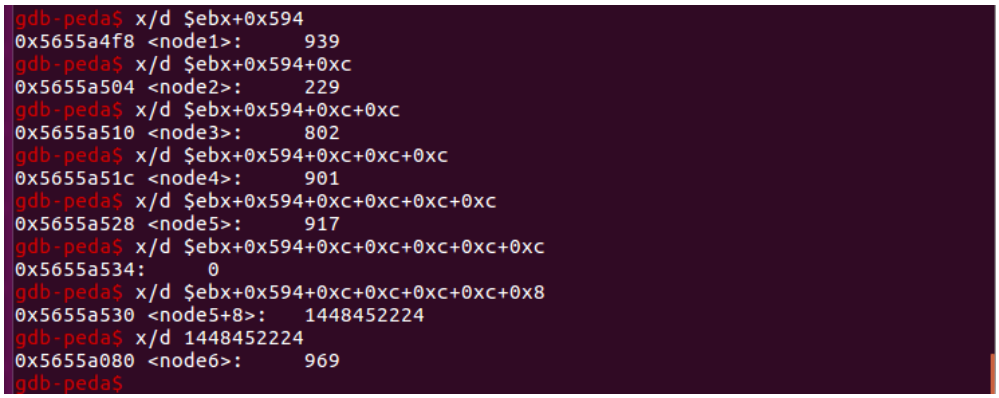
第二种方法，继续分析接下来的逻辑

```
    19fe:   be 00 00 00 00          mov    $0x0,%esi                //退出检验
循环的地址
    1a03:   89 f7                   mov    %esi,%edi
    1a05:   8b 4c b5 b4             mov    -0x4c(%ebp,%esi,4),%ecx
//ecx=输入的当前数
    1a09:   b8 01 00 00 00          mov    $0x1,%eax
    1a0e:   8d 93 94 05 00 00       lea    0x594(%ebx),%edx
//edx是链表头的地址
    1a14:   83 f9 01                cmp    $0x1,%ecx

    1a17:   7e 0a                   jle    1a23 <phase_6+0xb1>
    1a19:   8b 52 08                mov    0x8(%edx),%edx           //赋值下个
节点的地址
    1a1c:   83 c0 01                add    $0x1,%eax
    1a1f:   39 c8                   cmp    %ecx,%eax                //edx移动
到链表对应的ecx位置节点
    1a21:   75 f6                   jne    1a19 <phase_6+0xa7>

    1a23:   89 54 bd cc             mov    %edx,-0x34(%ebp,%edi,4)  //把节点地
址存到后面这个地址
    1a27:   83 c6 01                add    $0x1,%esi
    1a2a:   83 fe 06                cmp    $0x6,%esi
    1a2d:   75 d4                   jne    1a03 <phase_6+0x91>
```

`-0x34(%ebp)` 这个地址按输入的顺序，存放节点的地址

后面的代码就是读取这个地址上的节点地址，并检查节点的值是否是按顺序排列

因此可以去看链表头 `0x594(%ebx)` 及之后一系列结点的值，然后按大小判断输入顺序



因此节点的值分别是

```
1   939
2   229
3   802
4   901
5   917
6   969
```

按顺序排，答案是 `2 3 4 5 1 6`

- 完成截图

```
szz@szz: ~/桌面/ctf/c
szz@szz:~/桌面/ctf/c$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
0 191
Halfway there!
5 15
So you got that one.  Try this one.
BELDOG
Good work!  On to the next...
2 3 4 5 1 6
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
szz@szz:~/桌面/ctf/c$
```

7. 最终结果

- bomblab 完成截图

```
szz@szz: ~/桌面/ctf/c
szz@szz:~/桌面/ctf/c$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
0 191
Halfway there!
5 15
So you got that one.  Try this one.
BELDOG
Good work!  On to the next...
2 3 4 5 1 6
Congratulations! You've defused the bomb!
Your instructor has been notified and will verify your solution.
szz@szz:~/桌面/ctf/c$
```

- （可选）bomblab 隐藏关卡

  未完成

8. 备注

使用延期机会