哈爾濱工業大學

实验报告

实 验（三）

题　　目 Binary Bomb

二进制炸弹

专　　业 计算学部

学　　号 120L020701

班　　级 2003005

学　　生 董琦

指 导 教 师 吴锐

实 验 地 点 宿舍

实 验 日 期 2022/4/1

计算学部

目 录

第 1 章 实验基本信息

1.1 实验目的

1.熟练掌握计算机系统的 ISA 指令系统与寻址方式

2.熟练掌握 Linux 下调试器的反汇编调试跟踪分析机器语言的方法

3．增强对程序机器级表示、汇编语言、调试器和逆向工程等的理解

1.2 实验环境与工具

1.2.1 硬件环境

处理器　Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz　2.11 GHz　8 核
机带 RAM　16.0 GB (15.8 GB 可用
系统类型 64 位操作系统, 基于 x64 的处理器

1.2.2 软件环境

Windows:
版本　Windows 11 家庭中文版
版本　21H2
安装日期　2022/2/4
操作系统版本　22000.556
体验　Windows 功能体验包 1000.22000.556.0

Ubuntu:
版本　Ubuntu 20.04.3 LTS
类型　64 位

1.2.3 开发工具

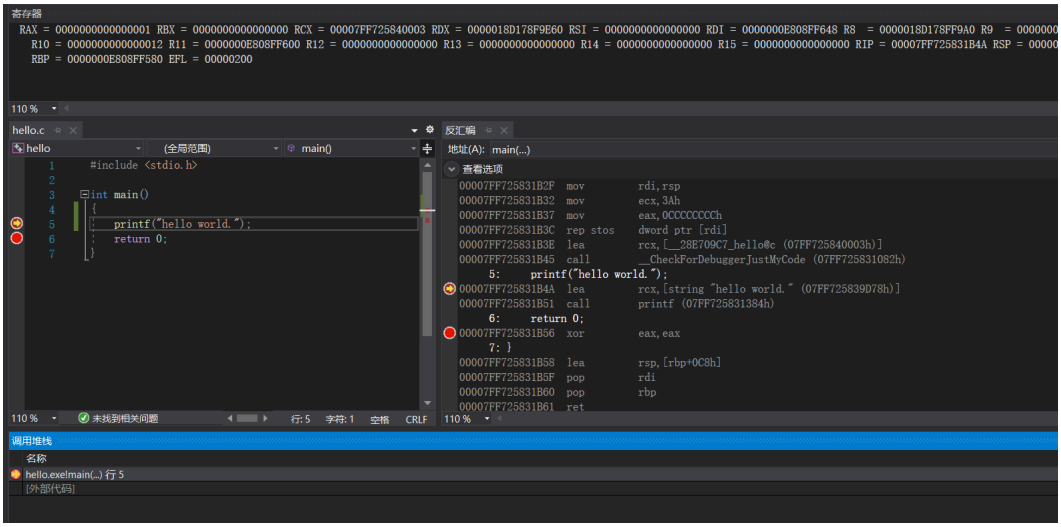
Windows: Visual Studio 2019
Ubuntu: VScode　gcc version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04)，edb,gdb

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 反汇编（10 分）

CodeBlocks 运行 hello.c。反汇编查看 printf 函数的实现。

要求：C、ASM、内存(显示 hello 等内容)、堆栈（call printf 前）、寄存器同时在一个窗口。



没下 CB，就用 VS 看了，道理应该一样吧。

图 2-1 Windows 下 VisualCode 反汇编截图

2.2 Ubuntu 下 EDB 运行环境建立（10 分）

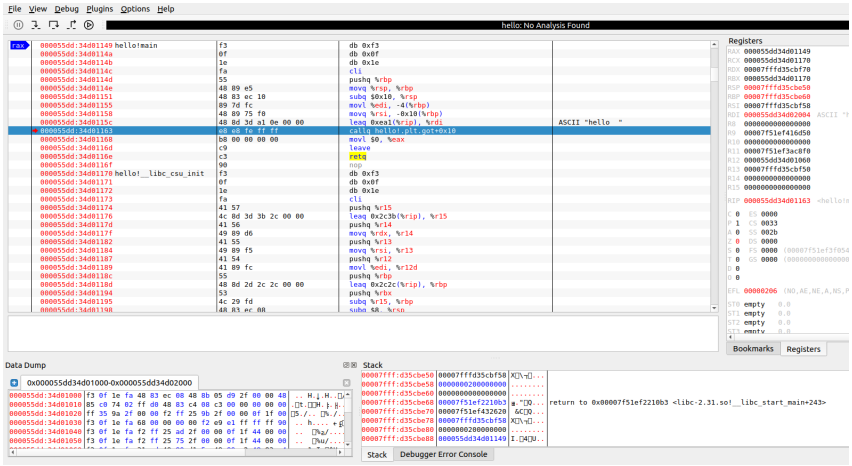


图 2-2 Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 30 分，密码 10 分，分析 20 分，总分不超过 80 分

3.1 阶段 1 的破解与分析

密码如下：I am the mayor. I can do anything I want.

破解过程：

由于主函数里面只有读取输入和函数调用，所以我们只需要看调用的函数即可。

```
00000000:004013ec 48 83 ec 08       subq $8, %rsp                  begin of phase1
00000000:004013f0 be 50 31 40 00    movl $0x403150, %esi           ASCII "I am the mayor. I can do anything I want."
00000000:004013f5 e8 cd 04 00 00    callq bomb!strings_not_equal
00000000:004013fa 85 c0             testl %eax, %eax
00000000:004013fc 75 05             jne 0x401403
00000000:004013fe 48 83 c4 08       addq $8, %rsp
00000000:00401402 c3                retq
00000000:00401403 e8 a1 05 00 00    callq bomb!explode_bomb
00000000:00401408 eb f4             jmp 0x4013fe                   end of phase1
```

该段为 phase1 的函数，可以看到里面调用了一个比较字符串是否相等的函数，然后检测返回值是否为 0，若不为 0 则炸弹爆炸，另外一个参数为右面的那个字符串地址，内容即为密码，由此第一题解。

3.2 阶段 2 的破解与分析

密码如下：1 2 4 8 16 32

破解过程：

```
00000000:0040140a 53                pushq %rbx                     begin of phase2
00000000:0040140b 48 83 ec 20       subq $0x20, %rsp
00000000:0040140f 48 89 e6          movq %rsp, %rsi
00000000:00401412 e8 b6 05 00 00    callq bomb!read_six_numbers
00000000:00401417 83 3c 24 01       cmpl $1, 0(%rsp)
00000000:0040141b 75 07             jne 0x401424
00000000:0040141d bb 01 00 00 00    movl $1, %ebx
00000000:00401422 eb 0f             jmp 0x401433
00000000:00401424 e8 80 05 00 00    callq bomb!explode_bomb
00000000:00401429 eb f2             jmp 0x40141d
00000000:0040142b e8 79 05 00 00    callq bomb!explode_bomb
00000000:00401430 83 c3 01          addl $1, %ebx
00000000:00401433 83 fb 05          cmpl $5, %ebx
00000000:00401436 7f 14             jg 0x40144c
00000000:00401438 48 63 d3          movslq %ebx, %rdx
00000000:0040143b 8d 43 ff          leal -1(%rbx), %eax
00000000:0040143e 48 98             cltq
00000000:00401440 8b 04 84          movl 0(%rsp, %rax, 4), %eax
00000000:00401443 01 c0             addl %eax, %eax
00000000:00401445 39 04 94          cmpl %eax, 0(%rsp, %rdx, 4)
00000000:00401448 74 e6             je 0x401430
00000000:0040144a eb df             jmp 0x40142b
00000000:0040144c 48 83 c4 20       addq $0x20, %rsp
00000000:00401450 5b                popq %rbx
00000000:00401451 c3                retq                           end of phase2
```

如图为 phase2 的函数。

首先看到他调用了 read_six_numbers 函数，将之保存在一个数组里，，因此将栈指针作为参数传入，可知我们应该输入六个数字。

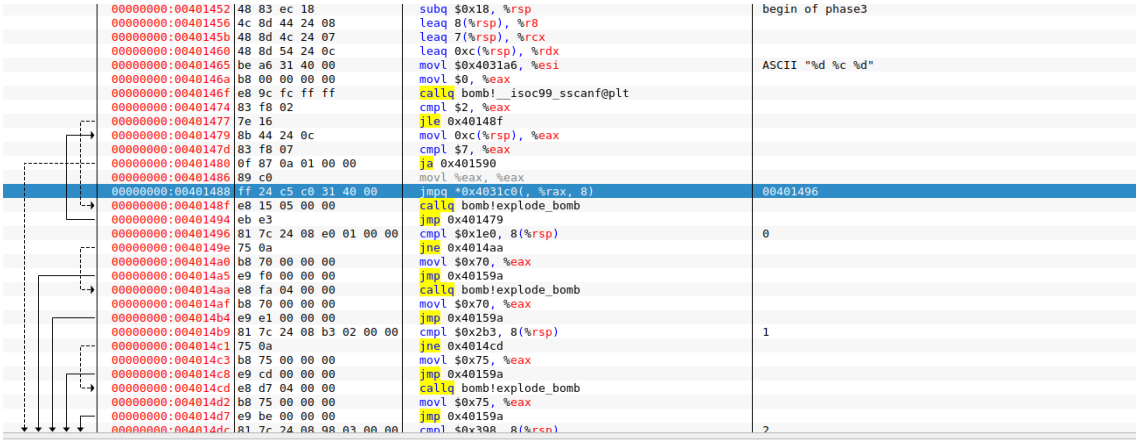
然后把数组第一个元素和 1 比较，若不相等则爆炸，可知第一个数字应该为 1.

之后函数进行一个循环，检测后一个元素是否为前一个元素的二倍，可知输入数列应该为一个公比为 2 的等比数列。

3.3 阶段 3 的破解与分析

密码如下：0 p 480

破解过程：



上图为 phase3 部分代码。

首先可以看到函数从字符串中提取三个参数，俩个数字和一个字符，然后检测第一个数字是否大于 7，可知第一个参数需要小于等于 7.然后跟着一个跳转指令，跳转目标以一个数据内存地址中所存的数为基地址的数组。

然后我们找到该内存位置，可以看到所存的为 0X00401496，为上图标 0 位置，后续还有七个元素，分别为代码的某个地址，而且各代码段结构相同。可以看出这是一个 switch 表，一共有 0 到 7 八个入口，由第一个参数决定。

在 switch 中，比较了第三个参数和给定数字，若不相等则炸弹爆炸，由此可以得到第二个数字的值，其取决于第一个参数。然后又在 eax 中保存了一个一字

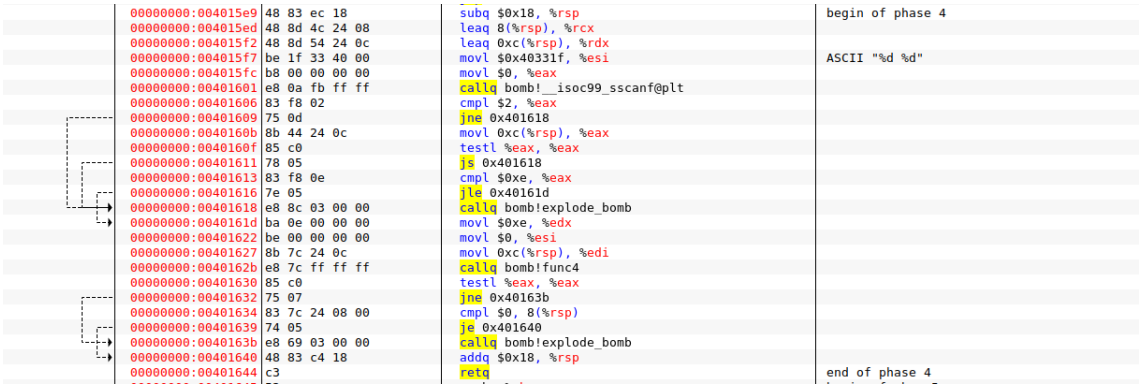
节的数字，不过扩展为 4 字节。在 switch 结束后，看到将第二个参数与这个数做比较，若不相等则炸弹爆炸，由此可知第二个参数的 ascii 编码应该等于这个数，由此得到第二个参数的值。

由于该关卡为一个 switch 表，因此有 8 种可能结果，我这里只写了第一种。

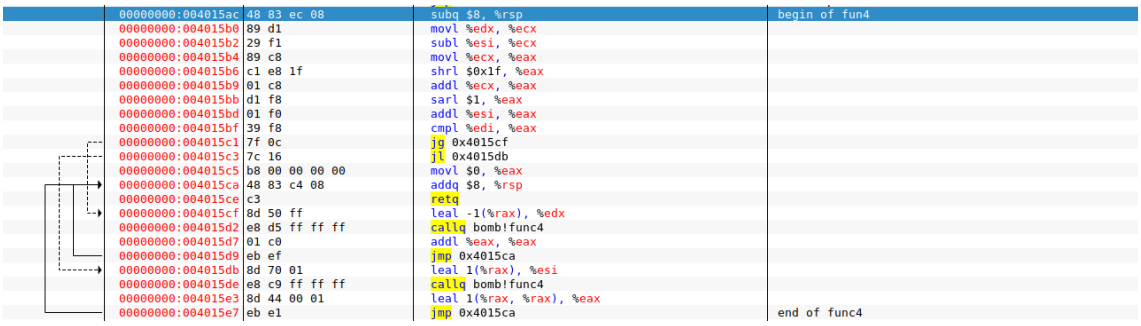
3.4 阶段 4 的破解与分析

密码如下：7 0 DrEvil

破解过程：



上图为 phase4 函数代码，可知需要读入俩个整数，存在临时变量中，然后将第一个整数和 0，14 作为参数传入调用 func4 函数，检测返回值是否为 0，不为 0 则爆炸。然后检测第二个整数是否为 0，可知第二个整数应该是 0.



然后我们看 fun4 内容。此时%edi 中为我们的第一个数字，%esi 中为 0，%edx 中为 14.开头几句的处理是为了得到俩个数的中间值，其中向右移 31 位是得到了差值的符号位，然后做一个偏置处理（由于他这个只除以 2，相当于向右移一位，因此直接把符号位的值重复利用了，所以一开始没咋看懂。）这样保证了这个算差值一半是向 0 舍入的。

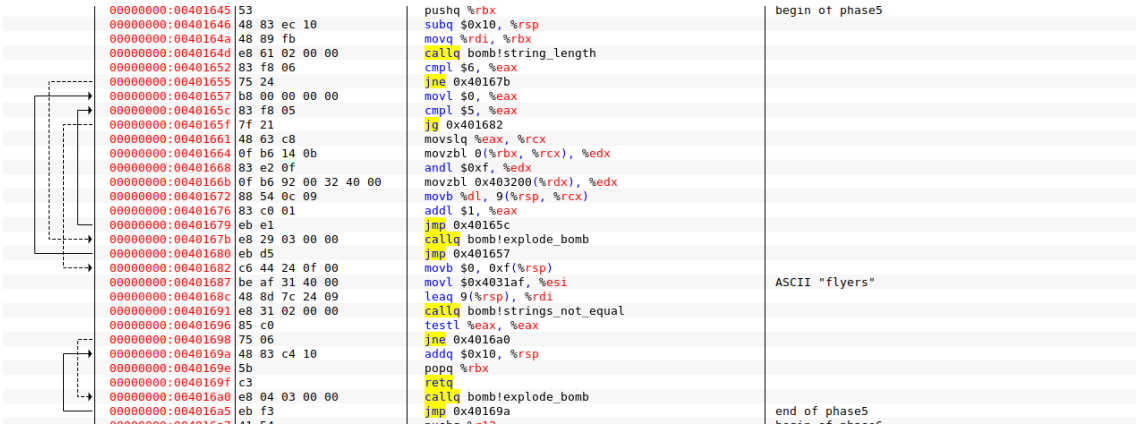
然后比较我们输入的第一个数字是否等于这个平均值。若小于，则将该中间值减 1 作为第三个参数递归调用 fun4，若大于，则炸弹爆炸。若等于，则拆弹成功。可知这是一个类似于折半查找的函数，可能的解有 7，3，1，0.

我这里直接将 7 做为答案输入。

3.5 阶段 5 的破解与分析

密码如下：IONEFG

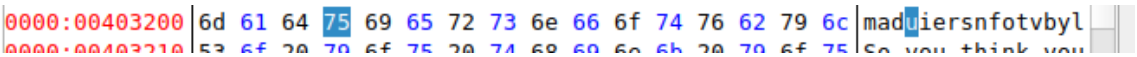
破解过程：



然后来到第 5 阶段，首先看到了 flyers 这个字段，不过尝试后不是答案，那么就得好好看看函数了。

首先检查参数读取，可以看到要求输入一个长度为 6 的字符串。然后对字符串各字符进行一个循环。然后我们看循环内部。

首先他对我们输入的字符做了一个掩码处理，只取了第四位，然后将得到的数字作为索引从 0X403200 这个地址中取值，存到一个临时数组中。那我们看看这个地址里存着什么。



如图，可以看到这个地址中存了一些乱序的字母，仔细观察，包含了 flyers 的所有字母。然后我们看到循环后将得到的字符数组与 flyers 做比较，可知我们应该按顺序取出 flyers 这六个字母，这要求我们输入的字符串低四位对应的值是这六个字母对应的位置下标，由此就可以得到应该输入的答案，此答案有多种可能，因为高四位是任意的，我这里只取了一种，即 IONEFG。

3.6 阶段 6 的破解与分析

密码如下：6 5 3 1 4 2

破解过程：

这个阶段的代码很长，我们慢慢分析。

```
00000000:004016a7 41 54          pushq %r12          begin of phase6
00000000:004016a9 55             pushq %rbp
00000000:004016aa 53             pushq %rbx
00000000:004016ab 48 83 ec 50    subq $0x50, %rsp
00000000:004016af 48 8d 74 24 30 leaq 0x30(%rsp), %rsi
00000000:004016b4 e8 14 03 00 00 callq bomb!read_six_numbers
```

首先从这部分可以看到我们需要读入六个数字，存在一个临时数组当中。

```
00000000:004016b9 bd 00 00 00 00 movl $0, %ebp
00000000:004016be eb 29          jmp 0x4016e9
00000000:004016c0 e8 e4 02 00 00 callq bomb!explode_bomb
00000000:004016c5 eb 36          jmp 0x4016fd
00000000:004016c7 e8 dd 02 00 00 callq bomb!explode_bomb
00000000:004016cc 83 c3 01       addl $1, %ebx
00000000:004016cf 83 fb 05       cmpl $5, %ebx
00000000:004016d2 7f 12          jg 0x4016e6
00000000:004016d4 48 63 c5       movslq %ebp, %rax
00000000:004016d7 48 63 d3       movslq %ebx, %rdx
00000000:004016da 8b 7c 94 30    movl 0x30(%rsp, %rdx, 4), %edi
00000000:004016de 39 7c 84 30    cmpl %edi, 0x30(%rsp, %rax, 4)
00000000:004016e2 75 e8          jne 0x4016cc
00000000:004016e4 eb e1          jmp 0x4016c7
00000000:004016e6 44 89 e5       movl %r12d, %ebp
00000000:004016e9 83 fd 05       cmpl $5, %ebp
00000000:004016ec 7f 18          jg 0x401706
00000000:004016ee 48 63 c5       movslq %ebp, %rax
00000000:004016f1 8b 44 84 30    movl 0x30(%rsp, %rax, 4), %eax
00000000:004016f5 83 e8 01       subl $1, %eax
00000000:004016f8 83 f8 05       cmpl $5, %eax
00000000:004016fb 77 c3          ja 0x4016c0
00000000:004016fd 44 8d 65 01    leal 1(%rbp), %r12d
00000000:00401701 44 89 e3       movl %r12d, %ebx
00000000:00401704 eb c9          jmp 0x4016cf
```

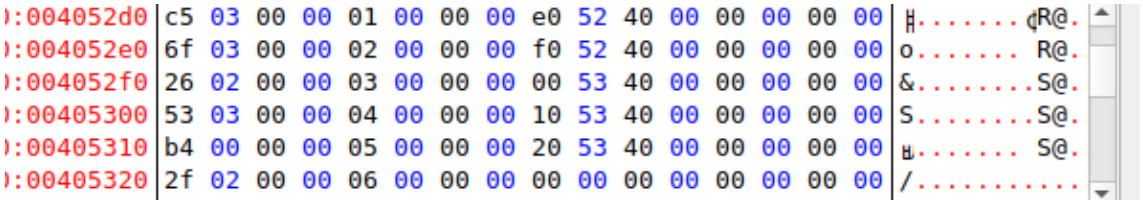
这段代码对我们输入的参数进行了检测，流程不仔细分析了，其作用就是依次检查这六个数字是否不大于 6 并且互不相等。（不过这里有点漏洞感觉，从后面我可以看出他是输入 1 到 6 的数字，这里没有限制大于 0.其次他也没有检测出一定读入了六个数字，可能是在 read_six_numbers 里检测了？）

```
00000000:0040170b eb 13          jmp 0x401720
00000000:0040170d 48 63 c8       movslq %eax, %rcx
00000000:00401710 ba 07 00 00 00 movl $7, %edx
00000000:00401715 2b 54 8c 30    subl 0x30(%rsp, %rcx, 4), %edx
00000000:00401719 89 54 8c 30    movl %edx, 0x30(%rsp, %rcx, 4)
00000000:0040171d 83 c0 01       addl $1, %eax
00000000:00401720 83 f8 05       cmpl $5, %eax
00000000:00401723 7e e8          jle 0x40170d
```

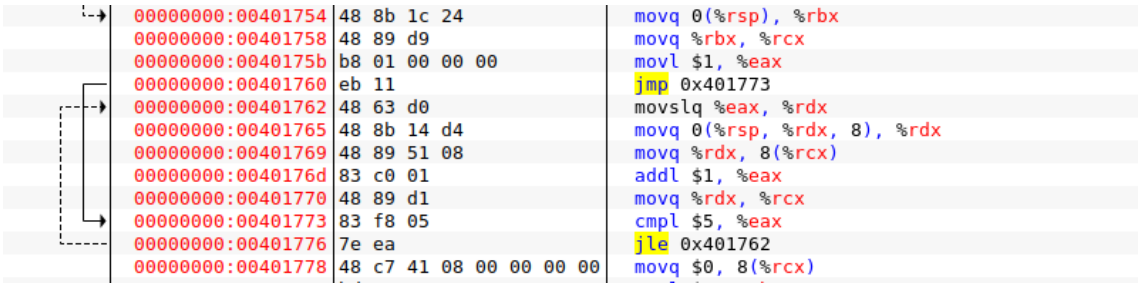
参数检测完毕后，又将各个数字对 7 取了一个补。

```
00000000:00401725 be 00 00 00 00 movl $0, %esi
00000000:0040172a eb 17          jmp 0x401743
00000000:0040172c 48 8b 52 08    movq 8(%rdx), %rdx
00000000:00401730 83 c0 01       addl $1, %eax
00000000:00401733 48 63 ce       movslq %esi, %rcx
00000000:00401736 39 44 8c 30    cmpl %eax, 0x30(%rsp, %rcx, 4)
00000000:0040173a 7f f0          jg 0x40172c
00000000:0040173c 48 89 14 cc    movq %rdx, 0(%rsp, %rcx, 8)
00000000:00401740 83 c6 01       addl $1, %esi
00000000:00401743 83 fe 05       cmpl $5, %esi
00000000:00401746 7f 0c          jg 0x401754
00000000:00401748 b8 01 00 00 00 movl $1, %eax
00000000:0040174d ba d0 52 40 00 movl $0x4052d0, %edx  list head7
00000000:00401752 eb df          jmp 0x401733
```

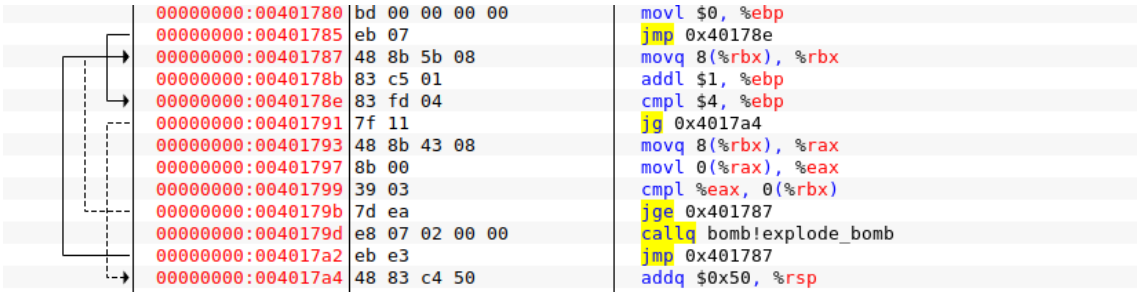
这段代码读了好久才读懂。他首先用到了一个地址中的内容，先去看看是什么：



观察后发现，这是一个长度为六的链表，并且很贴心的标注了 123456 这个序号。然后再看代码，发现他是将这链表的六个节点的地址存到了一个临时数组中，顺序即为我们输入六个数字所对应的顺序。（比如第 1 个数字为 6，那么就把第 6 个节点地址作为数组第一个元素，然后昂，他这个链表节点顺序也不是按图上那个顺序，是反着标的，比如标 1 的是第六个节点，这样就对上了）



然后这段代码就是把这个链表按照我们那个地址的顺序重排了一下，然后末节点指向 null。



这段代码是遍历了一遍列表，检测前一个节点元素是否大于后一个节点元素，若不大于则炸弹爆炸，由此可知这个阶段是让我们输入一个顺序，将这个链表从大到小重新排序。不过这里有一个需要注意的是，比较的数字是四字节的，因此需要忽略后面的四字节，只比较前四个字节，并且这是一个小端法机器，要注意高低位。故到这里我们便可以得到答案了，仔细对一下数字可以得

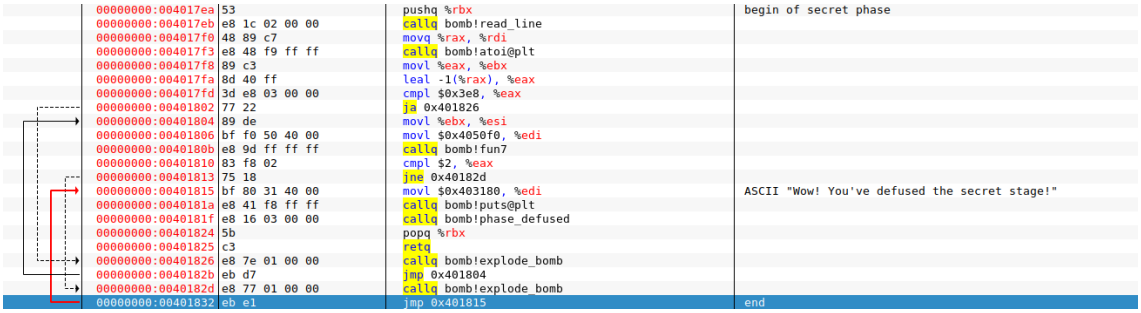
到 6 5 3 1 4 2。

3.7 阶段 7 的破解与分析(隐藏阶段)

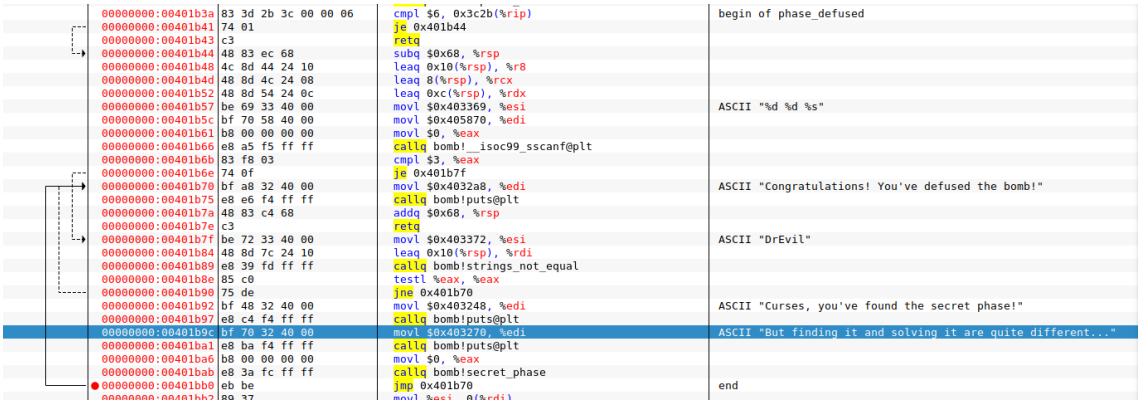
密码如下：22

破解过程：

首先看汇编代码可以找到 secret_phase 这个函数，但是不知道怎么进去。



然后我看了看源代码，发现只有 phase_defused 这个函数我没仔细看过了，而其他 1 到 6 阶段的代码我都仔细看过，没有调用这个函数的，由此我去瞅瞅 phase_defused 的源代码。

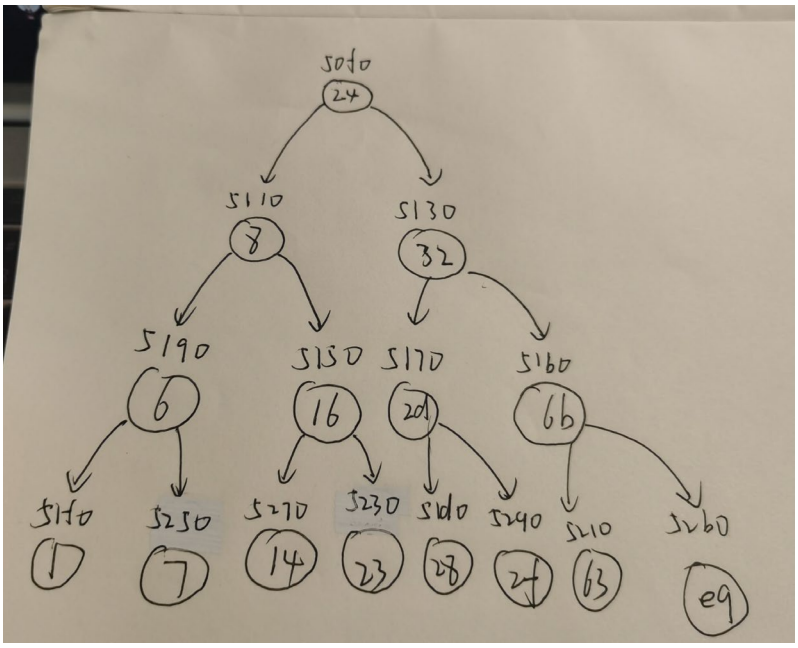


这里可以看到他一些字符串提示，显然这里是秘密阶段的入口。然后看一下代码，首先他把 6 和某一个值做了比较，应该是一个全局变量，检测我们通过的阶段数。如果通过六关，则进行一系列检测，分析略，总之我们应该在第四阶段附带一个 DrEvil 的字符串。

分析隐藏阶段的代码。首先可以看出我们需要输入一个数字，然后他将这个数字和一个地址作为参数传入到 fun7，然后检测返回值是否为 2，若不为 2 则爆炸。我们先看一下这个地址 0X4050F0 的内容。

```
:004050f0 24 00 00 00 00 00 00 00 10 51 40 00 00 00 00 00 $........Q@.
:00405100 30 51 40 00 00 00 00 00 00 00 00 00 00 00 00 00 0Q@.........
:00405110 08 00 00 00 00 00 00 00 90 51 40 00 00 00 00 00 .........Q@.
:00405120 50 51 40 00 00 00 00 00 00 00 00 00 00 00 00 00 PQ@.........
:00405130 32 00 00 00 00 00 00 00 70 51 40 00 00 00 00 00 2.......pQ@.
```

Emmm，乍一看发现，这个地址后面第一个八字节是一个数字，后俩个八字节是一个地址，而且这俩地址跟这个地方挺近的，就感觉是一个图型的结构，然后就跟着这个地址画了画结构。得到了下图：



发现是一个熟悉的结构，二叉搜索平衡树。然后看一下 fun7：

```
00000000:004017ad 48 85 ff       testq %rdi, %rdi          begin of fun7不
00000000:004017b0 74 32          je 0x4017e4
00000000:004017b2 48 83 ec 08    subq $8, %rsp
00000000:004017b6 8b 07          movl 0(%rdi), %eax
00000000:004017b8 39 f0          cmpl %esi, %eax
00000000:004017ba 7f 0c          jg 0x4017c8
00000000:004017bc 75 17          jne 0x4017d5
00000000:004017be b8 00 00 00 00 movl $0, %eax
00000000:004017c3 48 83 c4 08    addq $8, %rsp
00000000:004017c7 c3             retq
00000000:004017c8 48 8b 7f 08    movq 8(%rdi), %rdi
00000000:004017cc e8 dc ff ff ff callq bomb!fun7
00000000:004017d1 01 c0          addl %eax, %eax
00000000:004017d3 eb ee          jmp 0x4017c3
00000000:004017d5 48 8b 7f 10    movq 0x10(%rdi), %rdi
00000000:004017d9 e8 cf ff ff ff callq bomb!fun7
00000000:004017de 8d 44 00 01    leal 1(%rax, %rax), %eax
00000000:004017e2 eb df          jmp 0x4017c3
00000000:004017e4 b8 ff ff ff ff movl $0xffffffff, %eax
00000000:004017e9 c3             retq                      end俑
```

具体分析不说了，总之这是一个二叉搜索树递归搜索的过程，如果找到，则返回 0，如果是从右子树里面找到的，则返回递归调用的返回值的二倍并且加一，如果是从左子树里找到的，则返回递归调用的返回值的二倍。

然后他要求最后结果是 2，那么只有 0 1 2 这个序列符合结果，因此应该在 5150 这个节点里找到，这样跳到 5110 节点得到 1，再跳到根得到 2.然后昂他这里有一

个注意点是节点里的数字是十六进制数字，因此应该进行一个到十进制的转换，0x16=0d22，所以我们应该输入 22.

至此 炸弹全部拆除~ 🎉🎉

```
dqv587@ubuntu:~/Documents/Code/LAB2/bomb315$ ./bomb ans.txt
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Phase 1 defused. How about the next one?
That's number 2.  Keep going!
Halfway there!
So you got that one.  Try this one.
Good work!  On to the next...
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
```

另外，我还看了看其他没有看到的代码。发现了一个 sig_handler 函数，看了看内容，发现可以用 ctrl+c 来发送中断异常，从而中止这个炸弹。

```
dqv587@ubuntu:~/Documents/Code/LAB2/bomb315$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
```

此外还发现一个 submit 的函数，他链接一个服务器，不过 IP 地址已经被改掉了，应该是原版实验有一个汇报学生情况做统计的函数。

第 4 章 总结

4.1 请总结本次实验的收获

首先熟悉了汇编语言，能够从汇编代码中反推出函数的操作。然后我仔细看了一下。Plt 部分，也对动态链接有了更好的认识，另外也对程序的内存分布有了认识，比如代码区、只读区、可写区等。

4.2 请给出对本次实验内容的建议

没啥建议。

注：本章为酌情加分项。

参考文献

[1] Kernighan B W, Ritchie D M. The C Programming Language[M]. 2. Dennis Ritchie & Bell Labs, / June 2018.

[2] Bryant R E, David R. O'Hallaron. Computer Systems a Programmer's Perspective[M]. 3rd. Carnegie Mellon University, 2016.