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 计算机类

学　　号 1170300825

班　　级 1703008

学　　生 李大鑫

指 导 教 师 郑贵滨

实 验 地 点

实 验 日 期

计算机科学与技术学院

目 录

第 1 章 实验基本信息 ..... - 3 -

1.1 实验目的 ..... - 3 -
1.2 实验环境与工具 ..... - 3 -
1.2.1 硬件环境 ..... - 3 -
1.2.2 软件环境 ..... - 3 -
1.2.3 开发工具 ..... - 3 -
1.3 实验预习 ..... - 3 -

第 2 章 实验环境建立 ..... - 5 -

2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分） ..... - 5 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分） ..... - 5 -

第 3 章 各阶段炸弹破解与分析 ..... - 7 -

3.1 阶段 1 的破解与分析 ..... - 7 -
3.2 阶段 2 的破解与分析 ..... - 7 -
3.3 阶段 3 的破解与分析 ..... - 8 -
3.4 阶段 4 的破解与分析 ..... - 10 -
3.5 阶段 5 的破解与分析 ..... - 11 -
3.6 阶段 6 的破解与分析 ..... - 12 -
3.7 阶段 7 的破解与分析(隐藏阶段) ..... - 15 -

第 4 章 总结 ..... - 17 -

4.1 请总结本次实验的收获 ..... - 17 -
4.2 请给出对本次实验内容的建议 ..... - 17 -

参考文献 ..... - 17 -

第 1 章 实验基本信息

1.1 实验目的

- 熟练掌握计算机系统的 ISA 指令系统与寻址方式
- 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
- 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

- X64 CPU；2GHz；2G RAM；256GHD Disk 以上

1.2.2 软件环境

- Windows7 64 位以上；VirtualBox/Vmware 11 以上 Ubuntu 16.04 LTS 64 位/优麒麟 64 位；

1.2.3 开发工具

- Visual Studio 2010 64 位以上；GDB/OBJDUMP；KDD 等

1.3 实验预习

■ 上实验课前，必须认真预习实验指导书（**PPT** 或 **PDF**）

■ 了解实验的目的、实验环境与软硬件工具、实验操作步骤，复习与实验有关的理论知识。

■ 请写出 **C** 语言下包含字符串比较、循环、分支（含 **switch**）、函数

调用、递归、指针、结构、链表等的例子程序 **sample.c**。

- 生成执行程序 **sample.out**。
- 用 **gcc –S** 或 **CodeBlocks** 或 **GDB** 或 **OBJDUMP** 等，反汇编，比较。
- 列出每一部分的 **C** 语言对应的汇编语言。
- 修改编译选项**-O (**缺省 **2)**、**O0**、**O1**、**O2**、**O3**，**-m32/m64**。再次查看生成的汇编语言与原来的区别。
- 注意 **O1** 之后无栈帧，**EBP** 做别的用途。**-fno-omit-frame-pointer** 加上栈指针。
- **GDB** 命令详解 **–tui** 模式 **^XA** 切换 **layout** 改变等等
- 有目的地学习：看 **VS** 的功能 **GDB** 命令用什么？

第 2 章 实验环境建立

2.1 Ubuntu 下 CodeBlocks 反汇编（10 分）

CodeBlocks 运行 hellolinux.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。

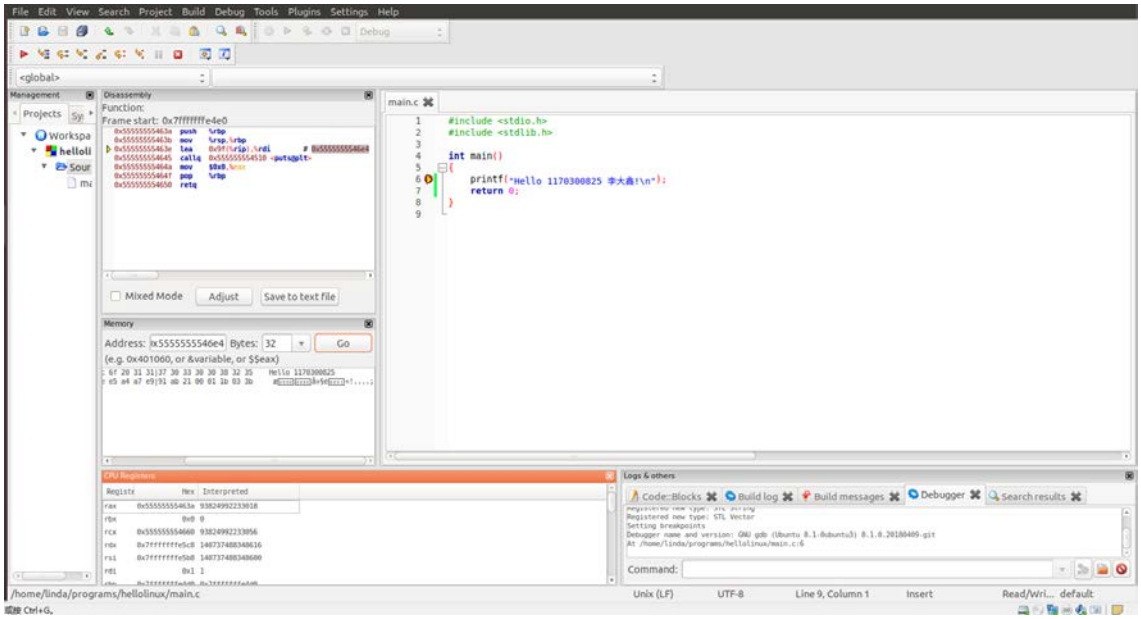


图 2-1 Ubuntu 下 CodeBlocks 反汇编截图

2.2 Ubuntu 下 EDB 运行环境建立（10 分）

用 EDB 调试 hellolinux.c 的执行文件，截图，要求同 2.1

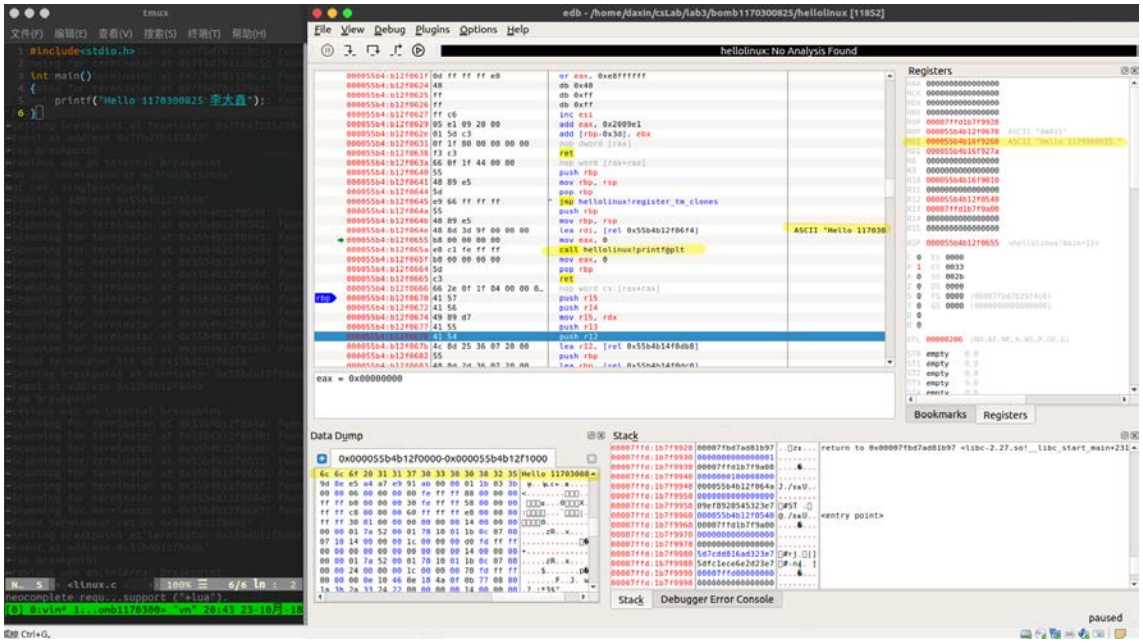


图 2-2　Ubuntu 下 EDB 截图

第 3 章 各阶段炸弹破解与分析

每阶段 15 分，密码 10 分，分析 5 分，总分不超过 80 分

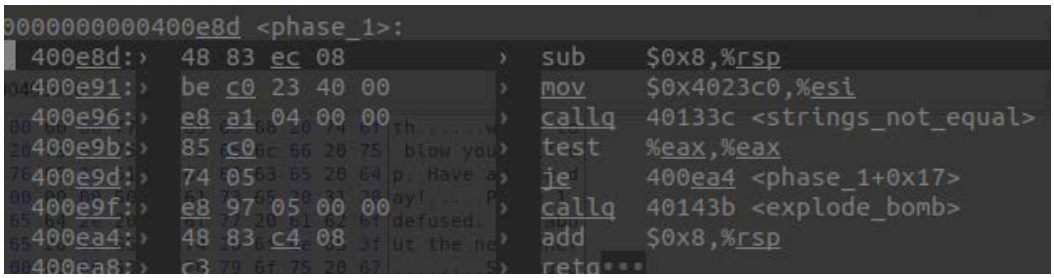
3.1 阶段 1 的破解与分析

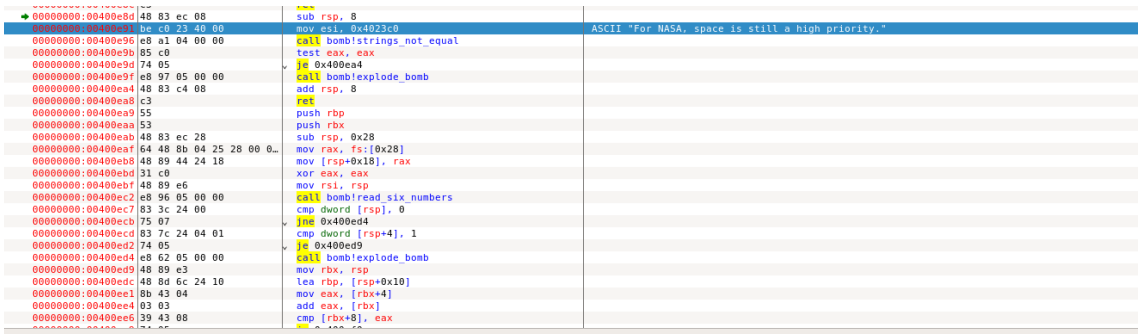
密码如下：

For NASA, space is still a high priority.

破解过程：

根据 main 函数的执行顺序首先找到第一个被执行的拆弹函数 phase_1，可以看到其主程序总体完成了一个字符串比较的操作，%edi 和%esi 分别指向输入的字符串和用来比较的字符串 S，推测出 S 就是第一个密码。通过 edp 跟踪执行过程可以看到 0x4023c0 这个地址指向的字符串为“For NASA, space is still a high priority.”





3.2 阶段 2 的破解与分析

密码如下：

0 1 1 2 3 5

破解过程：

可以看出 phase_2 的主要思路是读入 6 个整数，然后和程序产生的 6 个整数比较，如果不相等则爆炸。

首先进行两个 cmp，分别是和 0x0 和 0x1，之后根据 cmp %rbp,%ebx 和后面的跳转指令可以判断出这是一个循环语句段的跳转入口。分析循环内部，可以分析出%eax=(%rbx)+(%rbx+4)，而每次循环%rbx 都会自加 4，可以分析得出这是每次都会拿出出现的前两个数求和之后算出现在的数，因此密码是前 6 个菲波那切数列。

```
0000000000400ea9 <phase_2>:
  400ea9:   55                      push   %rbp
  400eaa:   53                      push   %rbx
  400eab:   48 83 ec 28             sub    $0x28,%rsp
  400eaf:   64 48 8b 04 25 28 00    mov    %fs:0x28,%rax
  400eb6:   00 00
  400eb8:   48 89 44 24 18          mov    %rax,0x18(%rsp)
  400ebd:   31 c0                   xor    %eax,%eax
  400ebf:   48 89 e6                mov    %rsp,%rsi
  400ec2:   e8 96 05 00 00          callq  40145d <read_six_numbers>
  400ec7:   83 3c 24 00             cmpl   $0x0,(%rsp)
  400ecb:   75 07                   jne    400ed4 <phase_2+0x2b>
  400ecd:   83 7c 24 04 01          cmpl   $0x1,0x4(%rsp)
  400ed2:   74 05                   je     400ed9 <phase_2+0x30>
  400ed4:   e8 62 05 00 00          callq  40143b <explode_bomb>
  400ed9:   48 89 e3                mov    %rsp,%rbx
  400edc:   48 8d 6c 24 10          lea    0x10(%rsp),%rbp
  400ee1:   8b 43 04                mov    0x4(%rbx),%eax
  400ee4:   03 03                   add    (%rbx),%eax
  400ee6:   39 43 08                cmp    %eax,0x8(%rbx)
  400ee9:   74 05                   je     400ef0 <phase_2+0x47>
  400eeb:   e8 4b 05 00 00          callq  40143b <explode_bomb>
  400ef0:   48 83 c3 04             add    $0x4,%rbx
  400ef4:   48 39 eb                cmp    %rbp,%rbx
  400ef7:   75 e8                   jne    400ee1 <phase_2+0x38>
  400ef9:   48 8b 44 24 18          mov    0x18(%rsp),%rax
  400efe:   64 48 33 04 25 28 00    xor    %fs:0x28,%rax
  400f05:   00 00
  400f07:   74 05                   je     400f0e <phase_2+0x65>
  400f09:   e8 f2 fb ff ff          callq  400b00 <__stack_chk_fail@plt>
  400f0e:   48 83 c4 28             add    $0x28,%rsp
  400f12:   5b                      pop    %rbx
  400f13:   5d                      pop    %rbp
  400f14:   c3                      retq***
```

3.3 阶段 3 的破解与分析

密码如下：

5 -610

破解过程：

本题核心破解思路是利用 edb 跟踪代码。

首先程序调用 sscanf 函数，观察%esi 指向的字符串，推测输入的是两个整数，分别设为 a、b。

```
00000000:00400f31 be 7f 25 40 00      mov esi, 0x40257f                ASCII "%d %d"
00000000:00400f36 e8 75 fc ff ff      call bomb!__isoc99_sscanf@plt
```

%eax 接收读到的数目，参数设置为“%d %d”所以只能读入两个，%eax 为 2，%ebp 指向读入的数据。

1）首先第一个检查要求%eax > 1，否则跳转爆炸。

```
00000000:00400f3b 83 f8 01            cmp eax, 1
00000000:00400f3e 7f 05               jg 0x400f45
00000000:00400f40 e8 f6 04 00 00      call bomb!explode_bomb
00000000:00400f45 83 3c 24 07         cmp dword [rsp], 7
```

2） 第二个检查要求输入 a <= 7，否则跳转爆炸。

```
00000000:00400f45 83 3c 24 07         cmp dword [rsp], 7
00000000:00400f49 77 65               ja 0x400fb0
```

3）根据第一个参数来计算下面对%rax 一系列操作从哪里开始，这里会直接影响到最终 b 的值，下面以输入 a=5 为例。

（设置的所有操作序列为+0x166->-0x3e6->+0x301->-0x262->+0x262->-0x262->+0x262->-0x262，因为设置 a<=7，a 用来选择从第几个操作开始进入，编号从 0 开始，0..7，然后顺序执行后面的操作，需要注意的是 a 不能为负数，否则会 jmp 到错误位置。）

其实是实现了一个没有 break 的 switch 功能。

```
00000000:00400f4b 8b 04 24            mov eax, [rsp]
00000000:00400f4e ff 24 c5 20 24 40 00  jmp qword [rax*8+0x402420]
```

4）经过一些跳转和计算、赋值之后，进行第三个检查，要求第一个输入 a<=5，否则爆炸。

```
00000000:00400fba 83 3c 24 05         cmp dword [rsp], 5
00000000:00400fbe 7f 06               jg 0x400fc6
```

5）然后进入最后一个检查，要求 b 与一系列计算之后的 eax 比较，如果相等则不会爆炸，因为 edb 的便利性我们可以直接忽略计算过程看到比较结果，

此时%eax 是 0xfffffd9e 即-610。

、

于是我们得到了第三个解(5(0<=a<=5),-610)。

当然也可求出当 a 为其他 5 个数的时候对应的 b 值，也是符合条件的解。

3.4 阶段 4 的破解与分析

密码如下：

108 2 （DrEvil） //DrEvil 用来开启进入 secret_phase 的大门

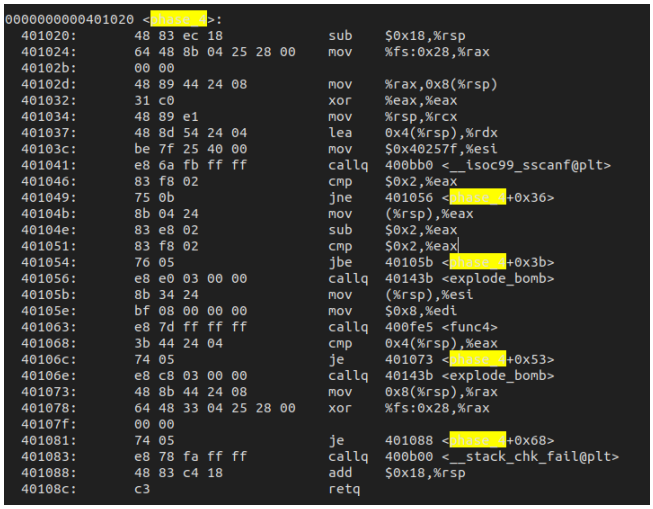
破解过程：

首先观察 phase_4 主函数，主要过程是读入两个数字 a,b（自左向右压入栈中），然后进行的第一个有效的验证是：(b-2)<=2，然后进行 func4 计算得到%eax，第二有效验证就是需要满足 func4==a。

首先调用过程中，传入了两个参数分别是%esi=b,%edi=8，然后观察 func4 函数，可以看出 func4 的函数是一个递归函数，递归函数如下：

func4(x)=func4(x-1)+func4(x-2)+b，其中 func4(0)=0,func4(1)=b

初始值 x=8，本题代入 b=2，则可以得出 func4（8）=108



```
0000000000400fe5 <func4>:
  400fe5:       85 ff                   test   %edi,%edi
  400fe7:       7e 2b                   jle    401014 <func4+0x2f>
  400fe9:       89 f0                   mov    %esi,%eax
  400feb:       83 ff 01                cmp    $0x1,%edi
  400fee:       74 2e                   je     40101e <func4+0x39>
  400ff0:       41 54                   push   %r12
  400ff2:       55                      push   %rbp
  400ff3:       53                      push   %rbx
  400ff4:       89 f5                   mov    %esi,%ebp
  400ff6:       89 fb                   mov    %edi,%ebx
  400ff8:       8d 7f ff                lea    -0x1(%rdi),%edi
  400ffb:       e8 e5 ff ff ff          callq  400fe5 <func4>
  401000:       44 8d 64 05 00          lea    0x0(%rbp,%rax,1),%r12d
  401005:       8d 7b fe                lea    -0x2(%rbx),%edi
  401008:       89 ee                   mov    %ebp,%esi
  40100a:       e8 d6 ff ff ff          callq  400fe5 <func4>
  40100f:       44 01 e0                add    %r12d,%eax
  401012:       eb 06                   jmp    40101a <func4+0x35>
  401014:       b8 00 00 00 00          mov    $0x0,%eax
  401019:       c3                      retq
  40101a:       5b                      pop    %rbx
  40101b:       5d                      pop    %rbp
  40101c:       41 5c                   pop    %r12
  40101e:       f3 c3                   repz retq
```

3.5 阶段 5 的破解与分析

密码如下：

m63487

破解过程：

首先输入的字符串存放到%rbx 为起始位置的栈中，首先调用了 bomb!string_length 函数计算输入字符串的长度，第一个验证需要满足：%eax==6，然后接下来进入一个循环，目的是将 0x402460+%rdx（%rdx+%rax）的字符存放到%rsp+%rax 指向的位置，rax 在循环中一次递加，将所有 6 个字符一次复制。然后进行第二个验证调用函数 strings_not_equal，需要满足%rsp 和%esi 指向的字符串相等，其中%esi 指向的字符串“bruins”。

每次循环%rdx 指向的数值为输入的字符串的循环，从这里可以看出输入的字符串其实起到一个偏移的作用，0x402460 存储的字符如下：

```
00000:00402460 6d 61 64 75 69 65 72 73 6e 66 6f 74 76 62 79 6c maduiersnfotvbyl
```

可以得出偏移字符串为 m63487，可以拼凑出“bruins”这个字符串。

注意 m 在内存中用 ascill 码存储，01101101，根据计算规则取最低 4 位得到 1101，为 13，即正确的偏移量。

```
00000000:0040108e 48 83 ec 10              sub rsp, 0x10
00000000:00401092 48 89 fb                 mov rbx, rdi
00000000:00401095 64 48 8b 04 25 28 00 0…  mov rax, fs:[0x28]
00000000:0040109e 48 89 44 24 08           mov [rsp+8], rax
00000000:004010a3 31 c0                    xor eax, eax
00000000:004010a5 e8 74 02 00 00           call bomb!string_length
00000000:004010aa 83 f8 06                 cmp eax, 6
00000000:004010ad 74 05                    je 0x4010b4
00000000:004010af e8 87 03 00 00           call bomb!explode_bomb
00000000:004010b4 b8 00 00 00 00           mov eax, 0
00000000:004010b9 0f b6 14 03              movzx edx, [rbx+rax]
00000000:004010bd 83 e2 0f                 and edx, 0xf
00000000:004010c0 0f b6 92 60 24 40 00     movzx edx, [rdx+0x402460]
00000000:004010c7 88 14 04                 mov [rsp+rax], dl
00000000:004010ca 48 83 c0 01              add rax, 1
00000000:004010ce 48 83 f8 06              cmp rax, 6
00000000:004010d2 75 e5                    jne 0x4010b9
00000000:004010d4 c6 44 24 06 00           mov byte [rsp+6], 0
00000000:004010d9 be 16 24 40 00           mov esi, 0x402416          ASCII "bruins"
00000000:004010de 48 89 e7                 mov rdi, rsp
00000000:004010e1 e8 56 02 00 00           call bomb!strings_not_equal
00000000:004010e6 85 c0                    test eax, eax
00000000:004010e8 74 05                    je 0x4010ef
00000000:004010ea e8 4c 03 00 00           call bomb!explode_bomb
00000000:004010ef 48 8b 44 24 08           mov rax, [rsp+8]
00000000:004010f4 64 48 33 04 25 28 00 0…  xor rax, fs:[0x28]
00000000:004010fd 74 05                    je 0x401104
00000000:004010ff e8 fc f9 ff ff           call bomb!__stack_chk_fail@plt
00000000:00401104 48 83 c4 10              add rsp, 0x10
00000000:00401108 5b                       pop rbx
00000000:00401109 c3                       ret
```

3.6 阶段 6 的破解与分析

密码如下：

5 3 1 6 4 2

破解过程：

将整个汇编代码划分成为不同的部分。

Part1.

```
000000000040110a <phase_6>:
 40110a:  41 55                 push   %r13
 40110c:  41 54                 push   %r12
 40110e:  55                    push   %rbp
 40110f:  53                    push   %rbx
 401110:  48 83 ec 68           sub    $0x68,%rsp
 401114:  64 48 8b 04 25 28 00  mov    %fs:0x28,%rax
 40111b:  00 00
 40111d:  48 89 44 24 58        mov    %rax,0x58(%rsp)
 401122:  31 c0                 xor    %eax,%eax
 401124:  48 89 e6              mov    %rsp,%rsi
 401127:  e8 31 03 00 00        callq  40145d <read_six_numbers>
 40112c:  49 89 e4              mov    %rsp,%r12
 40112f:  41 bd 00 00 00 00     mov    $0x0,%r13d
```

主要功能：读入 6 个整数

Part2.

```
401135:  4c 89 e5          mov    %r12,%rbp
401138:  41 8b 04 24       mov    (%r12),%eax
40113c:  83 e8 01          sub    $0x1,%eax
40113f:  83 f8 05          cmp    $0x5,%eax
401142:  76 05             jbe    401149 <phase_6+0x3f>
401144:  e8 f2 02 00 00    callq  40143b <explode_bomb>
401149:  41 83 c5 01       add    $0x1,%r13d
40114d:  41 83 fd 06       cmp    $0x6,%r13d
401151:  74 3d             je     401190 <phase_6+0x86>
401153:  44 89 eb          mov    %r13d,%ebx
401156:  48 63 c3          movslq %ebx,%rax
401159:  8b 04 84          mov    (%rsp,%rax,4),%eax
40115c:  39 45 00          cmp    %eax,0x0(%rbp)
40115f:  75 05             jne    401166 <phase_6+0x5c>
401161:  e8 d5 02 00 00    callq  40143b <explode_bomb>
401166:  83 c3 01          add    $0x1,%ebx
401169:  83 fb 05          cmp    $0x5,%ebx
40116c:  7e e8             jle    401156 <phase_6+0x4c>
40116e:  49 83 c4 04       add    $0x4,%r12
401172:  eb c1             jmp    401135 <phase_6+0x2b>
```

首先观察 j**跳转指令的地址和逻辑，可以判断出这部分嵌套了两个 for 语句，循环体主要功能段是，

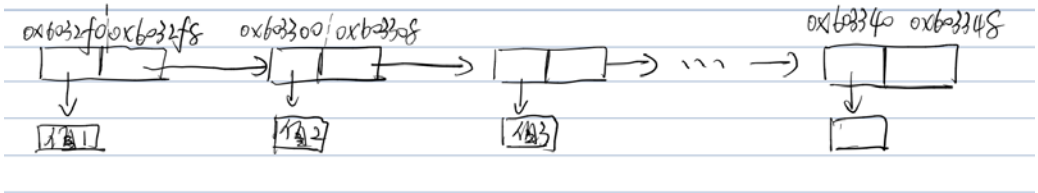
```
cmp    $0x5,%eax
jbe    401149 <phase_6+0x3f>
callq  40143b <explode_bomb>
```

，完成了比较输入的 6 个数的功能，要求 6 个整数两两不同。

Part3.

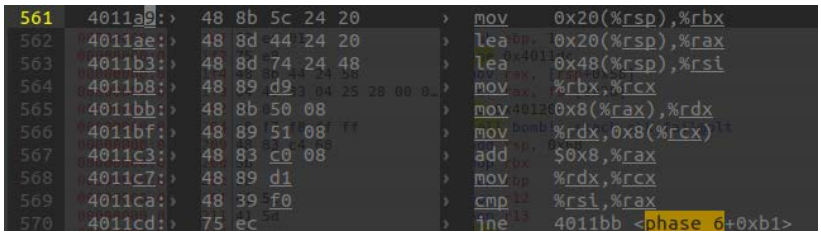
```
401174:  48 8b 52 08       mov    0x8(%rdx),%rdx
401178:  83 c0 01          add    $0x1,%eax
40117b:  39 c8             cmp    %ecx,%eax
40117d:  75 f5             jne    401174 <phase_6+0x6a>
40117f:  48 89 54 74 20    mov    %rdx,0x20(%rsp,%rsi,2)
401184:  48 83 c6 04       add    $0x4,%rsi
401188:  48 83 fe 18       cmp    $0x18,%rsi
40118c:  75 07             jne    401195 <phase_6+0x8b>
40118e:  eb 19             jmp    4011a9 <phase_6+0x9f>
401190:  be 00 00 00 00    mov    $0x0,%esi
401195:  8b 0c 34          mov    (%rsp,%rsi,1),%ecx
401198:  b8 01 00 00 00    mov    $0x1,%eax
40119d:  ba f0 32 60 00    mov    $0x6032f0,%edx
4011a2:  83 f9 01          cmp    $0x1,%ecx
4011a5:  7f cd             jg     401174 <phase_6+0x6a>
4011a7:  eb d6             jmp    40117f <phase_6+0x75>
```

主要功能：可以看出过程中访问的地址是基于链表的，链表形式如下：



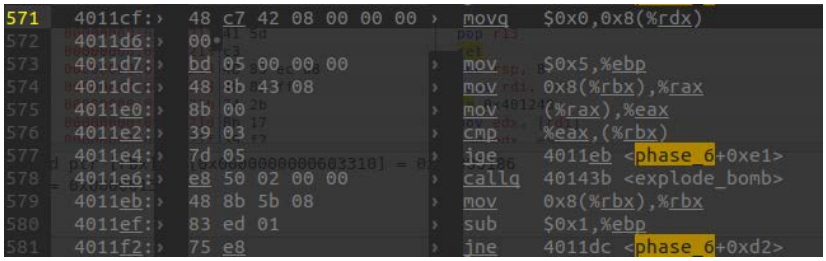
这段代码的主要功能是根据输入的 6 个数进行索引，将链表中对应的值依次放到内存中一段连续的空间，内存地址都在%rsp+0x20~%rsp+0x48。其实相当于实现了一个数组的功能，而输入的 6 个数就相当于这个“链表”数组的下标。

Part4.



主要功能：根据存放在%rsp+0x20~%rsp+0x48 的值重新改变链表中的指针，将指针指向排序之后的后继，从而获得经过排序之后的链表。

Part5.



主要功能：顺序比较，要求经过排序之后的存放在链表的六个数呈单调不减。

Part6.

所以可以总结出程序的目的是要求输入六个整数，这 6 个整数代表着一片内存区域中存放的大小为 6 的链表的索引值，如果能通过索引序列索引值，重新排列生成的数列是单调不增的则不会爆炸。

通过 edb 工具可以轻松获得原来在 0x6032f0-0x603340 存放的 6 个值分别是：0x16c,0xc5,0x186,0x11a,0x1be,0x13c，所以如果按照索引值 5 3 1 6 4 2 重新排列的话就可以使得到的新序列单调不增。

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：

1001

破解过程：

通过查找 secret_phase 在 phase_defused 中发现函数入口

```
40162e:   e8 1f fc ff ff        callq  401252 <secret_phase>
```

寻找进入条件：

发现 dufused 中的逻辑是当执行完 phase_6 之后判断第 4 次的输入中是否存在第三个为%s 的输入，如果存在则与 0x4025d2 存放的 DrEvil 字符串比较，如果比较成功则进入 secret_phase。

```
00000000:004015c1 c3              retq
00000000:004015c2 48 83 ec 78     subq $0x78, %rsp
00000000:004015c6 64 48 8b 04 25 28 00 0…  movq %fs:0x28, %rax
00000000:004015cf 48 89 44 24 68  movq %rax, 0x68(%rsp)
00000000:004015d4 31 c0           xorl %eax, %eax
00000000:004015d6 83 3d af 21 20 00 06  cmpl $6, 0x2021af(%rip)
00000000:004015dd 75 5e           jne 0x40163d
00000000:004015df 4c 8d 44 24 10  leaq 0x10(%rsp), %r8
00000000:004015e4 48 8d 4c 24 0c  leaq 0xc(%rsp), %rcx
00000000:004015e9 48 8d 54 24 08  leaq 8(%rsp), %rdx
00000000:004015ee be c9 25 40 00  movl $0x4025c9, %esi
00000000:004015f3 bf 90 38 60 00  movl $0x603890, %edi
00000000:004015f8 e8 b3 f5 ff ff  callq bomb!__isoc99_sscanf@plt
00000000:004015fd 83 f8 03        cmpl $3, %eax
00000000:00401600 75 31           jne 0x401633
00000000:00401602 be d2 25 40 00  movl $0x4025d2, %esi
00000000:00401607 48 8d 7c 24 10  leaq 0x10(%rsp), %rdi
```

Secret_phase 和 fun7 的代码如下：

```
0000000000401252 <secret_phase>:
  401252:  53                    push   %rbx
  401253:  e8 44 02 00 00        callq  40149c <read_line>
  401258:  ba 0a 00 00 00        mov    $0xa,%edx
  40125d:  be 00 00 00 00        mov    $0x0,%esi
  401262:  48 89 c7              mov    %rax,%rdi
  401265:  e8 26 f9 ff ff        callq  400b90 <strtol@plt>
  40126a:  48 89 c3              mov    %rax,%rbx
  40126d:  8d 40 ff              lea    -0x1(%rax),%eax
  401270:  3d e8 03 00 00        cmp    $0x3e8,%eax
  401275:  76 05                 jbe    40127c <secret_phase+0x2a>
  401277:  e8 bf 01 00 00        callq  40143b <explode_bomb>
  40127c:  89 de                 mov    %ebx,%esi
  40127e:  bf 10 31 60 00        mov    $0x603110,%edi
  401283:  e8 8c ff ff ff        callq  401214 <fun7>
  401288:  83 f8 07              cmp    $0x7,%eax
  40128b:  74 05                 je     401292 <secret_phase+0x40>
  40128d:  e8 a9 01 00 00        callq  40143b <explode_bomb>
  401292:  bf f0 23 40 00        mov    $0x4023f0,%edi
  401297:  e8 44 f8 ff ff        callq  400ae0 <puts@plt>
  40129c:  e8 21 03 00 00        callq  4015c2 <phase_defused>
  4012a1:  5b                    pop    %rbx
  4012a2:  c3                    retq
```

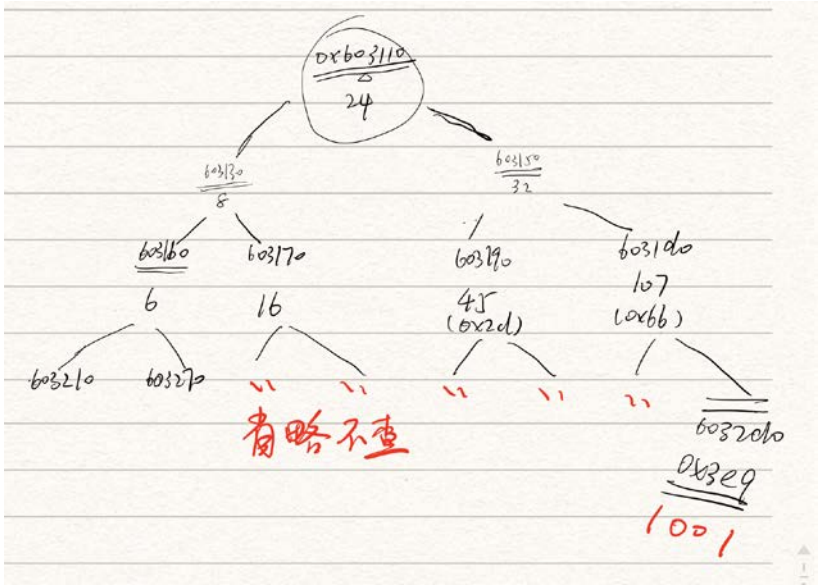
```
0000000000401214 <fun7>:
 401214:  48 83 ec 08     sub    $0x8,%rsp
 401218:  48 85 ff        test   %rdi,%rdi
 40121b:  74 2b           je     401248 <fun7+0x34>
 40121d:  8b 17           mov    (%rdi),%edx
 40121f:  39 f2           cmp    %esi,%edx
 401221:  7e 0d           jle    401230 <fun7+0x1c>
 401223:  48 8b 7f 08     mov    0x8(%rdi),%rdi
 401227:  e8 e8 ff ff ff  callq  401214 <fun7>
 40122c:  01 c0           add    %eax,%eax
 40122e:  eb 1d           jmp    40124d <fun7+0x39>
 401230:  b8 00 00 00 00  mov    $0x0,%eax
 401235:  39 f2           cmp    %esi,%edx
 401237:  74 14           je     40124d <fun7+0x39>
 401239:  48 8b 7f 10     mov    0x10(%rdi),%rdi
 40123d:  e8 d2 ff ff ff  callq  401214 <fun7>
 401242:  8d 44 00 01     lea    0x1(%rax,%rax,1),%eax
 401246:  eb 05           jmp    40124d <fun7+0x39>
 401248:  b8 ff ff ff ff  mov    $0xffffffff,%eax
 40124d:  48 83 c4 08     add    $0x8,%rsp
 401251:  c3              retq
```

输入经过 strol 之后将字符串转化为对应数字存储在 rax 中。

第一个验证：要求 eax-1<=0x3e8。

进入 fun7 之后要求返回值 eax==7，否则爆炸。

观察 fun7 函数结构，发现 fun7 的递归操作是基于如下表示的一棵链表实现的二叉搜索树的。



输入的值就是我们需要在二叉搜索树中查找的值 x，如果 x 比当前节点大进入右节点反之进入左节点，如果进入右节点则 fun7(now_node)=fun7(right_child)*2+1，否则 fun7=(now_node)=fun7(left_child)*2。如果需要满足 fun7==7 则到推出，递归返回结果依次应该是 0->1->3->7，可以看出每次递归走的都是右节点，所以最后的答案应该是 1001。

第 4 章 总结

4.1 请总结本次实验的收获

* 看汇编代码
* 汇编代码的结构、循环、过程的典型反汇编实现
* 耐心

4.2 请给出对本次实验内容的建议

* 如果老师在实验讲解的时候加一些 c 语言中典型语句在汇编中的对应的话对于同学来说可能会更好看懂项目。

注：本章为酌情加分项。

参考文献

为完成本次实验你翻阅的书籍与网站等

[1] 林来兴. 空间控制技术[M]. 北京：中国宇航出版社，1992：25-42.

[2] 辛希孟. 信息技术与信息服务国际研讨会论文集：A 集[C]. 北京：中国科学出版社，1999.

[3] 赵耀东. 新时代的工业工程师[M/OL]. 台北：天下文化出版社，1998 [1998-09-26]. http://www.ie.nthu.edu.tw/info/ie.newie.htm（Big5）.

[4] 谌颖. 空间交会控制理论与方法研究[D]. 哈尔滨：哈尔滨工业大学，1992：8-13.

[5] KANAMORI H. Shaking Without Quaking[J]. Science，1998，279（5359）：2063-2064.

[6] CHRISTINE M. Plant Physiology: Plant Biology in the Genome Era[J/OL]. Science，1998，281：331-332[1998-09-23]. http://www.sciencemag.org/cgi/

collection/anatmorp.