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 计算机科学与技

学　　号 L170300901

班　　级 1703009

学　　生 卢兑玳

指 导 教 师 史先俊

实 验 地 点 G712

实 验 日 期 10. 21

计算机科学与技术学院

目 录

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

GDB/OBJDUMP；EDB；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？

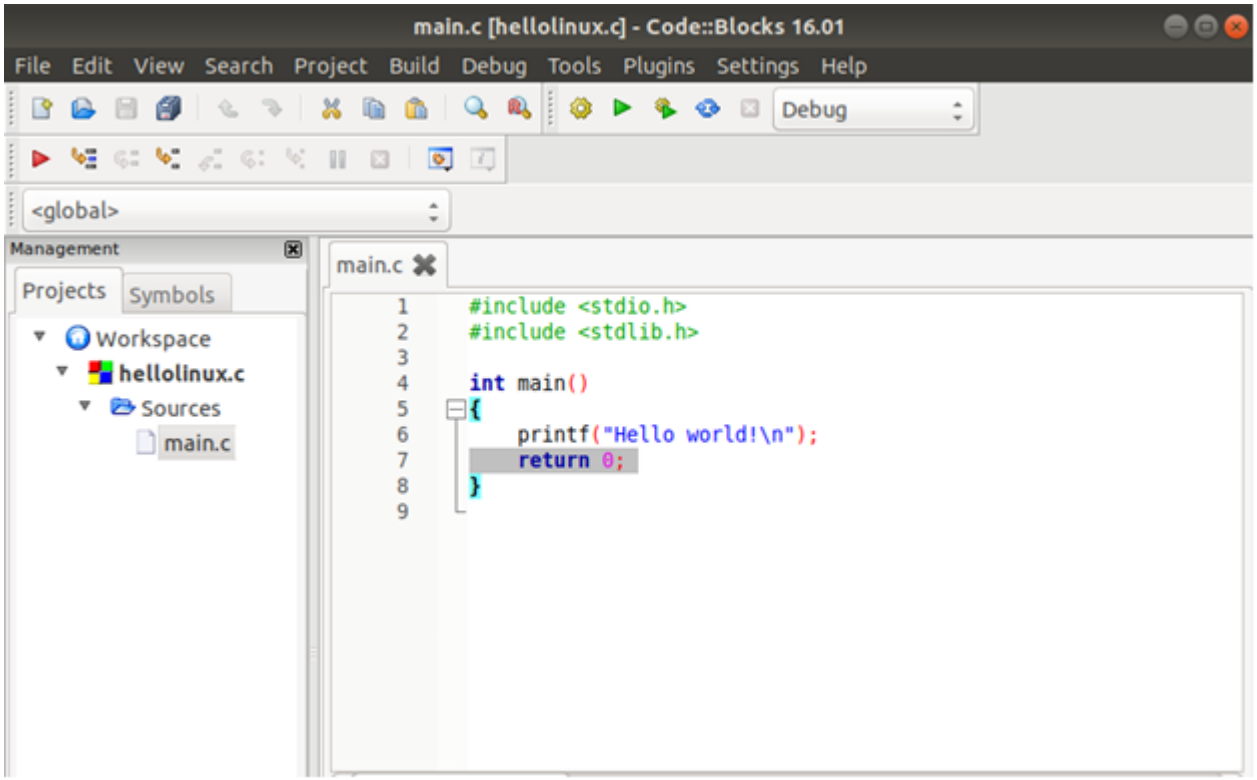
第 2 章 实验环境建立

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。

```
=> 0x555555554639 <main+4>:     lea    rdi,[rip+0x9f]        # 0x5555555546e4
   0x555555554645 <main+11>:    call   0x555555554510 <puts@plt>
   0x55555555464a <main+16>:    mov    eax,0x0
   0x55555555464f <main+21>:    pop    rbp
   0x555555554650 <main+22>:    ret
```



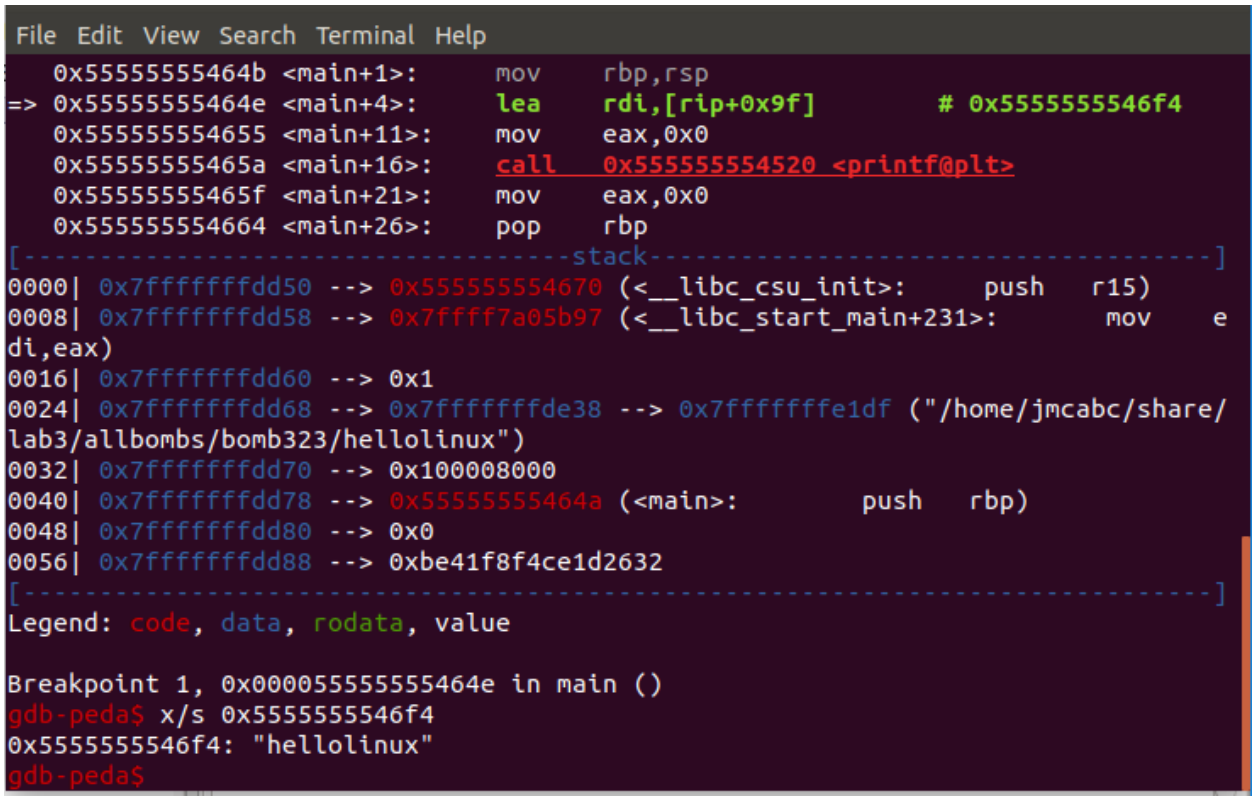


图 2-1　Ubuntu 下 CodeBlocks 反汇编截图

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

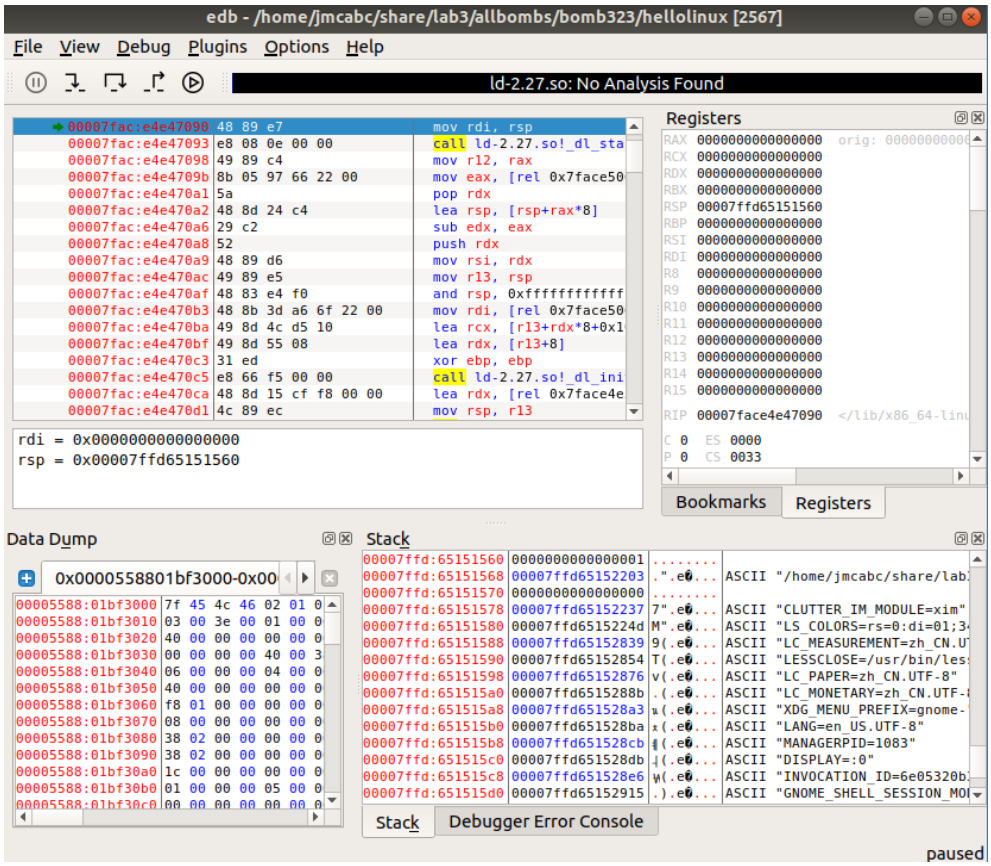


图 2-2　Ubuntu 下 EDB 截图

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：

```
gdb-peda$ x/s 0x555555556660
0x555555556660: "Brownie, you are doing a heck of a job."
gdb-peda$
```

破解过程：

第一阶段相关内容存在于第 phase_1 函数中。 disassemble 如下。用 mov 命令语搬字符串，呼叫函数然后，比较文字序列是否相同。根据这个结果， 决定去向 explode_bomb 函数。因此要和前面移动的字符串一样。 通过 x/s 字排列， 输入球地址， 下面的文字排列出来了。

```
[----------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x0000555555555174 in phase_1 ()
gdb-peda$ x/20i $rip
=> 0x555555555174 <phase_1>:    sub    rsp,0x8
   0x555555555178 <phase_1+4>:  lea    rsi,[rip+0x14e1]        # 0x555555556660
   0x55555555517f <phase_1+11>: call   0x5555555555c0 <strings_not_equal>
   0x555555555184 <phase_1+16>: test   eax,eax
   0x555555555186 <phase_1+18>: jne    0x55555555518d <phase_1+25>
   0x555555555188 <phase_1+20>: add    rsp,0x8
   0x55555555518c <phase_1+24>: ret
   0x55555555518d <phase_1+25>: call   0x5555555556cc <explode_bomb>
   0x555555555192 <phase_1+30>: jmp    0x555555555188 <phase_1+20>
   0x555555555194 <phase_2>:    push   rbp
   0x555555555195 <phase_2+1>:  push   rbx
   0x555555555196 <phase_2+2>:  sub    rsp,0x28
   0x55555555519a <phase_2+6>:  mov    rsi,rsp
   0x55555555519d <phase_2+9>:  call   0x5555555556f2 <read_six_numbers>
   0x5555555551a2 <phase_2+14>: cmp    DWORD PTR [rsp],0x0
   0x5555555551a6 <phase_2+18>: jne    0x5555555551af <phase_2+27>
   0x5555555551a8 <phase_2+20>: cmp    DWORD PTR [rsp+0x4],0x1
   0x5555555551ad <phase_2+25>: je     0x5555555551b4 <phase_2+32>
   0x5555555551af <phase_2+27>: call   0x5555555556cc <explode_bomb>
   0x5555555551b4 <phase_2+32>: mov    rbx,rsp
gdb-peda$ x/s 0x55555556660
0x55555556660:  <error: Cannot access memory at address 0x55555556660>
gdb-peda$ x/s 0x55555555660
0x55555555660:  <error: Cannot access memory at address 0x55555555660>
gdb-peda$ x/s 0x555555556660
0x555555556660: "Brownie, you are doing a heck of a job."
gdb-peda$
```

3.2 阶段 2 的破解与分析

密码如下：

```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
```

破解过程：

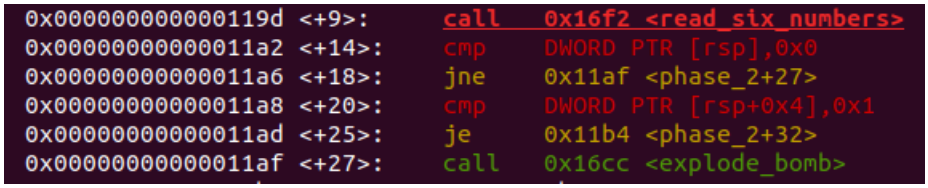
观察第 2 阶段的 phase_2 的函数，就如同下面的照片。

```
gdb-peda$ pdisas phase_2
Dump of assembler code for function phase_2:
   0x0000000000001194 <+0>:     push   rbp
   0x0000000000001195 <+1>:     push   rbx
   0x0000000000001196 <+2>:     sub    rsp,0x28
   0x000000000000119a <+6>:     mov    rsi,rsp
   0x000000000000119d <+9>:     call   0x16f2 <read_six_numbers>
   0x00000000000011a2 <+14>:    cmp    DWORD PTR [rsp],0x0
   0x00000000000011a6 <+18>:    jne    0x11af <phase_2+27>
   0x00000000000011a8 <+20>:    cmp    DWORD PTR [rsp+0x4],0x1
   0x00000000000011ad <+25>:    je     0x11b4 <phase_2+32>
   0x00000000000011af <+27>:    call   0x16cc <explode_bomb>
   0x00000000000011b4 <+32>:    mov    rbx,rsp
   0x00000000000011b7 <+35>:    lea    rbp,[rbx+0x10]
   0x00000000000011bb <+39>:    jmp    0x11c6 <phase_2+50>
   0x00000000000011bd <+41>:    add    rbx,0x4
   0x00000000000011c1 <+45>:    cmp    rbx,rbp
   0x00000000000011c4 <+48>:    je     0x11d7 <phase_2+67>
   0x00000000000011c6 <+50>:    mov    eax,DWORD PTR [rbx+0x4]
   0x00000000000011c9 <+53>:    add    eax,DWORD PTR [rbx]
   0x00000000000011cb <+55>:    cmp    DWORD PTR [rbx+0x8],eax
   0x00000000000011ce <+58>:    je     0x11bd <phase_2+41>
   0x00000000000011d0 <+60>:    call   0x16cc <explode_bomb>
   0x00000000000011d5 <+65>:    jmp    0x11bd <phase_2+41>
   0x00000000000011d7 <+67>:    add    rsp,0x28
   0x00000000000011db <+71>:    pop    rbx
   0x00000000000011dc <+72>:    pop    rbp
   0x00000000000011dd <+73>:    ret
End of assembler dump.
gdb-peda$
```

在 peda 强调的函数 read_siz_number 跟着，二阶段是按规定规则输入六个数的问题。输入形式按白点表示为空白。ex）123456 =》 1 2 3 4 5 6

在这纸张 2 函数中，大可分为两种。(以 explode_bomb 函数为准)

第一部分



接受输入数量后,根据比较结果,移动至失败函数。 这个比较部分是 DWORD PTR[rsp], 0x0。 如果 DWORD PTR[rsp]的话,就是 rsp 寄存器价格的地址。

与这个比 0。 在这里， 确认是否有 6 个字符串符合输入形式的空白状态。

因为有读六个数字的函数.。 而且如果所有数量为 0，则设定为炸弹爆炸。

第二部分

就是检查输入的数列是否符合算法的数列的反复句程序。 将 phase_2+41 的地址设定为寄存器的价格，再加上 eax 价格,经过比较达到季度. 设定为 mov 的寄存器价格为 rsp => rbp，ebx，eax => 1

1번째 반복	2번째 반복	3번째 반복
ebx = 1 eax = 1 eax = rbp[0] +1	eax = 2 eax = rbp[1] +2	eax = 3 eax = rbp[2] +3
cmp(비교) rbp[0x4] == eax? => 입력의 첫 번째값에 1을 더한 값과 두 번째 값이 같은지 비교 if 같다. : 아래 분기문 else 다르다. : 폭탄, 루틴에 맞지 않음	cmp(비교) rbp[0x8] == eax? => 입력의 두 번째 값에 2을 더한 값과 세 번째 값이 같은지 비교 if 같다. : 아래 분기문 else 다르다. : 폭탄, 루틴에 맞지 않음	cmp(비교) rbp[0x8] == eax? => 입력의 세 번째 값에 3을 더한 값과 네 번째 값이 같은지 비교 if 같다. : 아래 분기문 else 다르다. : 폭탄, 루틴에 맞지 않음
ebx = 2 rbp = 4, 입력값의 다음을 가르키도록	ebx = 3 rbp = 8, 입력값의 다음을 가르키도록	ebx = 4 rbp = 8, 입력값의 다음을 가르키도록
cmp(비교) if ebx == 6 : 비교가 끝났으니 종료 else ebx != 6 : 위 함수로 복귀	cmp(비교) if ebx == 6 : 비교가 끝났으니 종료 else ebx != 6 : 위 함수로 복귀	cmp(비교) if ebx == 6 : 비교가 끝났으니 종료 else ebx != 6 : 위 함수로 복귀

将随上表上的例证执行。 如果把输入值设为 1，2，3，4，5，6 然后进行调试，则根据应输入的例程，数列应为 0 1 1 2 3 5 。

```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
```

3.3 阶段 3 的破解与分析

密码如下：

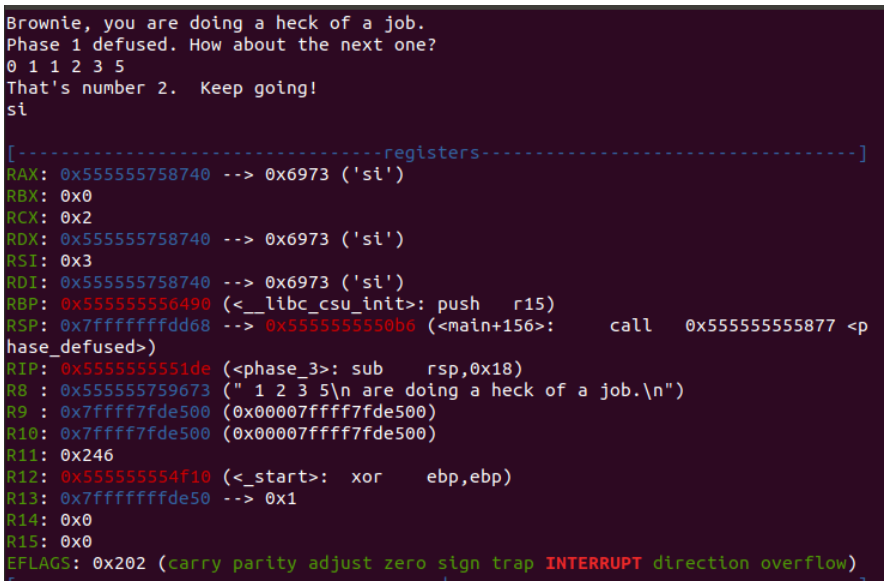
```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!

3 199
Halfway there!
```

破解过程：

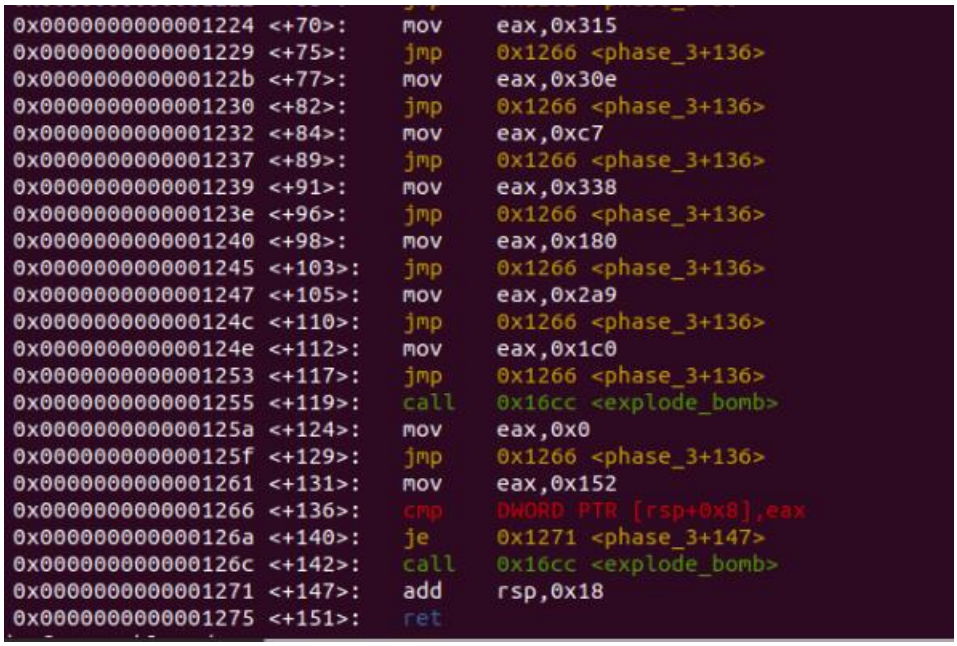
下图是三级函数的 disassemble。

```
l170300901@l170300901-VirtualBox: ~/share/bomb_L170300901/bomb104
파일(F)  편집(E)  보기(V)  검색(S)  터미널(T)  도움말(H)
gdb-peda$ pdisas phase_3
Dump of assembler code for function phase_3:
   0x00000000000011de <+0>:     sub    rsp,0x18
   0x00000000000011e2 <+4>:     lea    rcx,[rsp+0x8]
   0x00000000000011e7 <+9>:     lea    rdx,[rsp+0xc]
   0x00000000000011ec <+14>:    lea    rsi,[rip+0x160c]        # 0x27ff
   0x00000000000011f3 <+21>:    mov    eax,0x0
   0x00000000000011f8 <+26>:    call   0xe60 <__isoc99_sscanf@plt>
   0x00000000000011fd <+31>:    cmp    eax,0x1
   0x0000000000001200 <+34>:    jle    0x121d <phase_3+63>
   0x0000000000001202 <+36>:    cmp    DWORD PTR [rsp+0xc],0x7
   0x0000000000001207 <+41>:    ja     0x1255 <phase_3+119>
   0x0000000000001209 <+43>:    mov    eax,DWORD PTR [rsp+0xc]
   0x000000000000120d <+47>:    lea    rdx,[rip+0x14ac]        # 0x26c0
   0x0000000000001214 <+54>:    movsxd rax,DWORD PTR [rdx+rax*4]
   0x0000000000001218 <+58>:    add    rax,rdx
   0x000000000000121b <+61>:    jmp    rax
   0x000000000000121d <+63>:    call   0x16cc <explode_bomb>
   0x0000000000001222 <+68>:    jmp    0x1202 <phase_3+36>
   0x0000000000001224 <+70>:    mov    eax,0x315
   0x0000000000001229 <+75>:    jmp    0x1266 <phase_3+136>
   0x000000000000122b <+77>:    mov    eax,0x30e
   0x0000000000001230 <+82>:    jmp    0x1266 <phase_3+136>
   0x0000000000001232 <+84>:    mov    eax,0xc7
   0x0000000000001237 <+89>:    jmp    0x1266 <phase_3+136>
   0x0000000000001239 <+91>:    mov    eax,0x338
   0x000000000000123e <+96>:    jmp    0x1266 <phase_3+136>
   0x0000000000001240 <+98>:    mov    eax,0x180
   0x0000000000001245 <+103>:   jmp    0x1266 <phase_3+136>
   0x0000000000001247 <+105>:   mov    eax,0x2a9
   0x000000000000124c <+110>:   jmp    0x1266 <phase_3+136>
   0x000000000000124e <+112>:   mov    eax,0x1c0
   0x0000000000001253 <+117>:   jmp    0x1266 <phase_3+136>
   0x0000000000001255 <+119>:   call   0x16cc <explode_bomb>
   0x000000000000125a <+124>:   mov    eax,0x0
   0x000000000000125f <+129>:   jmp    0x1266 <phase_3+136>
   0x0000000000001261 <+131>:   mov    eax,0x152
   0x0000000000001266 <+136>:   cmp    DWORD PTR [rsp+0x8],eax
   0x000000000000126a <+140>:   je     0x1271 <phase_3+147>
   0x000000000000126c <+142>:   call   0x16cc <explode_bomb>
   0x0000000000001271 <+147>:   add    rsp,0x18
   0x0000000000001275 <+151>:   ret
```

先把刹车点挂在这个函数上。通过一个一个 ni 命令语来实行吧。



在这个状态下一直执行的话。在本函数中,总比起来有好几个部分是第一部分.将第一个比较的 eax 和 1 进行比较,结果大的话再分期。在这里非得分岔儿.从这里可以看出，eax 就是第一个，第一个数字要大于一。



在 phase_3+70 和 phase_3+75， 所以前两排部分将第一价移到 eax，根据 eax 的价格进行跳跃。 然后从下开始调价跳跃， 移动到 phase_3+136。 这是将第二笔输入值和 eax 进行比较,如果不同就会引爆炸弹的形式。

那么,如果在这里问到什么是 eax 的话,价格转移,跳跃的部分共有 7 个.

刚才的首批输入值是 1 到 7,根据各值移动到相应位置,进入在 eax 值中按代码输入后,再进入是否符合相应价格的比较位置进行比较,然后爆炸.

决定与否

如放入 3， 则可到，达第三部分， eax 要花 c7 的价格。

因此,c7 应加入转换为 10 进位数的值。

实行时，放进去，会通过阶段，出现下一阶段的指纹．当然,这个问题的答案有 7 个．根据第一价的不同,有 7 种正确答案．

```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!

3 199
Halfway there!
```

3.4 阶段 4 的破解与分析

密码如下：

```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
3 199
Halfway there!
264 3
So you got that one.  Try this one.
```

破解过程：

下图是 4 级函数的 disassemble。

```
gdb-peda$ pdisas phase_4
Dump of assembler code for function phase_4:
   0x00000000000012af <+0>:     sub    rsp,0x18
   0x00000000000012b3 <+4>:     lea    rcx,[rsp+0xc]
   0x00000000000012b8 <+9>:     lea    rdx,[rsp+0x8]
   0x00000000000012bd <+14>:    lea    rsi,[rip+0x153b]        # 0x27ff
   0x00000000000012c4 <+21>:    mov    eax,0x0
   0x00000000000012c9 <+26>:    call   0xe60 <__isoc99_sscanf@plt>
   0x00000000000012ce <+31>:    cmp    eax,0x2
   0x00000000000012d1 <+34>:    jne    0x12df <phase_4+48>
   0x00000000000012d3 <+36>:    mov    eax,DWORD PTR [rsp+0xc]
   0x00000000000012d7 <+40>:    sub    eax,0x2
   0x00000000000012da <+43>:    cmp    eax,0x2
   0x00000000000012dd <+46>:    jbe    0x12e4 <phase_4+53>
   0x00000000000012df <+48>:    call   0x16cc <explode_bomb>
   0x00000000000012e4 <+53>:    mov    esi,DWORD PTR [rsp+0xc]
   0x00000000000012e8 <+57>:    mov    edi,0x9
   0x00000000000012ed <+62>:    call   0x1276 <func4>
   0x00000000000012f2 <+67>:    cmp    DWORD PTR [rsp+0x8],eax
   0x00000000000012f6 <+71>:    je     0x12fd <phase_4+78>
   0x00000000000012f8 <+73>:    call   0x16cc <explode_bomb>
   0x00000000000012fd <+78>:    add    rsp,0x18
   0x0000000000001301 <+82>:    ret
End of assembler dump.
gdb-peda$
```

放 4 2，试试看。前面的 2 相比较的两个后， 在 phase_4+67， 和 a 做比较。前面对比 2 的部分如下。

```
0x00000000000012d3 <+36>:    mov    eax,DWORD PTR [rsp+0xc]
0x00000000000012d7 <+40>:    sub    eax,0x2
0x00000000000012da <+43>:    cmp    eax,0x2
0x00000000000012dd <+46>:    jbe    0x12e4 <phase_4+53>
```

把 eax 和 2 比较过去。 在这里比较 eax 和 2. 输入两种数字,依次输入前后数字后发现是后面数字。 在这里 rsp+0xc 地价格要调到 eax， . eax 和 2 相比，

将进入 jbe (jump on below or equal) 分期跳台。 在这里不跳起来， 就会炸裂。所以 a 某应该比 2 小或者更小。

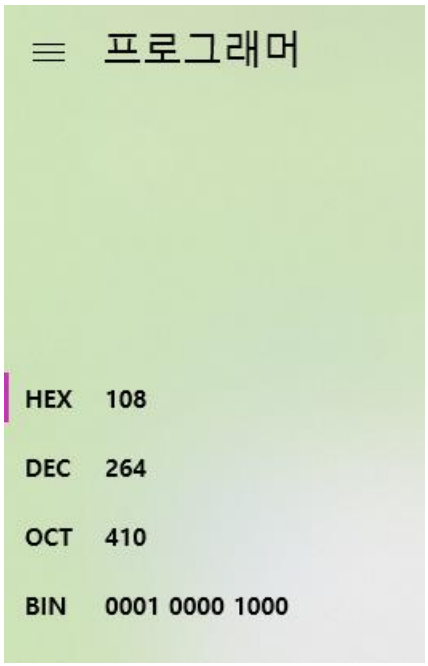
即， eax 值表示第二次输入值。第二个输入值 eax 除去 2,小于 2 的情况如下:

```
x-2 = 0 => x=2

x-2 = 1 => x=3

x-2 = 2 => x=4
```

并根据该输入值,在 func4 函数中更新 eax 的价格。 必须以第一笔输入来支付从这里得出的价格， 事实上,过去,那个函数的寄存器，rax 了价格的结果显示,eax，函数,再确认的值注册就可以了。 这和前问题一样,寄存在寄存器中的价格是 16 进位需，要更换 10 进位数。

```
[----------------------------------registers-----------------------------------]
RAX: 0x108
RBX: 0x0
RCX: 0x0
RDX: 0x7fffffffdd4c --> 0xffffde4000000003
RSI: 0x3
RDI: 0x1
RBP: 0x555555556490 (<__libc_csu_init>: push   r15)
RSP: 0x7fffffffdd40 --> 0x555555556490 (<__libc_csu_init>:      push   r15)
RIP: 0x5555555552f6 (<phase_4+71>:      je     0x5555555552fd <phase_4+78>)
R8 : 0x0
R9 : 0x0
R10: 0x7ffff7b82cc0 --> 0x2000200020002
R11: 0x555555556804 --> 0x203a726f72724500 ('')
R12: 0x555555554f10 (<_start>:  xor    ebp,ebp)
R13: 0x7fffffffde40 --> 0x1
R14: 0x0
R15: 0x0
EFLAGS: 0x283 (CARRY parity adjust zero SIGN trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x5555555552e8 <phase_4+57>: mov    edi,0x9
   0x5555555552ed <phase_4+62>: call   0x555555555276 <func4>
   0x5555555552f2 <phase_4+67>: cmp    DWORD PTR [rsp+0x8],eax
=> 0x5555555552f6 <phase_4+71>: je     0x5555555552fd <phase_4+78>
   0x5555555552f8 <phase_4+73>: call   0x5555555556cc <explode_bomb>
   0x5555555552fd <phase_4+78>: add    rsp,0x18
   0x555555555301 <phase_4+82>: ret
   0x555555555302 <phase_5>:    push   rbx
                                                          JUMP is NOT taken
[-------------------------------------stack-------------------------------------]
0000| 0x7fffffffdd40 --> 0x555555556490 (<__libc_csu_init>:     push   r15)
0008| 0x7fffffffdd48 --> 0x30000006c
0016| 0x7fffffffdd50 --> 0x7fffffffde40 --> 0x1
0024| 0x7fffffffdd58 --> 0x5555555550d4 (<main+186>:    call   0x555555555877 <phase_defused>)
0032| 0x7fffffffdd60 --> 0x0
0040| 0x7fffffffdd68 --> 0x7ffff7a05b97 (<__libc_start_main+231>:       mov    edi,eax)
0048| 0x7fffffffdd70 --> 0x1
0056| 0x7fffffffdd78 --> 0x7fffffffde48 --> 0x7fffffffe1df ("/home/l170300901/share/bomb_L170300901/bomb104/bom
b")
[------------------------------------------------------------------------------]
Legend: code, data, rodata, value
0x00005555555552f6 in phase_4 ()
gdb-peda$ q
```

转换的价格如下。



假定以下数字为第一个输入值,第二个输入值时

264, 3

```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
3 199
Halfway there!
264 3
So you got that one.  Try this one.
```

3.5 阶段 5 的破解与分析

密码如下：

```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
3 199
Halfway there!
264 3
So you got that one.  Try this one.
y_a@ew
Good work!  On to the next...
```

破解过程：

下图是 5 级函数的 disassemble。

```
gdb-peda$ pdisas phase_5
Dump of assembler code for function phase_5:
   0x0000000000001302 <+0>:     push   rbx
   0x0000000000001303 <+1>:     sub    rsp,0x10
   0x0000000000001307 <+5>:     mov    rbx,rdi
   0x000000000000130a <+8>:     call   0x15a3 <string_length>
   0x000000000000130f <+13>:    cmp    eax,0x6
   0x0000000000001312 <+16>:    jne    0x1359 <phase_5+87>
   0x0000000000001314 <+18>:    mov    eax,0x0
   0x0000000000001319 <+23>:    lea    rcx,[rip+0x13c0]        # 0x26e0 <array.3413>
   0x0000000000001320 <+30>:    movzx  edx,BYTE PTR [rbx+rax*1]
   0x0000000000001324 <+34>:    and    edx,0xf
   0x0000000000001327 <+37>:    movzx  edx,BYTE PTR [rcx+rdx*1]
   0x000000000000132b <+41>:    mov    BYTE PTR [rsp+rax*1+0x9],dl
   0x000000000000132f <+45>:    add    rax,0x1
   0x0000000000001333 <+49>:    cmp    rax,0x6
   0x0000000000001337 <+53>:    jne    0x1320 <phase_5+30>
   0x0000000000001339 <+55>:    mov    BYTE PTR [rsp+0xf],0x0
   0x000000000000133e <+60>:    lea    rdi,[rsp+0x9]
   0x0000000000001343 <+65>:    lea    rsi,[rip+0x1364]        # 0x26ae
   0x000000000000134a <+72>:    call   0x15c0 <strings_not_equal>
   0x000000000000134f <+77>:    test   eax,eax
   0x0000000000001351 <+79>:    jne    0x1360 <phase_5+94>
   0x0000000000001353 <+81>:    add    rsp,0x10
   0x0000000000001357 <+85>:    pop    rbx
   0x0000000000001358 <+86>:    ret
   0x0000000000001359 <+87>:    call   0x16cc <explode_bomb>
   0x000000000000135e <+92>:    jmp    0x1314 <phase_5+18>
   0x0000000000001360 <+94>:    call   0x16cc <explode_bomb>
   0x0000000000001365 <+99>:    jmp    0x1353 <phase_5+81>
End of assembler dump.
```

```
gdb-peda$ x/s 0x5555555566e0
0x5555555566e0 <array.3413>:    "maduiersnfotvbylSo you think you can stop the bomb with ctrl-c, do you?"
gdb-peda$ x/s 0x5555555566ae
0x5555555566ae: "flames"
```

可以确认有特定文字组合。 此布文末。如果从 Asky Code 中找到这个短信，

可以知道是 y_a@ew。

```
l170300901@l170300901-VirtualBox:~/share/bomb_L170300901/bomb104$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Brownie, you are doing a heck of a job.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2.  Keep going!
3 199
Halfway there!
264 3
So you got that one.  Try this one.
y_a@ew
Good work!  On to the next...
```

3.6 阶段 6 的破解与分析

密码如下：

破解过程：

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下：

破解过程：

第 4 章 总结

4.1 请总结本次实验的收获

我是韩国留学生. 因此用中文学习这个课程是非常困难的. 但是很多中国朋友告诉我和帮助我了解了课堂内容 所以我提高了很多汉语水平,也了解了很多 linux. 对我来说,这似乎是一次很好的经验,真的很幸福

4.2 请给出对本次实验内容的建议

我是留学生 。请多多关照。 谢谢老师

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.