实验 3　X86 汇编语言

本实验目标:

1. 学会使用 GDB 工具进行反向工程。
2. 学会理解汇编代码逻辑。

二进制炸弹

二进制炸弹是一个可执行文件，它有若干“炸弹”函数，每个炸弹函数接受一行输入，如果这行输入不符合要求，就会“爆炸”，

首先解压:

$ tar -xvf bomb.tar

然后切到 bomb 目录下:

$ cd bomb

本实验提供一个压缩包，里面包括可执行文件 bomb，和一个不完整的 c 文件 bomb.c 。

运行 boom:

$ ./bomb

如果输入的字符串或者数字与要求的不匹配，程序会显示:

```
BOOM!!!
The bomb has blown up.
```

然后程序结束运行，并退出。

查看 bomb.c 文件你可以看到它调用了六个函数, phase_1 到 phase-6 。每调用一个函数，都会判断当前输入的这一行输入，是否是期望的输入，如果是，就会进入下一阶段。例如:

```
shiyanlou:bomb/ $ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Border relations with Canada have never been better.
Phase 1 defused. How about the next one?
```

如果第一行的输入是: “ Border relations with Canada have never been better.“ 就通过了 phase_1，进入到下一阶段。 你总共需要输入七行正确的输入，才能顺利通过六个阶段的测试和一个隐藏测试，不引发炸弹爆炸。

由于 bomb.c 不完整，我们只能看到 main 函数的源代码了，却无法看到其他子程序的源代码。我们无法从源代码判断 phase_1 到 phase_6 应该输入的正确答案。

但我们可以通过 GDB，对可执行代码进行逆向工程，解析出其汇编代码。通过查看汇编代码，找出正确的输入，依次拆除若干个炸弹。

下面开始我们的实验:

然后就可以使用 GDB 进行逆向工程:

$ gdb -q bomb
你就可以看到已经进入了调试模式，你可以使用一些命令进行解析代码，如：
$ disas phase_1
就可以解析出 phase_1 函数的汇编代码。

```
shiyanlou:bomb/ $ gdb -q bomb
Reading symbols from bomb...done.
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    $0x8,%rsp
   0x0000000000400ee4 <+4>:     mov    $0x402400,%esi
   0x0000000000400ee9 <+9>:     callq  0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   %eax,%eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    callq  0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    $0x8,%rsp
   0x0000000000400efb <+27>:    retq
End of assembler dump.
```

同样的方法，可以依次解析出各个子函数的汇编代码。

另外，你也使用另外一个工具 objdump 进行反汇编，在本实验我们就不介绍了。
为了降低实验难度， 你可以查看网络上的资料，了解解题思路。
参考博客：https://www.jianshu.com/p/8de4e4641245
我们还给出了标准答案，答案所在文件为 bomb-solution.txt。有助于你阅读汇编代码和回答问题。如果你运行 $ ./bomb bomb-solution.txt 你会发现炸弹没有爆炸，安全运行到程序结束。

你的实验报告中，要求回答的问题只涉及前三个阶段，后面的四个阶段，在本次作业不要求，你可以通过课后自学完成。

要求：在实验报告中回答以下问题。回答问题时，使用文字说明，必要时可结合代码或者图片形式，表明主要逻辑即可。


```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400ee0 <+0>:     sub    $0x8,%rsp
   0x0000000000400ee4 <+4>:     mov    $0x402400,%esi
   0x0000000000400ee9 <+9>:     callq  0x401338 <strings_not_equal>
   0x0000000000400eee <+14>:    test   %eax,%eax
   0x0000000000400ef0 <+16>:    je     0x400ef7 <phase_1+23>
   0x0000000000400ef2 <+18>:    callq  0x40143a <explode_bomb>
   0x0000000000400ef7 <+23>:    add    $0x8,%rsp
   0x0000000000400efb <+27>:    retq
End of assembler dump.
```

1. 在 phase_1 中，地址____？____处存放的是用于和终端输入相比较的字符串（的首地址 ）
2. 在 gdb 中，用什么命令可以查看该字符串的内容?
3. 指令 test %eax, %eax 的作用是什么?

$ disas phase_2

解析出 phase_2 函数的汇编代码

```
Dump of assembler code for function phase_2:
   0x0000000000400efc <+0>:     push   %rbp
   0x0000000000400efd <+1>:     push   %rbx
   0x0000000000400efe <+2>:     sub    $0x28,%rsp
   0x0000000000400f02 <+6>:     mov    %rsp,%rsi
   0x0000000000400f05 <+9>:     callq  0x40145c <read_six_numbers>
   0x0000000000400f0a <+14>:    cmpl   $0x1,(%rsp)
   0x0000000000400f0e <+18>:    je     0x400f30 <phase_2+52>
   0x0000000000400f10 <+20>:    callq  0x40143a <explode_bomb>
   0x0000000000400f15 <+25>:    jmp    0x400f30 <phase_2+52>
   0x0000000000400f17 <+27>:    mov    -0x4(%rbx),%eax
   0x0000000000400f1a <+30>:    add    %eax,%eax
   0x0000000000400f1c <+32>:    cmp    %eax,(%rbx)
   0x0000000000400f1e <+34>:    je     0x400f25 <phase_2+41>
   0x0000000000400f20 <+36>:    callq  0x40143a <explode_bomb>
   0x0000000000400f25 <+41>:    add    $0x4,%rbx
   0x0000000000400f29 <+45>:    cmp    %rbp,%rbx
   0x0000000000400f2c <+48>:    jne    0x400f17 <phase_2+27>
   0x0000000000400f2e <+50>:    jmp    0x400f3c <phase_2+64>
   0x0000000000400f30 <+52>:    lea    0x4(%rsp),%rbx
   0x0000000000400f35 <+57>:    lea    0x18(%rsp),%rbp
   0x0000000000400f3a <+62>:    jmp    0x400f17 <phase_2+27>
   0x0000000000400f3c <+64>:    add    $0x28,%rsp
---Type <return> to continue, or q <return> to quit---
   0x0000000000400f40 <+68>:    pop    %rbx
   0x0000000000400f41 <+69>:    pop    %rbp
   0x0000000000400f42 <+70>:    retq
End of assembler dump.
```

回答问题：

4. 指令 mov -0x4(%rbx), %eax, 这条指令完成的功能是什么?
5. 为了比较输入的六个整数是否是预埋的答案，phase_2 函数从哪里开始循环？循环的出口在哪里?

6. `0x0000000000400f0a <+14>:    cmpl   $0x1,(%rsp)` 从这条指令我们可以看出，%rsp 中存储了第一个输入数据的地址，即%rsp 指向第一输入的整数，那么第 6 个输入的整数，存放在什么位置？

$ disas phase_3

解析出 phase_3 函数的汇编代码

```
Dump of assembler code for function phase_3:
   0x0000000000400f43 <+0>:     sub    $0x18,%rsp
   0x0000000000400f47 <+4>:     lea    0xc(%rsp),%rcx
   0x0000000000400f4c <+9>:     lea    0x8(%rsp),%rdx
   0x0000000000400f51 <+14>:    mov    $0x4025cf,%esi
   0x0000000000400f56 <+19>:    mov    $0x0,%eax
   0x0000000000400f5b <+24>:    callq  0x400bf0 <__isoc99_sscanf@plt>
   0x0000000000400f60 <+29>:    cmp    $0x1,%eax
   0x0000000000400f63 <+32>:    jg     0x400f6a <phase_3+39>
   0x0000000000400f65 <+34>:    callq  0x40143a <explode_bomb>
   0x0000000000400f6a <+39>:    cmpl   $0x7,0x8(%rsp)
   0x0000000000400f6f <+44>:    ja     0x400fad <phase_3+106>
   0x0000000000400f71 <+46>:    mov    0x8(%rsp),%eax
   0x0000000000400f75 <+50>:    jmpq   *0x402470(,%rax,8)
   0x0000000000400f7c <+57>:    mov    $0xcf,%eax
   0x0000000000400f81 <+62>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f83 <+64>:    mov    $0x2c3,%eax
   0x0000000000400f88 <+69>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f8a <+71>:    mov    $0x100,%eax
   0x0000000000400f8f <+76>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f91 <+78>:    mov    $0x185,%eax
   0x0000000000400f96 <+83>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f98 <+85>:    mov    $0xce,%eax
---Type <return> to continue, or q <return> to quit---
```

```
   0x0000000000400f9d <+90>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400f9f <+92>:    mov    $0x2aa,%eax
   0x0000000000400fa4 <+97>:    jmp    0x400fbe <phase_3+123>
   0x0000000000400fa6 <+99>:    mov    $0x147,%eax
   0x0000000000400fab <+104>:   jmp    0x400fbe <phase_3+123>
   0x0000000000400fad <+106>:   callq  0x40143a <explode_bomb>
   0x0000000000400fb2 <+111>:   mov    $0x0,%eax
   0x0000000000400fb7 <+116>:   jmp    0x400fbe <phase_3+123>
   0x0000000000400fb9 <+118>:   mov    $0x137,%eax
   0x0000000000400fbe <+123>:   cmp    0xc(%rsp),%eax
   0x0000000000400fc2 <+127>:   je     0x400fc9 <phase_3+134>
   0x0000000000400fc4 <+129>:   callq  0x40143a <explode_bomb>
   0x0000000000400fc9 <+134>:   add    $0x18,%rsp
   0x0000000000400fcd <+138>:   retq
End of assembler dump.
```

回答问题：

7. 第三行的输入 无论是“4 389“ 还是”1 311“，都是正确的，炸弹都不会爆炸，说明这给函数有多个正确答案。你能再罗列两个吗？

8. 这条指令的作用是什么？

0x0000000000400f75 <+50>: jmpq *0x402470(,%rax,8)

提示：分别试试

（gdb） x/x 0x402470
（gdb） x/x 0x402478
（gdb） x/x 0x402480
（gdb） x/x 0x402488
（gdb） x/x 0x402490
（gdb） x/x 0x402498
（gdb） x/x 0x4024A0

9. phase_3 中体现的是 switch 语句逻辑，还是 if-else 语句逻辑？

注：
GDB 查看内存(x 命令) 的解释，可以参考这里：
https://www.cnblogs.com/adamwong/p/10538019.html

本实验选自 CMU CASPP Bomb lab (二进制炸弹实验)
http://csapp.cs.cmu.edu/3e/labs.html