一、实验目的

1. 理解C语言程序的机器级表示。
2. 初步掌握GDB调试器的用法。
3. 阅读C编译器生成的x86-64机器代码，理解不同控制结构生成的基本指令模式，过程的实现。

二、 实验环境

1. Remote Terminal（1.9.2.24）
2. Linux
3. Objdump命令反汇编
4. GDB调试工具
5. Visual Studio Code

三、实验内容

登录bupt1服务器，在home目录下可以找到Evil博士专门为你量身定制的一个bomb，当运行时，它会要求你输入一个字符串，如果正确，则进入下一关，继续要求你输入下一个字符串；否则，炸弹就会爆炸，输出一行提示信息并向计分服务器提交扣分信息。因此，本实验要求你必须通过反汇编和逆向工程对bomb执行文件进行分析，找到正确的字符串来解除这个的炸弹。

本实验通过要求使用课程所学知识拆除一个“binary bombs”来增强对程序的机器级表示、汇编语言、调试器和逆向工程等方面原理与技能的掌握。 “binary bombs”是一个Linux可执行程序，包含了5个阶段（或关卡）。炸弹运行的每个阶段要求你输入一个特定字符串，你的输入符合程序预期的输入，该阶段的炸弹就被拆除引信；否则炸弹“爆炸”，打印输出 “BOOM!!!”。炸弹的每个阶段考察了机器级程序语言的一个不同方面，难度逐级递增。

为完成二进制炸弹拆除任务，需要使用gdb调试器和objdump来反汇编bomb文件，可以单步跟踪调试每一阶段的机器代码，也可以阅读反汇编代码，从中理解每一汇编语言代码的行为或作用，进而设法推断拆除炸弹所需的目标字符串。实验2的具体内容见实验2说明。

四、 实验步骤及实验分析

1. 准备工作

   使用`objdump`反汇编bomb文件，并将其保存为`asm.txt`。

   ```
   2018210547@bupt1:~/bomb459$ objdump -d bomb > asm.txt
   ```

2. phase_1

   搜索`phase_1`

   由`400f36`可以看出，函数进入了`<strings_not_equal>`,搜索`<strings_not_equal>`

   不难看出，该函数比较两个字符串是否相等，不等返回1，相等返回0

   我们输入的字符串被用于和存放在`0x402670`的字符串比较，说明该地址存有第一阶段的密码

   在gdb中打印该地址的值，找到密码

```
(gdb) x/s 0x402670
0x402670:       "In 2004, BUPT became one of the 56 universities which have
graduate school."
```

密码为

```
In 2004, BUPT became one of the 56 universities which have graduate school.
```

3. phase_2

   搜索`phase_2`

   由`400f62`，该函数进入了一个名为`<read_six_numbers>`的函数，搜索`<read_six_numbers>`。

   读取了六个int型整数，因此我们要输入六个整数，且首地址为`%rsp`。

   由`400f67`,若第一个数字不大于等于零，就会爆炸，所以`a[0]>=0`。

   `400f75`给`%ebx`赋初值1，然后进入`400f7a`~`400f93`的循环，其中由`400f7a`~`400f7f`可知，内存位置相差为4，即相邻的两个数字之间的差必须为`%ebx`。即相邻两数字之间的差为1，2，3，4，5。

   令第一个数字为0，可得密码序列为

   ```
   0 1 3 6 10 15
   ```

4. phase_3

   搜索`phase_3`

   由`400fcd`，在gdb中打印`0x4029cd`的值

   ```
   (gdb) x/s 0x4029cd
   0x4029cd:       "%d %d"
   ```

   输入的是两个int型整数，由`400fc5` `400fca`可知两个整数d1，d2分别存放在`%rsp`，`%rsp+4`中。

   由`400fe1`，若d1>7则跳转入爆炸函数，故d1<7。

   `400fea`是一个swtich（d1）的语句，用一个数组储存各个case标号的地址。在gdb中打印

   ```
   (gdb) x/8gx *0x402700
   0x400ff1 <phase_3+64>:  0xb805eb00000200b8      0x0002872d00000000

   0x401001 <phase_3+80>:  0x00000000b805eb00      0xb805eb0000015e05

   0x401011 <phase_3+96>:  0xeb78e88300000000      0xc08300000000b805

   0x401021 <phase_3+112>: 0x00000000b805eb78      0x0000b805eb78e883
   ```

分别是case 0 ~ case7语句所在位置。分析switch中的语句，发现每一个case内的jump都指向下一个case中的第二条语句，得知该switch不含break。一旦满足case条件，就要一直执行到switch结束。

同时分析`401052`可知，计算出的`%eax`必须等于d2才不会爆炸。

又由`40104c`可知，若d1>5则爆炸，故d1的取值为0，1，2，3，4，5.

$$
\begin{gathered}
d1=5, d2=-200+200-200=-200 \\
d1=4, d2=200-200=0 \\
d1=3, d2=200+0=200 \\
d1=2, d2=350+200=550 \\
d1=1, d2=-647+550=-97 \\
d1=0, d2=200-97=103
\end{gathered}
$$

所以，密码的所有情况为

$$
(d1, d2) \in \{(0,103),(1,-97),(2,550),(3,200),(4,0),(5,-200)\}
$$

5. phase_4

搜索`phase_4`

函数开头的读入与`phase_3`类似，读入两个int型整数d1，d2，分别在`%rsp`与`%rsp+4`中。

`4010e0`要求d1<=14，否则爆炸。

`4010eb`~`4010f5`中，给函数`func4`传参，其中`x=14,y=0,z=d1`

搜索`func4`

分析func4的逻辑（写在备注中），发现是一个递归的二分操作，写出对应的c++代码

```
#include <cstdio>
#define FOR(_i,_a,_b) for (Reg int _i = (_a) ; _i <= (_b) ; _i++)
using namespace std;

int func (int x, int y, int z){
  int mid = (x - y) / 2 + y, t;
  if (mid <= z) {
    t = 0;
    if (mid >= z) return t;
    else{
      y = mid + 1;
      t = func(x, y, z);
      return 2 * t + 1;
    }
  }
  else {
    x = mid - 1;
    t = func(x, y, z);
    return  t * 2;
  }
  return 0;
}
int main() {
  FOR (i, 0, 14){
```

```
    printf ("i=%d func=%d\n", i, func(14, 0, i));
  }
  return 0;
}
```

运行结果为

```
i=0 func=0
i=1 func=0
i=2 func=4
i=3 func=0
i=4 func=2
i=5 func=2
i=6 func=6
i=7 func=0
i=8 func=1
i=9 func=1
i=10 func=5
i=11 func=1
i=12 func=3
i=13 func=3
i=14 func=7
```

由`4010fd`可知，返回值func=0便不会爆炸，则d1可取的值为0，1，7

又由`401011`可知，d2=0便不会爆炸

则密码的所有可能取值为为

$$(d1, d2) \in \{(0,0),(0,1),(0,7)\}$$

6. phase_5

搜索`phase_5`

通过`<string_length>`求得字符串长度，`401133`表明字符串长度必须为6，否则发生爆炸。

`40113d`求出字符串结尾的地址，存入`%rdi`，作为循环结束的标志

在循环体内，每次将当前循环到的字符对应的ASCII码后四位取出（也就是mod16），以该值作为偏移量，`0x402740`作为基址，得到对应地址的数值，加到累加器`%rax`上。`40115f`表示这个和必须等于48，才不会爆炸。

查看`0x402740`处后16个数的值

```
(gdb) print *0x402740@16

$1 = {2, 10, 6, 1, 12, 16, 9, 3, 4, 7, 14, 5, 11, 8, 15, 13}
```

考虑从中挑选6个数构造处48

$$48 = 2 + 2 + 10 + 6 + 12 + 16 = a[0] + a[0] + a[1] + a[2] + a[4] + a[5]$$

于是只要求出$c \equiv k(mod\ 16)$,其中 k为数组a的下标，c为对应满足条件的字符的ASCII码

利用C++在小写字母内寻找

```
int main() {
    FOR (i, 0, 25){
        char a = 'a' + i;
        int x = a % 16;
        printf ("%c %d\n", a, x);
    }
}
```

输出：

```
a 1
b 2
c 3
d 4
e 5
f 6
g 7
h 8
i 9
j 10
k 11
l 12
m 13
n 14
o 15
p 0
q 1
r 2
s 3
t 4
u 5
v 6
w 7
x 8
y 9
z 10
```

构造字符串：

```
ppabde
```

即为密码。