二进制炸弹 实验报告

16307130006 陈幸豪

1. 准备工作

连接校园网内网，在Ubuntu内打开terminal，运行：

scp -r stu16307130006@10.48.76.8:~/bomblab ~/bomblabrev

输入密码，回车，即从服务器取回炸弹。解压bomb75.tar得到可执行文件bomb等。在bomb所在目录打开terminal，执行：

objdump -d bomb > bomb.asm

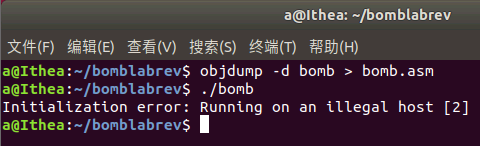
生成bomb.asm。打开bomb.asm，发现内容实在很多。换一种方法，直接执行：

./bomb

试图运行bomb。结果为

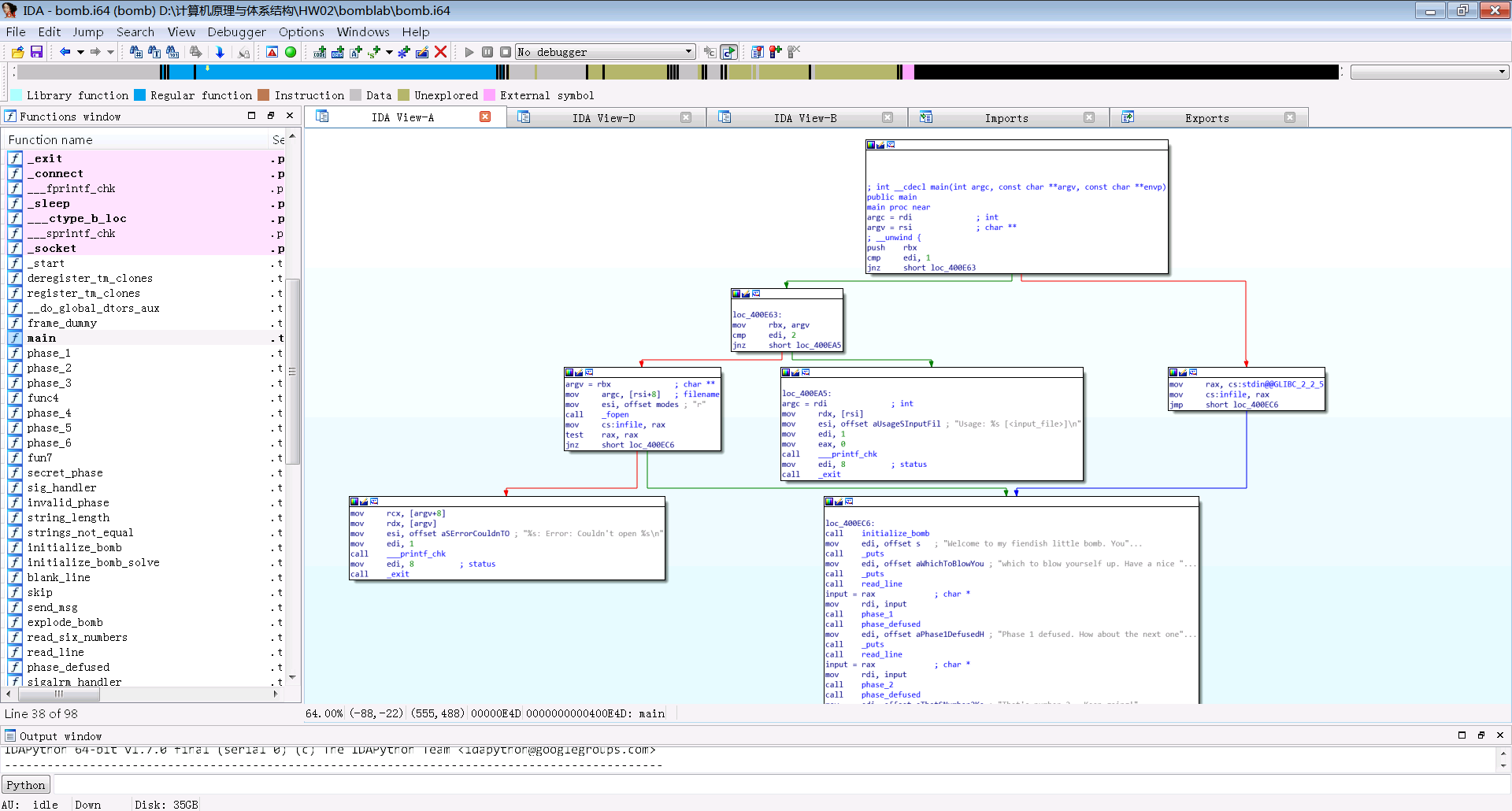
Initialization error: Running on an illegal host [2]

看起来bomb无法在服务器以外的环境运行。如下图：

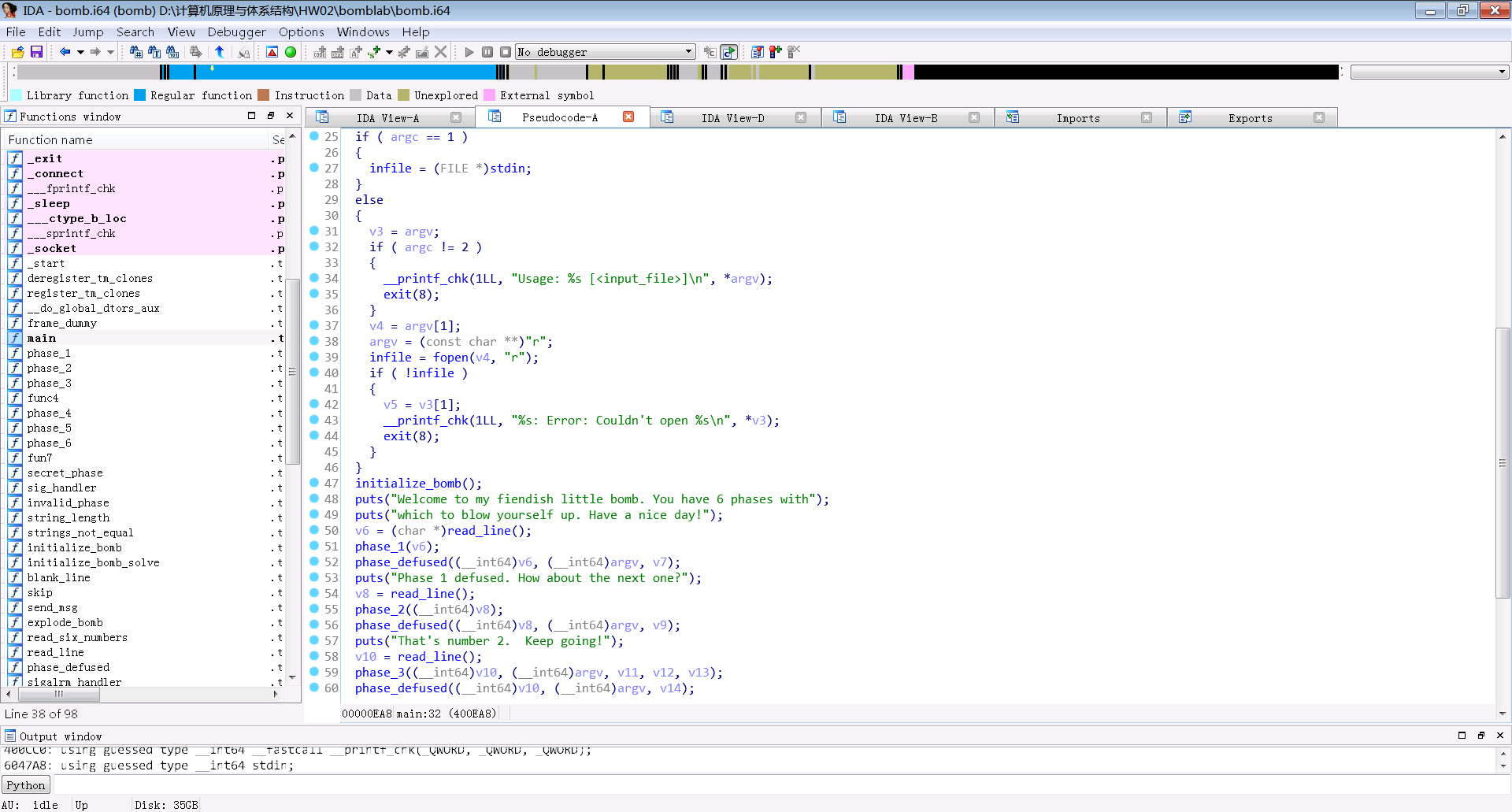


1. 试图让bomb可在本地运行

我做到phase\_5时才真正实现了本地运行bomb。**将bomb转移到Windows下**，用软件IDA打开，查看main函数的结构如下：

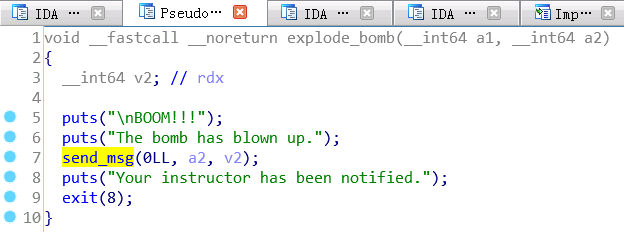
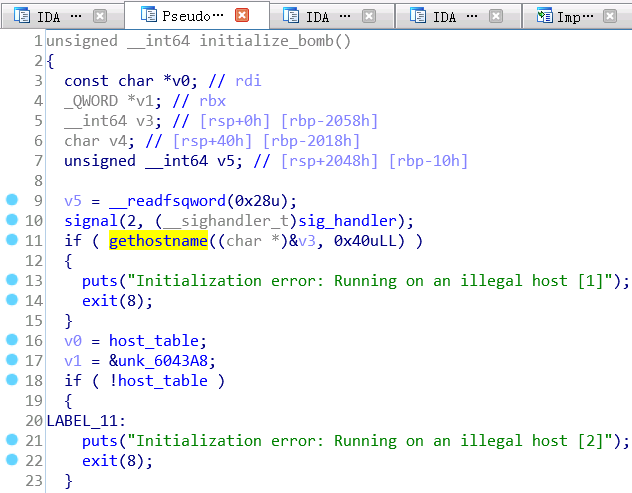


可以清楚地看到各种跳转。此时按下F5，显示main函数反汇编得到的C代码如下（省略变量定义的部分）：



**反编译得到的C代码对变量类型的识别经常错得很厉害，且会在各种地方出现严重内容性错误。**但反汇编到C的结果仍然能**写意地**描绘程序的结构。阅读C语言或汇编语言流程块，可以高效地把握程序的框架。

双击initialize\_bomb，phase\_1，phase\_defused等，可查看函数内反汇编得到的C代码。按Esc可从子函数中返回父函数。发现initialize\_bomb会利用gethostname函数收集宿主机信息，而explode\_bomb会用send\_msg函数发送消息，如下图：



因此考虑在本地的bomb上禁用掉这两个函数。可以通过让这两个函数直接return来达到目的。回到Linux上，在bomb所在目录打开terminal，运行如下命令：（可执行文件bomb在home/bomblabrev/文件夹中）

a@Ithea:~/bomblabrev$ gdb -write bomb

GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git

……（省略大量内容）

Reading symbols from bomb...done.

(gdb) disas initialize\_bomb

Dump of assembler code for function initialize\_bomb:

0x0000000000401422 <+0>: push %rbx

……

---Type <return> to continue, or q <return> to quit---q

Quit

(gdb) set \*(unsigned char \*)(0x0000000000401422)=0xc3

(gdb) disas send\_msg

Dump of assembler code for function send\_msg:

0x0000000000401585 <+0>: sub $0x4028,%rsp

……

---Type <return> to continue, or q <return> to quit---q

Quit

(gdb) set \*(unsigned char \*)(0x0000000000401585)=0xc3

上面这些命令中，gdb –write bomb以可写（而非只读）的方式打开bomb。disas <函数名>可以列出这个函数内部的汇编结果。为了使initialize\_bomb和send\_msg失效，我们可以set \*(unsigned char \*)(<函数首行地址>)=0xc3，这就将该函数执行时的第一条指令改为ret。检查一下修改结果：

(gdb) disas initialize\_bomb

Dump of assembler code for function initialize\_bomb:

0x0000000000401422 <+0>: retq

……

---Type <return> to continue, or q <return> to quit---q

Quit

(gdb) disas send\_msg

Dump of assembler code for function send\_msg:

0x0000000000401585 <+0>: retq

……

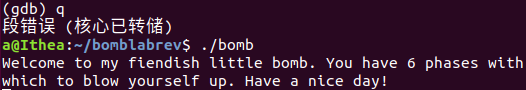
---Type <return> to continue, or q <return> to quit---q

Quit

(gdb) q

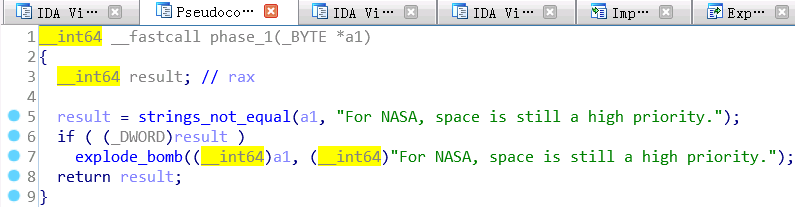
段错误 (核心已转储)

这时再执行./bomb，发现炸弹可以在本地运行了：

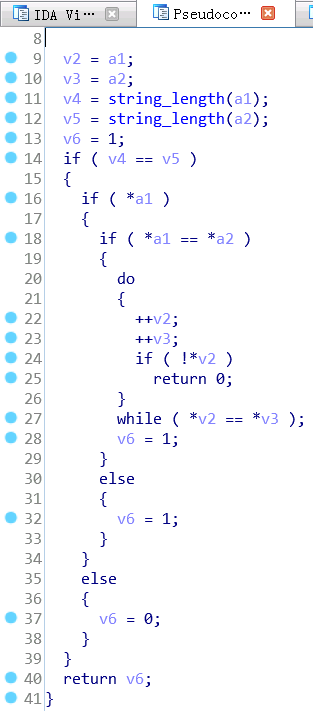


1. 拆弹
   1. **phase\_1**

反编译代码如下：



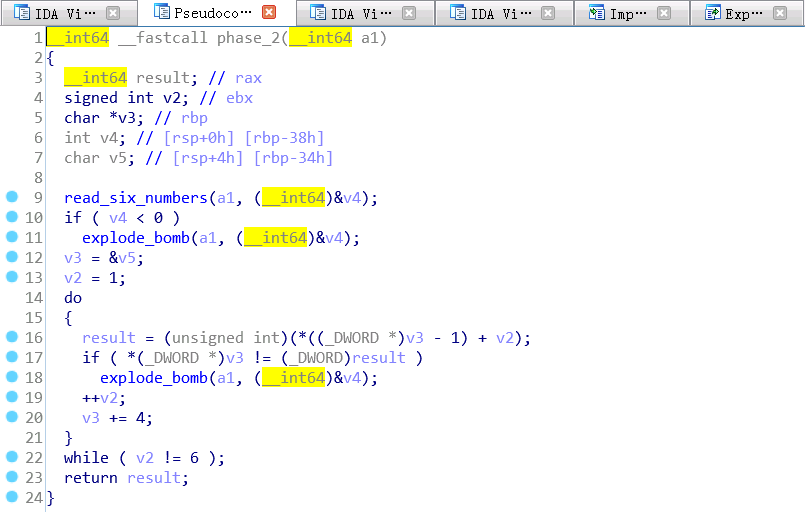
猜测只要输入一样的字符串即可。进入strings\_not\_equal，代码为：



确实是在逐位比较字符是否相等。相等时返回0。答案是：

For NASA, space is still a high priority.

* 1. **phase\_2**

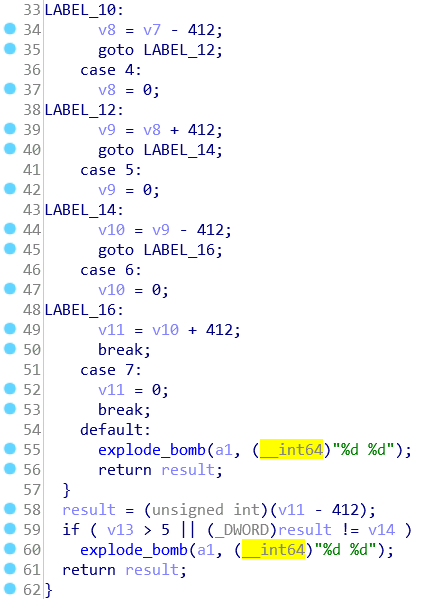


64位程序中，一个int占64bit（按C语言标准的建议，应与机器字长相等），或8Byte，或16位16进制数（每位16进制为4bit）。64位程序的指针每+1即移动4位16进制数的地址。此函数中栈顶为rsp，v3是数组指针，初值为rsp+4，即指向输入六个数的第2个（下标1）。要使炸弹不爆，应使数组前一个数\*(v3-1)加上v2后，与当前数\*v3相等。而v2又在不断+1，因此答案为相邻两数之差初始为1，每次差值+1的6个整数，例如：

1 2 4 7 11 16

* 1. **phase\_3**

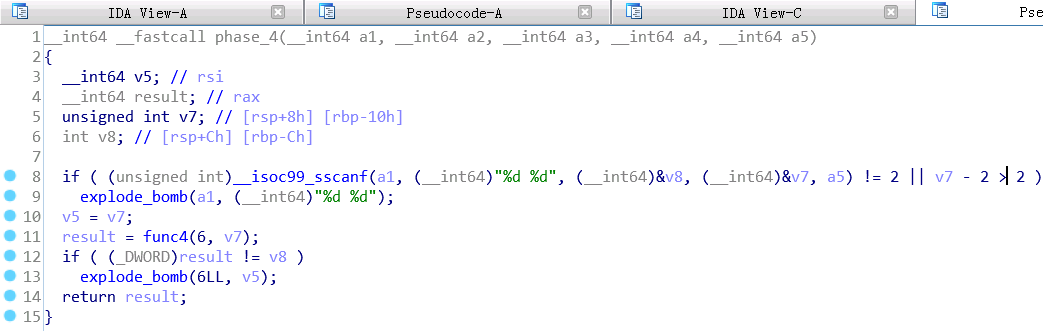
读入两个数并switch，主要内容如下（按顺序输入v13，v14）。



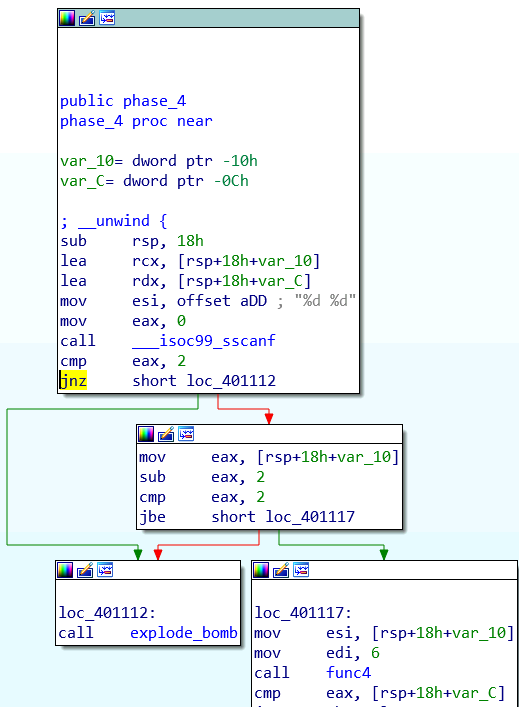
输入的第1个数v13显然应小于等于5，而v13将决定switch所走的分支。v13 == 5时，求得v14 == -412。于是得到一个答案：

5 -412

* 1. **phase\_4**

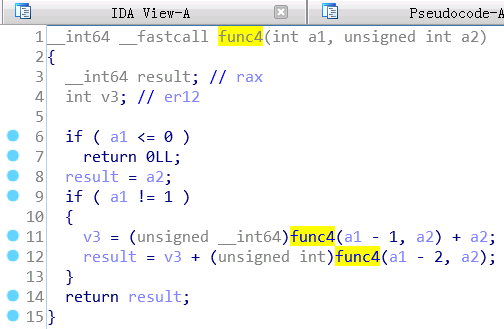


输入v8，v7。第二个数v7是func4的初始参数之一。**事实上这里v7-2>2是错误的**，因为原始汇编代码为



jnz是比较值不为0时跳转，也就是第二个输入数必须为4，才能使炸弹不爆。

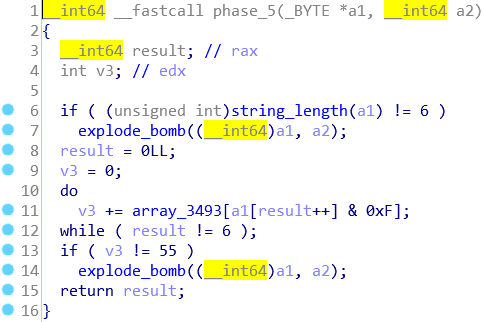
func4不断调用自身如下：



检查汇编代码认为C语言翻译无误。直接在Windows下抄一份func4的C语言代码，编译求出func4(6,4)的值为80，即答案为

80 4

* 1. **phase\_5**



涉及数组array\_3493。从数据区查到其定义为

int array\_3493[16] = { 2, 10, 6, 1, 12, 16, 9, 3, 4, 7, 14, 5, 11, 8, 15, 13 }; // idb

输入的字符串含6个字符，它们的ASCII码进行& 0xF（即取二进制最后四位，或者说除以16取余数）运算后，作为array\_3493的6个索引。用这6个索引从array\_3493取出的数之和必须为55。从一堆数中找出若干个数，使取出数的和为指定值的问题，可用以下代码解决：

//背包问题

//设有一背包可放入的物品重量为S

//现有n件物品，重量分别为w0，w1， …，wn-1

//问能否从这n件物品中选择若干件放入背包，使其重量之和正好为S。

//若存在一种符合上述要求的选择，则称此背包问题有解，否则无解。

#include <stdio.h>

int w[]={2, 10, 6, 1, 12, 16, 9, 3, 4, 7, 14, 5, 11, 8, 15, 13}; //若干件物品的重量w

int S=55; //重量总和S

int n=sizeof(w)/sizeof(int); //一共n件物品

int knap(int S,int n)

{

if (S==0)

return 1;

if ( (S<0) || (S>0 && n<1) )

return 0;

if (knap(S-w[n-1],n-1))

{

printf("%d ",w[n-1]);

return 1;

}

else

return knap(S,n-1);

}

int main()

{

if (knap(S,n))

{

printf("\n可以实现");

}

else

{

printf("\n不能实现");

}

}

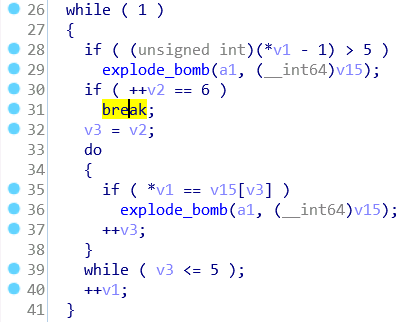
求得可取出数值为3,5,11,8,15,13，其下标分别为7和11~15。由于键盘上只能输入数字、字母等，所以应从ASCII表中查出[除以16所得余数为7和11~15，且便于从键盘输入]的字符。一种答案为：

GKLMNO

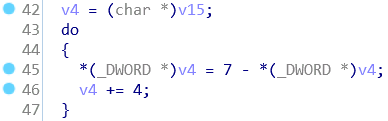
其中G的ASCII为71==64+7。

* 1. **phase\_6**

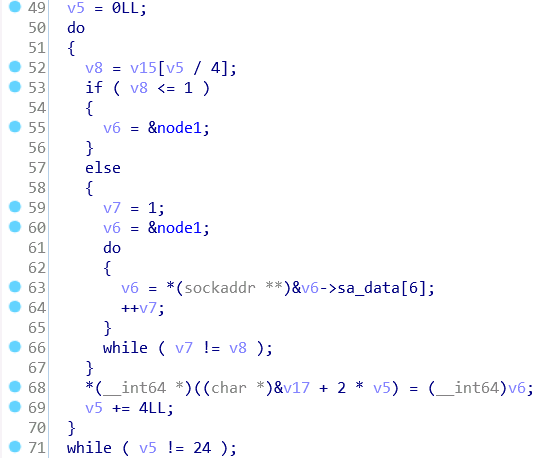
输入6个数构成数组v15。下图的第一重循环规定所有数均小于等于6且互不相等。v1以指针方式遍历数组，而v2（以及事实上没用的v3）初值为0，以索引方式遍历数组。



第二重循环以v4为指针遍历数组，使每个数均变为(7-那个数)



第三重循环调用了node1如下

  
从可执行文件bomb的数据区查到node1~node6如下：

(gdb) x/12xg 0x6043100x604310 <node1>: 0x0000000100000350 0x00000000006043200x604320 <node2>: 0x0000000200000110 0x00000000006043300x604330 <node3>: 0x00000003000000bc 0x00000000006043400x604340 <node4>: 0x0000000400000042 0x00000000006043500x604350 <node5>: 0x000000050000032a 0x00000000006043600x604360 <node6>: 0x00000006000001c4 0x0000000000000000

IDA软件也可以查到这些定义，但使用gdb可以在实验报告中表示得更清楚。

然而机器不知道node的定义方式（在编译过程中已经被抹除了）。推测node1~node6定义方式为：

struct node

{

\_\_int64 serial; //取值范围1到6

\_\_int64 payload; //在这些node中取了0x350，0x110等

\_\_int64 占位符; //全是0。可能是编译器进行内存对齐产生的

(struct node\*) p; //指向其他node //长度相当于\_\_int64

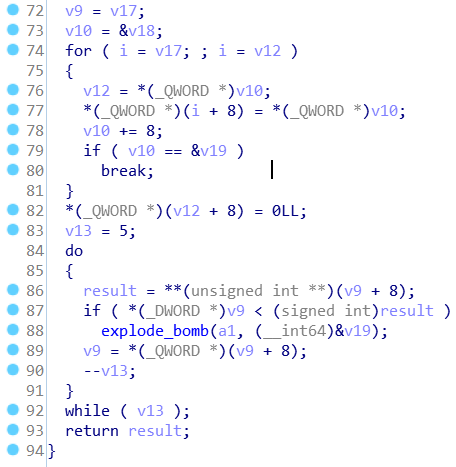
};

于是六个node构成链表。同时这段代码第63行还用了sa\_data[6]这一变量。查阅汇编，发现第63行实为：

mov rdx, [rdx+8]

其中rdx代表v6，变量类型应为struct node\*（node这一struct被软件命名为sockaddr）。也就是说这段代码是在按我们的输入数组v15来设定v17的内容：假设v17可指向一个指针数组(struct node\*) p[6]，则设定p[i]=&node(v15[i])。每个p[i]占8字节。例如v15的第1个数是6，则p[1]指向node6。

最后还有两段循环如下：

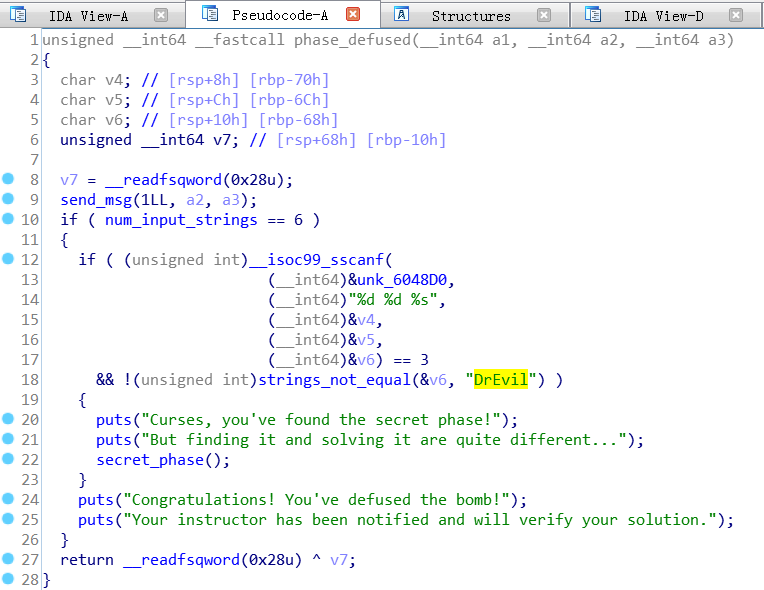


这里v18初始定义为p[1]。for循环将(struct node\*)p[i]在内存中的位置整体右移了一格。不太明白这样做的目的是什么。最后的do循环要求\*(p[i])随着i的递增而递增。由于p[i]==&node(v15[i])，也就是说这段循环要求node(v15[i])的payload递减。给node1~node6的payload排序，得到1 5 6 2 3 4的序号。由于序号是被7减过的，因此答案为：

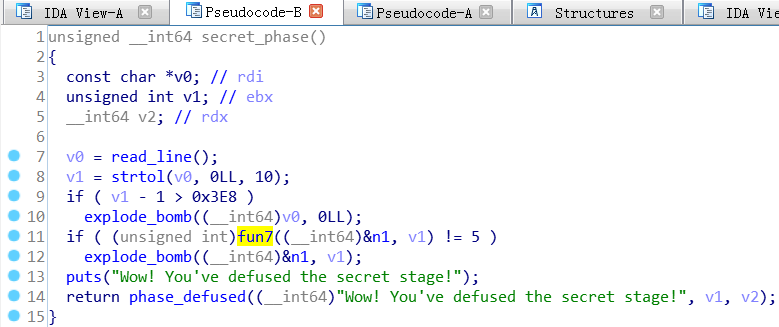
6 2 1 5 4 3

* 1. **secret\_phase**

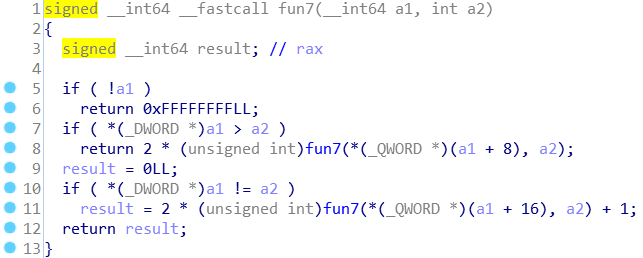
搜索字段phase时意外找到了secret\_phase这个函数，且发现它的入口在phase\_defused里。将phase\_defused反编译如下：



未知变量unk\_6048D0在read\_six\_numbers中出现过，作为输入字符串的指针。于是推测在phase\_3或phase\_4的答案中补充输入” DrEvil”即可解锁secret\_phase（其他phase的原始答案不止两个整数）。secret\_phase反编译如下：



strtol(v0,0,10)是将输入字符串v0（例如”0xB”）转为十进制数v1（例如11）。(&n1)实际作为一个\_\_int64常数来使用，其值为0x24。也就是secret\_phase调用fun7后要求结果为5。fun7反编译如下：



看起来逆推这个递归会是一件体力活。不想手动推测了！于是复制出fun7的C代码，自己写了一个程序暴力尝试，得到一个答案

47

汇总一份可行的答案如下：

For NASA, space is still a high priority.1 2 4 7 11 165 -41280 4 DrEvilGKLMNO

6 2 1 5 4 3

47

1. uid回到服务器回到服务器回到服务器正式解除炸弹
   1. 打开只回答启动**在服务器启动炸弹程序**

a@Ithea:~/bomblab$ ssh stu16307130006@10.48.76.8

stu16307130006@10.48.76.8's password: （输入密码）

Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.16.0-30-generic x86\_64)

......

and reboot your system.

stu16307130006@catlab-ubuntu:~/bomblab$ ls

bomb75.tar bomblab.pdf

stu16307130006@catlab-ubuntu:~/bomblab$ tar -xvf bomb75.tar

bomb75/README

bomb75/bomb.c

bomb75/bomb

stu16307130006@catlab-ubuntu:~/bomblab$ ls

bomb75 bomb75.tar bomblab.pdf

stu16307130006@catlab-ubuntu:~/bomblab$ cd bomb75

stu16307130006@catlab-ubuntu:~/bomblab/bomb75$ gdb bomb

GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.3) 7.7.1

......

Reading symbols from bomb...done.

* 1. **对explode\_bomb设置断点，防止误操作引爆**

(gdb) b explode\_bomb

Breakpoint 1 at 0x401691

(gdb) r

Starting program: /home/2019csapp/stu16307130006/bomblab/bomb75/bomb

Welcome to my fiendish little bomb. You have 6 phases with

which to blow yourself up. Have a nice day!

For NASA, space is still a high priority.

Phase 1 defused. How about the next one?

1 2 4 7 11 16

That's number 2. Keep going!

5 -412

Halfway there!

80 4 DrEvil

So you got that one. Try this one.

GKLMNO

Good work! On to the next...

* 1. **故意引爆一次**

GKLMNO

Breakpoint 1, 0x0000000000401691 in explode\_bomb ()

(gdb) q

A debugging session is active.

Inferior 1 [process 451] will be killed.

Quit anyway? (y or n) y

* 1. **重新拆弹**

stu16307130006@catlab-ubuntu:~/bomblab/bomb75$ gdb bomb

GNU gdb (Ubuntu 7.7.1-0ubuntu5~14.04.3) 7.7.1

...

Reading symbols from bomb...done.

(gdb) b explode\_bomb

Breakpoint 1 at 0x401691

(gdb) r

Starting program: /home/2019csapp/stu16307130006/bomblab/bomb75/bomb

Welcome to my fiendish little bomb. You have 6 phases with

which to blow yourself up. Have a nice day!

For NASA, space is still a high priority.

1 2 4 7 11 16

5 -412

80 4 DrEvil

GKLMNO

6 2 1 5 4 3

47

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!

Your instructor has been notified and will verify your solution.

[Inferior 1 (process 1009) exited normally]

(gdb) q

stu16307130006@catlab-ubuntu:~/bomblab/bomb75$ cd

stu16307130006@catlab-ubuntu:~$ ls

bomblab

stu16307130006@catlab-ubuntu:~$ exit

logout

Connection to 10.48.76.8 closed.