

综合实验 二进制炸弹实验

姓名: 林荣恩 学号: 191220060 院系: 计算机科学与技术系

一. 实验目的:

本实验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。一个“binary bombs”(二进制炸弹,下文将简称为炸弹)是一个 Linux 可执行程序,包含了 6 个阶段(或层次、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即解除了,否则炸弹“爆炸”打印输出“BOOM!!!”。

二. 实验内容:

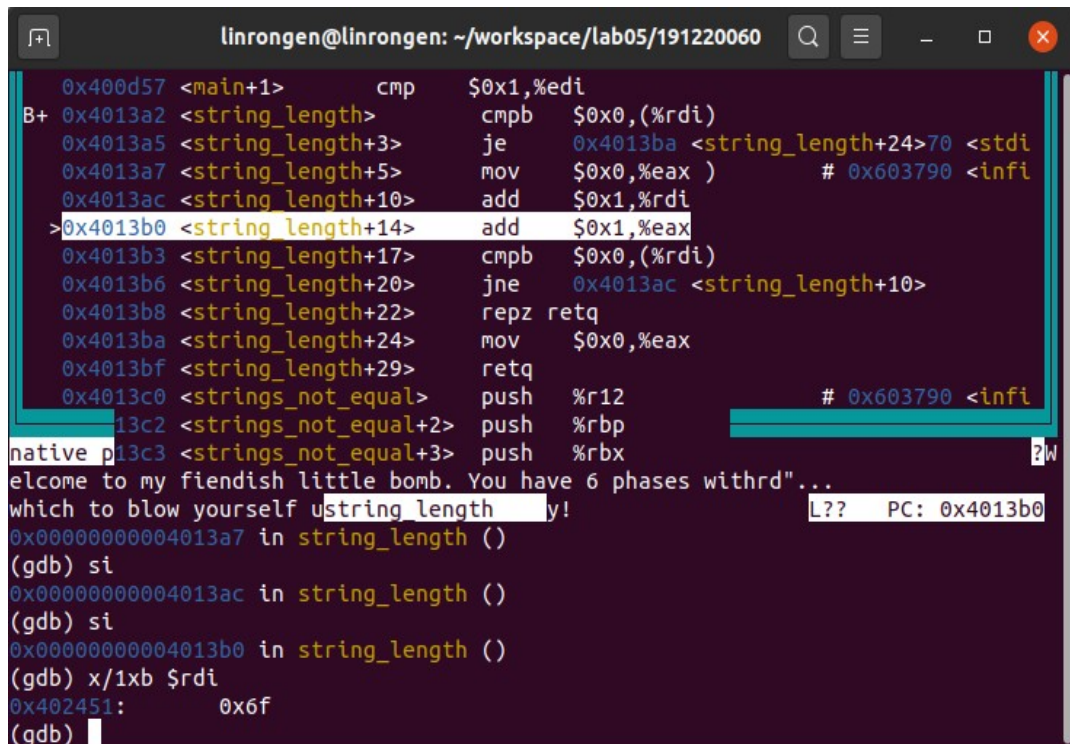
使用 objdump 对 bomb 进行反汇编,观察 bomb.c 与 bomb.s.

1. 阶段 1

本阶段要求输入一个字符串,使之与 bomb 内的字符串相等。

观察汇编代码可以发现,其中调用了函数 string_length 获取两个字符串的长度,其中每个循环内的 %rdi 表示当前字符的地址。对于第二次调用 string_length,使用 gdb 对 bomb 进行测试,获取循环内每个字符,即可拼凑出整个字符串。例如,下图中,使用 x/1xb 命令查询字符串的第二位,得到其 ascii 码为 0x6f,即字符 o。

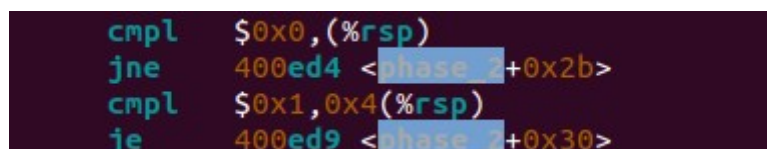
最终可得出原字符串为“For NASA, space is still a high priority.”。



```
linrongen@linrongen: ~/workspace/lab05/191220060
0x400d57 <main+1>      cmp     $0x1,%edi
B+ 0x4013a2 <string_length>  cmpb   $0x0,(%rdi)
0x4013a5 <string_length+3>  je     0x4013ba <string_length+24>70 <stdi
0x4013a7 <string_length+5>  mov     $0x0,%eax
0x4013ac <string_length+10>  add     $0x1,%rdi
>0x4013b0 <string_length+14>  add     $0x1,%eax
0x4013b3 <string_length+17>  cmpb   $0x0,(%rdi)
0x4013b6 <string_length+20>  jne     0x4013ac <string_length+10>
0x4013b8 <string_length+22>  repz   retq
0x4013ba <string_length+24>  mov     $0x0,%eax
0x4013bf <string_length+29>  retq
0x4013c0 <strings_not_equal>  push    %r12
0x4013c2 <strings_not_equal+2>  push    %rbp
native p13c3 <strings_not_equal+3>  push    %rbx
Welcome to my fiendish little bomb. You have 6 phases withrd"...
which to blow yourself ustring_length y!
0x00000000004013a7 in string_length ()
(gdb) si
0x00000000004013ac in string_length ()
(gdb) si
0x00000000004013b0 in string_length ()
(gdb) x/1xb $rdi
0x402451:      0x6f
(gdb)
```

2. 阶段 2

观察 phase_2 部分,可以看出读入的六个数为 (%rsp), 0x4(%rsp)...。首先,从下图可以看出,六个数中前两个数必须分别为 0 和 1。



```
cmpl     $0x0,(%rsp)
jne      400ed4 <phase_2+0x2b>
cmpl     $0x1,0x4(%rsp)
je       400ed9 <phase_2+0x30>
```

同时，下面的代码是一个循环，判断对所有的 $1 \leq i \leq 4$ 是否满足第 i 个数与第 $i+1$ 个数的和等于第 $i+2$ 个数（`%rbx` 为当前第 i 个数的位置）。

```

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 cf 05 00 00 callq   4014bf <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>

```

递推可知六个数为 “0 1 1 2 3 5”。

3. 阶段 3

首先，确定输入的格式。用 `gdb` 检查 `sscanf` 的参数，如下图，知需要输入三个元素，分别为一个数、一个字符、一个数，分别存放在 `0x10(%rsp)`, `0xf(%rsp)`, `0x14(%rsp)` 中。

```

(gdb) x/s 0x4024a6
0x4024a6: "%d %c %d"

```

容易看出主要结构为跳转表，判断依据为输入的第一个数（需满足 ≤ 7 ），可以使用 `gdb` 查看跳转到的地址（设第一个数为 i ，则查询 `x/1xw 0x4024c0+i*8`）。此处取输入的第一个数为 0，查询得跳转地址为 `400f62`。

接着可以看出要求第三个元素等于 `0x269=617`，字符等于 `0x71='q'`。

```

mov     $0x71,%eax
cmpl    $0x269,0x14(%rsp)

je      40105e <phase_3+0x13e>

cmp     0xf(%rsp),%al
je      40105e <phase_3+0x149>

```

因此 “0 q 617” 满足要求。若第一个数取 $[0,7]$ 中的其他整数，也可构造出相应的解。

4. 阶段 4

首先，与阶段 3 的开始部分类似，可以确定输入为两个整数。

分析汇编代码可以看出，第二个数必须为 5，第一个数必须小于或等于 14。同时，对 `func4` 进行分析，将其翻译成 C 语言，如下所示（ n 为输入的第一个数，`func4` 执行完毕后 `%eax` 的值为 `g(14,0)`）：

```

#include <stdio.h>
int n;
int g(int a,int b) {
    int t=(((a-b)>>31)+a-b)>>1)+b;
    printf("%d %d %d\n",a,b,t);
    getchar();
    if (t>n) return 2*g(t-1,b);
    if (t<n) return 2*g(a,t+1)+1;
    return 0;
}
int main() {
    scanf("%d",&n);
    printf("%d\n",g(14,0));
    return 0;
}

```

同时又知函数返回值%eax=5，在上面给出的 c 程序中进行枚举测试可知 n=10 满足要求。因此“10 5”为一组解。

5. 阶段 5

由汇编代码可以看出，该阶段需要输入 6 个字符。程序中包含一个循环（见下图），依次检查该字符串，并将 0x402500(,%rdx,4) 累加至%ecx，其中 %rdx 是字符 ascii 码 &0xf 的值。经 gdb 试验，当 %rdx=0,1,...,15 时，对应的值分别为 2,10,6,1,12,16,9,3,4,7,14,5,11,8,15,13。程序要求六个字符对应的值的和为 0x4e=46。此处取字符串为“111177”，对应的和为 10*4+3*2=46，满足要求。

```
movzbl (%rax),%edx
and     $0xf,%edx
add     0x402500(,%rdx,4),%ecx
add     $0x1,%rax
cmp     %rdi,%rax
jne     401148 <phase_5+0x1f>
cmp     $0x2e,%ecx
```

6. 阶段 6

该阶段要求输入六个整数。在汇编代码的头部，有一个两重循环用以检测是否给出的六个整数是 1~6 的一个排列。关键在于确定这六个整数包含的性质。

```
mov     $0x6032f0,%edx
```

在上图所示语句中包含一个立即数。在之后的程序里，0x8(%rdx) 等被多次访问。

用“2 5 4 6 3 1”作为输入进行测试。该序列在过程中每一位被用 6 减，更新后的数列为“5 2 3 1 4 6”。在程序执行过程中，使用 gdb 查询附近位置的值，得到下表：

i	i+%rdx	(i+%rdx)	0x4(i+%rdx)	0x8(i+%rdx)	修改后的 0x8(i+%rdx)
0	0x6032f0	0x238	1	0x603300	0x603320
1	0x603300	0x2da	2	0x603310	0x603310
2	0x603310	0x1a3	3	0x603320	0x6032f0
3	0x603320	0x09c	4	0x603330	0x603340
4	0x603330	0x319	5	0x603340	0x603300
5	0x603340	0x1cb	6	0	0

该表格反映出的数据特点暗示了这一部分数据的组织形式为链表。容易发现，在经过修改后的链表，依照链表组织顺序排列的 0x4(i+%rdx) 为“5 2 3 1 4 6”（例如，0x603330 为链表头，对应的数字为 5，为第一位）。这与更新后的输入数据相同。

在该部分程序结束后，代码中对 (i+%rdx) 间的大小关系进行判断，要求降序排列。为使该条件满足，更新后的输入数据应为“5 2 1 6 3 4”。因此，输入的原始数据应为“2 5 6 1 4 3”。

因此“2 5 6 1 4 3”为一组正确的解。

7. 隐藏阶段

在 phase_defused 函数中，出现了 sscanf 的调用。显然这一部分与隐藏阶段有关。使用与阶段一相同的方法，可以得到第三阶段后需要输入的字符串“DrEvil”。

容易发现, func7 相当于二叉搜索树的查询过程, 同时, 若令左儿子的权为 0, 右儿子的权为 1, 则要求从叶子到根的路径的点权组成的二进制数等于 2. 故目标数位于根的左儿子的右子树的最左链的底端。使用 gdb 查询该路径, 得目标数为 0x14=20.
故解为 20.

至此, 二进制炸弹被成功拆除。

```
linrongen@linrongen:~/workspace/lab05/191220060$ ./bomb solution
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
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!
```