

哈尔滨工业大学

实验报告

实验（三）

题 目 Binary Bomb

二进制炸弹

专 业 计算学部

学 号

班 级

学 生

指 导 教 师

实 验 地 点 G709

实 验 日 期 2021.4.22

计算机科学与技术学院

目 录

第 1 章 实验基本信息	- 3 -
1.1 实验目的.....	- 3 -
1.2 实验环境与工具.....	- 3 -
1.2.1 硬件环境.....	- 3 -
1.2.2 软件环境.....	- 3 -
1.2.3 开发工具.....	- 3 -
1.3 实验预习.....	- 3 -
第 2 章 实验环境建立	- 5 -
2.1 UBUNTU 下 CODEBLOCKS 反汇编（10 分）	- 5 -
2.2 UBUNTU 下 EDB 运行环境建立（10 分）	- 5 -
第 3 章 各阶段炸弹破解与分析	- 6 -
3.1 阶段 1 的破解与分析.....	- 6 -
3.2 阶段 2 的破解与分析.....	- 7 -
3.3 阶段 3 的破解与分析.....	- 8 -
3.4 阶段 4 的破解与分析.....	- 9 -
3.5 阶段 5 的破解与分析.....	- 11 -
3.6 阶段 6 的破解与分析.....	- 12 -
3.7 阶段 7 的破解与分析(隐藏阶段).....	- 14 -
第 4 章 总结.....	- 17 -
4.1 请总结本次实验的收获.....	- 17 -
4.2 请给出对本次实验内容的建议.....	- 17 -
参考文献.....	- 18 -

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; 512GHD Disk

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)、Og、O0、O1、O2、O3、Og, -m32/m64。再次查看生成的汇编语言与原来的区别。

堆栈访问[rbp+-n]或[rsp+n]。-fno-omit-frame-pointer。

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?

2.1 Ubuntu 下 CodeBlocks 反汇编 (10 分)

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。



用 EDB 调试 hellolinux.c 的执行文件，截图，要求同 2.1



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

整体破解情况：

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I was trying to give Tina Fey more material.
Phase 1 defused. How about the next one?
0 1 1 2 3 5
That's number 2. Keep going!
0 792
Halfway there!
24 2 DrEvil
So you got that one. Try this one.
ioa`eg
Good work! On to the next...
1 5 3 2 4 6
Curses, you've found the secret phase!
But finding it and solving it are quite different...
35
Wow! You've defused the secret stage!
Congratulations! You've defused the bomb!
[Inferior 1 (process 25552) exited normally]
```

3.1 阶段 1 的破解与分析

密码如下：I was trying to give Tina Fey more material.

破解过程：

```
00000000004013f9 <phase_1>:
4013f9: 55                    push    %rbp
4013fa: 48 89 e5              mov     %rsp,%rbp
4013fd: be 50 31 40 00        mov     $0x403150,%esi
401402: e8 0b 04 00 00        callq   401812 <strings_not_equal>
401407: 85 c0                test    %eax,%eax
401409: 75 02                jne     40140d <phase_1+0x14>
40140b: 5d                    pop     %rbp
40140c: c3                    retq
40140d: e8 fc 04 00 00        callq   40190e <explode_bomb>
401412: eb f7                jmp     40140b <phase_1+0x12>
```

从上图画红线的位置往下，可以看出在给%esi 传了一个值之后调用 strings_not_equal 函数，之后对寄存器之中的值进行判断，可以大胆猜测需要比较的炸弹的值就存在 0x403150 中，使用 gdb 查看一下该地址中的值：

```
Breakpoint 1, phase_1 (input=0x405780 <input_strings> "q") at phases.c:20
20      phases.c: 没有那个文件或目录.
(gdb)
(gdb) x/s 0x403150
0x403150:      "I was trying to give Tina Fey more material."
(gdb)
```

可以发现其中的值是“I was trying to give Tina Fey more material.”则可以判断这就是这一阶段的炸弹值，经过验证之后确实如此

3.2 阶段 2 的破解与分析

密码如下：0 1 1 2 3 5

破解过程：阶段 2 的反汇编代码如下

0000000000401414 <phase_2>:	
401414:	55
401415:	48 89 e5
401418:	53
401419:	48 83 ec 28
40141d:	48 8d 75 d0
401421:	e8 0a 05 00 00
401426:	83 7d d0 00
40142a:	75 06
40142c:	83 7d d4 01
401430:	74 05
401432:	e8 d7 04 00 00
401437:	bb 02 00 00 00
40143c:	eb 08
40143e:	e8 cb 04 00 00
401443:	83 c3 01
401446:	83 fb 05
401449:	7f 1e
40144b:	48 63 d3
40144e:	8d 4b fe
401451:	48 63 c9
401454:	8d 43 ff
401457:	48 98
401459:	8b 44 85 d0
40145d:	03 44 8d d0
401461:	39 44 95 d0
401465:	74 dc
401467:	eb d5
401469:	48 83 c4 28
40146d:	5b
40146e:	5d
40146f:	c3
	push %rbp
	mov %rsp,%rbp
	push %rbx
	sub \$0x28,%rsp
	lea -0x30(%rbp),%rsi
	callq 401930 <read_six_numbers> ①
	cmpl \$0x0,-0x30(%rbp)
	jne 401432 <phase_2+0x1e>
	cmpl \$0x1,-0x2c(%rbp) ②
	je 401437 <phase_2+0x23>
	callq 40190e <explode_bomb>
	mov \$0x2,%ebx
	jmp 401446 <phase_2+0x32>
	callq 40190e <explode_bomb>
	add \$0x1,%ebx
	cmp \$0x5,%ebx ③
	jg 401469 <phase_2+0x55>
	movslq %ebx,%rdx
	lea -0x2(%rbx),%ecx
	movslq %ecx,%rcx
	lea -0x1(%rbx),%eax
	cltq
	mov -0x30(%rbp,%rax,4),%eax
	add -0x30(%rbp,%rcx,4),%eax ④
	cmp %eax,-0x30(%rbp,%rdx,4)
	je 401443 <phase_2+0x2f>
	jmp 40143e <phase_2+0x2a>
	add \$0x28,%rsp
	pop %rbx
	pop %rbp
	retq

从标号①处可以看出，明显这一阶段需要输入的是六个数字，进入该函数可以看到六个数字应该是每两个之间有一个空格的格式输入的。从标号②处可以看出，比较了输入的第一个和第二个数字，可以看出第一个数字是 0，第二个数字是 1。而从标号③处可以看出这是一个循环，需要一共输入六个数字，这一点也和我们从标号①处看到的相同。最后就是标号④，可以看出标号④进行的操作就是将第 $n-1$ 与 $n-2$ 处的数加起来得到第 n 位的数，也就是一个斐波那契数列，那么容易得到需要输入的炸弹值就是 0 1 1 2 3 5。

3.3 阶段 3 的破解与分析

密码如下：

0 792 or 2 96（其中两个密码，还可能有多种组合）

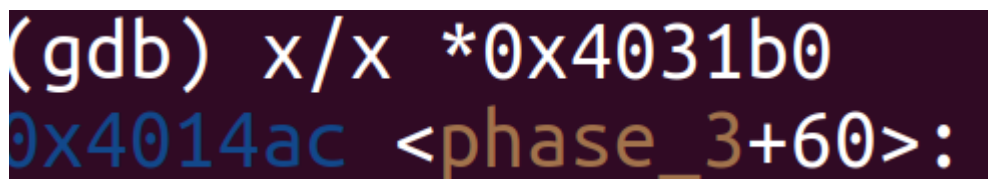
破解过程：

```

351 0000000000401470 <phase_3>:
352 401470: 55 push %rbp
353 401471: 48 89 e5 mov %rsp,%rbp
354 401474: 48 83 ec 10 sub $0x10,%rsp
355 401478: 48 8d 4d f8 lea -0x8(%rbp),%rcx
356 40147c: 48 8d 55 fc lea -0x4(%rbp),%rdx
357 401480: be 0f 33 40 00 mov $0x40330f,%esi
358 401485: b8 00 00 00 00 mov $0x0,%eax
359 40148a: e8 81 fc ff ff callq 401110 <__isoc99_sscanf@plt>
360 40148f: 83 f8 01 cmp $0x1,%eax
361 401492: 7e 11 jle 4014a5 <phase_3+0x35>
362 401494: 8b 45 fc mov -0x4(%rbp),%eax
363 401497: 83 f8 07 cmp $0x7,%eax
364 40149a: 77 46 ja 4014e2 <phase_3+0x72>
365 40149c: 89 c0 mov %eax,%eax
366 40149e: ff 24 c5 b0 31 40 00 jmpq *0x4031b0(,%rax,8)
367 4014a5: e8 64 04 00 00 callq 40190e <explode_bomb>
368 4014aa: eb e8 jmp 401494 <phase_3+0x24>
369 4014ac: b8 18 03 00 00 mov $0x318,%eax
370 4014b1: 39 45 f8 cmp %eax,-0x8(%rbp)
371 4014b4: 75 3f jne 4014f5 <phase_3+0x85>
372 4014b6: c9 leaveq
373 4014b7: c3 retq
374 4014b8: b8 60 00 00 00 mov $0x60,%eax
375 4014bd: eb f2 jmp 4014b1 <phase_3+0x41>
376 4014bf: b8 46 00 00 00 mov $0x46,%eax
377 4014c4: eb eb jmp 4014b1 <phase_3+0x41>
378 4014c6: b8 15 01 00 00 mov $0x115,%eax
379 4014cb: eb e4 jmp 4014b1 <phase_3+0x41>
380 4014cd: b8 b5 03 00 00 mov $0x3b5,%eax
381 4014d2: eb dd jmp 4014b1 <phase_3+0x41>
382 4014d4: b8 bf 01 00 00 mov $0x1bf,%eax
383 4014d9: eb d6 jmp 4014b1 <phase_3+0x41>
384 4014db: b8 42 01 00 00 mov $0x142,%eax
385 4014e0: eb cf jmp 4014b1 <phase_3+0x41>

```


通过汇编代码可以看出这是一个类似于 switch 跳转的跳转的类型，通过对输入类型的查看可以知道输入的应该是两个整型数字，且根据反汇编代码，可以发现第一个数字应该是在 0 到 6 之间，而我们发现 366 行有一个指针，经过对指针内容的查看，可以发现存储的地址是 0x4014ac，如下图。



```
(gdb) x/x *0x4031b0
0x4014ac <phase_3+60>:
0x4014ac
```

而根据 366 行的反汇编可以看出这一代码的目的就是将输入的数字乘以 8 之后作为地址的偏置量加到 0x4014ac 上，跳转到该步执行。从而如果输入的的第一个数字是 0 的话，那么第二个数字就应该是 792，如果第一个数字是 2 的话第二个数字就应该是 96，其余密码以此类推可得。

3.4 阶段 4 的破解与分析

密码如下：24 2

破解过程：

根据对第四阶段反汇编代码的分析可以很轻易得出这一阶段的代码实现的是一个递归函数，递归函数如下：

```

394 00000000004014fc <func4>:
395 4014fc: 85 ff          test    %edi,%edi
396 4014fe: 7e 3d          jle     40153d <func4+0x41>
397 401500: 55             push    %rbp
398 401501: 48 89 e5       mov     %rsp,%rbp
399 401504: 41 55          push    %r13
400 401506: 41 54          push    %r12
401 401508: 53             push    %rbx
402 401509: 48 83 ec 08    sub     $0x8,%rsp
403 40150d: 41 89 fc       mov     %edi,%r12d
404 401510: 89 f3          mov     %esi,%ebx
405 401512: 83 ff 01       cmp     $0x1,%edi
406 401515: 74 2c          je      401543 <func4+0x47>
407 401517: 8d 7f ff       lea     -0x1(%rdi),%edi
408 40151a: e8 dd ff ff ff callq   4014fc <func4>
409 40151f: 44 8d 2c 18    lea     (%rax,%rbx,1),%r13d
410 401523: 41 8d 7c 24 fe lea     -0x2(%r12),%edi
411 401528: 89 de          mov     %ebx,%esi
412 40152a: e8 cd ff ff ff callq   4014fc <func4>
413 40152f: 44 01 e8       add     %r13d,%eax
414 401532: 48 83 c4 08    add     $0x8,%rsp
415 401536: 5b             pop     %rbx
416 401537: 41 5c          pop     %r12
417 401539: 41 5d          pop     %r13
418 40153b: 5d             pop     %rbp
419 40153c: c3             retq
420 40153d: b8 00 00 00 00 mov     $0x0,%eax
421 401542: c3             retq
422 401543: 89 f0          mov     %esi,%eax
423 401545: eb eb          jmp     401532 <func4+0x36>

```

可以发现在 408 行与 412 行都对函数进行了递归调用。而通过对主函数的分析可以发现，输入的还是两个整数（两个整数后输入一个特定字符串可以进入隐藏关卡，这一点最后说），而第二个整数必须是 2，相关代码如下，

```

434 401566: 83 f8 02       cmp     $0x2,%eax
435 401569: 75 0d          jne     401578 <phase_4+0x31>
436 40156b: 8b 45 fc       mov     -0x4(%rbp),%eax
437 40156e: 83 f8 01       cmp     $0x1,%eax
438 401571: 7e 05          jle     401578 <phase_4+0x31>

```

确定了这一点之后就可以通过对递归函数的分析来得到输入的第一个数字是多少。由于这个递归函数的递归关系较为复杂，因此直接使用了一个 c 语言程序来实现这一功能，c 语言程序中的递归函数如下，

```

int fun4(int edi,int esi,int eax)
{
    if(edi<=0)return 0;
    int r13;
    int r12=edi;
    int ebx=esi;
    if(edi==1)
    {
        eax=esi;
        return eax;
    }
    edi--;
    eax=fun4(edi,esi,eax);
    r13=eax+ebx;
    edi=r12-2;
    esi=ebx;
    eax=fun4(edi,esi,eax);
    eax+=r13;
    return eax;
}

```

24请按任意键继续. . .

运行此函数可以发现输入的第二个数字应该是 24。这一问需要注意的一点是和上面几个阶段不同，这一阶段是根据输入的第二个数字来确定第一个数字是多少，这是需要注意的。

3.5 阶段 5 的破解与分析

密码如下：ioa`eg

破解过程：

根据对于源码的分析，可以发现这是一个对于指针的考察，而我们发现有两个比较可疑的地址 0x4031f0, 0x4031a6,通过 gdb 查看，可以发现这是如下的字符串：

```

(gdb) x/s 0x4031f0
0x4031f0 <array.3401>: "maduiersnfotvbylSo you think you can stop the bomb with
ctrl-c, do you?"

```

```
(gdb) x 0x4031a6
0x4031a6: "flames"
```

可以猜测我们需要用到的是 `maduiersnfotvbyl`，根据后面的句子，显然这两个字符串不是最终的答案，最后的结果应该是根据这两个字符串进行一定的处理之后的一个结果，现在只是完成了第一步。

```

461 4015ae: b8 00 00 00 00 mov $0x0,%eax
462 4015b3: 83 f8 05 cmp $0x5,%eax
463 4015b6: 7f 21 jg 4015d9 <phase_5+0x41>
464 4015b8: 48 63 c8 movslq %eax,%rcx
465 4015bb: 0f b6 14 0b movzbl (%rbx,%rcx,1),%edx
466 4015bf: 83 e2 0f and $0xf,%edx
467 4015c2: 0f b6 92 f0 31 40 00 movzbl 0x4031f0(%rdx),%edx
468 4015c9: 88 54 0d e9 mov %dl,-0x17(%rbp,%rcx,1)
469 4015cd: 83 c0 01 add $0x1,%eax
470 4015d0: eb e1 jmp 4015b3 <phase_5+0x1b>
471 4015d2: e8 37 03 00 00 callq 40190e <explode_bomb>

```

通过对上面这一部分代码的分析可以发现这是一个循环，则可以分析出输入的应该是一个 6 个字符组成的字符串，而通过对图中圈起的代码可以发现，这是对于读入的一个字符的处理，目的是得到该字符的 `ascii` 码低四位，通过对代码的分析我们可以清楚地发现，我们要以输入的六个字符的 `ASCII` 码的低四位作为字符串 “`maduiersnfotvbyl`” 的索引去除对应的六个字符，使这六个字符按顺序组成的新的字符串和 “`flames`” 相同。

我们发现 `flames` 中的字符在 `maduiersnfotvbyl` 中的顺序为 9,15,1,0,5,7 那么我们需要输入的字符就是 `ascii` 码低四位分别为 9,15,1,0,5,7 的字符，通过查找 `ascii` 码表我们可以查找到一个符合条件的字符串：`ioa`eg`

3.6 阶段 6 的破解与分析

密码如下：1 5 3 2 4 6

破解过程：

通过对第六阶段的反汇编代码的分析可以发现这一阶段需要输入的是六个整型数。由于这一阶段的反汇编代码很长，因此很难一步到位把握住这一阶段需要进行的任务是什么，那么可以先寻找是否有如上几问一样的地址存储的信息，可以先通过这一信息分析出一部分题解。可以发现有一个可疑的

地址：0x4052d0,通过 gdb 查看这个地址中的信息，可以发现这是一个如下链表：

```
(gdb) x/32wx 0x4052d0
0x4052d0 <node1>:      0x00000072      0x00000001      0x004052e0
00
0x4052e0 <node2>:      0x00000252      0x00000002      0x004052f0
00
0x4052f0 <node3>:      0x0000019a      0x00000003      0x00405300
00
0x405300 <node4>:      0x00000264      0x00000004      0x00405310
00
0x405310 <node5>:      0x000000d1      0x00000005      0x00405320
00
0x405320 <node6>:      0x000003ba      0x00000006      0x00000000
00
0x405330 <bomb_id>:    0x00000212      0x00000000      0x00000000
00
0x405340 <host_table>: 0x00403369      0x00000000      0x00403383
00
```

那么我们就可以大胆猜测这是对于这个链表进行的操作，那么我们再回头看看上面的代码，可以发现一个如下的循环，根据这个循环我们就可以大致得出这一段代码完成的任务是什么了。

525	401675:	7f 1c	jg	401693 <phase_6+0x96>
526	401677:	b8 01 00 00 00	mov	\$0x1,%eax
527	40167c:	ba d0 52 40 00	mov	\$0x4052d0,%edx
528	401681:	48 63 ce	movslq	%esi,%rcx
529	401684:	39 44 8d c0	cmp	%eax,-0x40(%rbp,%rcx,4)
530	401688:	7e e0	jle	40166a <phase_6+0x6d>
531	40168a:	48 8b 52 08	mov	0x8(%rdx),%rdx
532	40168e:	83 c0 01	add	\$0x1,%eax
533	401691:	eb ee	jmp	401681 <phase_6+0x84>
534	401693:	48 8b 5d 90	mov	-0x70(%rbp),%rbx
535	401697:	48 89 d9	mov	%rbx,%rcx
536	40169a:	b8 01 00 00 00	mov	\$0x1,%eax
537	40169f:	eb 12	jmp	4016b3 <phase_6+0xb6>
538	4016a1:	48 63 d0	movslq	%eax,%rdx
539	4016a4:	48 8b 54 d5 90	mov	-0x70(%rbp,%rdx,8),%rdx
540	4016a9:	48 89 51 08	mov	%rdx,0x8(%rcx)
541	4016ad:	83 c0 01	add	\$0x1,%eax
542	4016b0:	48 89 d1	mov	%rdx,%rcx
543	4016b3:	83 f8 05	cmp	\$0x5,%eax
544	4016b6:	7e e9	jle	4016a1 <phase_6+0xa4>

我们可以发现，这一段代码的对链表中的值进行了比较，最后达到炸弹不爆炸的要求应该是经过处理后的链表中存储的值应该是从小到大的，那么就可以大致猜测出这一段代码是对链表进行了重排之后希望使得链表中的值是从小到大排列的，那么如果真的是这样的话，我们可以得到一个数字列表：1 5 3

2 4 6, 经过测试, 确实是这样的。这一段代码实现的就是根据输入的数字与链表的节点的标号比较对链表进行重排后这一链表内存储的值应该从小到大的。

3.7 阶段 7 的破解与分析(隐藏阶段)

密码如下: 35

破解过程:

通过对第六阶段下面的代码进行分析, 可以发现这个炸弹应该是存在第七阶段的, 而第七阶段的入口应该是在前面的某一关中的, 首先我们找到 `phase_defused` 函数, 这一函数进行的就是判断是否进入隐藏阶段, 对于这一函数中出现的每一个地址查看内容, 我们可以发现三个有效的地址内容, 如下:

```
(gdb) x/s 0x403359
0x403359:      "%d %d %s"
(gdb) x/s 0x405870
0x405870 <input_strings+240>:  "24 2 "
```

24 与 2 就是我们在第四问的时候输入的内容, 而上面一个地址表明进入隐藏阶段的入口处输入的应该是两个整型数与一个字符串, 那么可以大胆猜测就是在第四阶段的时候多输入一个字符串就可以进入隐藏阶段, 而字符串信息如下:

```
(gdb) x/s 0x403362
0x403362:      "DrEvil"
```

经过求证, 确实进入了隐藏阶段, 那么接下来我们就对隐藏阶段的代码进行分析。可以发现这个同样是一个递归的函数, 根据如下反汇编我们得知我们应该输入一个 1~1001 之间的数,

598	401744:	8d 40 ff	lea	-0x1(%rax),%eax
599	401747:	3d e8 03 00 00	cmp	\$0x3e8,%eax
600	40174c:	77 27	ja	401775 <secret_phase+0x49>
601	40174e:	89 de	mov	%ebx,%esi
602	401750:	bf f0 50 40 00	mov	\$0x4050f0,%edi

而根据如下代码我们又得出 fun7 的返回值必须为 6,

```
40175a:      83 f8 06          cmp     $0x6,%eax
```

那么我们就可以对 fun7 进行分析, 很显然这是一个递归函数, 而 fun7 的代码大致可以用如下 c 语言代码模拟:

```

5  int fun7(int *x, int a)//%edi存放x, %esi存放a, 返回值存放在%rax
6  {
7      if(x == 0){
8          return -1;
9      }
10     int result = *x;
11     if (result > a) {
12         int temp = *(x+10);
13         result = fun7(&temp, a);
14         result *= 2;
15         return result;
16     }
17     else if(result < a){
18         int temp = *(x+8);
19         result = fun7(&temp, a);
20         result = result * 2 + 1;
21         return result;
22     }
23     return 0;
24 }
```

经过分析我们可以发现返回值应该为 $6 = ((0 * 2 + 1) * 2 + 1) * 2$, 共 4 次递归, 那么我们就可以直接使用 gdb 查看对应地址中的数值即可, 具体过程如下:

第一次时 %rsi 的值 <36, 并读取 0x4050f8 的中存的地址值供递归调用:

```

(gdb) x/ 0x4050f0
0x4050f0 <n1>: 0x00000024
(gdb) x/x 0x4050f8
0x4050f8 <n1+8>: 0x00405110
```

第二次时%rsi 的值>8，并读取 0x405120 中存的地址值供递归调用：

```
(gdb) x/ 0x405110
0x405110 <n21>: 0x00000008
(gdb) x/ 0x405120
0x405120 <n21+16>: 0x00405150
(gdb) x/ 0x405150
```

第三次时%rsi 的值>22，并读取 0x405160 中存的地址值供递归调用：

```
(gdb) x/ 0x405150
0x405150 <n32>: 0x00000016
(gdb) x/ 0x405160
0x405160 <n32+16>: 0x00405230
(gdb) x/ 0x405230
```

最后读取 0x405230 中的内容，这时这里的内容就应该是我们要输入的答案了。

```
(gdb) x/ 0x405230
0x405230 <n44>: 0x00000023
(gdb) x/ 0x405230
```


第 4 章 总结

4.1 请总结本次实验的收获

本次实验主要使我加深了对反汇编代码的理解，同时对于在 `gdb` 中的调试有了更加深入的了解。

4.2 请给出对本次实验内容的建议

可以增加更多具有趣味性的阶段。

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