北京郵電大學

实验报告

题目：拆解二进制炸弹

班　　级：2020211314

学　　号：2020211616

姓　　名：付容天

学　　院：计算机学院（国家示范性软件学院）

2021 年 11 月 3 日

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. SecureCRT（10.120.11.12）
2. Linux
3. Objdump 命令反汇编
4. GDB 调试工具
5. Windows Powershell

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

第一阶段

首先进行第一阶段的实验。设置断点在 main 函数处，并开始运行，得到如下结果：

```
For help, type "help".
Type "apropos word" to search for commands related to "word"...
bomb34: No such file or directory.
(gdb) file bomb34
bomb34: No such file or directory.
(gdb) file bomb
Reading symbols from bomb...
(gdb) break main
Breakpoint 1 at 0x400df6: file bomb.c, line 37.
(gdb) run
Starting program: /students/2020211616/bomb34/bomb

Breakpoint 1, main (argc=1, argv=0x7fffffffeb98) at bomb.c:37
37      {
(gdb) disass
Dump of assembler code for function main:
=> 0x0000000000400df6 <+0>:     push   %rbx
   0x0000000000400df7 <+1>:     cmp    $0x1,%edi
   0x0000000000400dfa <+4>:     jne    0x400e0c <main+22>
   0x0000000000400dfc <+6>:     mov    0x20398d(%rip),%rax        # 0x604790 <stdin@@GLIBC_2.2.5>
   0x0000000000400e03 <+13>:    mov    %rax,0x2039a6(%rip)        # 0x6047b0 <infile>
   0x0000000000400e0a <+20>:    jmp    0x400e6f <main+121>
   0x0000000000400e0c <+22>:    mov    %rsi,%rbx
   0x0000000000400e0f <+25>:    cmp    $0x2,%edi
   0x0000000000400e12 <+28>:    jne    0x400e4e <main+88>
   0x0000000000400e14 <+30>:    mov    0x8(%rsi),%rdi
   0x0000000000400e18 <+34>:    mov    $0x402564,%esi
   0x0000000000400e1d <+39>:    callq  0x400c60 <fopen@plt>
   0x0000000000400e22 <+44>:    mov    %rax,0x203987(%rip)        # 0x6047b0 <infile>
   0x0000000000400e29 <+51>:    test   %rax,%rax
   0x0000000000400e2c <+54>:    jne    0x400e6f <main+121>
   0x0000000000400e2e <+56>:    mov    0x8(%rbx),%rcx
   0x0000000000400e32 <+60>:    mov    (%rbx),%rdx
   0x0000000000400e35 <+63>:    mov    $0x402566,%esi
```

（图 1-阶段 1-1）

```
   0x0000000000400e79 <+131>:   callq  0x400b70 <puts@plt>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400e7e <+136>:   mov    $0x402628,%edi
   0x0000000000400e83 <+141>:   callq  0x400b70 <puts@plt>
   0x0000000000400e88 <+146>:   callq  0x401755 <read_line>
   0x0000000000400e8d <+151>:   mov    %rax,%rdi
   0x0000000000400e90 <+154>:   callq  0x400f2d <phase_1>
   0x0000000000400e95 <+159>:   callq  0x40187b <phase_defused>
   0x0000000000400e9a <+164>:   mov    $0x402658,%edi
   0x0000000000400e9f <+169>:   callq  0x400b70 <puts@plt>
   0x0000000000400ea4 <+174>:   callq  0x401755 <read_line>
   0x0000000000400ea9 <+179>:   mov    %rax,%rdi
   0x0000000000400eac <+182>:   callq  0x400f40 <phase_2>
   0x0000000000400eb1 <+187>:   callq  0x40187b <phase_defused>
   0x0000000000400eb6 <+192>:   mov    $0x40259d,%edi
   0x0000000000400ebb <+197>:   callq  0x400b70 <puts@plt>
   0x0000000000400ec0 <+202>:   callq  0x401755 <read_line>
   0x0000000000400ec5 <+207>:   mov    %rax,%rdi
   0x0000000000400ec8 <+210>:   callq  0x400fb5 <phase_3>
   0x0000000000400ecd <+215>:   callq  0x40187b <phase_defused>
   0x0000000000400ed2 <+220>:   mov    $0x4025bb,%edi
   0x0000000000400ed7 <+225>:   callq  0x400b70 <puts@plt>
   0x0000000000400edc <+230>:   callq  0x401755 <read_line>
   0x0000000000400ee1 <+235>:   mov    %rax,%rdi
   0x0000000000400ee4 <+238>:   callq  0x4010b8 <phase_4>
   0x0000000000400ee9 <+243>:   callq  0x40187b <phase_defused>
   0x0000000000400eee <+248>:   mov    $0x402688,%edi
   0x0000000000400ef3 <+253>:   callq  0x400b70 <puts@plt>
   0x0000000000400ef8 <+258>:   callq  0x401755 <read_line>
   0x0000000000400efd <+263>:   mov    %rax,%rdi
   0x0000000000400f00 <+266>:   callq  0x40112b <phase_5>
   0x0000000000400f05 <+271>:   callq  0x40187b <phase_defused>
   0x0000000000400f0a <+276>:   mov    $0x4025ca,%edi
   0x0000000000400f0f <+281>:   callq  0x400b70 <puts@plt>
```

（图 2-阶段 1-2）

从上面的两幅图中分析下一个断点的设置位置，观察得到：应该在 read_line 函数之后、phase_1 函数之前设置断点，一开始我尝试了三次均失败，失败截图如下：

```
   0x0000000000400ef3 <+253>:   callq  0x400b70 <puts@plt>
   0x0000000000400ef8 <+258>:   callq  0x401755 <read_line>
   0x0000000000400efd <+263>:   mov    %rax,%rdi
   0x0000000000400f00 <+266>:   callq  0x40112b <phase_5>
   0x0000000000400f05 <+271>:   callq  0x40187b <phase_defused>
   0x0000000000400f0a <+276>:   mov    $0x4025ca,%edi
   0x0000000000400f0f <+281>:   callq  0x400b70 <puts@plt>
   0x0000000000400f14 <+286>:   callq  0x401755 <read_line>
   0x0000000000400f19 <+291>:   mov    %rax,%rdi
   0x0000000000400f1c <+294>:   callq  0x4011b7 <phase_6>
   0x0000000000400f21 <+299>:   callq  0x40187b <phase_defused>
   0x0000000000400f26 <+304>:   mov    $0x0,%eax
   0x0000000000400f2b <+309>:   pop    %rbx
   0x0000000000400f2c <+310>:   retq
End of assembler dump.
(gdb) break 0x0000000000400e90
Function "0x0000000000400e90" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (0x0000000000400e90) pending.
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) n
Program not restarted.
(gdb) continue
Continuing.
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
si

BOOM!!!
The bomb has blown up.
Your instructor has been notified.
[Inferior 1 (process 2682853) exited with code 010]
(gdb)
```

（图 3-阶段 1-3）

```
   0x0000000000400ee1 <+235>:   mov    %rax,%rdi
   0x0000000000400ee4 <+238>:   callq  0x4010b8 <phase_4>
   0x0000000000400ee9 <+243>:   callq  0x40187b <phase_defused>
   0x0000000000400eee <+248>:   mov    $0x402688,%edi
   0x0000000000400ef3 <+253>:   callq  0x400b70 <puts@plt>
   0x0000000000400ef8 <+258>:   callq  0x401755 <read_line>
   0x0000000000400efd <+263>:   mov    %rax,%rdi
   0x0000000000400f00 <+266>:   callq  0x40112b <phase_5>
   0x0000000000400f05 <+271>:   callq  0x40187b <phase_defused>
   0x0000000000400f0a <+276>:   mov    $0x4025ca,%edi
   0x0000000000400f0f <+281>:   callq  0x400b70 <puts@plt>
   0x0000000000400f14 <+286>:   callq  0x401755 <read_line>
   0x0000000000400f19 <+291>:   mov    %rax,%rdi
   0x0000000000400f1c <+294>:   callq  0x4011b7 <phase_6>
   0x0000000000400f21 <+299>:   callq  0x40187b <phase_defused>
   0x0000000000400f26 <+304>:   mov    $0x0,%eax
   0x0000000000400f2b <+309>:   pop    %rbx
   0x0000000000400f2c <+310>:   retq
End of assembler dump.
(gdb) break 0x0000000000400e88
Function "0x0000000000400e88" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (0x0000000000400e88) pending.
(gdb) continue
Continuing.
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
123

BOOM!!!
The bomb has blown up.
Your instructor has been notified.
[Inferior 1 (process 2683436) exited with code 010]
(gdb)
```

（图 4-阶段 1-4）

```
   0x0000000000400edc <+230>:   callq  0x401755 <read_line>
   0x0000000000400ee1 <+235>:   mov    %rax,%rdi
   0x0000000000400ee4 <+238>:   callq  0x4010b8 <phase_4>
   0x0000000000400ee9 <+243>:   callq  0x40187b <phase_defused>
   0x0000000000400eee <+248>:   mov    $0x402688,%edi
   0x0000000000400ef3 <+253>:   callq  0x400b70 <puts@plt>
   0x0000000000400ef8 <+258>:   callq  0x401755 <read_line>
   0x0000000000400efd <+263>:   mov    %rax,%rdi
   0x0000000000400f00 <+266>:   callq  0x40112b <phase_5>
   0x0000000000400f05 <+271>:   callq  0x40187b <phase_defused>
   0x0000000000400f0a <+276>:   mov    $0x4025ca,%edi
   0x0000000000400f0f <+281>:   callq  0x400b70 <puts@plt>
   0x0000000000400f14 <+286>:   callq  0x401755 <read_line>
   0x0000000000400f19 <+291>:   mov    %rax,%rdi
   0x0000000000400f1c <+294>:   callq  0x4011b7 <phase_6>
   0x0000000000400f21 <+299>:   callq  0x40187b <phase_defused>
   0x0000000000400f26 <+304>:   mov    $0x0,%eax
   0x0000000000400f2b <+309>:   pop    %rbx
   0x0000000000400f2c <+310>:   retq
End of assembler dump.
(gdb) break 0x0000000000400e8d
Function "0x0000000000400e8d" not defined.
Make breakpoint pending on future shared library load? (y or [n]) y
Breakpoint 2 (0x0000000000400e8d) pending.
(gdb) continue
Continuing.
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
12345

BOOM!!!
The bomb has blown up.
Your instructor has been notified.
[Inferior 1 (process 2683774) exited with code 010]
```

（图 5-阶段 1-5）

一开始我认为是断点设置有问题，所以我三次设置了不同位置的断点，但是均失败了。我回顾了老师在课上的教学内容，很快发现这三次的失误均是由一个很基本的问题引起的：设置断点的格式不正确。之后，我使用了符合规范的命令来设置断点，很快就解决了第一个问题，过程截图如下：

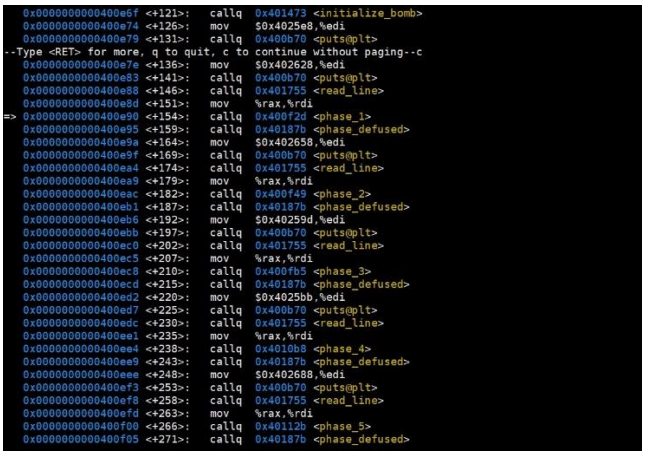
```
   0x0000000000400f21 <+299>:   callq  0x40187b <phase_defused>
   0x0000000000400f26 <+304>:   mov    $0x0,%eax
   0x0000000000400f2b <+309>:   pop    %rbx
   0x0000000000400f2c <+310>:   retq
End of assembler dump.
(gdb) break *0x400e8d
Breakpoint 2 at 0x400e8d: file bomb.c, line 74.
(gdb) c
Continuing.
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
12345

Breakpoint 2, main (argc=<optimized out>, argv=<optimized out>) at bomb.c:74
74          phase_1(input);                 /* Run the phase             */
(gdb) disass
Dump of assembler code for function main:
   0x0000000000400df6 <+0>:     push   %rbx
   0x0000000000400df7 <+1>:     cmp    $0x1,%edi
   0x0000000000400dfa <+4>:     jne    0x400e0c <main+22>
   0x0000000000400dfc <+6>:     mov    0x20398d(%rip),%rax        # 0x604790 <stdin@@GLIBC_2.2.5>
   0x0000000000400e03 <+13>:    mov    %rax,0x2039a6(%rip)        # 0x6047b0 <infile>
   0x0000000000400e0a <+20>:    jmp    0x400e6f <main+121>
   0x0000000000400e0c <+22>:    mov    %rsi,%rbx
   0x0000000000400e0f <+25>:    cmp    $0x2,%edi
   0x0000000000400e12 <+28>:    jne    0x400e4e <main+88>
   0x0000000000400e14 <+30>:    mov    0x8(%rsi),%rdi
   0x0000000000400e18 <+34>:    mov    $0x402564,%esi
   0x0000000000400e1d <+39>:    callq  0x400c60 <fopen@plt>
   0x0000000000400e22 <+44>:    mov    %rax,0x203987(%rip)        # 0x6047b0 <infile>
   0x0000000000400e29 <+51>:    test   %rax,%rax
   0x0000000000400e2c <+54>:    jne    0x400e6f <main+121>
   0x0000000000400e2e <+56>:    mov    0x8(%rbx),%rcx
   0x0000000000400e32 <+60>:    mov    (%rbx),%rdx
```

（图 6-阶段 1-6）

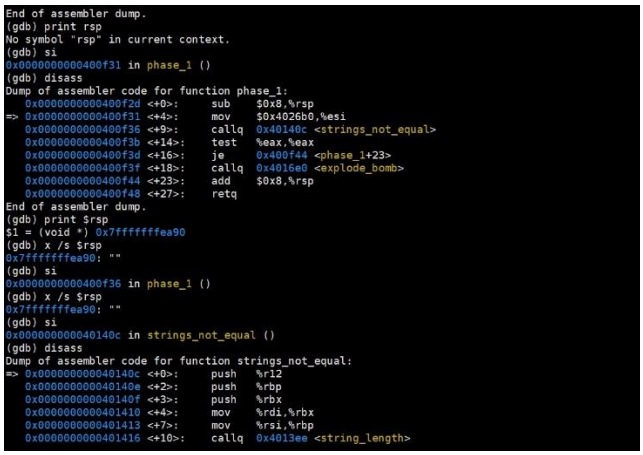
```
   0x0000000000400e6a <+116>:   callq  0x400c80 <exit@plt>
   0x0000000000400e6f <+121>:   callq  0x401473 <initialize_bomb>
   0x0000000000400e74 <+126>:   mov    $0x4025e8,%edi
   0x0000000000400e79 <+131>:   callq  0x400b70 <puts@plt>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x0000000000400e7e <+136>:   mov    $0x402628,%edi
   0x0000000000400e83 <+141>:   callq  0x400b70 <puts@plt>
   0x0000000000400e88 <+146>:   callq  0x401755 <read_line>
=> 0x0000000000400e8d <+151>:   mov    %rax,%rdi
   0x0000000000400e90 <+154>:   callq  0x400f2d <phase_1>
   0x0000000000400e95 <+159>:   callq  0x40187b <phase_defused>
   0x0000000000400e9a <+164>:   mov    $0x402658,%edi
   0x0000000000400e9f <+169>:   callq  0x400b70 <puts@plt>
   0x0000000000400ea4 <+174>:   callq  0x401755 <read_line>
   0x0000000000400ea9 <+179>:   mov    %rax,%rdi
   0x0000000000400eac <+182>:   callq  0x400f49 <phase_2>
   0x0000000000400eb1 <+187>:   callq  0x40187b <phase_defused>
   0x0000000000400eb6 <+192>:   mov    $0x40259d,%edi
   0x0000000000400ebb <+197>:   callq  0x400b70 <puts@plt>
   0x0000000000400ec0 <+202>:   callq  0x401755 <read_line>
   0x0000000000400ec5 <+207>:   mov    %rax,%rdi
   0x0000000000400ec8 <+210>:   callq  0x400fb5 <phase_3>
   0x0000000000400ecd <+215>:   callq  0x40187b <phase_defused>
   0x0000000000400ed2 <+220>:   mov    $0x4025bb,%edi
   0x0000000000400ed7 <+225>:   callq  0x400b70 <puts@plt>
   0x0000000000400edc <+230>:   callq  0x401755 <read_line>
   0x0000000000400ee1 <+235>:   mov    %rax,%rdi
   0x0000000000400ee4 <+238>:   callq  0x4010b8 <phase_4>
   0x0000000000400ee9 <+243>:   callq  0x40187b <phase_defused>
   0x0000000000400eee <+248>:   mov    $0x402688,%edi
   0x0000000000400ef3 <+253>:   callq  0x400b70 <puts@plt>
   0x0000000000400ef8 <+258>:   callq  0x401755 <read_line>
   0x0000000000400efd <+263>:   mov    %rax,%rdi
   0x0000000000400f00 <+266>:   callq  0x40112b <phase_5>
```

（图 7-阶段 1-7）

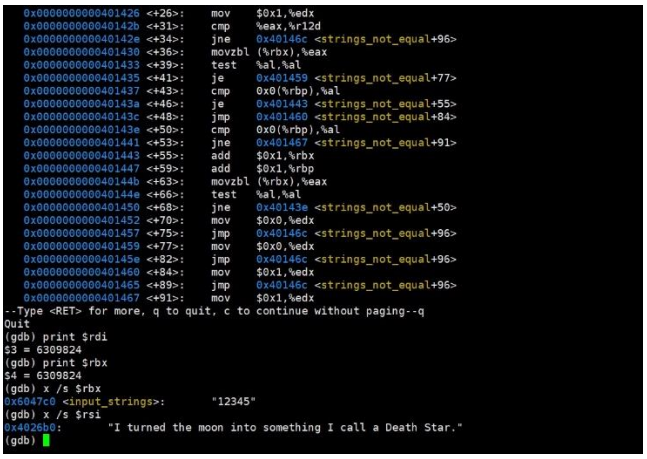
在图 7 中可以看到，此时 gdb 即将跟进到 phase_1 函数中，此时使用 si 指令使其进入到 phase_1 函数中，再进行后续的调试，过程截图如下：



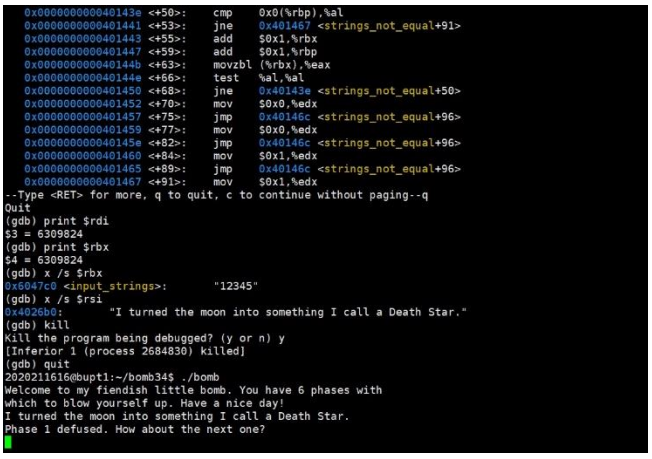
（图 8-阶段 1-8）



（图 9-阶段 1-9）



（图 10-阶段 1-10）

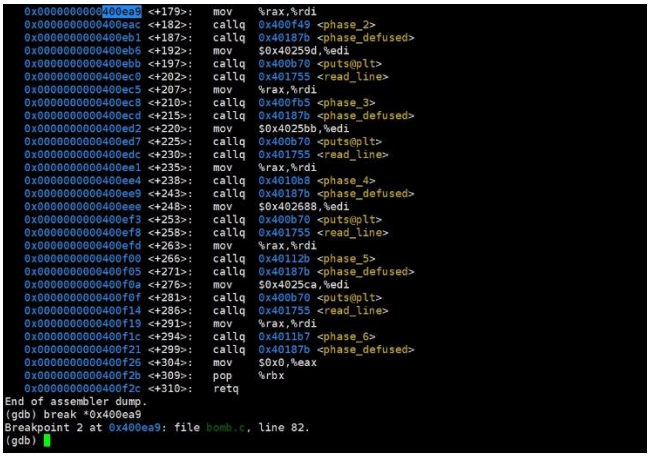


（图 11-阶段 1-11）

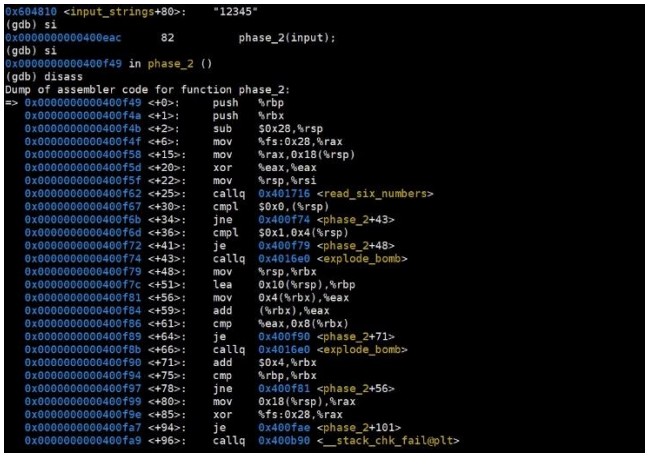
在图 8 和图 9 中猜测其涉及到的寄存器中含有我们输入的值和正确的答案，因此多次输出，检测是否能够输出有效值，如果输出的是有效值，那么就得到了正确答案。在图 10 中，我们得到了正确答案（I turned the moon into something I call a Death Star.），并且由图 11 知道，我们已经正确地破解了第一阶段。

第二阶段

然后进行第二阶段的实验，过程截图如下：



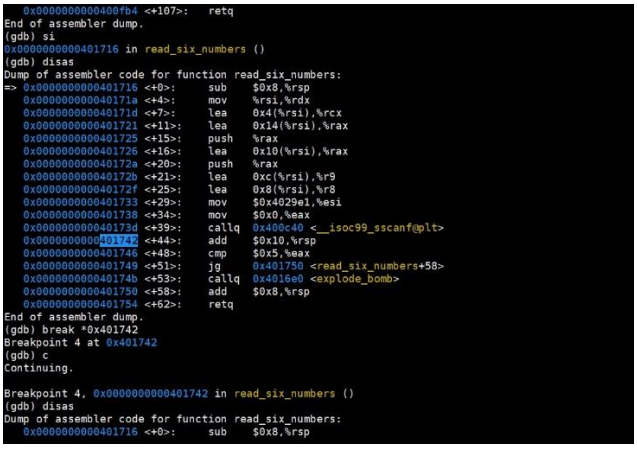
（图 12-阶段 2-1）



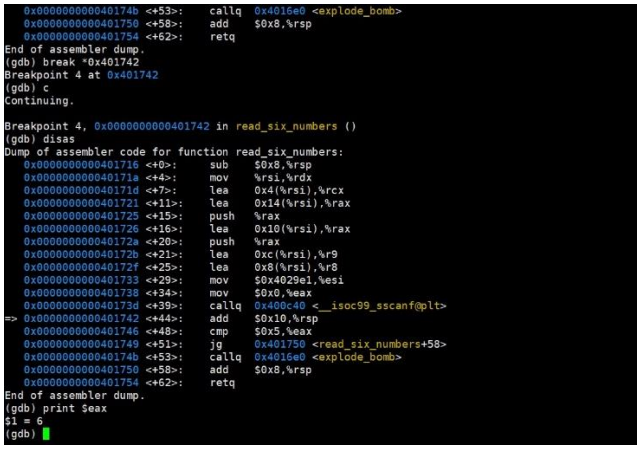
（图 13-阶段 2-2）

图 12 中展示了我们的断点设置位置，输入第一阶段的密码后，进入 phase_2 中（如图 13 所示），观察发现，欲使<+43>处的 explode_bomb 函数跳过，应使<+34>处的跳转指令不执行，于是设置断点在<+34>处，continue

之后发现炸弹爆炸，分析原因，可能是<+25>处的读取函数中有问题，令 gdb 跟进此函数，看到如下的情形：

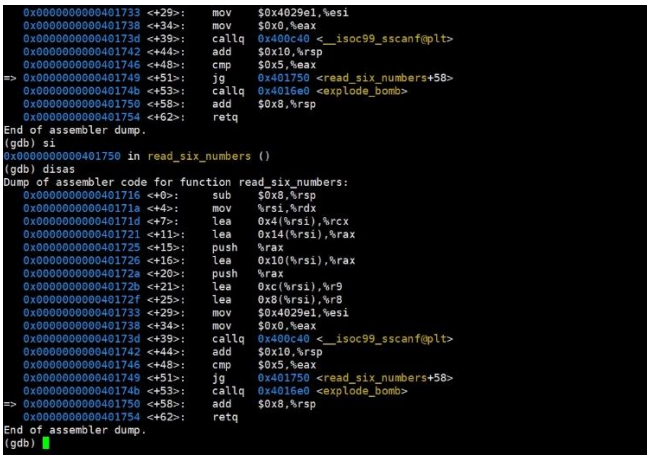


（图 14-阶段 2-3）

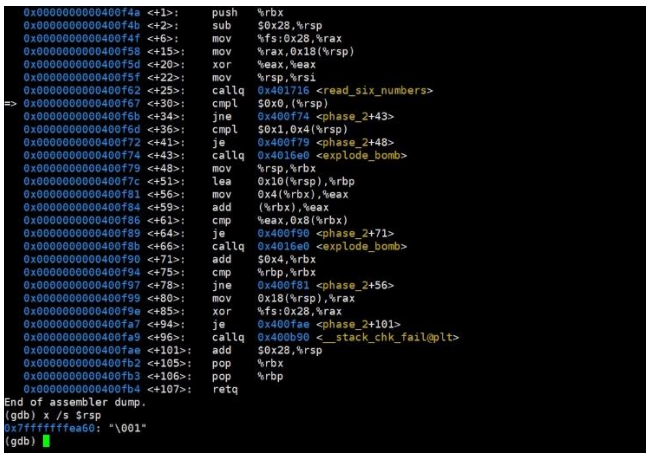


（图 15-阶段 2-4）

如图 14,从<+51>处可以知道,只有当输入的 number 数目大于 5 的时候,才能够跳过<+53>处的 explode_bomb 函数，因此在运行第二阶段的时候，输入 1 2 3 4 5 6 7 共七个数字，得到寄存器 eax 的值为 6（如图 15 所示），此时<+53>处的 explode_bomb 函数不执行（如下图 16 所示，explode_bomb 函数确实被跳过了）。

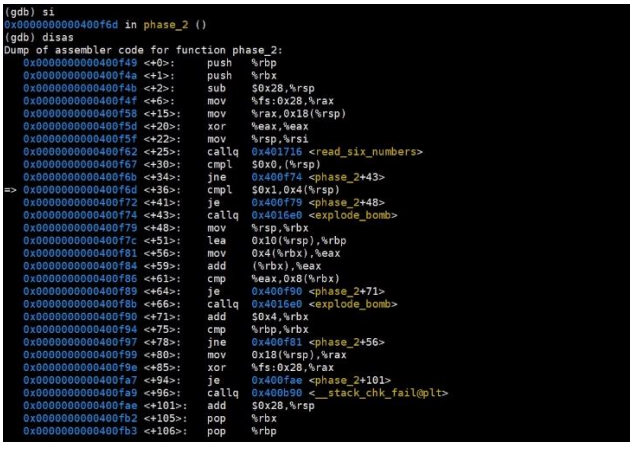


（图 16-阶段 2-5）

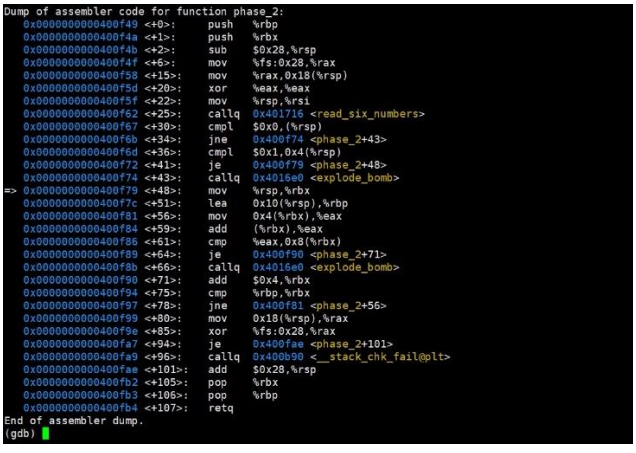


（图 17-阶段 2-6）

接下来回到了 phase_2 函数当中，如果要使<+43>处的 explode_bomb 函数不执行，那么<+34>处的跳转指令就应当不被执行，那么由<+30>处知道，输入的第一个数字必须是 0（从下图 18 知道<+34>处的跳转语句确实不被执行）；然后欲使<+41>处的跳转语句被执行，那么从<+36>处可以知道输入的第二个数字必须为 1（从下图 19 知道<+43>处的 explode_bomb 函数确实被跳过了）。上述两个过程的截图如下：



（图 18-阶段 2-7）



（图 19-阶段 2-8）

然后对图 19 中剩下的指令进行分析如下：rsp 指向第 1 个数字，第一个数字必须为 0；rbx 指向第 1 个数字；rbp 指向第 5 个数字；eax 的值为第 2 个数字；第 2 个数字必须为 1；eax 的值更新为第 2 个数字加上第 3 个数字；第三个数字必须等于 eax 的值，因此第三个数字为 1；rbx 指向第 2 个；第 2 个与第 5 个数字进行比较，假设相等，那么将 rax 的值更新为第 7 个数字（实际上不存在），查阅相关资料知道%fs:0x28 实际上实现的是“程序哨兵”的功能。之后使用类似的分析方法，最后可以得到正确答案（0 1 1 2 3 5，后面可以添加更多的输入，但是前 6 个输入必须如此），下面是破解成功的实验截图：

```
   0x0000000000400eee <+248>:   mov    $0x402688,%edi
   0x0000000000400ef3 <+253>:   callq  0x400b70 <puts@plt>
   0x0000000000400ef8 <+258>:   callq  0x401755 <read_line>
   0x0000000000400efd <+263>:   mov    %rax,%rdi
   0x0000000000400f00 <+266>:   callq  0x40112b <phase_5>
   0x0000000000400f05 <+271>:   callq  0x40187b <phase_defused>
   0x0000000000400f0a <+276>:   mov    $0x4025ca,%edi
   0x0000000000400f0f <+281>:   callq  0x400b70 <puts@plt>
   0x0000000000400f14 <+286>:   callq  0x401755 <read_line>
   0x0000000000400f19 <+291>:   mov    %rax,%rdi
   0x0000000000400f1c <+294>:   callq  0x4011b7 <phase_6>
   0x0000000000400f21 <+299>:   callq  0x40187b <phase_defused>
   0x0000000000400f26 <+304>:   mov    $0x0,%eax
   0x0000000000400f2b <+309>:   pop    %rbx
   0x0000000000400f2c <+310>:   retq
End of assembler dump.
(gdb) si
0x000000000040187b in phase_defused ()
(gdb) quit
A debugging session is active.

        Inferior 1 [process 2719909] will be killed.

Quit anyway? (y or n) y
2020211616@bupt1:~/bomb34$ ls
bomb  bomb.c  README
2020211616@bupt1:~/bomb34$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
I turned the moon into something I call a Death Star.
Phase 1 defused. How about the next one?
0 1 1 2 3 5 6 7
That's number 2.  Keep going!
```

（图 20-阶段 2-9）

第三阶段

在输入了前两关的答案之后，我们进入 phase_3 函数之中，过程截图如下：

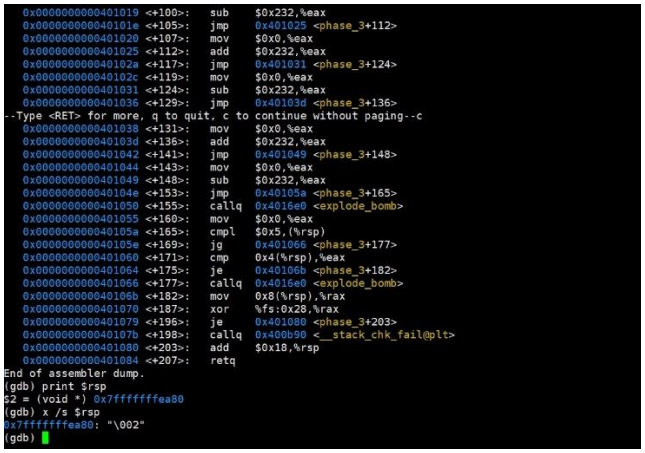
```
   0x0000000000400eb1 <+187>:   callq  0x40187b <phase_defused>
   0x0000000000400eb6 <+192>:   mov    $0x40259d,%edi
   0x0000000000400ebb <+197>:   callq  0x400b70 <puts@plt>
   0x0000000000400ec0 <+202>:   callq  0x401755 <read_line>
   0x0000000000400ec5 <+207>:   mov    %rax,%rdi
   0x0000000000400ec8 <+210>:   callq  0x400fb5 <phase_3>
   0x0000000000400ecd <+215>:   callq  0x40187b <phase_defused>
   0x0000000000400ed2 <+220>:   mov    $0x4025bb,%edi
   0x0000000000400ed7 <+225>:   callq  0x400b70 <puts@plt>
   0x0000000000400edc <+230>:   callq  0x401755 <read_line>
   0x0000000000400ee1 <+235>:   mov    %rax,%rdi
   0x0000000000400ee4 <+238>:   callq  0x4010b8 <phase_4>
   0x0000000000400ee9 <+243>:   callq  0x40187b <phase_defused>
   0x0000000000400eee <+248>:   mov    $0x402688,%edi
   0x0000000000400ef3 <+253>:   callq  0x400b70 <puts@plt>
   0x0000000000400ef8 <+258>:   callq  0x401755 <read_line>
   0x0000000000400efd <+263>:   mov    %rax,%rdi
   0x0000000000400f00 <+266>:   callq  0x40112b <phase_5>
   0x0000000000400f05 <+271>:   callq  0x40187b <phase_defused>
   0x0000000000400f0a <+276>:   mov    $0x4025ca,%edi
   0x0000000000400f0f <+281>:   callq  0x400b70 <puts@plt>
   0x0000000000400f14 <+286>:   callq  0x401755 <read_line>
   0x0000000000400f19 <+291>:   mov    %rax,%rdi
   0x0000000000400f1c <+294>:   callq  0x4011b7 <phase_6>
   0x0000000000400f21 <+299>:   callq  0x40187b <phase_defused>
   0x0000000000400f26 <+304>:   mov    $0x0,%eax
   0x0000000000400f2b <+309>:   pop    %rbx
   0x0000000000400f2c <+310>:   retq
End of assembler dump.
(gdb) break *0x400ec5
Breakpoint 2 at 0x400ec5: file bomb.c, line 89.
(gdb) c
Continuing.
Welcome to my fiendish little bomb. You have 6 phases with
```

（图 21-阶段 3-1）

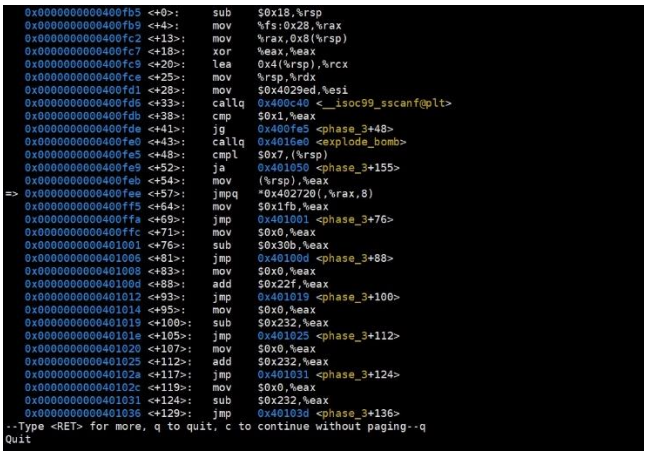
```
   0x0000000000400fb5 <+0>:     sub    $0x18,%rsp
   0x0000000000400fb9 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fc2 <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400fc7 <+18>:    xor    %eax,%eax
   0x0000000000400fc9 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400fce <+25>:    mov    %rsp,%rdx
   0x0000000000400fd1 <+28>:    mov    $0x4029ed,%esi
   0x0000000000400fd6 <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
=> 0x0000000000400fdb <+38>:    cmp    $0x1,%eax
   0x0000000000400fde <+41>:    jg     0x400fe5 <phase_3+48>
   0x0000000000400fe0 <+43>:    callq  0x4016e0 <explode_bomb>
   0x0000000000400fe5 <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400fe9 <+52>:    ja     0x401050 <phase_3+155>
   0x0000000000400feb <+54>:    mov    (%rsp),%eax
   0x0000000000400fee <+57>:    jmpq   *0x402720(,%rax,8)
   0x0000000000400ff5 <+64>:    mov    $0x1fb,%eax
   0x0000000000400ffa <+69>:    jmp    0x401001 <phase_3+76>
   0x0000000000400ffc <+71>:    mov    $0x0,%eax
   0x0000000000401001 <+76>:    sub    $0x30b,%eax
   0x0000000000401006 <+81>:    jmp    0x40100d <phase_3+88>
   0x0000000000401008 <+83>:    mov    $0x0,%eax
   0x000000000040100d <+88>:    add    $0x22f,%eax
   0x0000000000401012 <+93>:    jmp    0x401019 <phase_3+100>
   0x0000000000401014 <+95>:    mov    $0x0,%eax
   0x0000000000401019 <+100>:   sub    $0x232,%eax
   0x000000000040101e <+105>:   jmp    0x401025 <phase_3+112>
   0x0000000000401020 <+107>:   mov    $0x0,%eax
   0x0000000000401025 <+112>:   add    $0x232,%eax
   0x000000000040102a <+117>:   jmp    0x401031 <phase_3+124>
   0x000000000040102c <+119>:   mov    $0x0,%eax
   0x0000000000401031 <+124>:   sub    $0x232,%eax
   0x0000000000401036 <+129>:   jmp    0x40103d <phase_3+136>
--Type <RET> for more, q to quit, c to continue without paging--q
Quit
```

（图 22-阶段 3-2）

上图 21 显示了我们的断点设置位置。之后我们进入了 phase_3 函数，有了 phase_2 中的失败经历，我们首先注意输入数字的数目，注意到只有输入的数字数目至少为 2 个的时候，<+41>处的跳转语句才能被执行，从而<+43>处的 explode_bomb 函数才能被跳过。如上图 22 所示，我们重新开始调试，输入 2 3，并在<+48>处设置断点，打印出寄存器 rsp 所指的内存地址的值，我们得到的结果是 2，如下图 23 所示：

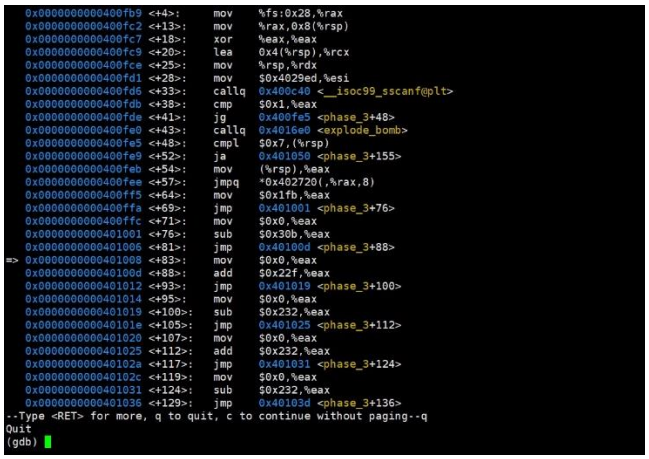


（图 23-阶段 3-3）



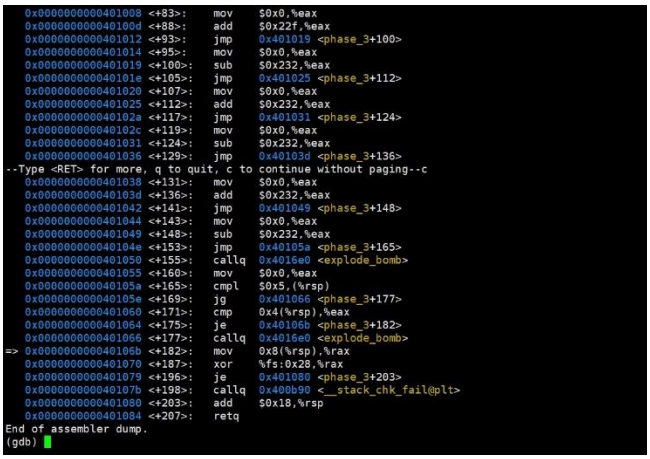
（图 24-阶段 3-4）

那么我们就可以知道寄存器 rsp 指向的地址存放的是我们输入的第一个数字。之后继续使用 si 指令逐步执行，直到遇到<+57>处的跳转语句（如图 24 所示），此时的跳转指令是间接跳转指令，并且以一个地址作为立即数偏移量、以寄存器 rax 作为变址寄存器、以 8 作为比例因子，那么就可以知道 phase_3 函数实际上上的主体就是一则 switch 语句，根据跳转表进行跳转。现在我们限定输入为 2，继续分析如下：

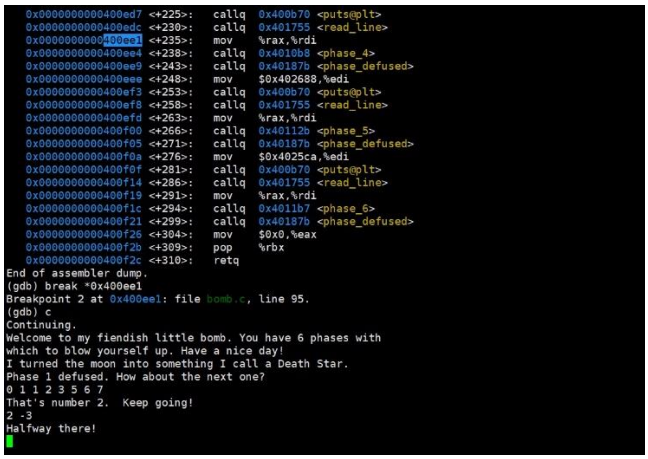


（图 25-阶段 3-5）

（图 26-阶段 3-6）



（图 27-阶段 3-7）



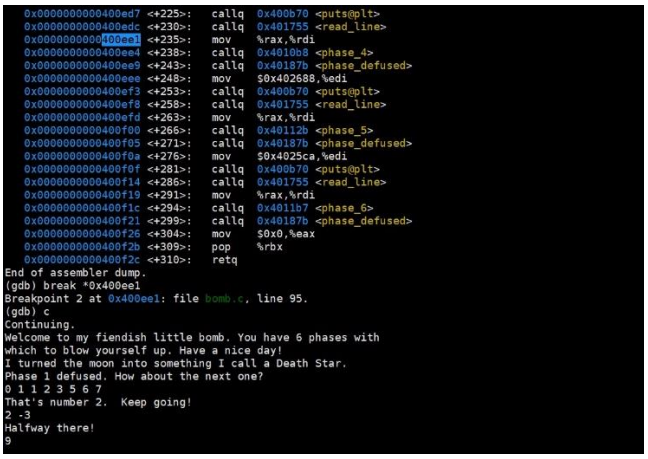
（图 28 阶段 3-8）

执行<+57>处之后的简介跳转语句，我们来到了<+83>处，如图 25 所示。之后使用 si 指令逐步执行，直到<+165>处的比较语句，如果寄存器 rsp 中的值所指的地址上的值大于 5，那么<+169>处的跳转语句便会执行，程序跳转到<+177>处的调用 explode_bomb 函数的语句，此时实验失败。在我们假设的那个输入下，<+169>处的跳转语句不会被执行。分析知道，寄存器 eax 中的值由<+88>、<+100>、<+112>、<+124>、<+136>、<+148>

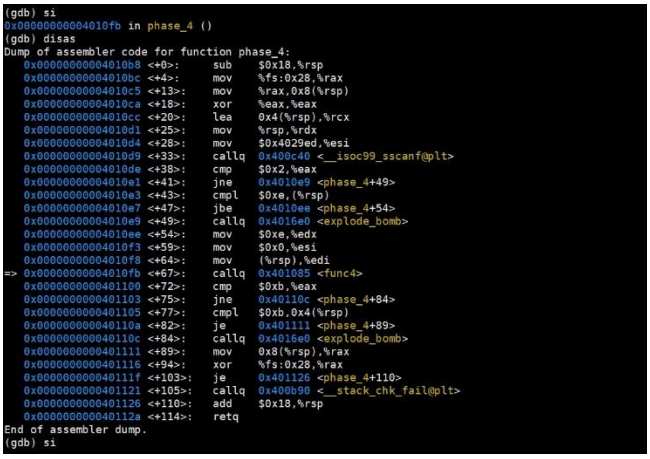
处共六条运算语句所决定，而地址为 0x4(%rsp)的值就是我们输入的第二个数，由<+171>和<+175>两处的语句知道：当且仅当我们输入的第二个数和寄存器 eax 中的值相等的时候，<+177>处的调用 explode_bomb 函数的语句才能被跳过，因为这是最后一个可能调用 explode_bomb 函数的语句，所以当此语句被跳过之后，我们实际上已经解决了第三阶段的问题。最后计算六条运算语句，得到：当第一个输入为 2 的时候，第二个输入应该为-3，成功解决截图见上图 27。

第四阶段

首先输入已经得到的第一至第三阶段的答案，进入第四阶段的机器指令界面如下：

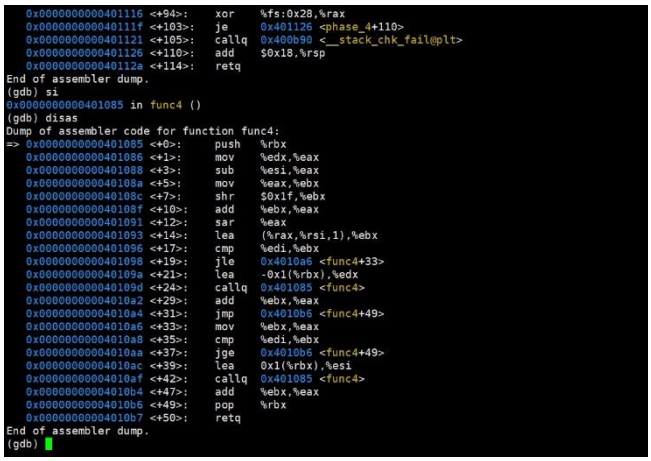


（图 29-阶段 4-1）

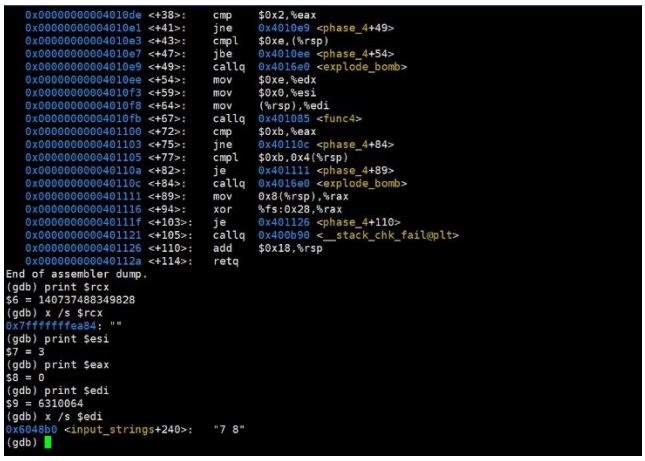


（图 30-阶段 4-2）

图 29 展示了我们进入 phase_4 的断点设置位置。从图 30 中<+38>看出，只有输入的数字个数等于 2，才不会调用<+49>处的 explode_bomb 函数，并且，从<+72>处知道该函数的返回值一定要是 11 才行，并且根据<+77>处可以知道第二个输入一定要是 11 才行，因此我们输入 1 11 两个数字进行调试，过程截图如下：



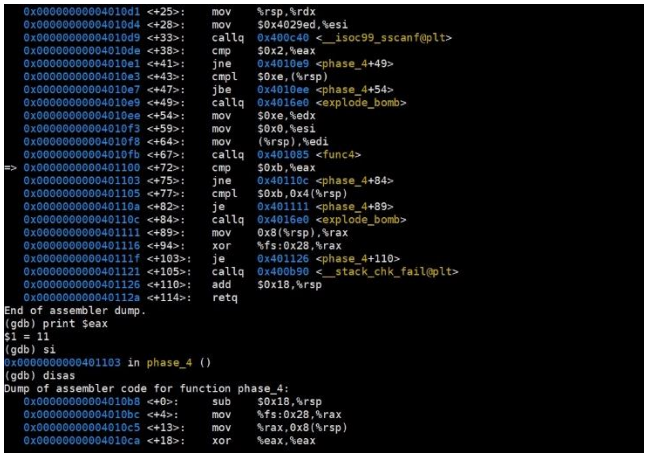
（图 31-阶段 4-3）



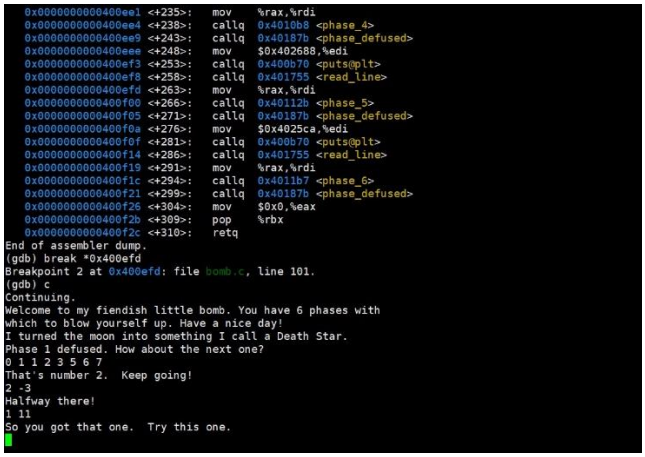
（图 32-阶段 4-4）

当程序执行到图 30 所示的<+67>处的时候，这里涉及到了一个未知的 fun4 函数，为了进一步了解程序实现的过程，同时也为了避免意外地调用 explode_bomb 函数，我们进入到这个函数内部去，如图 31 所示。

我们首先确定了输入的两个数字存储在了寄存器 edi 所指的内存区域（这时截图对应的输入仍然是尝试输入的 7 8 而非前面所说的 1 11）。从 fun4 函数的形式可以看出：这是一个递归调用的函数，因为其多次调用自己。从图 30 中<+43>处可知输入的第一个数字不能大于 15，因此使用暴力方法进行破解便可以解决这个问题。分别尝试输入 1 11、2 11、…、14 11，可以知道，当输入为 1 11 的时候可以破解成功（如下图 33 和图 34），截图如下：



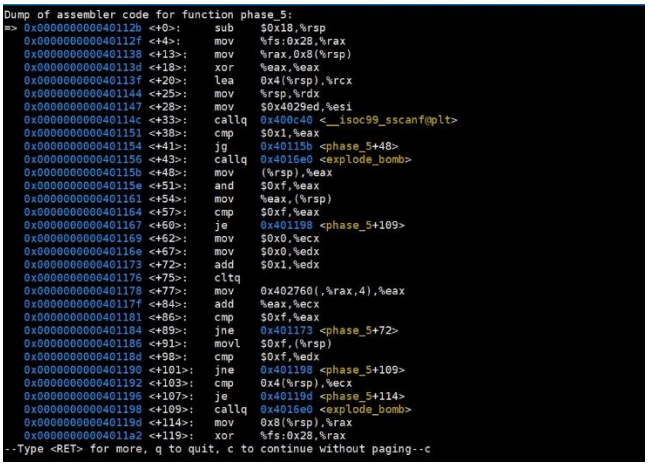
（图 33-阶段 4-5）



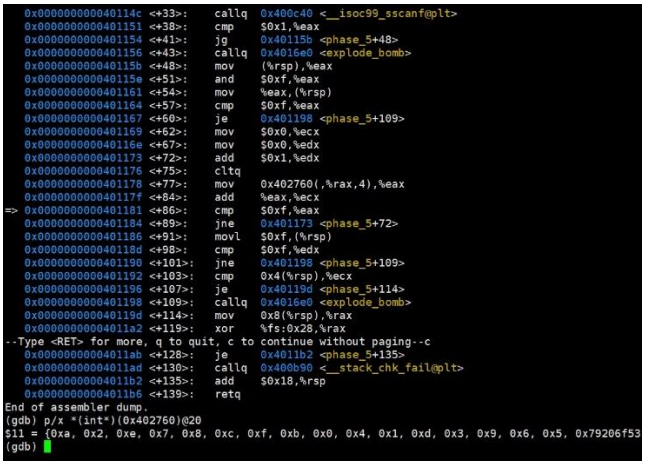
（图 34-阶段 4-6）

第五阶段

首先进入 phase_5 函数，下图 34 和 35 分别展示了 phase_5 函数的第一部分和第二部分：

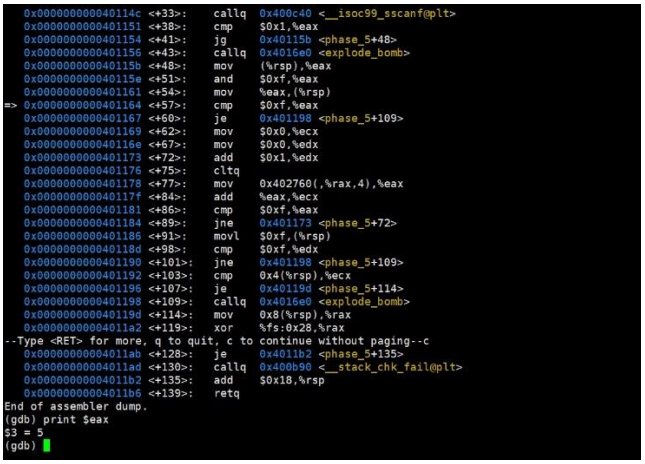


（图 35-阶段 5-1）

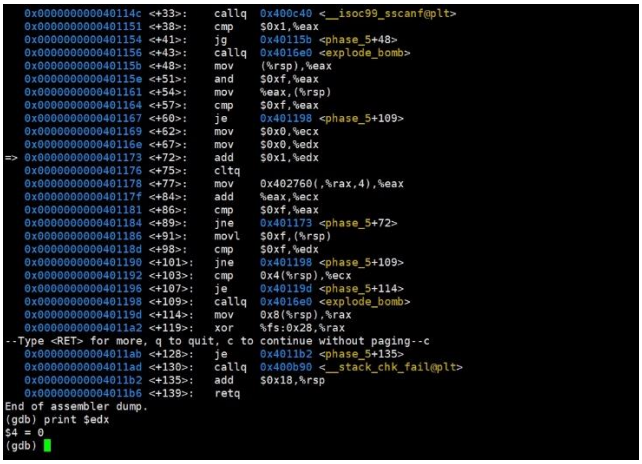


（图 36 阶段 5-2）

从图 35 中的<+38>处可以知道输入的数字数目至少是 2 个，否则会调用<+43>处的 explode_bomb 函数。其次，当程序运行到<+77>处的时候，发现这条赋值语句中含有常量地址，也就是说，这里的地址 0x402760 是程序本身的值，而且这条赋值语句在形式上很像从数组中取值（int 数长为 4，恰好对应其中的第三项），那么以数组形式打印出从地址 0x402760 开始的若干个数如上图 36 所示，发现这确实是一个数组，其中有 16 个元素，从 0 号元素到 15 号元素分别为 10，2，14，7，8，12，15，11，0，4，1，13，3，9，6，5。

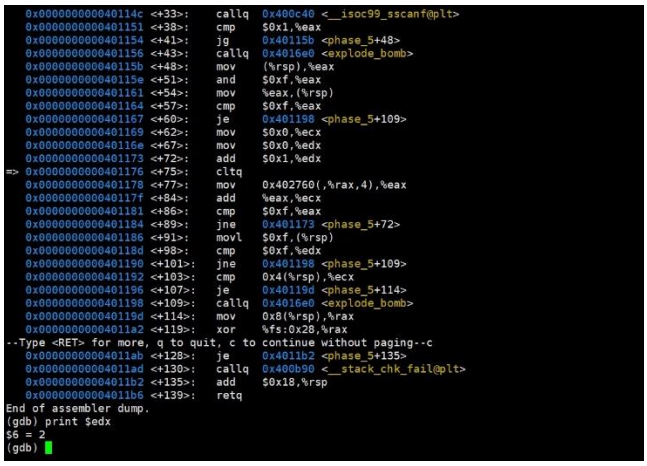


（图 37-阶段 5-3）

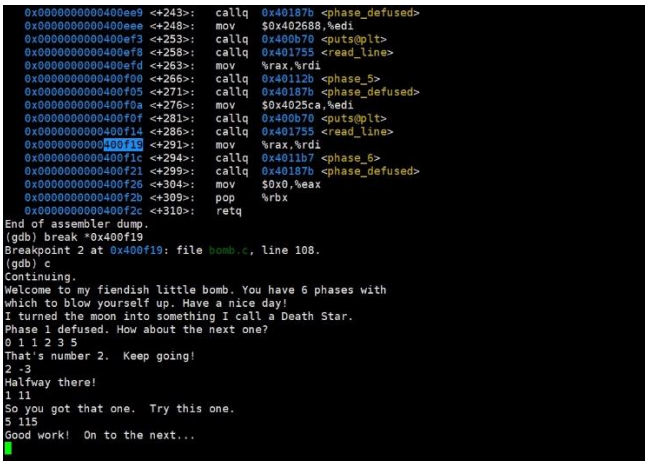


（图 38-阶段 5-4）

从图 37 中的<+57>处可以知道，输入的第一个数字不能超过 15，那么我们输入 5 作为第一个数字。进一步分析知道该程序实际上实现了数组链表的遍历：以当前访问元素的值作为下一次访问的元素的数组下标，并且从<+72>处可以知道寄存器 edx 起到了循环计数器的作用（如上图 38 所示第一次进入循环之前其值为 0；如下图 39 所示第二次循环结束时其值为 2）。并且，从<+84>处知道，每次访问的值都被加进了寄存器 ecx 中。又，从<+98>处可知正确的循环次数是 15 次，从<+107>处可知输入的第二个数字应该是该 15 次遍历中求和的结果，也即 115。因此，正确的输入为 5 115，解决成功的截图如下图 40 所示：



（图 39-阶段 5-5）



（图 40-阶段 5-6）

五、总结体会

本次实验耗费时间长，花费精力大，但是收获也非常多。回顾整个实验过程，我印象最深的不是实验过程中遇到的种种问题，而是在一开始由于基础知识不牢固导致三次 BOOM，后续实验中也因为一次操作失误导致 BOOM，这就说明其实错误往往在我意想不到的地方发生，在进入调试之后，explode_bomb 函数的调用是可以预见进而避免的，但是操作失误导致的 BOOM 却是很容易发生的。这给了我一个教训：一定要保证基础知识的牢固，这样才不会因为非实验因素导致实验出问题。

本次实验的第一阶段花费了我最多的时间，一开始我想要不断尝试输出不同寄存器的值，并且在程序执行的每一步都这么做，我进行了很久这种无意义的重复操作，最终我还是放弃了这种做法。我沉下心来分析机器指令对应的实现功能，并结合不断的猜测和验证，这样才解决了这一阶段。有了第一阶段的教训，在后面的阶段中我不再尝试“穷举法”，而是细心分析其实现的功能，完成了本次的实验。

在 gdb 调试指令方面，我得到了很多的练习，但是有些指令我并不熟悉，比如 p/x *(int*)指令，通过查阅相关资料，我解决了这个问题。