北京郵電大學

实验报告

题目：拆解二进制炸弹（高阶）

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 14 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

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

有了基础实验的经验和教训，我很快找到了正确的方法，首先进入 phase_1 函数，截图如下：

```
(gdb) disas
Dump of assembler code for function phase_1:
   0x0000000000401028 <+0>:     stp     x29, x30, [sp, #-16]!
   0x000000000040102c <+4>:     mov     x29, sp
   0x0000000000401030 <+8>:     adrp    x1, 0x402000 <submitr+1072>
   0x0000000000401034 <+12>:    add     x1, x1, #0x670
=> 0x0000000000401038 <+16>:    bl      0x401568 <strings_not_equal>
   0x000000000040103c <+20>:    cbnz    w0, 0x401048 <phase_1+32>
   0x0000000000401040 <+24>:    ldp     x29, x30, [sp], #16
   0x0000000000401044 <+28>:    ret
   0x0000000000401048 <+32>:    bl      0x40186c <explode_bomb>
   0x000000000040104c <+36>:    b       0x401040 <phase_1+24>
End of assembler dump.
(gdb) si
0x0000000000401568 in strings_not_equal ()
(gdb) disas
Dump of assembler code for function strings_not_equal:
=> 0x0000000000401568 <+0>:     stp     x29, x30, [sp, #-48]!
   0x000000000040156c <+4>:     mov     x29, sp
   0x0000000000401570 <+8>:     stp     x19, x20, [sp, #16]
   0x0000000000401574 <+12>:    str     x21, [sp, #32]
   0x0000000000401578 <+16>:    mov     x20, x0
   0x000000000040157c <+20>:    mov     x19, x1
   0x0000000000401580 <+24>:    bl      0x40153c <string_length>
   0x0000000000401584 <+28>:    mov     w21, w0
   0x0000000000401588 <+32>:    mov     x0, x19
   0x000000000040158c <+36>:    bl      0x40153c <string_length>
   0x0000000000401590 <+40>:    cmp     w21, w0
   0x0000000000401594 <+44>:    b.eq    0x4015ac <strings_not_equal+68>  // b.none
   0x0000000000401598 <+48>:    mov     w0, #0x1                         // #1
   0x000000000040159c <+52>:    ldp     x19, x20, [sp, #16]
   0x00000000004015a0 <+56>:    ldr     x21, [sp, #32]
   0x00000000004015a4 <+60>:    ldp     x29, x30, [sp], #48
   0x00000000004015a8 <+64>:    ret
```

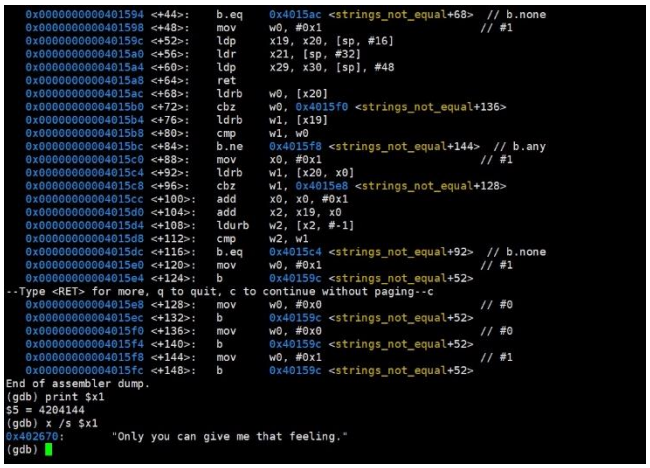
（图 1-阶段 1-1）

```
   0x0000000000401584 <+28>:    mov     w21, w0
   0x0000000000401588 <+32>:    mov     x0, x19
   0x000000000040158c <+36>:    bl      0x40153c <string_length>
   0x0000000000401590 <+40>:    cmp     w21, w0
   0x0000000000401594 <+44>:    b.eq    0x4015ac <strings_not_equal+68>  // b.none
   0x0000000000401598 <+48>:    mov     w0, #0x1                         // #1
   0x000000000040159c <+52>:    ldp     x19, x20, [sp, #16]
   0x00000000004015a0 <+56>:    ldr     x21, [sp, #32]
   0x00000000004015a4 <+60>:    ldp     x29, x30, [sp], #48
   0x00000000004015a8 <+64>:    ret
   0x00000000004015ac <+68>:    ldrb    w0, [x20]
   0x00000000004015b0 <+72>:    cbz     w0, 0x4015f0 <strings_not_equal+136>
   0x00000000004015b4 <+76>:    ldrb    w1, [x19]
   0x00000000004015b8 <+80>:    cmp     w1, w0
   0x00000000004015bc <+84>:    b.ne    0x4015f8 <strings_not_equal+144>  // b.any
   0x00000000004015c0 <+88>:    mov     x0, #0x1                         // #1
   0x00000000004015c4 <+92>:    ldrb    w1, [x20, x0]
   0x00000000004015c8 <+96>:    cbz     w1, 0x4015e8 <strings_not_equal+128>
   0x00000000004015cc <+100>:   add     x0, x0, #0x1
   0x00000000004015d0 <+104>:   add     x2, x19, x0
   0x00000000004015d4 <+108>:   ldurb   w2, [x2, #-1]
   0x00000000004015d8 <+112>:   cmp     w2, w1
   0x00000000004015dc <+116>:   b.eq    0x4015c4 <strings_not_equal+92>  // b.none
   0x00000000004015e0 <+120>:   mov     w0, #0x1                         // #1
   0x00000000004015e4 <+124>:   b       0x40159c <strings_not_equal+52>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x00000000004015e8 <+128>:   mov     w0, #0x0                         // #0
   0x00000000004015ec <+132>:   b       0x40159c <strings_not_equal+52>
   0x00000000004015f0 <+136>:   mov     w0, #0x0                         // #0
   0x00000000004015f4 <+140>:   b       0x40159c <strings_not_equal+52>
   0x00000000004015f8 <+144>:   mov     w0, #0x1                         // #1
   0x00000000004015fc <+148>:   b       0x40159c <strings_not_equal+52>
End of assembler dump.
(gdb)
```

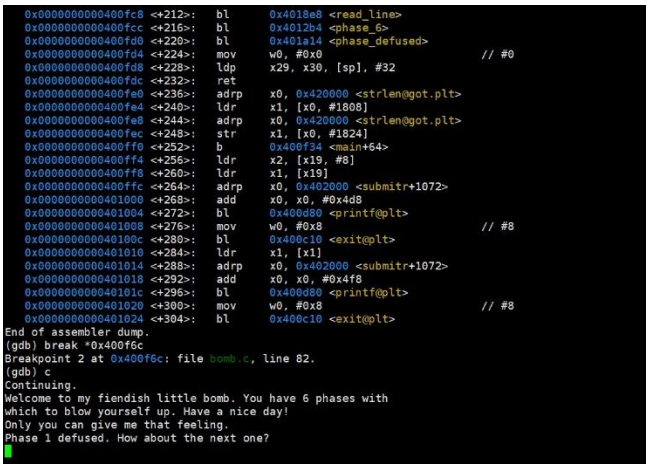
（图 2-阶段 1-2）

本阶段的关键过程就在于图 1 中 phase_1 函数内<+16>处的 strings_not_equal 函数，该函数的汇编代码如

图 1 和图 2 所示。在逐步执行的同时对其中涉及到的所有寄存器进行逐一检查，很快发现寄存器 x1 所指的地址处存放了它的密码（Only you can give me that feeling.），输出其值如下图 3 所示，成功解决该阶段的截图如下图 4 所示：



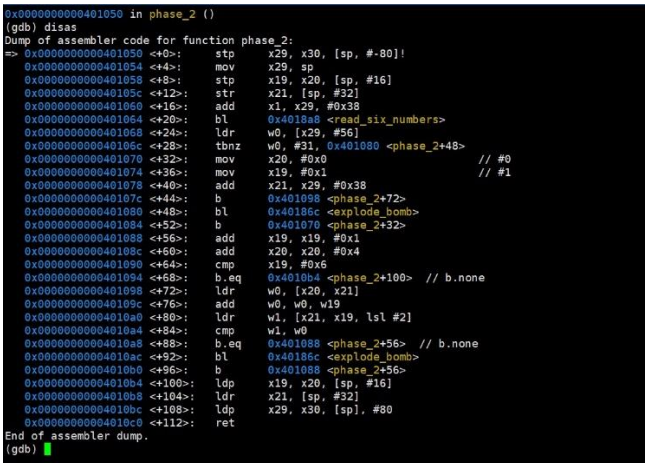
（图 3-阶段 1-3）



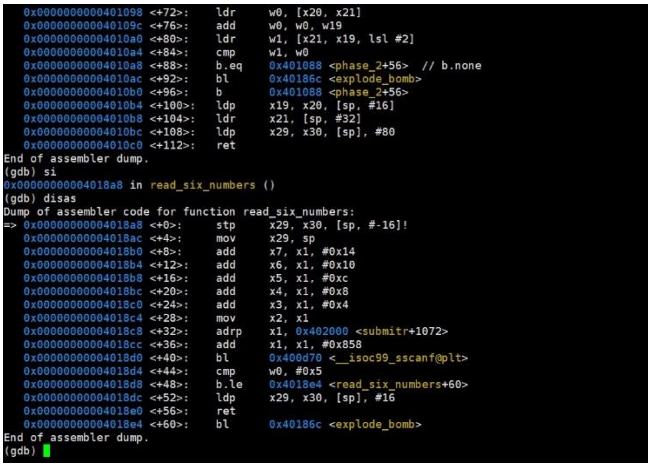
（图 4-阶段 1-4）

第二阶段

首先输入第一阶段的密码，进入到第二阶段的调试界面中，如下图所示：

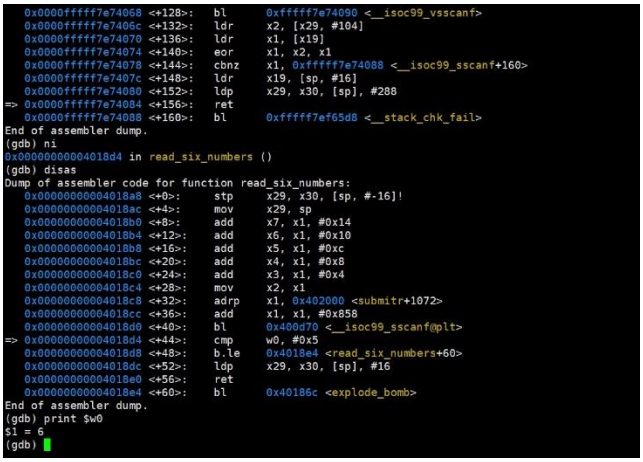


（图 5-阶段 2-1）

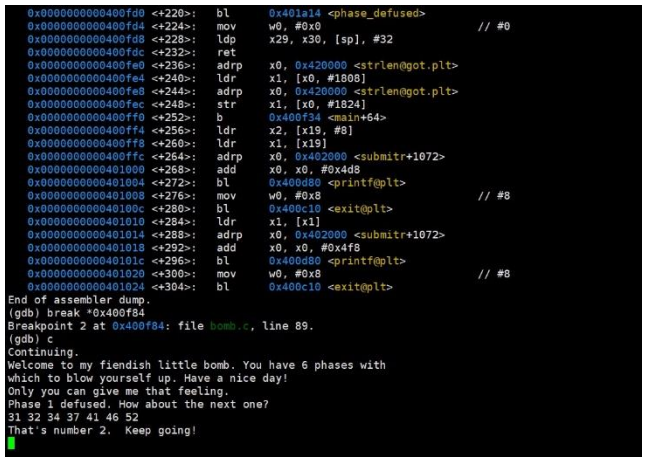


（图 6-阶段 2-2）

有了初级初级实验中那次 BOOM 的经历，我对 read 函数格外小心，上图 6 展示了 read 函数的内部结构，发现其中确实隐藏了一个 explode_bomb 函数，该函数要求输入的数字不得少于 5 个，否则便会调用 explode_bomb 函数导致实验失败。下面分析 phase_2 函数本身的功能：从<+88>处可以看出，该函数存在“回跳”的行为，因此可以猜测该函数实现的功能是涉及到循环的，并且由<+32>和<+36>、<+56>和<+60>处可以知道寄存器 x19 和 x20 中的变量是在每次循环中都被更新的（其中，<+32>和<+36>处是初始化变量，而<+56>和<+60>是在更新变量）。具体来说，寄存器 x20 中的值起到了指针偏置量更新的作用，其每次更新时候增加 4，恰好是一个 int 数的内存长度；而寄存器 x19 中的值则在计算中有着重要的作用。由<+28>处可以知道第一个输入的数字一定是 31，并且结合寄存器 x19 在程序执行过程中的行为可以知道后续输入的数字与第一个输入的数字呈现出“差”逐次增加 1 的特征，也就是说，正确的输入是 31，32，34，37，41，46，后面可以增加其他输入，但前 6 个输入一定要是这 6 个。下图 7 展示了 read 函数正确读入了 6 个数字的情形，下图 8 展示了该阶段被解决的情形：



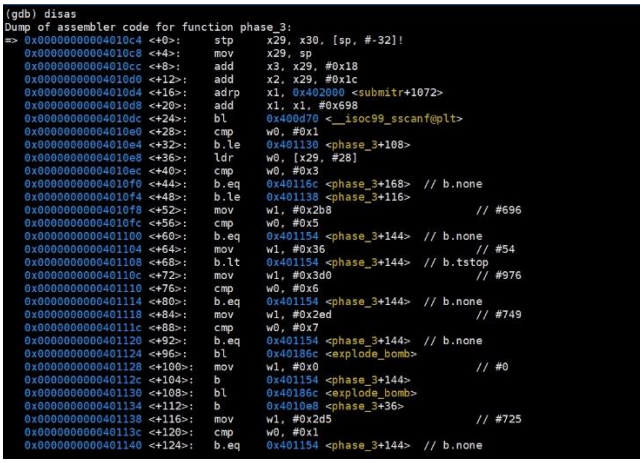
（图 7-阶段 2-3）



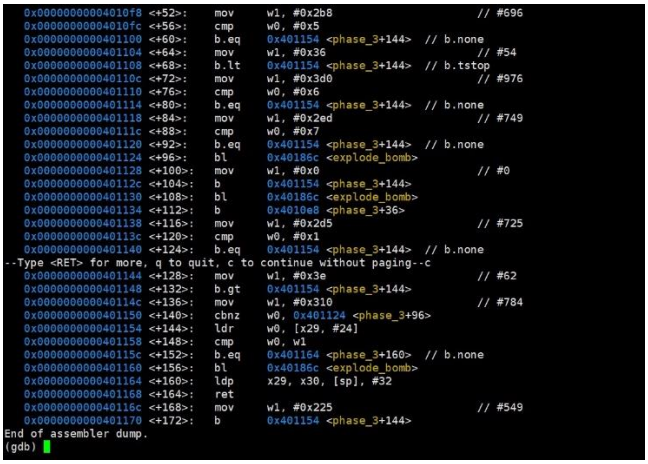
（图 8-阶段 2-4）

第三阶段

输入前面已经得到的答案，进入该阶段，下图 9 和 10 展示了 phase_3 函数的内容：

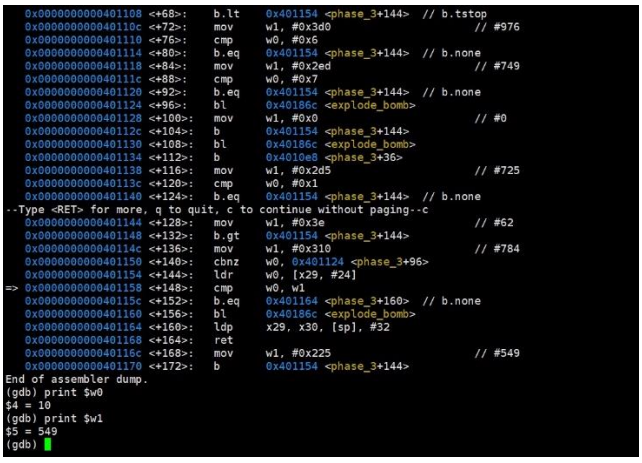


（图 9-阶段 3-1）

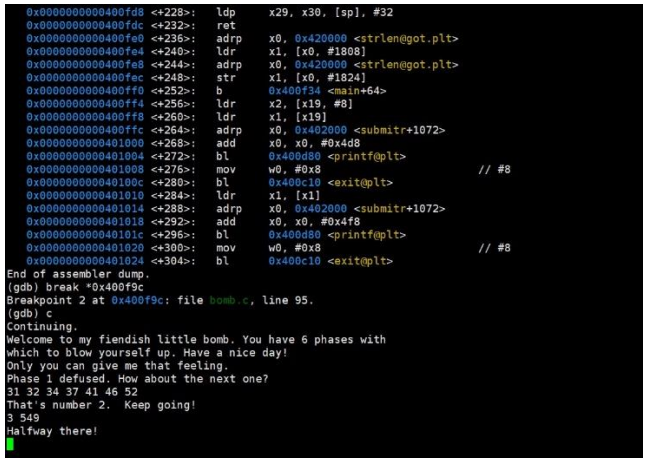


（图 10-阶段 3-2）

从 phase_3 函数的内容不难看出这是一个实现 switch 过程的代码段，假设输入为 3，那么从图 9 的<+40>和<+44>以及图 10 的<+168>处可以知道此时输入的第二个数字一定要是 549，这就是正确答案应该输入的两个数字。下图 11 中展示了寄存器 w0 中存放了我们输入的第二个数字，而寄存器 w1 则存放了待比较的（正确输入）549；下图 12 展示了解决该阶段的截图：



（图 11-阶段 3-3）



（图 12-阶段 3-4）

第四阶段

输入已经得到的答案，进入第四阶段的调试：

```
   0x0000000000401020 <+300>:   mov     w0, #0x8                        // #8
   0x0000000000401024 <+304>:   bl      0x400c10 <exit@plt>
End of assembler dump.
(gdb) si
0x00000000004011c8 in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
=> 0x00000000004011c8 <+0>:     stp     x29, x30, [sp, #-32]!
   0x00000000004011cc <+4>:     mov     x29, sp
   0x00000000004011d0 <+8>:     add     x3, x29, #0x18
   0x00000000004011d4 <+12>:    add     x2, x29, #0x1c
   0x00000000004011d8 <+16>:    adrp    x1, 0x402000 <submitr+1072>
   0x00000000004011dc <+20>:    add     x1, x1, #0x698
   0x00000000004011e0 <+24>:    bl      0x400d70 <__isoc99_sscanf@plt>
   0x00000000004011e4 <+28>:    cmp     w0, #0x2
   0x00000000004011e8 <+32>:    b.ne    0x4011f8 <phase_4+48>  // b.any
   0x00000000004011ec <+36>:    ldr     w0, [x29, #28]
   0x00000000004011f0 <+40>:    cmp     w0, #0xe
   0x00000000004011f4 <+44>:    b.ls    0x4011fc <phase_4+52>  // b.plast
   0x00000000004011f8 <+48>:    bl      0x40186c <explode_bomb>
   0x00000000004011fc <+52>:    mov     w2, #0xe                        // #14
   0x0000000000401200 <+56>:    mov     w1, #0x0                        // #0
   0x0000000000401204 <+60>:    ldr     w0, [x29, #28]
   0x0000000000401208 <+64>:    bl      0x401174 <func4>
   0x000000000040120c <+68>:    cmp     w0, #0x7
   0x0000000000401210 <+72>:    b.ne    0x401220 <phase_4+88>  // b.any
   0x0000000000401214 <+76>:    ldr     w0, [x29, #24]
   0x0000000000401218 <+80>:    cmp     w0, #0x7
   0x000000000040121c <+84>:    b.eq    0x401224 <phase_4+92>  // b.none
   0x0000000000401220 <+88>:    bl      0x40186c <explode_bomb>
   0x0000000000401224 <+92>:    ldp     x29, x30, [sp], #32
   0x0000000000401228 <+96>:    ret
End of assembler dump.
(gdb)
```

（图 13-阶段 4-1）

```
   0x0000000000401214 <+76>:    ldr     w0, [x29, #24]
   0x0000000000401218 <+80>:    cmp     w0, #0x7
   0x000000000040121c <+84>:    b.eq    0x401224 <phase_4+92>  // b.none
   0x0000000000401220 <+88>:    bl      0x40186c <explode_bomb>
   0x0000000000401224 <+92>:    ldp     x29, x30, [sp], #32
   0x0000000000401228 <+96>:    ret
End of assembler dump.
(gdb) si
0x0000000000401174 in func4 ()
(gdb) disas
Dump of assembler code for function func4:
=> 0x0000000000401174 <+0>:     stp     x29, x30, [sp, #-16]!
   0x0000000000401178 <+4>:     mov     x29, sp
   0x000000000040117c <+8>:     sub     w3, w2, w1
   0x0000000000401180 <+12>:    add     w3, w3, w3, lsr #31
   0x0000000000401184 <+16>:    add     w3, w1, w3, asr #1
   0x0000000000401188 <+20>:    cmp     w3, w0
   0x000000000040118c <+24>:    b.gt    0x4011a4 <func4+48>
   0x0000000000401190 <+28>:    mov     w1, #0x0                        // #0
   0x0000000000401194 <+32>:    b.lt    0x4011b4 <func4+64>  // b.tstop
   0x0000000000401198 <+36>:    mov     w0, w1
   0x000000000040119c <+40>:    ldp     x29, x30, [sp], #16
   0x00000000004011a0 <+44>:    ret
   0x00000000004011a4 <+48>:    sub     w2, w3, #0x1
   0x00000000004011a8 <+52>:    bl      0x401174 <func4>
   0x00000000004011ac <+56>:    lsl     w1, w0, #1
   0x00000000004011b0 <+60>:    b       0x401198 <func4+36>
   0x00000000004011b4 <+64>:    add     w1, w3, #0x1
   0x00000000004011b8 <+68>:    bl      0x401174 <func4>
   0x00000000004011bc <+72>:    lsl     w0, w0, #1
   0x00000000004011c0 <+76>:    add     w1, w0, #0x1
   0x00000000004011c4 <+80>:    b       0x401198 <func4+36>
End of assembler dump.
(gdb)
```

（图 14-阶段 4-2）

从图 13 不难看出，本题的关键是<+64>处的 func4 函数，图 14 进一步展示了 func4 的内部结构，不难发现这是一个递归函数。由于图 13<+40>处可以知道第一个输入的数字不得超过 14，并根据<+80>处知道第二个输入必须是 7，而且<+68>处告诉我们该递归函数最后的返回值应该是 7。由于第一个输入的数字有限，所以考虑穷举法进行破解，分别尝试输入 0 7、1 7、…、14 7，最后发现输入 14 7 是正确的。下图 15 展示了第一个输入为 14 时的递归函数返回值是 7；下图 16 展示了该阶段破解成功的截图：

```
End of assembler dump.
(gdb) ni
0x000000000040120c in phase_4 ()
(gdb) disas
Dump of assembler code for function phase_4:
   0x00000000004011c8 <+0>:     stp     x29, x30, [sp, #-32]!
   0x00000000004011cc <+4>:     mov     x29, sp
   0x00000000004011d0 <+8>:     add     x3, x29, #0x18
   0x00000000004011d4 <+12>:    add     x2, x29, #0x1c
   0x00000000004011d8 <+16>:    adrp    x1, 0x402000 <submitr+1072>
   0x00000000004011dc <+20>:    add     x1, x1, #0x698
   0x00000000004011e0 <+24>:    bl      0x400d70 <__isoc99_sscanf@plt>
   0x00000000004011e4 <+28>:    cmp     w0, #0x2
   0x00000000004011e8 <+32>:    b.ne    0x4011f8 <phase_4+48>  // b.any
   0x00000000004011ec <+36>:    ldr     w0, [x29, #28]
   0x00000000004011f0 <+40>:    cmp     w0, #0xe
   0x00000000004011f4 <+44>:    b.ls    0x4011fc <phase_4+52>  // b.plast
   0x00000000004011f8 <+48>:    bl      0x40186c <explode_bomb>
   0x00000000004011fc <+52>:    mov     w2, #0xe                        // #14
   0x0000000000401200 <+56>:    mov     w1, #0x0                        // #0
   0x0000000000401204 <+60>:    ldr     w0, [x29, #28]
   0x0000000000401208 <+64>:    bl      0x401174 <func4>
=> 0x000000000040120c <+68>:    cmp     w0, #0x7
   0x0000000000401210 <+72>:    b.ne    0x401220 <phase_4+88>  // b.any
   0x0000000000401214 <+76>:    ldr     w0, [x29, #24]
   0x0000000000401218 <+80>:    cmp     w0, #0x7
   0x000000000040121c <+84>:    b.eq    0x401224 <phase_4+92>  // b.none
   0x0000000000401220 <+88>:    bl      0x40186c <explode_bomb>
   0x0000000000401224 <+92>:    ldp     x29, x30, [sp], #32
   0x0000000000401228 <+96>:    ret
End of assembler dump.
(gdb) print $w0
$1 = 7
(gdb)
```

（图 15-阶段 4-3）

```
   0x0000000000400fe0 <+236>:   adrp    x0, 0x420000 <strlen@got.plt>
   0x0000000000400fe4 <+240>:   ldr     x1, [x0, #1808]
   0x0000000000400fe8 <+244>:   adrp    x0, 0x420000 <strlen@got.plt>
   0x0000000000400fec <+248>:   str     x1, [x0, #1824]
   0x0000000000400ff0 <+252>:   b       0x400f34 <main+64>
   0x0000000000400ff4 <+256>:   ldr     x2, [x19, #8]
   0x0000000000400ff8 <+260>:   ldr     x1, [x19]
   0x0000000000400ffc <+264>:   adrp    x0, 0x402000 <submitr+1072>
   0x0000000000401000 <+268>:   add     x0, x0, #0x4d8
   0x0000000000401004 <+272>:   bl      0x400d80 <printf@plt>
   0x0000000000401008 <+276>:   mov     w0, #0x8                        // #8
   0x000000000040100c <+280>:   bl      0x400c10 <exit@plt>
   0x0000000000401010 <+284>:   ldr     x1, [x1]
   0x0000000000401014 <+288>:   adrp    x0, 0x402000 <submitr+1072>
   0x0000000000401018 <+292>:   add     x0, x0, #0x4f8
   0x000000000040101c <+296>:   bl      0x400d80 <printf@plt>
   0x0000000000401020 <+300>:   mov     w0, #0x8                        // #8
   0x0000000000401024 <+304>:   bl      0x400c10 <exit@plt>
End of assembler dump.
(gdb) break *0x400fb4
Breakpoint 2 at 0x400fb4: file bomb.c, line 101.
(gdb) c
Continuing.
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Only you can give me that feeling.
Phase 1 defused. How about the next one?
31 32 34 37 41 46 52
That's number 2.  Keep going!
3 549
Halfway there!
14 7
So you got that one.  Try this one.
```

（图 16-阶段 4-4）

第五阶段

首先输入之前得到的答案，进入到第五阶段的调试中（图 17 和 18 展示了 phase_5 的代码段）：

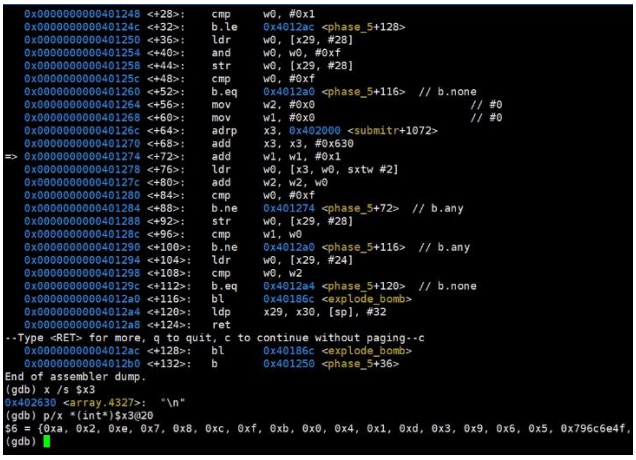
```
0x000000000040122c in phase_5 ()
(gdb) disas
Dump of assembler code for function phase_5:
=> 0x000000000040122c <+0>:     stp     x29, x30, [sp, #-32]!
   0x0000000000401230 <+4>:     mov     x29, sp
   0x0000000000401234 <+8>:     add     x3, x29, #0x18
   0x0000000000401238 <+12>:    add     x2, x29, #0x1c
   0x000000000040123c <+16>:    adrp    x1, 0x402000 <submitr+1072>
   0x0000000000401240 <+20>:    add     x1, x1, #0x698
   0x0000000000401244 <+24>:    bl      0x400d70 <__isoc99_sscanf@plt>
   0x0000000000401248 <+28>:    cmp     w0, #0x1
   0x000000000040124c <+32>:    b.le    0x4012ac <phase_5+128>
   0x0000000000401250 <+36>:    ldr     w0, [x29, #28]
   0x0000000000401254 <+40>:    and     w0, w0, #0xf
   0x0000000000401258 <+44>:    str     w0, [x29, #28]
   0x000000000040125c <+48>:    cmp     w0, #0xf
   0x0000000000401260 <+52>:    b.eq    0x4012a0 <phase_5+116>  // b.none
   0x0000000000401264 <+56>:    mov     w2, #0x0                        // #0
   0x0000000000401268 <+60>:    mov     w1, #0x0                        // #0
   0x000000000040126c <+64>:    adrp    x3, 0x402000 <submitr+1072>
   0x0000000000401270 <+68>:    add     x3, x3, #0x630
   0x0000000000401274 <+72>:    add     w1, w1, #0x1
   0x0000000000401278 <+76>:    ldr     w0, [x3, w0, sxtw #2]
   0x000000000040127c <+80>:    add     w2, w2, w0
   0x0000000000401280 <+84>:    cmp     w0, #0xf
   0x0000000000401284 <+88>:    b.ne    0x401274 <phase_5+72>  // b.any
   0x0000000000401288 <+92>:    str     w0, [x29, #28]
   0x000000000040128c <+96>:    cmp     w1, w0
   0x0000000000401290 <+100>:   b.ne    0x4012a0 <phase_5+116>  // b.any
   0x0000000000401294 <+104>:   ldr     w0, [x29, #24]
   0x0000000000401298 <+108>:   cmp     w0, w2
   0x000000000040129c <+112>:   b.eq    0x4012a4 <phase_5+120>  // b.none
   0x00000000004012a0 <+116>:   bl      0x40186c <explode_bomb>
   0x00000000004012a4 <+120>:   ldp     x29, x30, [sp], #32
```

（图 17-阶段 5-1）

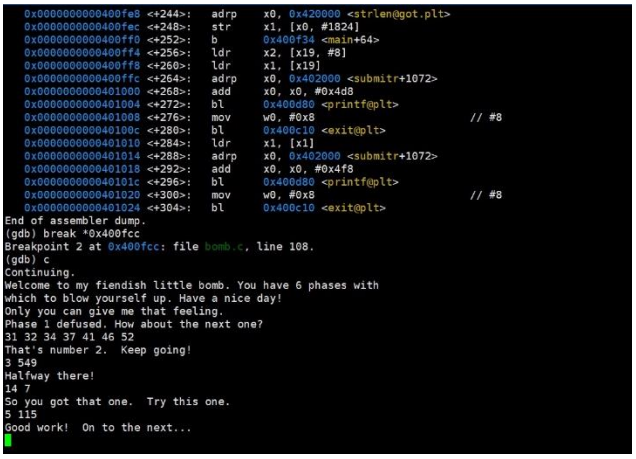
```
   0x0000000000401238 <+12>:    add     x2, x29, #0x1c
   0x000000000040123c <+16>:    adrp    x1, 0x402000 <submitr+1072>
   0x0000000000401240 <+20>:    add     x1, x1, #0x698
   0x0000000000401244 <+24>:    bl      0x400d70 <__isoc99_sscanf@plt>
   0x0000000000401248 <+28>:    cmp     w0, #0x1
   0x000000000040124c <+32>:    b.le    0x4012ac <phase_5+128>
   0x0000000000401250 <+36>:    ldr     w0, [x29, #28]
   0x0000000000401254 <+40>:    and     w0, w0, #0xf
   0x0000000000401258 <+44>:    str     w0, [x29, #28]
   0x000000000040125c <+48>:    cmp     w0, #0xf
   0x0000000000401260 <+52>:    b.eq    0x4012a0 <phase_5+116>  // b.none
   0x0000000000401264 <+56>:    mov     w2, #0x0                        // #0
   0x0000000000401268 <+60>:    mov     w1, #0x0                        // #0
   0x000000000040126c <+64>:    adrp    x3, 0x402000 <submitr+1072>
   0x0000000000401270 <+68>:    add     x3, x3, #0x630
   0x0000000000401274 <+72>:    add     w1, w1, #0x1
   0x0000000000401278 <+76>:    ldr     w0, [x3, w0, sxtw #2]
   0x000000000040127c <+80>:    add     w2, w2, w0
   0x0000000000401280 <+84>:    cmp     w0, #0xf
   0x0000000000401284 <+88>:    b.ne    0x401274 <phase_5+72>  // b.any
   0x0000000000401288 <+92>:    str     w0, [x29, #28]
   0x000000000040128c <+96>:    cmp     w1, w0
   0x0000000000401290 <+100>:   b.ne    0x4012a0 <phase_5+116>  // b.any
   0x0000000000401294 <+104>:   ldr     w0, [x29, #24]
   0x0000000000401298 <+108>:   cmp     w0, w2
   0x000000000040129c <+112>:   b.eq    0x4012a4 <phase_5+120>  // b.none
   0x00000000004012a0 <+116>:   bl      0x40186c <explode_bomb>
   0x00000000004012a4 <+120>:   ldp     x29, x30, [sp], #32
   0x00000000004012a8 <+124>:   ret
--Type <RET> for more, q to quit, c to continue without paging--c
   0x00000000004012ac <+128>:   bl      0x40186c <explode_bomb>
   0x00000000004012b0 <+132>:   b       0x401250 <phase_5+36>
End of assembler dump.
(gdb)
```

（图 18-阶段 5-2）

首先对输入进行分析，从图 17 中<+28>处可知输入的数字至少有 2 个，并且根据<+48>处可知输入的第一个数字不能是 15。当程序运行到<+64>处的时候，发现这条赋值语句中含有常量地址（如上图 18 所示），也就是说，这里的地址 0x402000 是程序本身的值，该值经过操作后赋给了寄存器 x3，而且<+76>处的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“#2”），那么以数组形式打印出寄存器 x3 所指向的若干个数如下图 19 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 19-阶段 5-3）



（图 20-阶段 5-4）

该片段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 w1 起到了循环计数器的作用（<+60>处所示第一次进入循环之前其值为初始值 0）。并且，从<+76>处知道每次访问的值存放在了寄存器 w0 中，从<+80>处知道每次访问的值都被加进了寄存器 w2 中。又，从<+84>处可知最后一次循环取得的值应该是 15，并且<+96>处告诉我们正确的循环次数是 15 次，从<+108>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，解决成功的截图如上图 20 所示。

五、总结体会

总的来说，这次实验完成较好，有了初级实验的经验和教训，这次实验不论是完成时间还是正确率都达到了较好的水平，共用时 3.5 小时，并且没有触发任何一次 BOOM，总体完成较好。从实验报告本身也可以看出这一点：我的初级实验的实验报告总计 4086 字、附图 40 张；而本次高级实验实验报告总计 2786 字、附图 20 张。

本次实验中我一开始遇到的困难是对 ARM 的指令集不熟悉，以至于在第一阶段的破解花费了较长的时间，认真学习了老师发的相关资料之后，我才对 ARM 的指令集渐渐熟悉起来，第一阶段完成之后，后面四个阶段的实验的完成速度就有了大幅度的提高，并且由于高级实验和初级实验中涉及到的功能类似，我也成功规避了我在初级实验中所犯的一些错误，提高了正确率。

纵观两个实验，我的完成情况在同学中并不算优秀，但是我相信在这两次实验中我所学到的东西、我所得到的锻炼是达到了我的预期，我在这两个实验中是真正有收获的，这才是初级实验和高级实验两个实验真正的意义所在。希望后续的计算机系统基础的学习能够更进一步！