北京郵電大學

实验报告

题目：拆解二进制炸弹

班　　级：2020211310

学　　号：2020211502

姓　　名：王小龙

学　　院：计算机学院

2021 年 11 月 17 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

准备工作

通过 ls 指令显示当前目录下的文件，找到压缩包 bomb115.tar，即目标炸弹。通过 tar xvf bomb115.tar 解压此压缩包，然后再通过 ls 显示解压出的文件：bomb、bomb.c 和 README。

阶段一

操作步骤：
1.gdb bomb;
2.设置炸弹断点 break explode_bomb;
3.设置断点 break phase_1;
4.输入地址：x/s 0x4026a0,查看值；
5.r 运行;
6.输入 Forget the memories, continue to be life, miss, just pass by.即可

分析过程：

通过指令 disas phase_1 查看第一关的汇编代码，发现需要输入与地址 0x4026a0 上一样的字符串

```
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400f2d <+0>:     sub    $0x8,%rsp
   0x0000000000400f31 <+4>:     mov    $0x4026a0,%esi
   0x0000000000400f36 <+9>:     callq  0x4013f8 <strings_not_equal>
   0x0000000000400f3b <+14>:    test   %eax,%eax
   0x0000000000400f3d <+16>:    je     0x400f44 <phase_1+23>
   0x0000000000400f3f <+18>:    callq  0x4016cc <explode_bomb>
   0x0000000000400f44 <+23>:    add    $0x8,%rsp
   0x0000000000400f48 <+27>:    retq
End of assembler dump.
```

运行截图：

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Forget the memories, continue to be life, miss, just pass by.
Phase 1 defused. How about the next one?
```

阶段二

操作步骤

1. gdb bomb;
2. 设置炸弹断点 break explode_bomb;
3. 设置断点 break phase_2;
4. r 运行
5. 输入：1 2 4 8 16 32.即可

分析过程：

1.查看关卡的反汇编代码

```
(gdb) disas phase_2
Dump of assembler code for function phase_2:
   0x0000000000400f49 <+0>:     push   %rbp
   0x0000000000400f4a <+1>:     push   %rbx
   0x0000000000400f4b <+2>:     sub    $0x28,%rsp
   0x0000000000400f4f <+6>:     mov    %fs:0x28,%rax
   0x0000000000400f58 <+15>:    mov    %rax,0x18(%rsp)
   0x0000000000400f5d <+20>:    xor    %eax,%eax
   0x0000000000400f5f <+22>:    mov    %rsp,%rsi
   0x0000000000400f62 <+25>:    callq  0x401702 <read_six_numbers>
   0x0000000000400f67 <+30>:    cmpl   $0x1,(%rsp)
   0x0000000000400f6b <+34>:    je     0x400f72 <phase_2+41>
   0x0000000000400f6d <+36>:    callq  0x4016cc <explode_bomb>
   0x0000000000400f72 <+41>:    mov    %rsp,%rbx
   0x0000000000400f75 <+44>:    lea    0x14(%rsp),%rbp
   0x0000000000400f7a <+49>:    mov    (%rbx),%eax
   0x0000000000400f7c <+51>:    add    %eax,%eax
   0x0000000000400f7e <+53>:    cmp    %eax,0x4(%rbx)
   0x0000000000400f81 <+56>:    je     0x400f88 <phase_2+63>
   0x0000000000400f83 <+58>:    callq  0x4016cc <explode_bomb>
   0x0000000000400f88 <+63>:    add    $0x4,%rbx
   0x0000000000400f8c <+67>:    cmp    %rbp,%rbx
   0x0000000000400f8f <+70>:    jne    0x400f7a <phase_2+49>
   0x0000000000400f91 <+72>:    mov    0x18(%rsp),%rax
   0x0000000000400f96 <+77>:    xor    %fs:0x28,%rax
   0x0000000000400f9f <+86>:    je     0x400fa6 <phase_2+93>
   0x0000000000400fa1 <+88>:    callq  0x400b90 <__stack_chk_fail@plt
   0x0000000000400fa6 <+93>:    add    $0x28,%rsp
   0x0000000000400faa <+97>:    pop    %rbx
   0x0000000000400fab <+98>:    pop    %rbp
   0x0000000000400fac <+99>:    retq
End of assembler dump.
(gdb) r
```

2.

```
0x0000000000400f62 <+25>:    callq  0x401702 <read_six_numbers>
```

输入为六个数字;

3.

第一个数和 1 比较；

4.

eax = eax *2；

5.

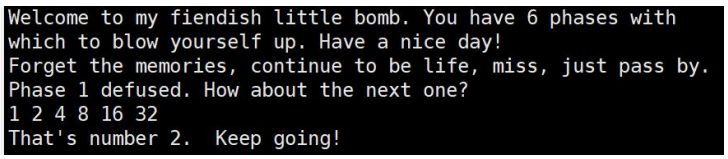
比较此数两倍和后一个数；

6.

循环；

由上述语句可知这是一个首项为 1，公比为 2 的等比数列，故答案为“1 2 4 8 16 32”。

运行截图：



阶段三

操作步骤

1.gdb bomb;

2.设置炸弹断点 break explode_bomb;

3.设置断点 break phase_3;

4.r 运行;

5.输入 2 -246.即可

分析过程：

1. 查看关卡的反汇编代码

```
(gdb) disas
Dump of assembler code for function phase_3:
=> 0x0000000000400fad <+0>:     sub    $0x18,%rsp
   0x0000000000400fb1 <+4>:     mov    %fs:0x28,%rax
   0x0000000000400fba <+13>:    mov    %rax,0x8(%rsp)
   0x0000000000400fbf <+18>:    xor    %eax,%eax
   0x0000000000400fc1 <+20>:    lea    0x4(%rsp),%rcx
   0x0000000000400fc6 <+25>:    mov    %rsp,%rdx
   0x0000000000400fc9 <+28>:    mov    $0x4029ad,%esi
   0x0000000000400fce <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x0000000000400fd3 <+38>:    cmp    $0x1,%eax
   0x0000000000400fd6 <+41>:    jg     0x400fdd <phase_3+48>
   0x0000000000400fd8 <+43>:    callq  0x4016cc <explode_bomb>
   0x0000000000400fdd <+48>:    cmpl   $0x7,(%rsp)
   0x0000000000400fe1 <+52>:    ja     0x401048 <phase_3+155>
   0x0000000000400fe3 <+54>:    mov    (%rsp),%eax
   0x0000000000400fe6 <+57>:    jmpq   *0x402710(,%rax,8)
   0x0000000000400fed <+64>:    mov    $0x1b2,%eax
   0x0000000000400ff2 <+69>:    jmp    0x400ff9 <phase_3+76>
   0x0000000000400ff4 <+71>:    mov    $0x0,%eax
   0x0000000000400ff9 <+76>:    sub    $0xd4,%eax
   0x0000000000400ffe <+81>:    jmp    0x401005 <phase_3+88>
   0x0000000000401000 <+83>:    mov    $0x0,%eax
   0x0000000000401005 <+88>:    add    $0x220,%eax
   0x000000000040100a <+93>:    jmp    0x401011 <phase_3+100>
   0x000000000040100c <+95>:    mov    $0x0,%eax
   0x0000000000401011 <+100>:   sub    $0x316,%eax
   0x0000000000401016 <+105>:   jmp    0x40101d <phase_3+112>
   0x0000000000401018 <+107>:   mov    $0x0,%eax
   0x000000000040101d <+112>:   add    $0x316,%eax
   0x0000000000401022 <+117>:   jmp    0x401029 <phase_3+124>
   0x0000000000401024 <+119>:   mov    $0x0,%eax
   0x0000000000401029 <+124>:   sub    $0x316,%eax
   0x000000000040102e <+129>:   jmp    0x401035 <phase_3+136>
   0x0000000000401030 <+131>:   mov    $0x0,%eax
   0x0000000000401035 <+136>:   add    $0x316,%eax
   0x000000000040103a <+141>:   jmp    0x401041 <phase_3+148>
   0x000000000040103c <+143>:   mov    $0x0,%eax
   0x0000000000401041 <+148>:   sub    $0x316,%eax
   0x0000000000401046 <+153>:   jmp    0x401052 <phase_3+165>
```

```
   0x0000000000401048 <+155>:   callq  0x4016cc <explode_bomb>
--Type <RET> for more, q to quit, c to continue without paging--c
   0x000000000040104d <+160>:   mov    $0x0,%eax
   0x0000000000401052 <+165>:   cmpl   $0x5,(%rsp)
   0x0000000000401056 <+169>:   jg     0x40105e <phase_3+177>
   0x0000000000401058 <+171>:   cmp    0x4(%rsp),%eax
   0x000000000040105c <+175>:   je     0x401063 <phase_3+182>
   0x000000000040105e <+177>:   callq  0x4016cc <explode_bomb>
   0x0000000000401063 <+182>:   mov    0x8(%rsp),%rax
   0x0000000000401068 <+187>:   xor    %fs:0x28,%rax
   0x0000000000401071 <+196>:   je     0x401078 <phase_3+203>
   0x0000000000401073 <+198>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000401078 <+203>:   add    $0x18,%rsp
   0x000000000040107c <+207>:   retq
End of assembler dump.
```

2. 查看输入格式，输入为两个整数

```
(gdb) x/s 0x4029ad
0x4029ad:       "%d %d"
```

3. 确定第一个数的取值范围为大于 1 小于 7 的整数

4.

```
0x0000000000400fe6 <+57>:    jmpq   *0x402710(,%rax,8)
```

switch 语句，尝试第一个数为“2”，经计算得“0x220-0x316=246”；

5.

```
0x0000000000401058 <+171>:   cmp    0x4(%rsp),%eax
```

比较 case 运算后的数 eax 与第二个数是否相等，故其中一组可行的答案为“2 -246”。

运行截图：

```
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Forget the memories, continue to be life, miss, just pass by.
Phase 1 defused. How about the next one?
1 2 4 8 16 32
That's number 2.  Keep going!
2 -246

Breakpoint 2, 0x0000000000400fad in phase_3 ()
(gdb) c
Continuing.
Halfway there!
```

阶段四

操作步骤

1.gdb bomb;

2.设置炸弹断点 break explode_bomb;

3.设置断点 break phase_4;

4.r 运行;

5.输入“352 4”.即可

分析过程

1.查看关卡的反汇编代码

```
(gdb) disas phase_4
Dump of assembler code for function phase_4:
   0x00000000004010b8 <+0>:     sub    $0x18,%rsp
   0x00000000004010bc <+4>:     mov    %fs:0x28,%rax
   0x00000000004010c5 <+13>:    mov    %rax,0x8(%rsp)
   0x00000000004010ca <+18>:    xor    %eax,%eax
   0x00000000004010cc <+20>:    mov    %rsp,%rcx
   0x00000000004010cf <+23>:    lea    0x4(%rsp),%rdx
   0x00000000004010d4 <+28>:    mov    $0x4029ad,%esi
   0x00000000004010d9 <+33>:    callq  0x400c40 <__isoc99_sscanf@plt>
   0x00000000004010de <+38>:    cmp    $0x2,%eax
   0x00000000004010e1 <+41>:    jne    0x4010ee <phase_4+54>
   0x00000000004010e3 <+43>:    mov    (%rsp),%eax
   0x00000000004010e6 <+46>:    sub    $0x2,%eax
   0x00000000004010e9 <+49>:    cmp    $0x2,%eax
   0x00000000004010ec <+52>:    jbe    0x4010f3 <phase_4+59>
   0x00000000004010ee <+54>:    callq  0x4016cc <explode_bomb>
   0x00000000004010f3 <+59>:    mov    (%rsp),%esi
   0x00000000004010f6 <+62>:    mov    $0x9,%edi
   0x00000000004010fb <+67>:    callq  0x40107d <func4>
   0x0000000000401100 <+72>:    cmp    0x4(%rsp),%eax
   0x0000000000401104 <+76>:    je     0x40110b <phase_4+83>
   0x0000000000401106 <+78>:    callq  0x4016cc <explode_bomb>
   0x000000000040110b <+83>:    mov    0x8(%rsp),%rax
   0x0000000000401110 <+88>:    xor    %fs:0x28,%rax
   0x0000000000401119 <+97>:    je     0x401120 <phase_4+104>
   0x000000000040111b <+99>:    callq  0x400b90 <__stack_chk_fail@plt>
   0x0000000000401120 <+104>:   add    $0x18,%rsp
   0x0000000000401124 <+108>:   retq
End of assembler dump.
```

2.查看输入格式，输入为两个整数;

```
(gdb) x/s 0x4029ad
0x4029ad:       "%d %d"
```

3.传入参数，然后调用递归函数 func4;

4.递归函数的返回值必须等于 9，否则爆炸;

5.则根据已经给的定值 9,我们可以确定递归调用的第二个参数值;

6.再根据反汇编写出源程序即可

递归函数 c 语言实现如下：

```c
#include<stdio.h>

int Function(int a1,int a2){
    if(a1 == 0)

    return 0;

    if(a1 == 1)

    return a2;
    return a2 + Function(a1 - 1,a2) + Function(a1 - 2,a2);
}
int main(){
    int a1 = 9,a2 = 4;
    int sum = Function(a1,a2);
    printf("sum = %d" ,sum);
    return 0;
}
```

运行截图：

```
352 4

Breakpoint 3, 0x00000000004010b8 in phase_4 ()
(gdb) c
Continuing.
So you got that one.  Try this one.
```

阶段五

操作步骤

1. gdb bomb;
2. 设置炸弹断点 break explode_bomb;
3. 设置断点 break phase_5;
4. 查看两个地址中的字符串内容

```
(gdb) x/s 0x402750
0x402750 <array.3599>:  "maduiersnfotvbylSo you think you can stop the
 bomb with ctrl-c, do you?"
```

```
(gdb) x/s 0x402706
0x402706:       "flyers"
```

5. r 运行;
6. 输入 IONEFG.即可

过程分析

1. 查看关卡的反汇编代码

```
(gdb) disas phase_5
Dump of assembler code for function phase_5:
   0x0000000000401125 <+0>:     push   %rbx
   0x0000000000401126 <+1>:     sub    $0x10,%rsp
   0x000000000040112a <+5>:     mov    %rdi,%rbx
   0x000000000040112d <+8>:     mov    %fs:0x28,%rax
   0x0000000000401136 <+17>:    mov    %rax,0x8(%rsp)
   0x000000000040113b <+22>:    xor    %eax,%eax
   0x000000000040113d <+24>:    callq  0x4013da <string_length>
   0x0000000000401142 <+29>:    cmp    $0x6,%eax
   0x0000000000401145 <+32>:    je     0x40114c <phase_5+39>
   0x0000000000401147 <+34>:    callq  0x4016cc <explode_bomb>
   0x000000000040114c <+39>:    mov    $0x0,%eax
   0x0000000000401151 <+44>:    movzbl (%rbx,%rax,1),%edx
   0x0000000000401155 <+48>:    and    $0xf,%edx
   0x0000000000401158 <+51>:    movzbl 0x402750(%rdx),%edx
   0x000000000040115f <+58>:    mov    %dl,(%rsp,%rax,1)
   0x0000000000401162 <+61>:    add    $0x1,%rax
   0x0000000000401166 <+65>:    cmp    $0x6,%rax
   0x000000000040116a <+69>:    jne    0x401151 <phase_5+44>
   0x000000000040116c <+71>:    movb   $0x0,0x6(%rsp)
   0x0000000000401171 <+76>:    mov    $0x402706,%esi
   0x0000000000401176 <+81>:    mov    %rsp,%rdi
   0x0000000000401179 <+84>:    callq  0x4013f8 <strings_not_equal>
   0x000000000040117e <+89>:    test   %eax,%eax
   0x0000000000401180 <+91>:    je     0x401187 <phase_5+98>
   0x0000000000401182 <+93>:    callq  0x4016cc <explode_bomb>
   0x0000000000401187 <+98>:    mov    0x8(%rsp),%rax
   0x000000000040118c <+103>:   xor    %fs:0x28,%rax
   0x0000000000401195 <+112>:   je     0x40119c <phase_5+119>
   0x0000000000401197 <+114>:   callq  0x400b90 <__stack_chk_fail@plt>
   0x000000000040119c <+119>:   add    $0x10,%rsp
   0x00000000004011a0 <+123>:   pop    %rbx
   0x00000000004011a1 <+124>:   retq
```

2.由下图

```
 0x000000000040113d <+24>:    callq  0x4013da <string_length>
 0x0000000000401142 <+29>:    cmp    $0x6,%eax
```

知输入的是 6 个字符的字符串；

3.由下图

```
0x0000000000401151 <+44>:    movzbl (%rbx,%rax,1),%edx
0x0000000000401155 <+48>:    and    $0xf,%edx
```

知与运算取低四位；

5.由下图

```
0x0000000000401158 <+51>:    movzbl 0x402750(%rdx),%edx
```

知将 edx 后四位作为 0x402760 字符数组的索引值；

6.由下图

```
0x0000000000401179 <+84>:    callq  0x4013f8 <strings_not_equal>
```

知“和字符串常量比较，比较字符串是否相等”；

7.读取字符串

8.“flyers”的各字母位于“maduiersnfotvbyl”的 9,15,14,5,6,7 位，对比 ASCII 码表，取一组 ASC 值化为二进制低四位符合的字符，如“IONEFG”为一组可能的答案。

运行截图：

```
IONEFG

Breakpoint 4, 0x0000000000401125 in phase_5 ()
(gdb) c
Continuing.
Good work!  On to the next...
```

五、总结体会

1.实验过程中遇到的问题、如何解决的：

关卡四递归运算复杂，起初用笔纸穷举计算，算错几次后，改用 C 语言编程运算，解决了问题；

2.过关或挫败的感受：

更深入地了解了汇编操作符的用法，对 CSAPP 书上内容有了更深的理解；

3.意见和建议：

建议课件增加内容,使操作更容易上手.