Computer Architecture

Bomb Lab Report

2016 Spring

李琛
微电子科学与工程
13307130163@fudan.edu.cn

Table of Content

1 introduction

1.1 Overview

A "binary bombïs a program provided to students as an object code file. When run, it prompts the user to type in 6 different strings. If any of these is incorrect, the bomb “explodes,” printing an error message and logging the event on a grading server. Students must “defuse” their own unique bomb by disassembling and reverse engineering the program to determine what the 6 strings should be. The lab teaches students to understand assembly language, and also forces them to learn how to use a debugger. It’s also great fun. A legendary lab among the CMU undergrads.

1.2 Preparation

在拆除炸弹之前，首先进行一些前期准备：

- 通过 putty 登陆服务器，发现我本次 Lab 的实验文件：**bomb51.tar**
- 执行 **tar xvf bomb51.tar** 解压文件，得到四个文件：bomb bomb.c ID README
- 其中 ID README 分别是学生的编号和这次的说明文档。

  于是查看 **bomb.c**, 发现其头文件声明如下：

```
1 #include <stdio.h>
2 #include "support.h"
3 #include "phases.h"
```

  phase.h 包含了这次炸弹的全部关卡，而它并没有在 Lab 中给出。因此我们只能从可执行文件 **bomb** 下手

- 执行 **objdump -d bomb > bomb.txt**，反编译可执行文件，将汇编代码输出到 **bomb.txt**
- 反编译之后的代码非常长，不过在仔细研究之后，发现其中有六个函数 <**phase_1..6**>，分别对应六个关卡。因此，这次 Lab 的关键就是破解这六个关卡对应的汇编代码，分析这六个函数的功能。

2 phase_1

- 第一关的汇编代码如下：

```
08048c10 <phase_1>:
 8048c10:       83 ec 1c                sub    $0x1c,%esp
 8048c13:       c7 44 24 04 ec 99 04    movl   $0x80499ec,0x4(%esp)
 8048c1a:       08
 8048c1b:       8b 44 24 20             mov    0x20(%esp),%eax
 8048c1f:       89 04 24                mov    %eax,(%esp)
 8048c22:       e8 1d 04 00 00          call   8049044 <strings_not_equal>
 8048c27:       85 c0                   test   %eax,%eax
 8048c29:       74 05                   je     8048c30 <phase_1+0x20>
 8048c2b:       e8 a3 09 00 00          call   80495d3 <explode_bomb>
 8048c30:       83 c4 1c                add    $0x1c,%esp
 8048c33:       c3                      ret
```

- 这个函数较短，分析发现，它调用了一个名为 **strings_not_equal** 函数。

  分析 **strings_not_equal** 这个函数，发现它实现这样一个功能：

  比较两个字符串，相等返回 0，不相等返回 1

- 分析 call 之后的程序，发现如果返回值（即 %*eax*）不为 0 的话，会调用 **explode_bomb** 函数，炸弹会被引爆。因此，我们输入的字符串应该与某个字符串相同。

- 这样，破解 phase_1 的方法就很清晰了：

  输入存在地址 **0x80499ec** 中的字符串

- 于是现在的问题是找出存放在地址 0x80499ec 中的字符串。

  从调用 strings_not_equal 的语句向上看：

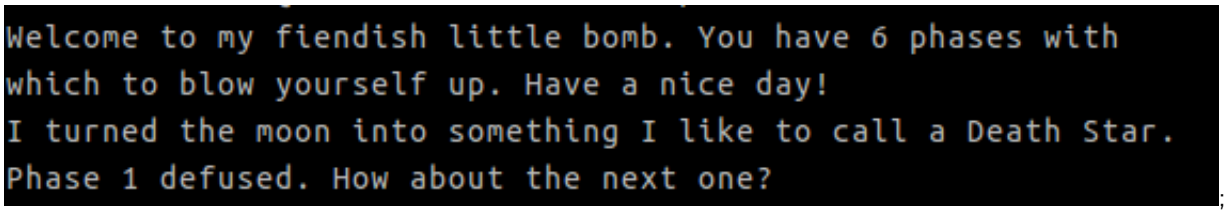
```
8048c13:       c7 44 24 04 ec 99 04    movl   $0x80499ec,0x4(%esp)
8048c1a:       08
8048c1b:       8b 44 24 20             mov    0x20(%esp),%eax
8048c1f:       89 04 24                mov    %eax,(%esp)
```

  这表示，这个函数的两个参数，一个是程序自身地址 **0x80499ec**，另一个 **0x20(%esp)** 就是我们的输入参数。

- 我们首先需要知道 **0x80499ec** 地址里存放的数据。于是使用 **gdb**，输入命令 **p (char *) 0x80499ec**, 查看它存放的数据：

```
(gdb) p (char *) 0x80499ec
$1 = 0x80499ec "I turned the moon into something I like to call a Death Star."
```

- 第一关的答案水落石出：输入 **I turned the moon into something I like to call a Death Star.** ，顺利过关。



;

主要复习知识点：常量的存储、函数的参数传递

3 phase_2

- 第二关的汇编代码如下：

```
08048c34 <phase_2>:
 8048c34:       53                      push   %ebx
 8048c35:       83 ec 38                sub    $0x38,%esp
 8048c38:       8d 44 24 18             lea    0x18(%esp),%eax
 8048c3c:       89 44 24 04             mov    %eax,0x4(%esp)
 8048c40:       8b 44 24 40             mov    0x40(%esp),%eax
 8048c44:       89 04 24                mov    %eax,(%esp)
 8048c47:       e8 8c 0a 00 00          call   80496d8 <read_six_numbers>
 8048c4c:       83 7c 24 18 01          cmpl   $0x1,0x18(%esp)
 8048c51:       74 05                   je     8048c58 <phase_2+0x24>
 8048c53:       e8 7b 09 00 00          call   80495d3 <explode_bomb>
 8048c58:       bb 01 00 00 00          mov    $0x1,%ebx
 8048c5d:       89 d8                   mov    %ebx,%eax
 8048c5f:       83 c3 01                add    $0x1,%ebx
 8048c62:       89 da                   mov    %ebx,%edx
 8048c64:       0f af 54 84 14          imul   0x14(%esp,%eax,4),%edx
 8048c69:       39 54 84 18             cmp    %edx,0x18(%esp,%eax,4)
 8048c6d:       74 05                   je     8048c74 <phase_2+0x40>
 8048c6f:       e8 5f 09 00 00          call   80495d3 <explode_bomb>
 8048c74:       83 fb 06                cmp    $0x6,%ebx
 8048c77:       75 e4                   jne    8048c5d <phase_2+0x29>
 8048c79:       83 c4 38                add    $0x38,%esp
 8048c7c:       5b                      pop    %ebx
 8048c7d:       c3                      ret
```

- 我们首先发现，它调用了一个 **read_six_numbers** 函数。经过分析，这个函数从输入中读入六个整数，并按地址从低到高存放在 **0x18(%esp)** - **(0x30%esp)** 这 24 个字节中。也就是说，**a[0]** 在 **0x18(%esp)**，**a[1]** 在 **0x1c(%esp)**，以此类推。
- 然后观察下面一段代码：

```
8048c4c:       83 7c 24 18 01          cmpl   $0x1,0x18(%esp)
8048c51:       74 05                   je     8048c58 <phase_2+0x24>
8048c53:       e8 7b 09 00 00          call   80495d3 <explode_bomb>
```

- 这段代码将 **0x18(%esp)** 即 **a[0]** 与 1 进行比较，如果不等则炸弹爆炸。这一段说明，我们输入的第一个

数必须为 1。

- 接下来是一个循环：

```
8048c58:    bb 01 00 00 00          mov    $0x1,%ebx
8048c5d:    89 d8                   mov    %ebx,%eax
8048c5f:    83 c3 01                add    $0x1,%ebx
8048c62:    89 da                   mov    %ebx,%edx
8048c64:    0f af 54 84 14          imul   0x14(%esp,%eax,4),%edx
8048c69:    39 54 84 18             cmp    %edx,0x18(%esp,%eax,4)
8048c6d:    74 05                   je     8048c74 <phase_2+0x40>
8048c6f:    e8 5f 09 00 00          call   80495d3 <explode_bomb>
8048c74:    83 fb 06                cmp    $0x6,%ebx
8048c77:    75 e4                   jne    8048c5d <phase_2+0x29>
```

这个循环的循环变量是 %**ebx**，从 2 循环到 5。同时，%**eax** 始终为 %**ebx** - **1** 中间的判断语句，可以发现它是将 **0x14(%esp, %eax, 4) *** %**edx** 相乘，并与 **0x18(%esp, %eax, 4)** 比较。由于 a 数组的起始位置为 **0x18(%esp)**，这两个地址就是 **a[_eax - 1]** 和 **a[_eax]**。

因此，其对应的 c 语言代码如下：

```c
void phase_2() {
        if(num[0] != 1) explode_bomb();
        int b = 1;
        while(b <= 5) {
                a = b + 1;
                if(num[b - 1] * a != num[b])
                        explode_bomb();
                b++;
        }
}
```

- 这个函数表明，我们输入的数组 a 要满足以下条件：
  - 数组长度为 6
  - a[0] = 1
  - a[i] = a[i - 1] * (i + 1)
- 所以，第二关的答案就水落石出了：**1 2 6 24 120 720**

```
1 2 6 24 120 720
That's number 2.  Keep going!
```

考察知识点：数组的存储

4 phase_3

- 第三关的汇编代码如下：

```
08048c7e <phase_3>:
 8048c7e:       83 ec 2c                sub    $0x2c,%esp
 8048c81:       8d 44 24 1c             lea    0x1c(%esp),%eax
 8048c85:       89 44 24 0c             mov    %eax,0xc(%esp)
 8048c89:       8d 44 24 18             lea    0x18(%esp),%eax
 8048c8d:       89 44 24 08             mov    %eax,0x8(%esp)
 8048c91:       c7 44 24 04 c6 9d 04    movl   $0x8049dc6,0x4(%esp)
 8048c98:       08
 8048c99:       8b 44 24 30             mov    0x30(%esp),%eax
 8048c9d:       89 04 24                mov    %eax,(%esp)
 8048ca0:       e8 6b fc ff ff          call   8048910 <__isoc99_sscanf@plt>
 8048ca5:       83 f8 01                cmp    $0x1,%eax
 8048ca8:       7f 05                   jg     8048caf <phase_3+0x31>
 8048caa:       e8 24 09 00 00          call   80495d3 <explode_bomb>
 8048caf:       83 7c 24 18 07          cmpl   $0x7,0x18(%esp)
 8048cb4:       77 64                   ja     8048d1a <phase_3+0x9c>
 8048cb6:       8b 44 24 18             mov    0x18(%esp),%eax
 8048cba:       ff 24 85 5c 9a 04 08    jmp    *0x8049a5c(,%eax,4)
 8048cc1:       b8 00 00 00 00          mov    $0x0,%eax
 8048cc6:       eb 05                   jmp    8048ccd <phase_3+0x4f>
 8048cc8:       b8 8e 01 00 00          mov    $0x18e,%eax
 8048ccd:       2d 66 01 00 00          sub    $0x166,%eax
 8048cd2:       eb 05                   jmp    8048cd9 <phase_3+0x5b>
 8048cd4:       b8 00 00 00 00          mov    $0x0,%eax
 8048cd9:       05 92 00 00 00          add    $0x92,%eax
 8048cde:       eb 05                   jmp    8048ce5 <phase_3+0x67>
 8048ce0:       b8 00 00 00 00          mov    $0x0,%eax
 8048ce5:       2d e9 00 00 00          sub    $0xe9,%eax
 8048cea:       eb 05                   jmp    8048cf1 <phase_3+0x73>
 8048cec:       b8 00 00 00 00          mov    $0x0,%eax
 8048cf1:       05 28 01 00 00          add    $0x128,%eax
 8048cf6:       eb 05                   jmp    8048cfd <phase_3+0x7f>
 8048cf8:       b8 00 00 00 00          mov    $0x0,%eax
 8048cfd:       2d df 01 00 00          sub    $0x1df,%eax
 8048d02:       eb 05                   jmp    8048d09 <phase_3+0x8b>
```

```
8048d04:    b8 00 00 00 00          mov    $0x0,%eax
8048d09:    05 df 01 00 00          add    $0x1df,%eax
8048d0e:    eb 05                   jmp    8048d15 <phase_3+0x97>
8048d10:    b8 00 00 00 00          mov    $0x0,%eax
8048d15:    83 c0 80                add    $0xffffff80,%eax
8048d18:    eb 0a                   jmp    8048d24 <phase_3+0xa6>
8048d1a:    e8 b4 08 00 00          call   80495d3 <explode_bomb>
8048d1f:    b8 00 00 00 00          mov    $0x0,%eax
8048d24:    83 7c 24 18 05          cmpl   $0x5,0x18(%esp)
8048d29:    7f 06                   jg     8048d31 <phase_3+0xb3>
8048d2b:    3b 44 24 1c             cmp    0x1c(%esp),%eax
8048d2f:    74 05                   je     8048d36 <phase_3+0xb8>
8048d31:    e8 9d 08 00 00          call   80495d3 <explode_bomb>
8048d36:    83 c4 2c                add    $0x2c,%esp
8048d39:    c3                      ret
```

- 首先看到它调用了 **sscanf** 函数，参数存放在 **0x8049dc6** 地址中。

  于是首先用 gdb 查看 **0x8049dc6** 地址中的参数：

```
(gdb) p (char *) 0x8049dc6
$3 = 0x8049dc6 "%d %d"
```

  “%*d*%*d*”代表读入了两个整数。调用 sscanf 结束后，将 sscanf 的返回值与 1 比较，若不大于 1 则炸弹爆炸。

  由此可以得到我们这关的目标是输入两个整数 **a b**。

- 输入的两个整数 **a b** 存放在 **0x18(%esp)** 和 **0x1c(%esp)** 中。接着往下看程序：

```
8048caf:    83 7c 24 18 07          cmpl   $0x7,0x18(%esp)
8048cb4:    77 64                   ja     8048d1a <phase_3+0x9c>
```

  这里可以发现，程序将 **0x18(%esp)**，即输入的第一个整数 **a** 与 7 进行比较，如果大于 7 则炸弹爆炸。同时，由于使用的是 **ja** 命令，即 **a** 为无符号数，**a** 应该不小于 0

  这样我们可以得出第一个限制条件：输入的第一个整数 $0 \leq a \leq 7$

  继续往下看，注意到这一段：

- 这段具有很明显的 **switch** 语句的特征，而跳转表存放在 **0x8049a5c** 中。

于是调用 gdb，先查看对应的跳转表：

这样可以将接下来的代码划分模块：

- 于是可以得出对应的 c 语言代码：

```
int fun(int a) {
        int ret = 0;
        switch (a) {
                case 1: ret = 0;
                case 0:
                        if (a != 1) ret = 0x18e;
                        ret -= 0x166;
                case 2: ret += 0x92;
                case 3: ret -= 0xe9;
                case 4: ret += 0x128;
                case 5: ret -= 0x1df;
                case 6: ret += 0x1df;
                case 7: ret += 0xffffff80;
        }
        return ret;
}
```

经过整理后的代码如下：

```
int fun(int a) {
        int ret = 0;
        switch(a) {
                case 0 : ret = 121; break;
                case 1 : ret = -277; break;
                case 2 : ret = 81; break;
                case 3 : ret = -65; break;
                case 4 : ret = 168; break;
                case 5 : ret = -128; break;
                case 6 : ret = 351; break;
                case 7 : ret = -128; break;
        }
        return ret;
}
```

- 在 switch 语句段后，有这么一段：

  这里给出了第二个限制条件：输入的第一个整数 $a \leq 5$

  最后，程序将输入的第二个整数 **b** 与 switch 的结果进行比较，不同则炸弹爆炸。

- 因此，这关的答案就水落石出了：

输入一个整数 $a(0 \leq a \leq 5)$，和 a 经过 switch 之后的结果 b

选用 $\mathbf{a = 4, b = 168}$, 顺利过关

```
8048cb6:       8b 44 24 18             mov    0x18(%esp),%eax
8048cba:       ff 24 85 5c 9a 04 08    jmp    *0x8049a5c(,%eax,4)
```

```
(gdb) p /x *0x8049a5c
$4 = 0x8048cc8
(gdb) p /x *(0x8049a5c + 4)
$5 = 0x8048cc1
(gdb) p /x *(0x8049a5c + 8)
$6 = 0x8048cd4
(gdb) p /x *(0x8049a5c + 12)
$7 = 0x8048ce0
(gdb) p /x *(0x8049a5c + 16)
$8 = 0x8048cec
(gdb) p /x *(0x8049a5c + 20)
$9 = 0x8048cf8
(gdb) p /x *(0x8049a5c + 24)
$10 = 0x8048d04
(gdb) p /x *(0x8049a5c + 28)
$11 = 0x8048d10
```

```
8048cc1:    b8 00 00 00 00          mov    $0x0,%eax
8048cc6:    eb 05                   jmp    8048ccd <phase_3+0x4f>
```
Case 0

```
8048cc8:    b8 8e 01 00 00          mov    $0x18e,%eax
8048ccd:    2d 66 01 00 00          sub    $0x166,%eax
8048cd2:    eb 05                   jmp    8048cd9 <phase_3+0x5b>
```
Case 1

```
8048cd4:    b8 00 00 00 00          mov    $0x0,%eax
8048cd9:    05 92 00 00 00          add    $0x92,%eax
8048cde:    eb 05                   jmp    8048ce5 <phase_3+0x67>
```
Case 2

```
8048ce0:    b8 00 00 00 00          mov    $0x0,%eax
8048ce5:    2d e9 00 00 00          sub    $0xe9,%eax
8048cea:    eb 05                   jmp    8048cf1 <phase_3+0x73>
```
Case 3

```
8048cec:    b8 00 00 00 00          mov    $0x0,%eax
8048cf1:    05 28 01 00 00          add    $0x128,%eax
8048cf6:    eb 05                   jmp    8048cfd <phase_3+0x7f>
```
Case 4

```
8048cf8:    b8 00 00 00 00          mov    $0x0,%eax
8048cfd:    2d df 01 00 00          sub    $0x1df,%eax
8048d02:    eb 05                   jmp    8048d09 <phase_3+0x8b>
```
Case 5

```
8048d04:    b8 00 00 00 00          mov    $0x0,%eax
8048d09:    05 df 01 00 00          add    $0x1df,%eax
8048d0e:    eb 05                   jmp    8048d15 <phase_3+0x97>
```
Case 6

```
8048d10:    b8 00 00 00 00          mov    $0x0,%eax
8048d15:    83 c0 80                add    $0xffffff80,%eax
```
Case 7

```
8048d1f:    b8 00 00 00 00          mov    $0x0,%eax
8048d24:    83 7c 24 18 05          cmpl   $0x5,0x18(%esp)
8048d29:    7f 06                   jg     8048d31 <phase_3+0xb3>
8048d2b:    3b 44 24 1c             cmp    0x1c(%esp),%eax
8048d2f:    74 05                   je     8048d36 <phase_3+0xb8>
8048d31:    e8 9d 08 00 00          call   80495d3 <explode_bomb>
```

5 phase_4

第四关的汇编代码如下：

```
08048d64 <phase_4>:
 8048d64:       83 ec 2c                sub    $0x2c,%esp
 8048d67:       8d 44 24 1c             lea    0x1c(%esp),%eax
 8048d6b:       89 44 24 08             mov    %eax,0x8(%esp)
 8048d6f:       c7 44 24 04 c9 9d 04    movl   $0x8049dc9,0x4(%esp)
 8048d76:       08
 8048d77:       8b 44 24 30             mov    0x30(%esp),%eax
 8048d7b:       89 04 24                mov    %eax,(%esp)
 8048d7e:       e8 8d fb ff ff          call   8048910 <__isoc99_sscanf@plt>
 8048d83:       83 f8 01                cmp    $0x1,%eax
 8048d86:       75 07                   jne    8048d8f <phase_4+0x2b>
 8048d88:       83 7c 24 1c 00          cmpl   $0x0,0x1c(%esp)
 8048d8d:       7f 05                   jg     8048d94 <phase_4+0x30>
 8048d8f:       e8 3f 08 00 00          call   80495d3 <explode_bomb>
 8048d94:       8b 44 24 1c             mov    0x1c(%esp),%eax
 8048d98:       89 04 24                mov    %eax,(%esp)
 8048d9b:       e8 9a ff ff ff          call   8048d3a <func4>
 8048da0:       3d a7 41 00 00          cmp    $0x41a7,%eax
 8048da5:       74 05                   je     8048dac <phase_4+0x48>
 8048da7:       e8 27 08 00 00          call   80495d3 <explode_bomb>
 8048dac:       83 c4 2c                add    $0x2c,%esp
 8048daf:       c3                      ret
```

首先，它和 phase_3 一样，调用了 sscanf 函数。和上一个函数的处理方法一样，运用 gdb，发现 sscanf 的参数是**"%d"**，即读入一个整数 n，存放在 **0x1c(%esp)**。

读入后，将 n 与 0 比较，如果 $n \leq 0$ 炸弹就会爆炸。

因此，我们得到了第一个限制条件：$n > 0$

随后，将 n 作为参数传入函数 **func4** 中。

这段代码表示，如果函数的返回值不为 **0x41a7**，则炸弹爆炸。

这样就知道了第四关的要求：找出使 **func4** 的结果为 **0x41a7** 的输入值。

于是我们开始研究 **func4** 函数。

分析这一个函数，可以得出它的作用如下：

- 首先，函数将返回值置为 1。
- 如果传入参数 $x \leq 0$，则直接返回。

```
08048d3a <func4>:
 8048d3a:       83 ec 1c                sub    $0x1c,%esp
 8048d3d:       8b 54 24 20             mov    0x20(%esp),%edx
 8048d41:       b8 01 00 00 00          mov    $0x1,%eax
 8048d46:       85 d2                   test   %edx,%edx
 8048d48:       7e 16                   jle    8048d60 <func4+0x26>
 8048d4a:       83 ea 01                sub    $0x1,%edx
 8048d4d:       89 14 24                mov    %edx,(%esp)
 8048d50:       e8 e5 ff ff ff          call   8048d3a <func4>
 8048d55:       8d 14 c5 00 00 00 00    lea    0x0(,%eax,8),%edx
 8048d5c:       29 c2                   sub    %eax,%edx
 8048d5e:       89 d0                   mov    %edx,%eax
 8048d60:       83 c4 1c                add    $0x1c,%esp
 8048d63:       c3                      ret
```

- 递归调用 **fun4**
- 将递归的结果乘 7 后作为返回值

对应的 C 语言代码如下：

```c
int fun4(int x) {
        int ret = 1;
        if (x <= 0)
                return ret;
        return ret * 7;
}
```

因此，我们的问题变成：$\log_7(0x41a7) = ?$

经过计算得出，$7^5 = 16807 = 0x41a7$

输入 5，成功过关。

6 phase_5

7 phase_6

8 secret_phase

9 conclusion