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

```
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
```

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**

考察知识点：数组的存储

4 phase_3

5 phase_4

6 phase_4

7 phase_6

8 secret_phase

9 conclusion