二进制程序分析实验

（苏丰，南京大学）

一、实验介绍

本实验通过对一个二进制程序的构成与运行逻辑的分析，加深对理论课程中关于程序的机器级表示各方面知识点的理解，以及反汇编、跟踪/调试等常用技能的掌握。

实验包含多个阶段的任务，每个阶段对应一个可执行程序，程序运行时要求输入一个特定字符串，如果该输入字符串符合该阶段程序的要求，该阶段即通过，否则输出失败提示。

每个阶段的程序考察了程序与数据的机器级表示的不同方面：

- 阶段 1：字符串比较
- 阶段 2：浮点数表示
- 阶段 3：循环
- 阶段 4：选择（条件分支）
- 阶段 5：递归调用
- 阶段 6：数组
- 阶段 7：指针
- 阶段 8：结构与链表/树

实验的目标是通过对各阶段程序中相关函数的执行逻辑及其所操作数据的分析与理解，尽可能多地构造和获得能够成功通过相应阶段的目标字符串。

- 实验环境：Linux i386
- 实验语言：汇编、C

二、实验数据

在本实验中，每位学生可从 Lab 实验网站下载包含本实验相关文件的一个 tar 文件。可在 Linux 实验环境中使用命令“**`tar xvf tar 文件名`**”将其中包含的文件提取到当前目录中。该 tar 文件中包含如下实验所需文件：

- main.c：包含程序 main 函数的 C 语言源程序
- phase1, ……, phase8：程序各阶段对应的二进制可执行程序

三、实验过程

实验中需要针对各阶段程序，分别进行反汇编分析、调试/跟踪和运行测试。

1）对第 ***n*** 阶段（n=1..8）的可执行程序 phase[***n***]进行反汇编，获得并分析其对应的汇编源程序，以了解其中主函数 phase（及其调用函数——C 标准库函数除外，其功能可查阅相关文献）的执行逻辑；

2）按照阶段主函数 phase 对作为其输入参数的一个字符串的测试逻辑，构造满足测试要求的输入字符串内容（可保存于一个文本文件 phase***n***.txt 中的第一且唯一一行中，n=1..8），注意该字符串的长度不能超过 255 字节；

3）运行、测试所构造解答字符串的正确性：可执行程序 phase[***n***]在运行时可接受 0 或 1 个命令行参数（可参见 main.c 源文件中 main 函数）：

- 如果运行时不指定参数，则程序打印出欢迎信息后，期望你输入当前阶段 ***n***（n=1..8）的解答字符串（以换行标识结束），程序使用输入字符串调用阶段主函数 phase 以测试决定该阶段通过与否，并相应输出提示信息。

- 如果已将当前阶段 ***n*** 的解答字符串记录在一个文本文件 phase***n***.txt（n=1..8）中（其中行尾必须以 Unix/Linux 格式的换行结束），则可将该文件作为运行程序时的一个命令行参数：

  ./phase***n*** phase***n***.txt

  程序将读取文件第一行中的字符串以测试阶段通过与否。

注意：实验各阶段之间没有直接、必然的联系，可按自己选择的任意顺序完成各实验阶段。另一方面，由于前面阶段中考察的知识可能也出现于后面阶段，建议大体按照序号顺序完成实验的各个阶段。

四、实验结果提交

将保存了各阶段的解答字符串的文本文件（phase1.txt、phase2.txt、phase3.txt、phase4.txt、phase5.txt、phase6.txt、phase7.txt、phase8.txt）用 tar 工具打包（注意其中不能包含任何目录结构），并命名为“学号.tar”的单一文件提交。

可只提交已完成阶段的对应文本文件，如验证正确可获得相应阶段的实验分数。

注意：提交的文本文件中只能包含解答字符串一行文本，除此之外不要包含任何其它字符，且该行的行尾必须采用 Unix/Linux 的换行字符格式（不同于 Windows 上默认使用的换行字符），因此建议在实验所用的 Linux 环境中编写该文件，并在提交前对其进行检查确认。

五、实验工具

为完成二进制程序分析任务，可使用 objdump 工具程序反汇编可执行程序，并使用 gdb 工具程序单步跟踪每一阶段的机器指令，从中理解每一指令的行为和作用，进而推断满足阶段要求的解答字符串的内容组成。例如，可在每一阶段主函数 phase 的起始指令处和返回 False（0）的指令前设置断点。

下面简要说明完成本实验所需要的一些实验工具：

GDB

为从二进制可执行程序中找出导致返回 False（0）的条件，可使用 GDB 程序帮助对程序的分析和跟踪。GDB 是 GNU 开源组织发布的一个强大的交互式程序调试工具。一般来说，GDB 可帮助完成以下几方面的调试工作（更详细描述可参看 GDB 文档和相关资料）：

- 装载、启动被调试的程序
- 使被调试程序在指定的调试断点处中断执行，以方便查看程序变量、寄存器、栈内容等程序运行的现场数据
- 动态改变程序的执行环境，如修改变量的值

objdump

- –t 选项：打印指定二进制程序的符号表，其中包含了程序中的函数、全局变量的名称和存储地址
- –d 选项：对二进制程序中的机器指令代码进行反汇编。通过分析汇编源代码了解程序是如何运行的

strings

该命令显示二进制程序中的所有可打印字符串

六、实验步骤演示

下面以阶段 1 为例演示基本的实验步骤（以下示例中的代码、数据、地址等可能不同于实际获得的程序，仅供参考）：

首先调用“objdump –d phase1 > disassemble.txt”对阶段 1 的可执行程序 phase1 进行反汇编，并将汇编代码输出到“disassemble.txt”文本文件中。

查看该汇编代码文件，可以在 main 函数中找到如下指令，从而得知阶段 1 的处理程序包含在 main 函数所调用的函数 phase 中：

```
……
8049317:      68 e0 c0 04 08       push   $0x804c0e0
804931c:      e8 80 00 00 00       call   80493a1 <phase>
8049321:      83 c4 10             add    $0x10,%esp
8049324:      85 c0                test   %eax,%eax
8049326:      74 18                je     8049340 <main+0x148>
……
```

接下来在汇编代码文件中继续查找函数 phase 的具体定义，如下所示：

```
080493a1 <phase>:
 80493a1:      55                   push   %ebp
 80493a2:      89 e5                mov    %esp,%ebp
 80493a4:      83 ec 08             sub    $0x8,%esp
 80493a7:      83 ec 08             sub    $0x8,%esp
 80493aa:      68 1c a1 04 08       push   $0x804a11c
 80493af:      ff 75 08             pushl  0x8(%ebp)
 80493b2:      e8 79 fc ff ff       call   8049030 <strcmp@plt>
 80493b7:      83 c4 10             add    $0x10,%esp
 80493ba:      85 c0                test   %eax,%eax
 80493bc:      0f 94 c0             sete   %al
 80493bf:      0f b6 c0             movzbl %al,%eax
 80493c2:      c9                   leave
 80493c3:      c3                   ret
……
```

从上面的汇编代码中可以看出函数 strcmp（C 标准库函数，其具体功能请自行查阅相关资料）所需要的两个字符串参数中的一个的存储首地址是 0x804a11c，另一个的存储首地址保存于地址 0x8(%ebp)所指向的栈存储单元里，由课程知识可知该单元实际存放的是 phase 函数的第一个也是唯一一个参数的值，进一步由 main 函数的源代码（或汇编代码）可知该参数实际是你输入的解答字符串的地址。因此，结合 strcmp 库函数的功能和其后测试 test 与条件设置 sete 指令的功能，可知如想要 phase 函数返回 True（非 0），必须使 strcmp 返回 0，即其两个字符串参数的内容必须相同。因此，可基于其中存储于地址 0x804a11c 的字符串参数的内容来构造解答字符串。

为此，可使用 gdb 查看该地址存储的字符串内容，具体过程如下，其中所设置的断点位于地址 0x80493b2（即上列函数 phase 代码中的 call strcmp 指令处）：

```
$ gdb phase1
GNU gdb (Debian 10.1-1.7) 10.1.90.20210103-git
Copyright (C) 2021 Free Software Foundation, Inc.
```

```
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from phase1...
(No debugging symbols found in phase1)
(gdb) b *0x80493b2
Breakpoint 1 at 0x80493b2
(gdb) r
Starting program: ……/phase1
Welcome to the binary program analysis lab.
Here begins the task. Please input your answer ...
```

【输入字符串】

```
Breakpoint 1, 0x080493b2 in phase ()
(gdb) x/1s 0x804a11c
0x804a11c:      "SRAM stores each bit in a bistable memory cell."
```

从上可知自地址 0x804a11c 处开始存放的字符串内容是“SRAM stores each bit in a bistable memory cell.”，也就是解答字符串应该包含的内容，至此完成了该阶段的求解。

七、参考信息

实验阶段 2 中的浮点相关操作采用了 SSE2 指令（不同于 X87 浮点指令）加以实现，请自行参考相关资料了解这些指令的功能。例如：

- **cvttsd2si** 指令：Converts a double-precision floating-point value in the source operand to a signed double-word integer in the destination operand. The source operand can be an XMM register or a memory location. The destination operand is a general-purpose register. When the source operand is an XMM register, the double-precision floating-point value is contained in the low quadword of the register.
- **movsd** 指令：Moves a scalar double-precision floating-point value from the source operand to the destination operand. The source and destination operands can be XMM registers or memory locations. This instruction can be used to move a double-precision floating-point value to and from the low quadword of an XMM register and a memory location, or to move a double-precision floating-point value between the

low quadwords of two XMM registers. The instruction cannot be used to transfer data between memory locations.

- **pxor** 指令：Performs a bitwise logical exclusive-OR (XOR) operation on the source operand and the destination operand and stores the result in the destination operand. Each bit of the result is 1 if the corresponding bits of the two operands are different; each bit is 0 if the corresponding bits of the operands are the same.