1 实验 1: 拆弹

1.1 环境安装

sudo apt install qemu-user

1.2 简介

在实验 1 中，你需要通过阅读汇编代码以及使用调试工具来拆除一个二进制炸弹程序。本实验分为两个部分：第一部分介绍拆弹实验的基本知识，包括 ARM 汇编语言、QEMU 模拟器、GDB 调试器的使用；第二部分需要分析炸弹程序，推测正确的输入来使得炸弹程序能正常退出。

1.3 第一部分：实验基本知识

本部分旨在熟悉 ARM 汇编语言，以及使用 QEMU 和 QEMU/GDB 调试。

1.3.1 熟悉 AArch64 汇编

AArch64 是 ARMv8 ISA 的 64 位执行状态。《ARM 指令集参考指南》（点此打开）是一个帮助入门 ARM 语法的手册。在 ChCore 实验中，只需要在提示下可以理解一些关键汇编和编写简单的汇编代码即可。

1.3.2 使用 QEMU 运行炸弹程序

我们在实验中提供了`bomb`二进制文件，但该文件只能运行在基于 AArch64 的 Linux 中。通过 QEMU，我们可以在其他架构上模拟运行。同时，QEMU 可以结合 GDB 进行调试（如打印输出、单步调试等）。

小知识： QEMU 不仅可以模拟运行用户态程序，也可以模拟运行在内核态的操作系统。在前一种模式下，QEMU 会模拟运行用户态的汇编代码，同时将系统调用等翻译为对宿主机的调用。在后一种模式下，QEMU 将在虚拟硬件上模拟一整套计算机启动的过程。

在`bomb-lab`目录下，输入以下命令可以在 QEMU 中运行炸弹程序：

```
os-bomb$ make qemu
```

炸弹程序的标准输出将会显示在 QEMU 中:

```
Type in your defuse password:
```

1.3.3 QEMU 与 GDB

在实验中，由于需要在 x86-64 平台上使用 GDB 来调试 AArch64 代码，因此使用`gdb-multiarch`代替了普通的`gdb`。使用 GDB 调试的原理是，QEMU 可以启动一个 GDB **远程目标（remote target）**（使用`-s`或`-S`参数启动），QEMU 会在真正执行镜像中的指令前等待 GDB 客户端的连接。开启远程目标之后，可以开启 GDB 进行调试，它会在某个端口上进行监听。

打开两个终端，在`bomb-lab`目录下，输入`make qemu-gdb`和`make gdb`命令可以分别打开带有 GDB 调试的 QEMU 以及 GDB，在 GDB 中将会看到如下的输出:

```
...
0x0000000000400540 in ?? ()
...
(gdb)
```

1.4 第二部分：二进制炸弹的拆除

我们在实验中提供了一个二进制炸弹程序`bomb`以及它的部分源码`bomb.c`。在 bomb.c 中，你可以看到一共有 6 个 phase。对每个 phase，`bomb`程序将从标准中输入中读取一行用户输入作为这一阶段的拆弹密码。若这一密码错误，炸弹程序将异常退出。你的任务是通过 GDB 以及阅读汇编代码，判断怎样的输入可以使得炸弹程序正常通过每个 phase。以下是一次失败尝试的例子:

```
[user@osbook bomb-lab] $ make qemu
qemu-aarch64 bomb
Type in your defuse password:
1234
BOOM!!!
```

Listing 1: 一次让炸弹爆炸的错误示范

1.4.1 提示

你需要学习gdb、objdump的使用来查看炸弹程序对应的汇编，并通过断点等方法来查看炸弹运行时的状态（寄存器、内存的值等）。以下是使用gdb来查看炸弹运行状态的例子。在这个例子中，我们在main函数的开头打了一个断点，通过continue让程序运行直至遇到我们设置的断点，使用info查看了寄存器中的值，最终通过x查看了x0寄存器中的地址指向的字符串的内容。

```
add symbol table from file "bomb"
(y or n) y
Reading symbols from bomb...
(gdb) break main
Breakpoint 1 at 0x4006a4
(gdb) continue
Continuing.

Breakpoint 1, 0x00000000004006a4 in main ()
(gdb) disassemble
Dump of assembler code for function main:
   0x0000000000400694 <+0>:     stp     x29, x30, [sp, #-16]!
   0x0000000000400698 <+4>:     mov     x29, sp
   0x000000000040069c <+8>:     adrp    x0, 0x464000 <free_mem+64>
   0x00000000004006a0 <+12>:    add     x0, x0, #0x778
=> 0x00000000004006a4 <+16>:    bl      0x413b20 <puts>
   0x00000000004006a8 <+20>:    bl      0x400b10 <read_line>
   0x00000000004006ac <+24>:    bl      0x400734 <phase_0>
   0x00000000004006b0 <+28>:    bl      0x400708 <phase_defused>
   0x00000000004006b4 <+32>:    bl      0x400b10 <read_line>
   0x00000000004006b8 <+36>:    bl      0x400760 <phase_1>
   0x00000000004006bc <+40>:    bl      0x400708 <phase_defused>
   0x00000000004006c0 <+44>:    bl      0x400b10 <read_line>
   0x00000000004006c4 <+48>:    bl      0x400788 <phase_2>
   0x00000000004006c8 <+52>:    bl      0x400708 <phase_defused>
   0x00000000004006cc <+56>:    bl      0x400b10 <read_line>
   0x00000000004006d0 <+60>:    bl      0x400800 <phase_3>
   0x00000000004006d4 <+64>:    bl      0x400708 <phase_defused>
   0x00000000004006d8 <+68>:    bl      0x400b10 <read_line>
   0x00000000004006dc <+72>:    bl      0x4009e4 <phase_4>
   0x00000000004006e0 <+76>:    bl      0x400708 <phase_defused>
   0x00000000004006e4 <+80>:    bl      0x400b10 <read_line>
   0x00000000004006e8 <+84>:    bl      0x400ac0 <phase_5>
   0x00000000004006ec <+88>:    bl      0x400708 <phase_defused>
   0x00000000004006f0 <+92>:    adrp    x0, 0x464000 <free_mem+64>
   0x00000000004006f4 <+96>:    add     x0, x0, #0x798
```

```
   0x0000000000004006f8 <+100>:   bl      0x413b20 <puts>
   0x0000000000004006fc <+104>:   mov     w0, #0x0
          // #0
   0x0000000000400700 <+108>:   ldp     x29, x30, [sp], #16
   0x0000000000400704 <+112>:   ret
End of assembler dump.
(gdb) info registers x0
x0             0x464778            4605816
(gdb) x /s 0x464778
0x464778:       "Type in your defuse password!"
```

Listing 2: 使用 gdb 进行简单的调试

在破解后续阶段时，为了避免每次都需要输入先前阶段的拆弹密码，你可以通过重定向的方式来让炸弹程序读取文件中的密码：

```
[user@osbook bomb-lab] $ make qemu < ans.txt
qemu-aarch64 bomb
Type in your defuse password:
5 phases to go
4 phases to go
3 phases to go
2 phases to go
1 phases to go
0 phases to go
Congrats! You have defused all phases!
```

Listing 3: 通过重定向来从文件读取密码

在这个例子中，我们将每一阶段的密码逐行保存在`ans.txt`中。若这些密码完全正确，你将看到上述拆弹成功的提示。