HPCL
State Key Laboratory of
High Performance Computing

二进制炸弹实验

2020年春计算机系统实验二

我们学习了什么?

- 3.2 程序编码
- 3.3 数据格式
- 3.4 访问信息
- 3.5 算术和逻辑操作

实验说明

- 锻炼大家阅读x86汇编的能力
- 实验主要内容
  - 我们会给每个人发一个二进制文件（所谓的炸弹）
  - 这个炸弹会接收几组输入
    - 如果输入与预期不同，则炸弹会爆炸。
  - 为使炸弹不爆炸，大家反汇编该二进制文件，找到正确的输入
- 实验环境：Linux
- 使用的主要工具
  - gdb
  - objdump

实验过程说明

- 同学们收到的文档有两个：
  - extra materials: 包含多个额外的参考手册
    - 其中A Readers Guide to x86 Assembly包含了常用命令的说明
  - bomb文件夹
    - bomb_xxxx: 二进制炸弹，共68个。每个人使用自己的那个
    - bomb.c: 对这个炸弹程序的整体框架的说明

实验过程说明

- 想试试是什么样子的?
  - ./bomb 直接运行
- 想得到正确的结果
  - 请使用gdb进行调试并阅读相应的汇编
- 使用的主要工具
  - gdb
  - objdump

常用工具

- objdump
  - 获得某个binary的反汇编
    - objdump –d <binary>
- 利用objdump得到的文件信息会非常长
  - 无法得知内存数据信息
  - 可做参考，但无法解题

GDB常用命令说明

- 对程序prog启动GDB
  - gdb prog
- 带参数启动

```
gdb --args executablename arg1 arg2 arg3
```

- 常用命令
  - stepi (si): 单条指令执行
  - next (n): 类似stepi，但以函数调用为单位
  - continue (c): 继续执行

`gdb` *program* [*core*]	debug *program* [using coredump *core*]
`b` [*file*`:`]*function*	set breakpoint at *function* [in *file*]
`run` [*arglist*]	start your program [with *arglist*]
`bt`	backtrace: display program stack
`p` *expr*	display the value of an expression
`c`	continue running your program
`n`	next line, stepping over function calls
`s`	next line, stepping into function calls

GDB使用说明

- 第3.10.2节：使用GDB调试器
  - 教材193页

命令	效果
开始和停止	
quit	退出 GDB
run	运行程序(在此给出命令行参数)
kill	停止程序
断点	
break multstore	在函数 multstore 入口处设置断点
break * 0x400540	在地址 0x400540 处设置断点
delete 1	删除断点 1
delete	删除所有断点
执行	
stepi	执行 1 条指令
stepi 4	执行 4 条指令
nexti	类似于 stepi，但以函数调用为单位
continue	继续执行
finish	运行到当前函数返回

GDB使用说明

检查代码	
disas	反汇编当前函数
disas multstore	反汇编函数 multstore
disas 0x400544	反汇编位于地址 0x400544 附近的函数
disas 0x400540,0x40054d	反汇编指定地址范围内的代码
print /x $rip	以十六进制输出程序计数器的值
检查数据	
print $rax	以十进制输出%rax的内容
print /x $rax	以十六进制输出%rax的内容
print /t $rax	以二进制输出%rax的内容
print 0x100	输出 0x100 的十进制表示
print /x 555	输出 555 的十六进制表示
print /x ($rsp+ 8)	以十六进制输出%rsp的内容加上 8
print *(long *) 0x7fffffffe818	输出位于地址 0x7fffffffe818 的长整数
print *(long *)($rsp+ 8)	输出位于地址%rsp+8 处的长整数
x/2g 0x7fffffffe818	检查从地址 0x7fffffffe818 开始的双(8 字节)字
x/20bmultstore	检查函数 multstore 的前 20 个字节
有用的信息	
info frame	有关当前栈帧的信息
info registers	所有寄存器的值
help	获取有关 GDB 的信息

图 3-39 GDB 命令示例。说明了一些 GDB 支持机器级程序调试的方式

- gdbnotes-x86-64: 基本上与该表相同，但有额外的打印说明
- gdb-refcard：包含了各种快速说明

GDB中对比查看

- layout split
  - 同时查看C代码与汇编代码
- layout prev / regs
  - 同时查看汇编代码与寄存器值
- layout next / src
  - 同时查看C代码
- layout asm
  - 同时查看汇编代码
- backtrace (bt)
  - 显示函数调用栈

GDB查看寄存器、内存值

- 查看寄存器的值
  - print /x $rdi
    - /x : 十六进制， /d: 十进制， /t: 二进制
  - print /x ($rdi+8)
    - 打印 (%rdi的值 + 8) 的结果
- 查看内存的值
  - x/w 0xbffff890 查看内存中以该地址起始的4个字节的数
  - x/s 0xbffff890 查看以该地址起始的内存中保存的字符串内容

x86 - 寄存器

- 传递参数时
  - 前6个参数分别使用
    - %rdi, %rsi, %rdx, %rcx
    - %r8, %r9
  - 如果多于6个参数
    - 剩余参数通过栈传递
- 几个重要的寄存器
  - %rax (%eax)：返回值
  - %rsp： 与栈有关
- %rip

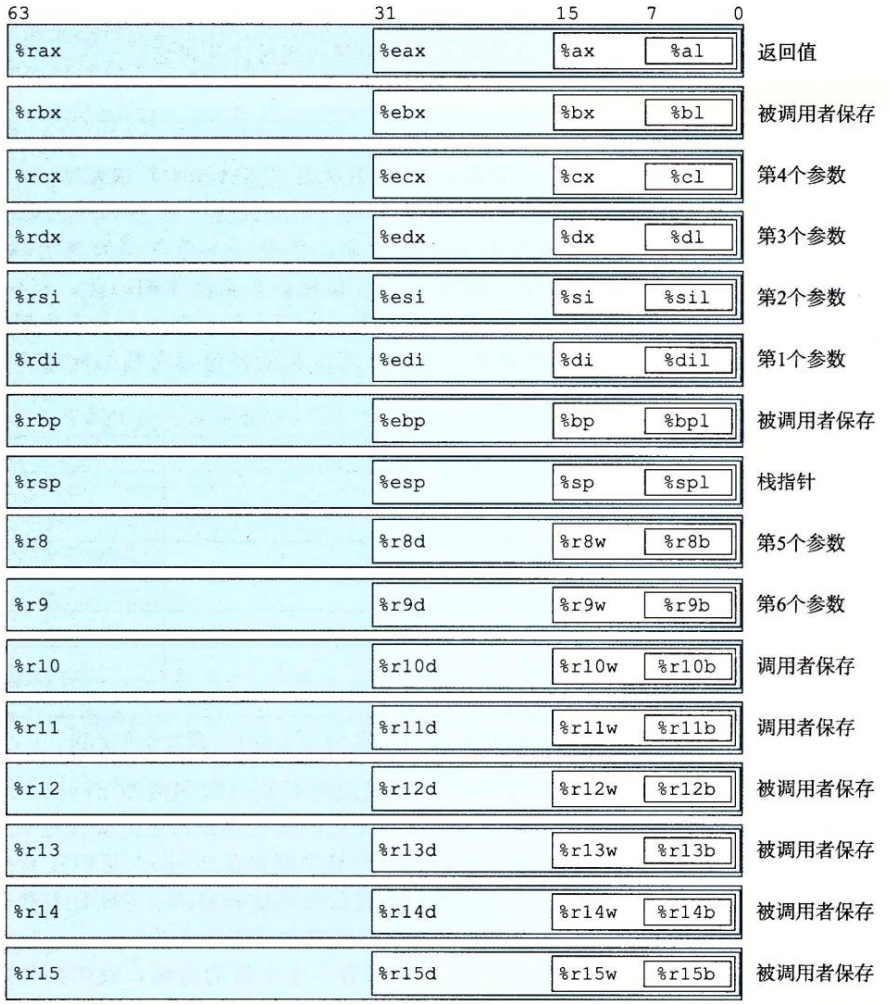

图 3-2 整数寄存器。所有 16 个寄存器的低位部分都可以作为字节、字(16 位)、双字(32 位)和四字(64 位)数字来访问

函数调用的基本方法

- 例子：

```c
// a: %rdi
// b: %rsi
int func(int a, int b)
{

    int c = a + b;

    // the result `c` will be put into %rax
    return c;
}
```

前6个参数: %rdi, %rsi, %rdx, %rcx, %r8, %r9

返回值: %rax (%eax)

```c
/* Hmm...  Four phases must be more secure than one phase! */
input = read_line(); /* Get input                  */
phase_1(input);      /* Run the phase              */
printf("Phase 1 defused. How about the next one?\n");
```

- 在调用phase_1时，第一个参数为input

类型: `char* input;`

- 所以input对应的寄存器会被放在%rdi寄存器中

x86 - 常用指令说明

- lea: 加载有效地址
  - 详见教材3.5.1节（129页）
- test a, b：计算 a & b， 并根据计算结果设置相应的标志位
  - 如果结果为0，置ZF为1
  - 如果结果为负数，则置SF为1
  - CF与OF都置为0
- cmp a, b：计算 b - a， 并根据计算结果设置相应的标志位
  - 如果结果为0，置ZF为1
  - 如果结果为负数，则置SF为1
  - CF与OF都置为0
- CF: 进位标志 ZF: 零标志 SF: 符号标志 OF: 溢出标志

内存寻址格式

类型	格式	操作数值	名称
立即数	$\$Imm$	Imm	立即数寻址
寄存器	r_a	$R[r_a]$	寄存器寻址
存储器	Imm	$M[Imm]$	绝对寻址
存储器	(r_a)	$M[R[r_a]]$	间接寻址
存储器	$Imm(r_b)$	$M[Imm+R[r_b]]$	（基址 + 偏移量）寻址
存储器	(r_b, r_i)	$M[R[r_b]+R[r_i]]$	变址寻址
存储器	$Imm(r_b, r_i)$	$M[Imm+R[r_b]+R[r_i]]$	变址寻址
存储器	$(,r_i, s)$	$M[R[r_i] \cdot s]$	比例变址寻址
存储器	$Imm(,r_i,s)$	$M[Imm+R[r_i] \cdot s]$	比例变址寻址
存储器	(r_b, r_i, s)	$M[R[r_b]+R[r_i] \cdot s]$	比例变址寻址
存储器	$Imm(r_b, r_i, s)$	$M[Imm+R[r_b]+R[r_i] \cdot s]$	比例变址寻址

图 3-3　操作数格式。操作数可以表示立即数(常数)值、寄存器值或是来自内存的值。比例因子 s 必须是 1、2、4 或者 8

指令	效果	描述
MOV S，D	D←S	传送
movb		传送字节
movw		传送字
movl		传送双字
movq		传送四字
movabsq I，R	R←I	传送绝对的四字

图 3-4　简单的数据传送指令

指令	效果	描述
MOVZ S，R	R←零扩展(S)	以零扩展进行传送
movzbw		将做了零扩展的字节传送到字
movzbl		将做了零扩展的字节传送到双字
movzwl		将做了零扩展的字传送到双字
movzbq		将做了零扩展的字节传送到四字
movzwq		将做了零扩展的字传送到四字

图 3-5 零扩展数据传送指令。这些指令以寄存器或内存地址作为源，以寄存器作为目的

指令	效果	描述
MOVS S，R	R←符号扩展(S)	传送符号扩展的字节
movsbw		将做了符号扩展的字节传送到字
movsbl		将做了符号扩展的字节传送到双字
movswl		将做了符号扩展的字传送到双字
movsbq		将做了符号扩展的字节传送到四字
movswq		将做了符号扩展的字传送到四字
movslq		将做了符号扩展的双字传送到四字
cltq	%rax ←符号扩展(%eax)	把%eax 符号扩展到%rax

图 3-6 符号扩展数据传送指令。MOVS 指令以寄存器或内存地址作为源，以寄存器作为目的。cltq 指令只作用于寄存器%eax 和%rax

指令	效果	描述
pushq S	R[%rsp]←R[%rsp]−8; M[R[%rsp]]←S	将四字压入栈
popq D	D←M[R[%rsp]]; R[%rsp]←R[%rsp]+8	将四字弹出栈

图 3-8　入栈和出栈指令

指令		效果	描述
leaq	S, D	$D \leftarrow \&S$	加载有效地址
INC	D	$D \leftarrow D+1$	加1
DEC	D	$D \leftarrow D-1$	减1
NEG	D	$D \leftarrow -D$	取负
NOT	D	$D \leftarrow \sim D$	取补
ADD	S, D	$D \leftarrow D+S$	加
SUB	S, D	$D \leftarrow D-S$	减
IMUL	S, D	$D \leftarrow D*S$	乘
XOR	S, D	$D \leftarrow D \hat{} S$	异或
OR	S, D	$D \leftarrow D \mid S$	或
AND	S, D	$D \leftarrow D \& S$	与
SAL	k, D	$D \leftarrow D << k$	左移
SHL	k, D	$D \leftarrow D << k$	左移（等同于SAL）
SAR	k, D	$D \leftarrow D >>_A k$	算术右移
SHR	k, D	$D \leftarrow D >>_L k$	逻辑右移

图 3-10 整数算术操作。加载有效地址(leaq)指令通常用来执行简单的算术操作。其余的指令是更加标准的一元或二元操作。我们用$>>_A$和$>>_L$来分别表示算术右移和逻辑右移。注意，这里的操作顺序与 ATT 格式的汇编代码中的相反

HPCL
State Key Laboratory of
High Performance Computing

指令		基于	描述
CMP	S_1，S_2	$S_2 - S_1$	比较
cmpb			比较字节
cmpw			比较字
cmpl			比较双字
cmpq			比较四字
TEST	S_1，S_2	$S_1 \& S_2$	测试
testb			测试字节
testw			测试字
testl			测试双字
testq			测试四字

图 3-13　比较和测试指令。这些指令不修改任何寄存器的值，只设置条件码

HPCL
State Key Laboratory of
High Performance Computing

指令	同义名	跳转条件	描述
jmp *Label*		1	直接跳转
jmp *Operand		1	间接跳转
je *Label*	jz	ZF	相等/零
jne *Label*	jnz	~ZF	不相等/非零
js *Label*		SF	负数
jns *Label*		~SF	非负数
jg *Label*	jnle	~(SF ^ OF) & ~ZF	大于（有符号>）
jge *Label*	jnl	~(SF ^ OF)	大于或等于（有符号>=）
jl *Label*	jnge	SF ^ OF	小于（有符号<）
jle *Label*	jng	(SF ^ OF) | ZF	小于或等于（有符号<=）
ja *Label*	jnbe	~CF & ~ZF	超过（无符号>）
jae *Label*	jnb	~CF	超过或相等（无符号>=）
jb *Label*	jnae	CF	低于（无符号<）
jbe *Label*	jna	CF | ZF	低于或相等（无符号<=）

图 3-15 jump 指令。当跳转条件满足时，这些指令会跳转到一条带标号的目的地。有些指令有“同义名”，也就是同一条机器指令的别名

指令	效果	描述
imulq S mulq S	R[%rdx]：R[%rax]←S×R[%rax] R[%rdx]：R[%rax]←S×R[%rax]	有符号全乘法 无符号全乘法
clto	R[%rdx]：R[%rax]←符号扩展(R[%rax])	转换为八字
idivq S	R[%rdx]←R[%rdx]：R[%rax] mod S R[%rdx]←R[%rdx]：R[%rax]÷S	有符号除法
divq S	R[%rdx]←R[%rdx]：R[%rax] mod S R[%rdx]←R[%rdx]：R[%rax]÷S	无符号除法

图 3-12　特殊的算术操作。这些操作提供了有符号和无符号数的全 128 位乘法和除法。一对寄存器%rdx 和%rax 组成一个 128 位的八字

HPCL
State Key Laboratory of
High Performance Computing

作业提交方式

- 提交内容
  - 你求解得到的属于自己的字符串文件，每行对应一个问题的解
  - 文件编码统一使用UTF-8，否则会有扣分
  - 文件名：<student_id>_<full_name>_input.txt
    - 比如说：付祥，学号13325，则提交文件名为13325_fuxiang_input.txt
- 关于实验报告
  - 用于说明自己在实验过程中遇到的问题、思考和解决方案。
  - 提交自己认为在解题过程中运用到的好的思路和技巧
  - 我们会找时间统一进行讲解，并和大家分享一些好的思路。
- 如何提交：SPOC
- Deadline：待定