CSAPP-期末考试速通版

注意分清 16 进制和 10 进制!! 不要忘记 0x!! 不要忘记 0x!! 不要忘记 0x!! 第二章

- 1、大端法(最高有效字节在前)/小端法(最低有效字节在前)
- 2、逻辑右移(左边为0) 算术右移(整数为0负数为1)
- 3、ieee 标准:符号位s,指数位exp,偏移量bias,尾数位f(尾数要加隐藏值)

规格化:(-1)^(s)*2^(exp-bias)*(1+f)非规格化(-1)^(s)*2^(1 -bias)*f

舍入规则: 向偶数舍入 char1short2int4long8 int 转 double 不会发生舍入 float 转 double 不会发生舍入

GDB lab:

- 1、ir+寄存器名(info register)查看寄存器内容 disas (disassemble)反编译函数
- 2、函数的返回值: %rax,栈指针%rsp,第一个参数%rdi,第二个%rsi,第三个%rdx,第四个%rcx 3.bt(backtrace)可以查看调用栈上活跃的实例

Bomb lab:

- 1、 print(char*)与 x/(数字) d 查看地址内容
- 2、 b设置断点(*地址或者函数名), ib 查看断点,

常用指令 add 右加左 sub 右减左 mov 右=左

条件跳转后缀: ja 无符号大于, jb 无符号小于, jg 有符号大于, jl 有符号小于,je 相等,jz 为 0,n 否定, 可部分组合 js 为负则跳转

条件传送: cmov.后缀处理同上(根据条件判断是否运行)

Sar 算术右移 shr 逻辑右移 sal 算术左移 shl 逻辑左移

数据后缀: q:字节: 1b2w4l(32 位寄存器)8q(64 位寄存器)

movs(左+右)左边扩展到右边(带符号扩展) movz 同理不带符号扩展

cltq 32 位寄存器扩展到 64 位置

访问寄存器例:%rax,访问内存例(%rax)

内存寻址模式:n(寄存器 a,寄存器 b,times):把(a+b*times+n)地址的值存入右边 lea 数字(%rdi)寄存器 2 则是将 rdi+数字存入寄存器 2

Push pop 栈指针+-8 保护数据

调用者保存: %rax、%rex 被调用者保存: %rbp、%rbx、%r12-%r15

数据对齐注意点: 1、结构体嵌套时,两种情况(举例b里面有 a) 1、a 里面所有都比b里面的小/等: a 自己对齐, b 按照最大的对齐/2、a 里面有比b里面大的:按照最大的对齐 2、结构体里面的数组/字符数组合并对齐(比如同一个struct 里 int a 与 char b[2],b[2]一起按照 int 对齐)

test 指令: (做按位与操作,两种情况) 1、左右寄存器相同+je: 比较该寄存器是否为02、左右不同: 做按位与操作,搭配 jnz 比较0

CF 进位标志 (无符号) SF 符号标志 (有符号) ZF 零标志 OF 溢出标志 (有符号)

嵌套数组计算公式: 三维数组 A[d1][d2][d3] 中元素 A[i][i][k] 的地址:

 $LOC(i,j,k)=LOC(0,0,0)+w\times[i\times d2\times d3+j\times d3+k]$, w 为数据类型 跳转表中,如果跳转值等于 0,等效为跳转到第一个

Attack lab

画栈图求解,通过缓冲区溢出更改函数返回地址,使栈指针跳跃执行自行设计的代码/对照表拼好饭代码攻击 缓冲区溢出对抗:栈随机化,栈破坏检测,限制可执行代码区域

Performance lab

优化方法: 代码移动,减少过程调用,消除内存引用,循环展开,多个累计变量,重新结合变换(加括号,称为m*na循环展开)(示例为2*4循环展开)

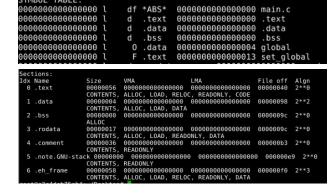
```
void combine(vec *v, data_t *dest) {
    size_t len = v->len;//代码移动
data_t *data = v->data;
    data_t acc0 = IDENT, acc1 = IDENT, acc2 = IDENT, acc3 = IDENT;
    size_t i;
    for (i = 0; i < len-3; i += 4) {//循环展升
                                                       void testcombine(vec *v, data_t *dest) {
        acc0 = acc0 OP data[i]; //m*n循环展开
                                                          long int i:
         acc1 = acc1 OP data[i+1];
                                                           dest = IDENT;
        acc2 = acc2 OP data[i+2];
        acc3 = acc3 OP data[i+3];
                                                          for(i = 0; i < vec_length(v); i++) {</pre>
                                                               data_t val;
                                                               get_vec_element(v, i, &val);//消除过程调用
*dest = *dest OP val; //减少内存引用
    for (; i < len; i++) {
        acc0 = acc0 OP data[i];
    *dest = acc0 OP acc1 OP acc2 OP acc3;
```

Linking lab

gcc-E 预处理-S编译.s(显示类型和源代码)

objdump -t == readelf -s

```
global:
.long 15122
.text
.type set_global, @function
set_global:
.IFBO:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
```



.s 文件

gcc - E 预处理 - S 生成汇编 - C 生成目标文件 - O 生成可执行文件 objdump - O 反汇编

-t 符号表(此处地址:相对于节的偏移量)左:地址 右:大小-h(节属性表(若尚未链接,地址不能确定):

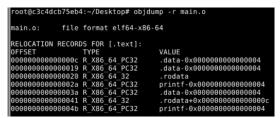
.data 已初始化(不为 0) (编译器为什么会对零初始化变量进行特殊处理,降低EFL 文件大小) 的全局变量和静态变量 .bss 未初始化的全局变量 (可能是COMMON) 和静态变量 (不占实际空间) .text 可执行指令.rodata 只读数据file-off 偏移量 (系统函数的位置为 UND (main 不是系统函数))

CONTENTS 有实际内容、ALLOC 需分配内存、LOAD 加载到内存、RELOC 需重定位、READONLY 只读、CODE 代码段(可执行)、DATA 数据段

-r 重定位表

左: 偏移量 右: 修正值

重定位引用值计算公式:符号地址-(指令地址+offset (偏移 量)) +addend (加数)



链接规则: (强符号:函数和已经初始化且不为0的数据,弱符号反之)1.强符号同名链接报错2.强弱同名选强符号3.弱符号同名任选其一

跨文件使用函数需要先声明;符号在脱离 C 语言后会失去类型信息,浮点型和整型使用不同寄存器,所以程序可以编译,但是运行赋值会失败

Shell lab

常用函数: waitpid, kill, fork, execve, setpgid, sigprocmask, sigsuspend 主流程: 解析命令行-eval 函数与信号处理

pid_t pid 大部分情况,参数中 pid >0 进程=-1 所有进程<-1 取绝对值/进程组kill (pid,信号) 发送信号 常用:SIGKILL SIGCONT SIGTSTP

fork()创建子进程 父进程返回子进程 pid(>0) 子进程返回 0(子进程与父进程输出顺序使用进程图+拓扑排序判断)

execve (路径名 char*, 参数列表 char* argv[],环境列表 char*[] envp) waitpid(pid, (int)&status, option)

1.status: WIFEXITED(status)返回是否退出, WEXITSTATUS(status): 返回子进程的退出状态码 WIFSIGNALED(status): 子进程因未捕获的信号终止时返回真--WTERMSIG(status): 返回导致子进程终止的信号编号 WIFSTOPPED(status): 子进程因信号而停止时返回真 WSTOPSIG(status): 返回导致子进程停止的信号编号 WIFCONTINUED(status): 子进程因收到 SIGCONT 信号恢复执行时返回真 2.Options: WNOHANG 非阻塞等待, 无子进程变化直接返回 0 WUNTRACED 报告被信号停止(不是终止)的子进程 WCONTINUED 报告因收到 SIGCONT 信号恢复执行的子进程

setpgid(pid, pgid)pgid=0 把 pid 进程设为组长>0 加入进程组为 pgid 的组

sigprocmask (,&sigset_t,old &sigset_t) SIG_BLOCK SIG_UNBLOCK SIG_SETMASK

exit 退出进程, 刷新缓冲区

用 sigemptyset、sigfillset、sigaddset、sigdelset 处理 sigset_t Sigsuspend (&sigset t) 等效如下过程的原子化操作

// 伪代码、展示sigsuspend()的等效逻辑(非原子操作)
sigset_t original_mask;

// 1. 保存当前信号接码
sigprocmask(SIG_SETMASK, &new_mask, &original_mask);

// 2. 排起进程,直到收到未被屏蔽的信号
pause();

// 3. 恢复原始信号接码(在信号处理函数返回后)
sigprocmask(SIG_SETMASK, &original_mask, NULL);

Sleep 返回剩下要休眠的秒数

ar-t 库文件名查看目标文件数目