长亭科技
CHAITIN.CN

二进制漏洞挖掘与利用

课时1：从C语言到汇编指令

总体课程目标

- 理解二进制内存漏洞的基本概念
- 了解二进制内存漏洞挖掘的方法
- 掌握Linux下二进制漏洞利用的编写

总体课程大纲(一)

01. 从C语言到汇编指令
02. 调用约定与ELF
03. 延迟绑定与GOT表劫持
04. 软件逆向工程
05. 漏洞类型与挖掘技术
06. 栈溢出与Shellcode
07. 从Return to libc到ROP

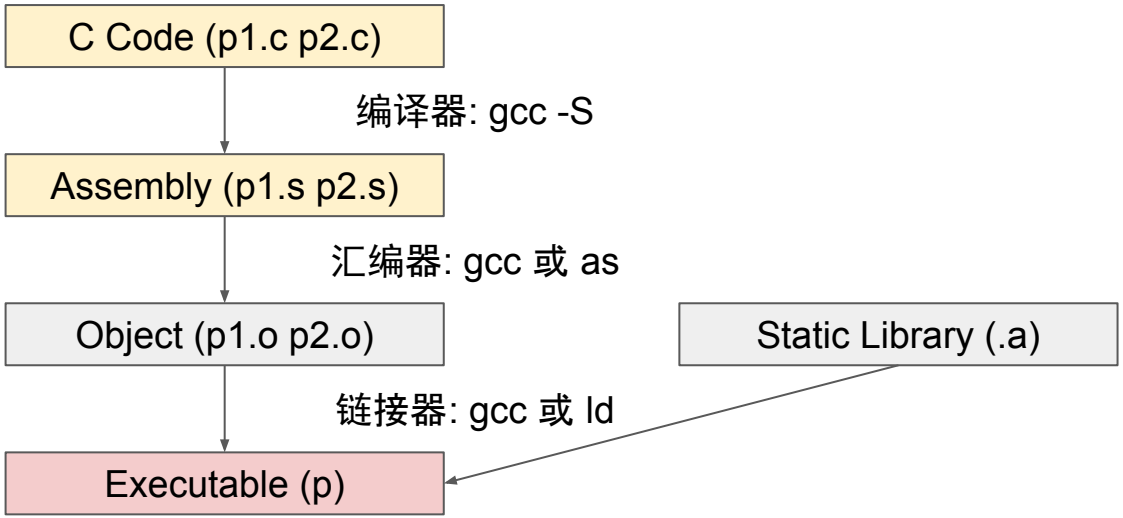
总体课程大纲(二)

08. 实战ROP
09. 格式化字符串
10. GLibc堆的实现
11. 堆溢出利用技术
12. Off-by-one与UAF
13. 实战Ghost漏洞——CVE-2015-0235
14. Linux内核漏洞利用技术

课时大纲

- 从C语言到汇编指令
  - 编译与链接
  - 机器是如何执行指令的
  - 栈
  - x86寻址模式与指令介绍
  - Intel语法与AT&T语法

编译与链接


C Code (p1.c p2.c)
编译器: gcc -S
Assembly (p1.s p2.s)
汇编器: gcc 或 as
Object (p1.o p2.o)
Static Library (.a)
链接器: gcc 或 ld
Executable (p)

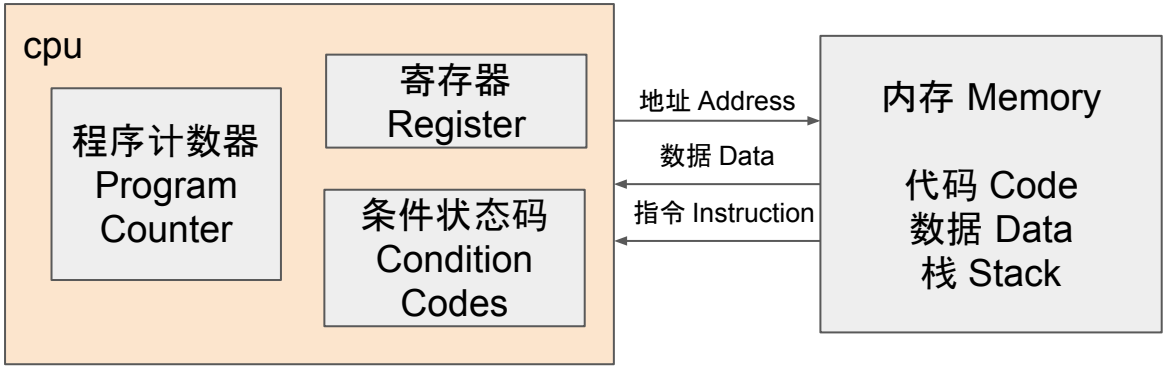
编译: 从C语言到汇编指令

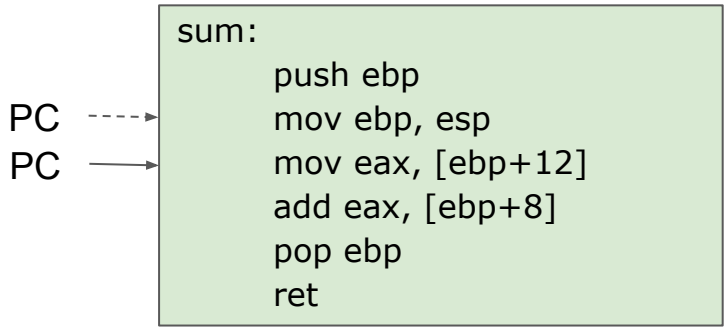

```
int sum(int x, int y)
{
      int t = x + y;
      return t;
}
```



```
sum:
      push ebp
      mov ebp, esp
      mov eax, [ebp+12]
      add eax, [ebp+8]
      pop ebp
      ret
```

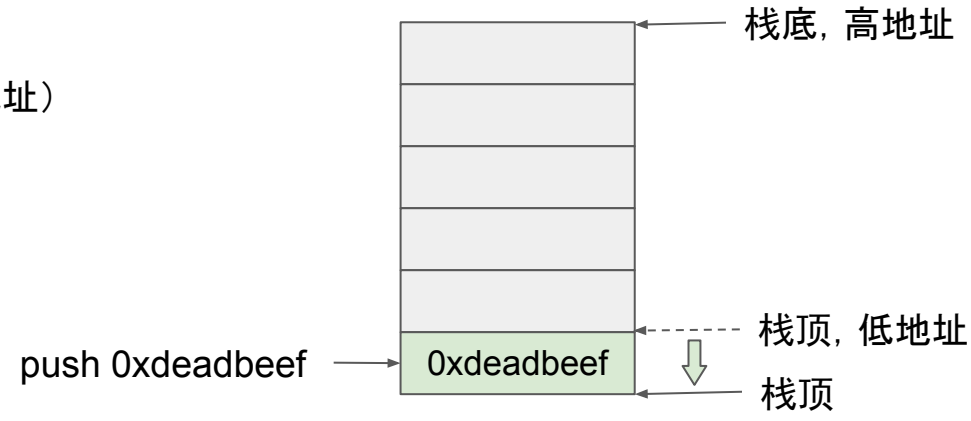
机器指令是如何执行的？





栈

- 一种先进后出的数据结构
- 被用于函数的局部内存管理
  - 保存局部变量
  - 保存函数的调用信息(例如返回地址)
- 栈往低地址方向增长
- esp寄存器永远指向栈顶
- 栈操作
  - 入栈，Push: esp = esp - 4
  - 出栈，Pop: esp = esp + 4



数据转移指令

- mov
- push
- pop
- lea

寻址模式

- 立即数寻址（Immediate Addressing）
  - 操作数包含在指令中，紧跟在操作码之后，作为指令的一部分
  - 举例
    - mov al, 5
    - mov eax, 1000h
- 寄存器寻址（Register Addressing）
  - 操作数在寄存器中，指令指定寄存器
  - 举例
    - mov ax, bx
    - mov ebp, esp

寻址模式

- 直接内存寻址（Direct/Displacement Only Addressing）
  - 操作数在内存中，指令直接指定内存地址
  - 举例
    - mov ax, [2000h]
- 寄存器间接寻址（Register Indirect Addressing）
  - 操作数在内存中，操作数的地址在寄存器中
  - 举例
    - mov eax, [ebx]

寻址模式

- 索引寻址（Indexed Addressing）
  - 通过基址寄存器内容加上一个索引 值来寻址内存中的数据
  - 举例
    - mov ax, [di+100h]
- 相对基址索引寻址（Based Indexed Addressing）
  - 用一个基址寄存器加上一个 变址寄存器的内容再加上一个偏移量来完成内容 单元的寻址
  - 举例
    - mov dh, [bx+si+10h]

寻址模式

- 比例寻址变址
  - 通过基址寄存器的内容加上 变址寄存器的内容与一个比例因子的乘 积来寻址内存中的数据
  - 举例
    - mov eax, [ebx+4*ecx]

mov

- 语法
  - mov <reg>,<reg>
  - mov <reg>,<mem>
  - mov <mem>,<reg>
  - mov <reg>,<const>
  - mov <mem>,<const>
- 举例
  - mov eax, ebx
  - mov byte ptr [var], 5

不同的寻址方式

- 举例
  - mov eax, [ebx]
  - mov [var], ebx
  - mov eax, [esi-4]
  - mov [esi+eax], cl
  - mov edx, [esi+4*ebx]

push

- 语法
  - push <reg32> == sub esp, 4; mov [esp], <reg32>
  - push <mem>
  - push <con32>
- 举例
  - push eax
  - push [var]

pop

- 语法
  - pop <reg32>
  - pop <mem>
- 举例
  - pop edi
  - pop [ebx]

lea - 加载有效地址（Load Effective Address）

- 语法
  - lea <reg32>,<mem>
- 举例
  - lea eax, [var] — 将地址var放入寄存器eax中
  - lea edi, [ebx+4*esi] — edi = ebx+4*esi
    - 某些编译器会使用lea指令来进行算术运算，因为速度更快

算数与逻辑指令

- add/sub
- inc/dec
- imul/idiv
- and/or/xor
- not/neg
- shl/shr

控制转移指令

- jmp - 无条件跳转
- j[condition] - 条件跳转
- cmp - 比较
- call/ret - 函数调用/函数返回

Intel语法与 AT&T语法

Intel	AT&T
`mov eax, 8`	`movl $8, %eax`
`mov ebx, 0ffffh`	`movl $0xffff, %ebx`
`int 80h`	`int $80`
`mov eax, [ecx]`	`movl (%ecx), %eax`

```
sum:
      push ebp
      mov ebp, esp
      mov eax, [ebp+12]
      add eax, [ebp+8]
      pop ebp
      retn
```

```
sum:
      pushl %ebp
      movl %esp,%ebp
      movl 12(%ebp),%eax
      addl 8(%ebp),%eax
      popl %ebp
      ret
```