恶意代码分析

第一章:编译、链接与运行

总体课程目标

- 掌握二进制程序的分析方法
- 理解二进制内存漏洞的基本概念
- 了解二进制内存漏洞的挖掘的方法
- 掌握Linux下二进制漏洞利用的编写

总体课程大纲

第一章：编译、链接与运行
第二章：漏洞类型与挖掘技术
第三章：栈溢出、Shellcode与ROP
第四章：堆溢出利用技术

课程实验与考试

- 实验一：逆向分析（5分）
- 实验二：栈溢出、Shellcode与ROP（10分）
- 实验三：堆溢出利用技术（5分）
- 闭卷考试：（50分）
- 平时表现（5-10分）
- 项目课（20-25分）

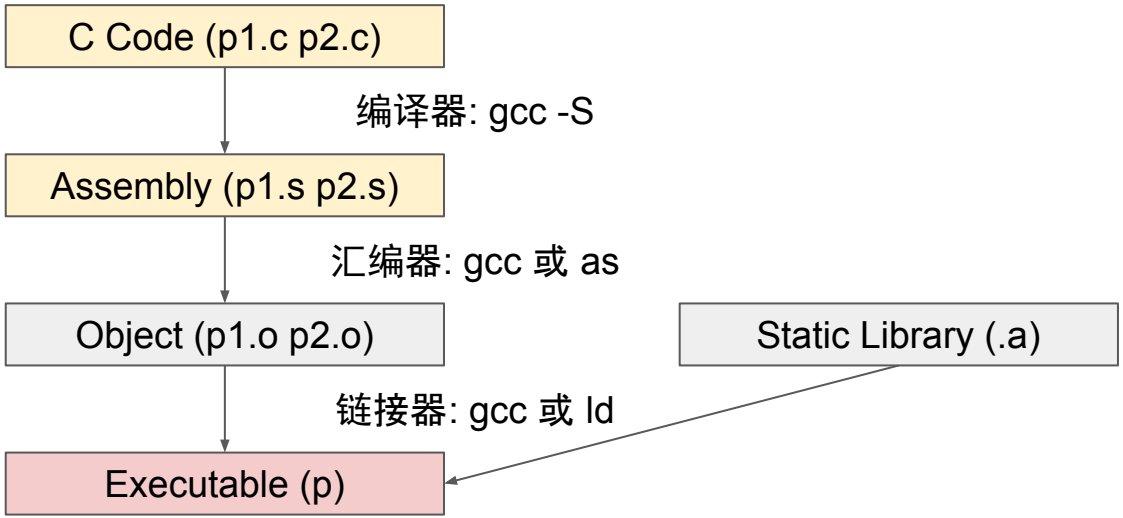
章节大纲

- 第1节：从C语言到汇编指令
- 第2节：调用约定与ELF文件格式
- 第3节：延迟绑定与GOT表劫持
- 第4节：软件逆向工程

第1节: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

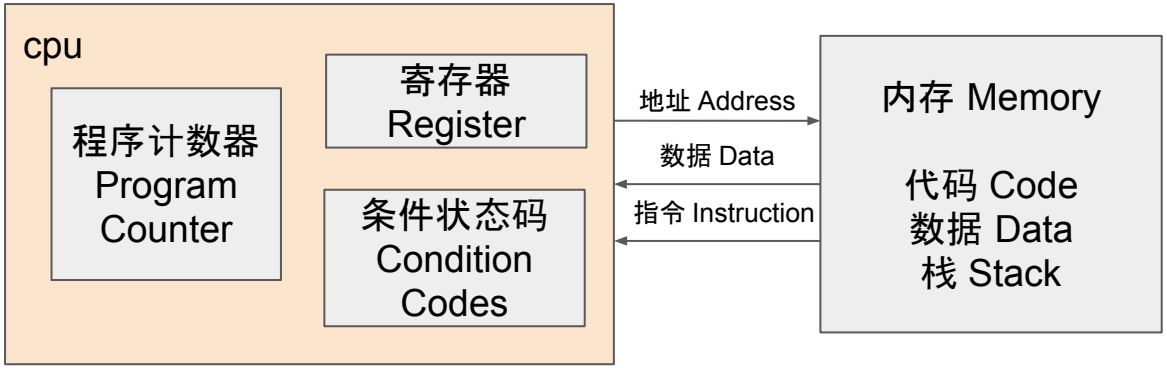
```
int sum(int x, int y)
{
      int t = x + y;
      return t;
}
```

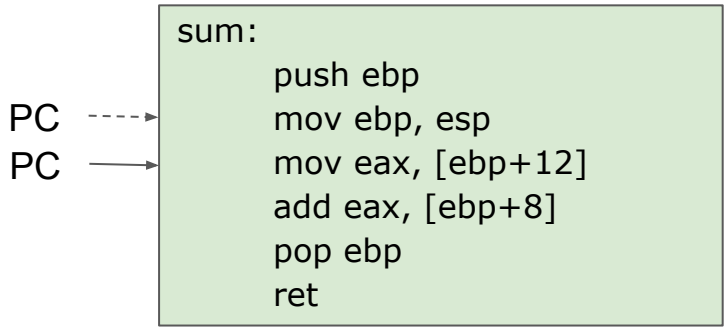


```
sum:
      push ebp
      mov ebp, esp
      mov eax, [ebp+12]
      add eax, [ebp+8]
      pop ebp
      ret
```

机器指令是如何执行的？

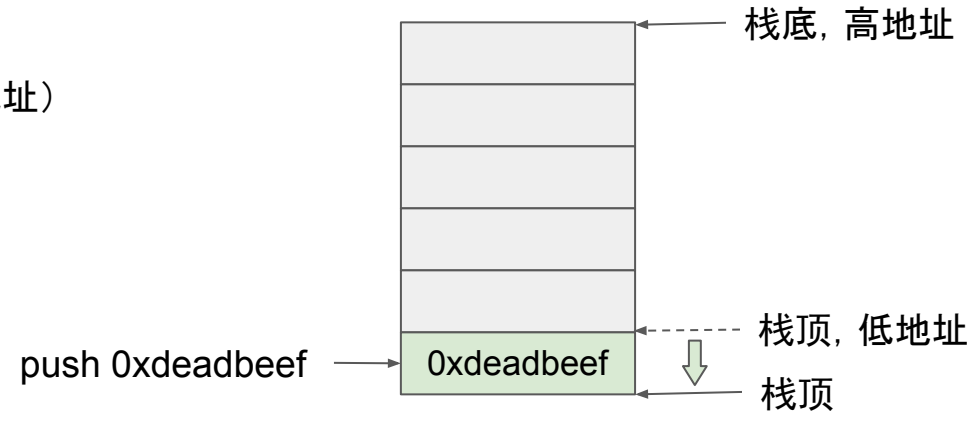
长亭科技
CHAITIN.CN





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
```

第2节:调用约定与ELF

- 什么是调用约定
- 调用约定 cdecl
- Linux 进程空间内存布局
- ELF 文件格式
- ELF 程序的启动过程

调用约定

- 什么是调用约定?
  - 实现层面(底层)的规范
  - 约定了函数之间如何传递参数
  - 约定了函数如何传递返回值
- 常见 x86 调用约定
  - 调用者负责清理栈上的参数(Caller Clean-up)
    - cdecl
    - optlink
  - 被调者负责清理栈上的参数(Callee Clean-up)
    - stdcall
    - fastcall

调用约定 cdecl（x86，32位）

```
int callee(int a, int b, int c) {
    return a + b + c;
}

int caller(void) {
    int ret;

    ret = callee(1, 2, 3);
    ret += 4;
    return ret;
}
```

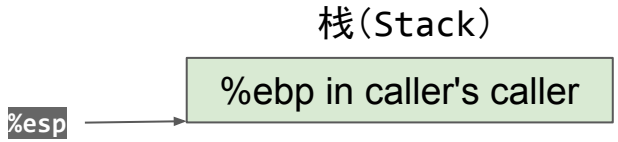
```
00000012 <caller>:
  12:   55                push   %ebp
  13:   89 e5             mov    %esp,%ebp
  15:   83 ec 10          sub    $0x10,%esp
  18:   6a 03             push   $0x3
  1a:   6a 02             push   $0x2
  1c:   6a 01             push   $0x1
  1e:   e8 fc ff ff ff    call   1f <caller+0xd>
  23:   83 c4 0c          add    $0xc,%esp
  26:   89 45 fc          mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04       addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc          mov    -0x4(%ebp),%eax
  30:   c9                leave
  31:   c3                ret

00000000 <callee>:
   0:   55           push   %ebp
   1:   89 e5        mov    %esp,%ebp
   3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c     mov    0xc(%ebp),%eax
   9:   01 c2        add    %eax,%edx
   b:   8b 45 10     mov    0x10(%ebp),%eax
   e:   01 d0        add    %edx,%eax
  10:   5d           pop    %ebp
  11:   c3           ret
```

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
%eip →
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <caller+0xd>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55          push   %ebp
   1:   89 e5       mov    %esp,%ebp
   3:   8b 55 08    mov    0x8(%ebp),%edx
   6:   8b 45 0c    mov    0xc(%ebp),%eax
   9:   01 c2       add    %eax,%edx
   b:   8b 45 10    mov    0x10(%ebp),%eax
   e:   01 d0       add    %edx,%eax
  10:   5d          pop    %ebp
  11:   c3          ret
```

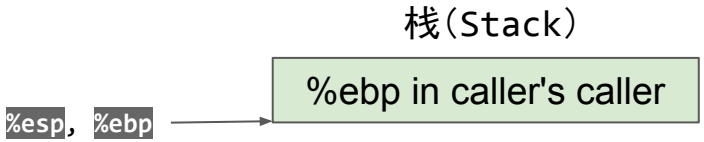


在栈上保存栈帧寄存器%ebp

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
%eip → 15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <caller+0xd>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55           push   %ebp
   1:   89 e5        mov    %esp,%ebp
   3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c     mov    0xc(%ebp),%eax
   9:   01 c2        add    %eax,%edx
   b:   8b 45 10     mov    0x10(%ebp),%eax
   e:   01 d0        add    %edx,%eax
  10:   5d           pop    %ebp
  11:   c3           ret
```

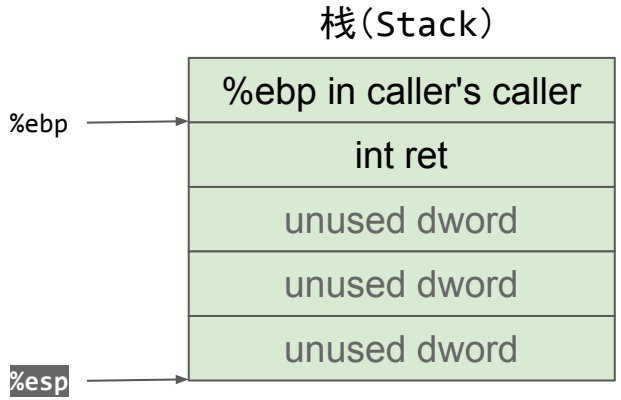


将当前%esp存入%ebp

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
%eip → 18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <caller+0xd>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55          push   %ebp
   1:   89 e5       mov    %esp,%ebp
   3:   8b 55 08    mov    0x8(%ebp),%edx
   6:   8b 45 0c    mov    0xc(%ebp),%eax
   9:   01 c2       add    %eax,%edx
   b:   8b 45 10    mov    0x10(%ebp),%eax
   e:   01 d0       add    %edx,%eax
  10:   5d          pop    %ebp
  11:   c3          ret
```

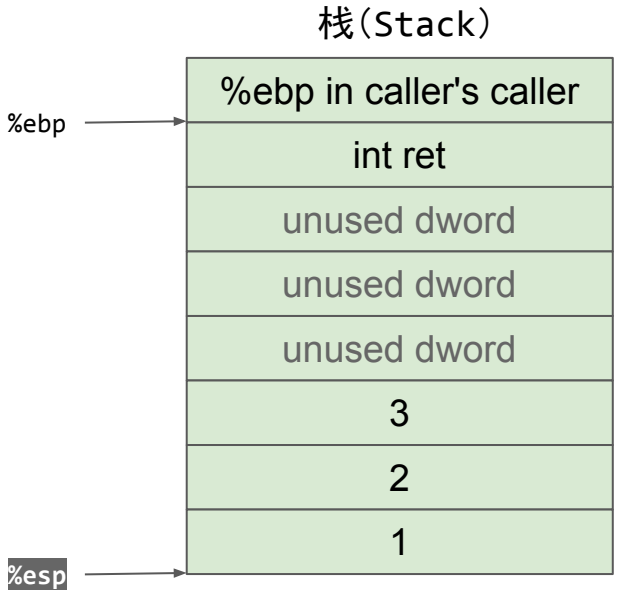


在栈上为局部变量开辟空间

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
%eip ->1e:   e8 fc ff ff ff   call   1f <caller+0xd>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55          push   %ebp
   1:   89 e5       mov    %esp,%ebp
   3:   8b 55 08    mov    0x8(%ebp),%edx
   6:   8b 45 0c    mov    0xc(%ebp),%eax
   9:   01 c2       add    %eax,%edx
   b:   8b 45 10    mov    0x10(%ebp),%eax
   e:   01 d0       add    %edx,%eax
  10:   5d          pop    %ebp
  11:   c3          ret
```

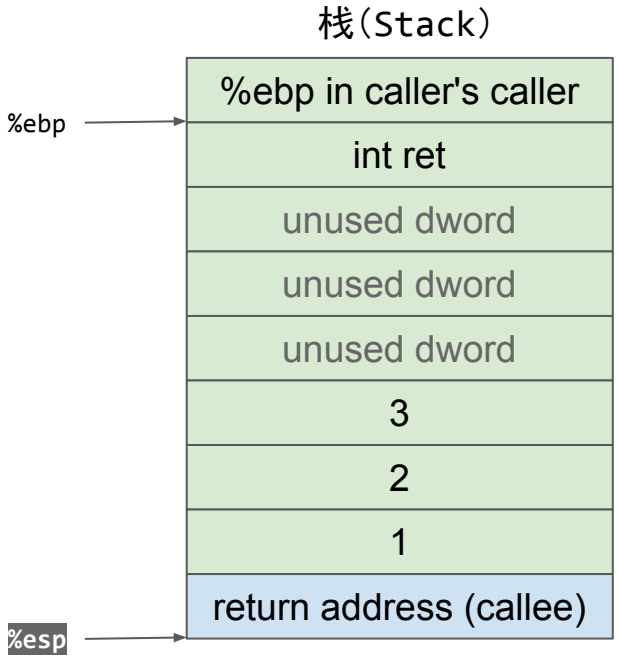


往栈上push传入callee()的参数

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
%eip →  0:   55           push   %ebp
   1:   89 e5        mov    %esp,%ebp
   3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c     mov    0xc(%ebp),%eax
   9:   01 c2        add    %eax,%edx
   b:   8b 45 10     mov    0x10(%ebp),%eax
   e:   01 d0        add    %edx,%eax
  10:   5d           pop    %ebp
  11:   c3           ret
```

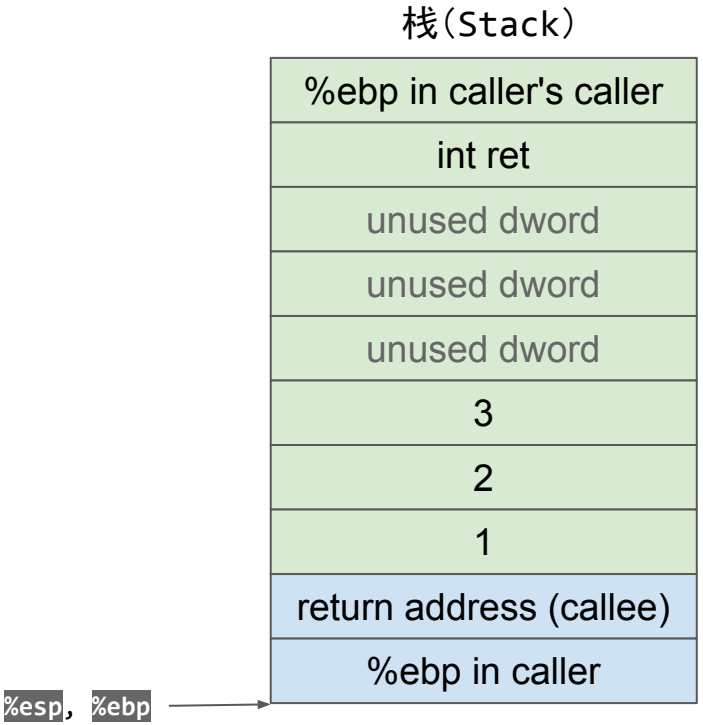


调用callee()，在栈上保存返回地址

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55              push   %ebp
   1:   89 e5           mov    %esp,%ebp
%eip → 3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c        mov    0xc(%ebp),%eax
   9:   01 c2           add    %eax,%edx
   b:   8b 45 10        mov    0x10(%ebp),%eax
   e:   01 d0           add    %edx,%eax
  10:   5d              pop    %ebp
  11:   c3              ret
```

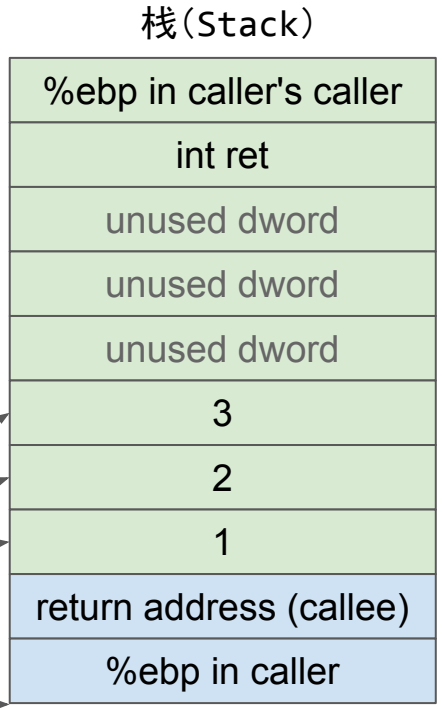


进入callee()，同样保存%ebp和%esp

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55              push   %ebp
   1:   89 e5           mov    %esp,%ebp
   3:   8b 55 08        mov    0x8(%ebp),%edx
   6:   8b 45 0c        mov    0xc(%ebp),%eax
   9:   01 c2           add    %eax,%edx
   b:   8b 45 10        mov    0x10(%ebp),%eax
   e:   01 d0           add    %edx,%eax
%eip --> 10:   5d       pop    %ebp
  11:   c3              ret
```

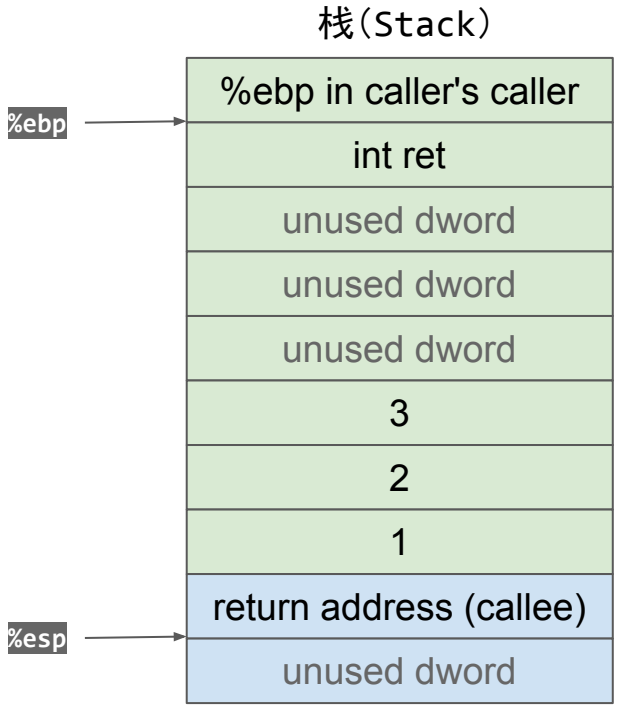


通过%ebp找到参数相加，结果存入%eax

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55          push   %ebp
   1:   89 e5       mov    %esp,%ebp
   3:   8b 55 08    mov    0x8(%ebp),%edx
   6:   8b 45 0c    mov    0xc(%ebp),%eax
   9:   01 c2       add    %eax,%edx
   b:   8b 45 10    mov    0x10(%ebp),%eax
   e:   01 d0       add    %edx,%eax
  10:   5d          pop    %ebp
%eip → 11:   c3          ret
```

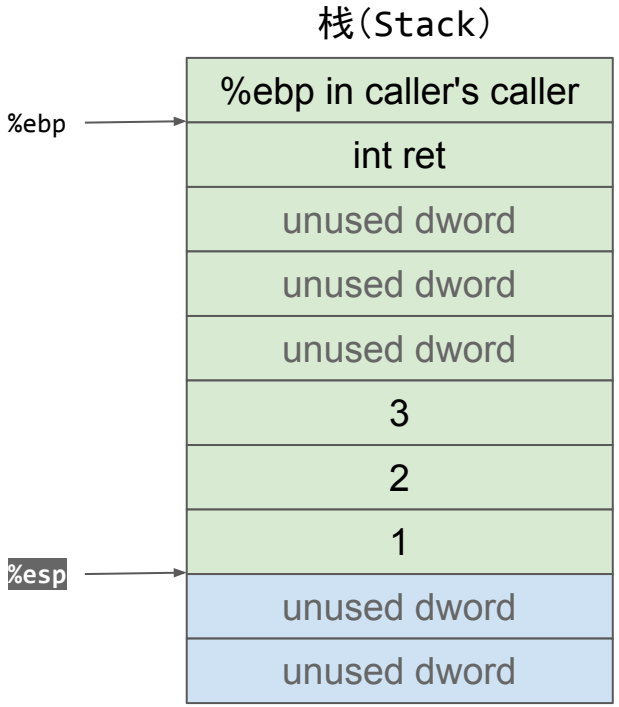


通过栈上保存的值恢复%ebp

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
%eip→ 23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55           push   %ebp
   1:   89 e5        mov    %esp,%ebp
   3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c     mov    0xc(%ebp),%eax
   9:   01 c2        add    %eax,%edx
   b:   8b 45 10     mov    0x10(%ebp),%eax
   e:   01 d0        add    %edx,%eax
  10:   5d           pop    %ebp
  11:   c3           ret
```



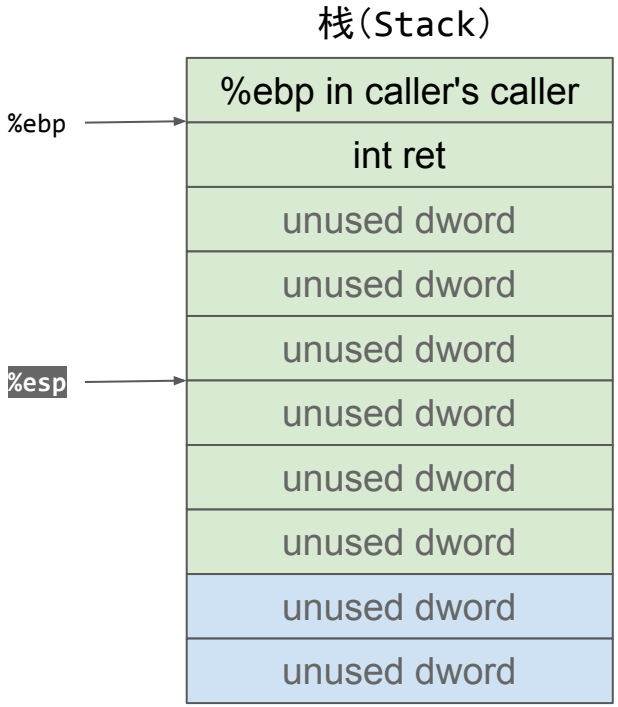
通过栈上的返回地址，返回 caller函数

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55          push   %ebp
   1:   89 e5       mov    %esp,%ebp
   3:   8b 55 08    mov    0x8(%ebp),%edx
   6:   8b 45 0c    mov    0xc(%ebp),%eax
   9:   01 c2       add    %eax,%edx
   b:   8b 45 10    mov    0x10(%ebp),%eax
   e:   01 d0       add    %edx,%eax
  10:   5d          pop    %ebp
  11:   c3          ret
```

%eip →



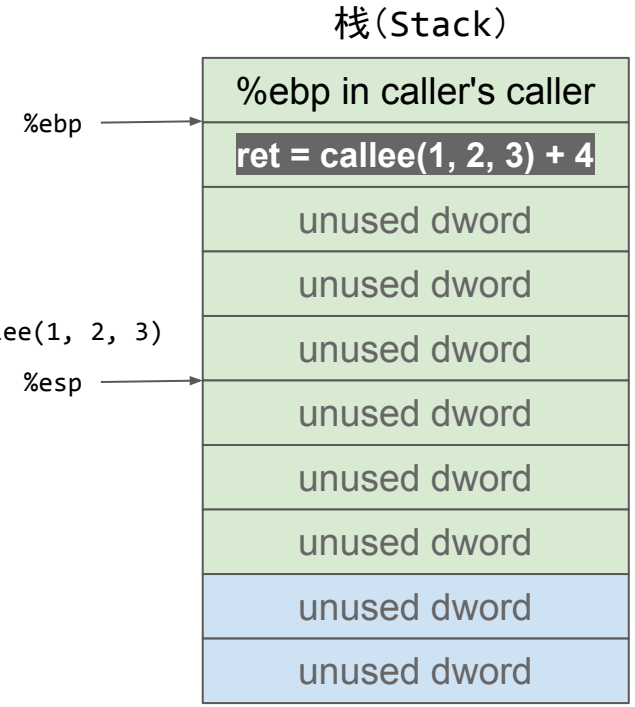
清理调用callee()时push在栈上的参数

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
%eip → 30:   c9         leave
  31:   c3              ret

00000000 <callee>:
   0:   55           push   %ebp
   1:   89 e5        mov    %esp,%ebp
   3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c     mov    0xc(%ebp),%eax
   9:   01 c2        add    %eax,%edx
   b:   8b 45 10     mov    0x10(%ebp),%eax
   e:   01 d0        add    %edx,%eax
  10:   5d           pop    %ebp
  11:   c3           ret
```

%eax = callee(1, 2, 3)



callee()返回结果与4相加，
存入栈上局部变量

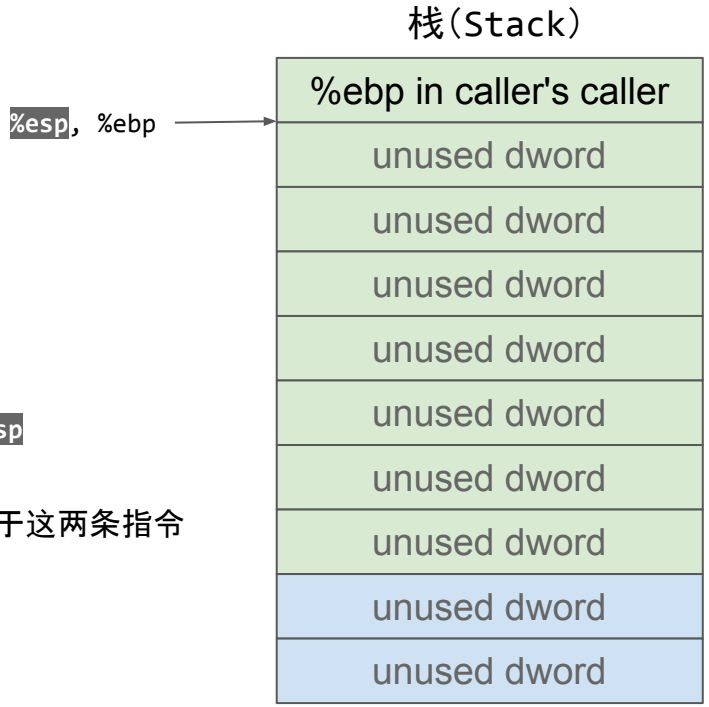
调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
%eip→ 31:   c3              ret
```

```
mov   %ebp, $esp
pop   %ebp
```

leave指令等价于这两条指令

```
00000000 <callee>:
   0:   55           push   %ebp
   1:   89 e5        mov    %esp,%ebp
   3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c     mov    0xc(%ebp),%eax
   9:   01 c2        add    %eax,%edx
   b:   8b 45 10     mov    0x10(%ebp),%eax
   e:   01 d0        add    %edx,%eax
  10:   5d           pop    %ebp
  11:   c3           ret
```

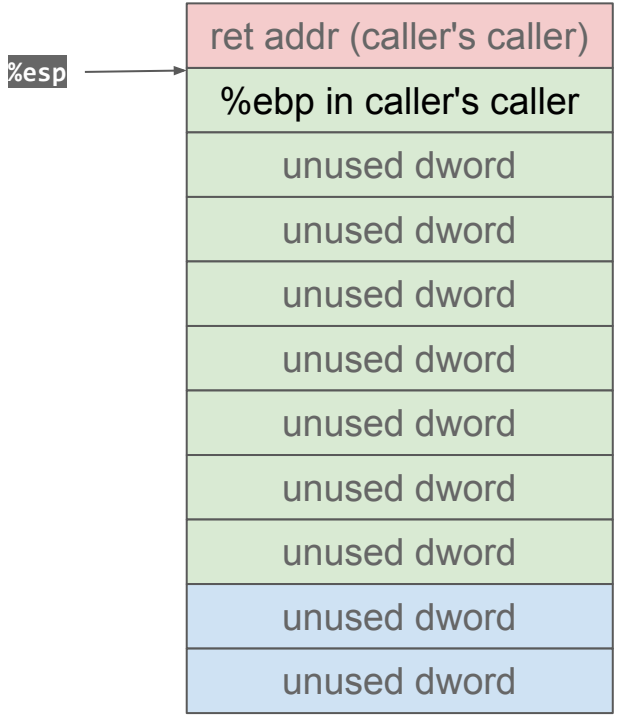


通过%ebp恢复%esp

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave                    mov   %ebp, $esp
%eip ->  31:   c3              ret                      pop   %ebp

00000000 <callee>:
   0:   55           push   %ebp
   1:   89 e5        mov    %esp,%ebp
   3:   8b 55 08     mov    0x8(%ebp),%edx
   6:   8b 45 0c     mov    0xc(%ebp),%eax
   9:   01 c2        add    %eax,%edx
   b:   8b 45 10     mov    0x10(%ebp),%eax
   e:   01 d0        add    %edx,%eax
  10:   5d           pop    %ebp
  11:   c3           ret
```

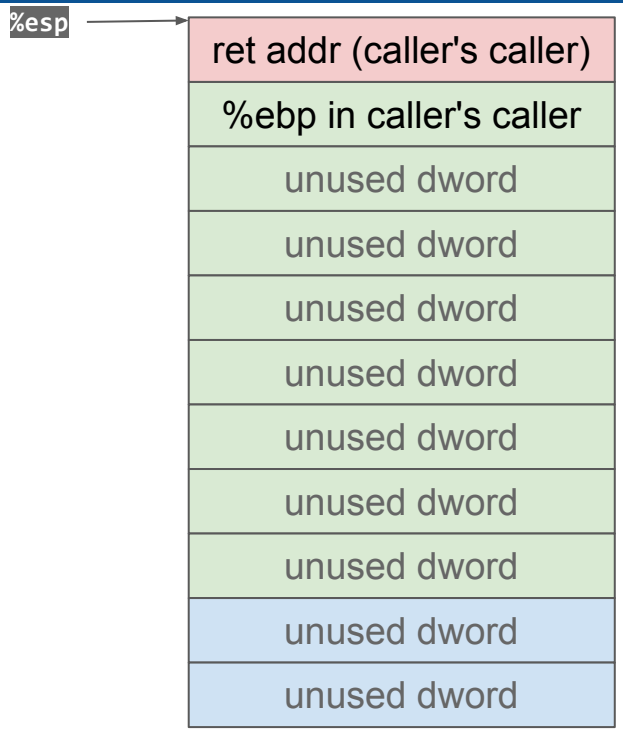


通过栈上保存的值恢复%ebp

调用约定 cdecl（x86，32位）

```
00000012 <caller>:
  12:   55              push   %ebp
  13:   89 e5           mov    %esp,%ebp
  15:   83 ec 10        sub    $0x10,%esp
  18:   6a 03           push   $0x3
  1a:   6a 02           push   $0x2
  1c:   6a 01           push   $0x1
  1e:   e8 fc ff ff ff  call   1f <callee>
  23:   83 c4 0c        add    $0xc,%esp
  26:   89 45 fc        mov    %eax,-0x4(%ebp)
  29:   83 45 fc 04     addl   $0x4,-0x4(%ebp)
  2d:   8b 45 fc        mov    -0x4(%ebp),%eax
  30:   c9              leave
  31:   c3              ret

00000000 <callee>:
   0:   55          push   %ebp
   1:   89 e5       mov    %esp,%ebp
   3:   8b 55 08    mov    0x8(%ebp),%edx
   6:   8b 45 0c    mov    0xc(%ebp),%eax
   9:   01 c2       add    %eax,%edx
   b:   8b 45 10    mov    0x10(%ebp),%eax
   e:   01 d0       add    %edx,%eax
  10:   5d          pop    %ebp
  11:   c3          ret
```

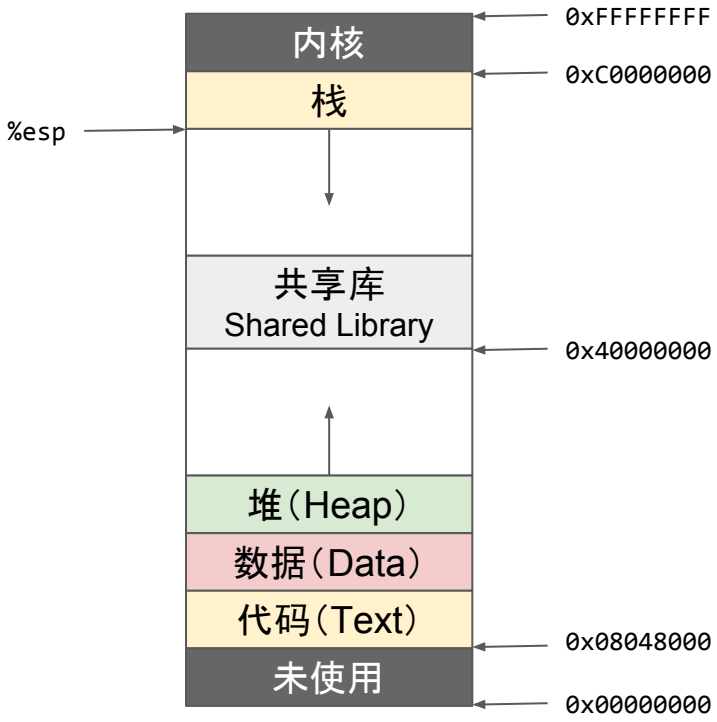


caller()执行完毕，返回上层

调用约定 cdecl

- x86（32位）cdecl 调用约定
  - 用栈来传递参数
  - 用寄存器$eax来保存返回值
- amd64（64位）cdecl 调用约定
  - 使用寄存器 %rdi, %rsi, %rdx, %rcx, %r8, %r9 来传递前6个参数
  - 第七个及以上的参数通过栈来传递
- 栈桢指针 %ebp (%rbp) 的用途
  - 索引栈上的参数（例如x86下，%ebp + 8指向第一个参数）
  - 保存栈顶位置 %esp (%rsp)

进程空间内存布局（Linux x86）



内存空间中的栈桢（Stack Frame）