长亭科技
CHAITIN.CN

二进制漏洞挖掘与利用

课时2：调用约定与ELF

课时大纲

长亭科技 CHAITIN.CN

- 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

长亭科技
CHAITIN.CN

- 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）

```c
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）

长亭科技 CHAITIN.CN

```
          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
```

栈（Stack）

%esp → | %ebp in caller's caller |

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
  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
```

%eip → 15

```
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
```

栈（Stack）

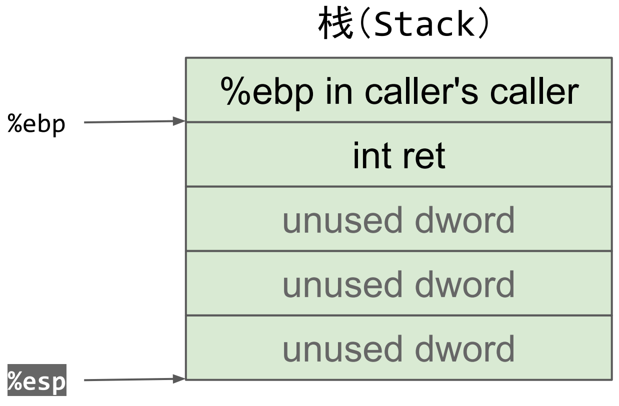
%esp，%ebp → %ebp in caller's caller

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

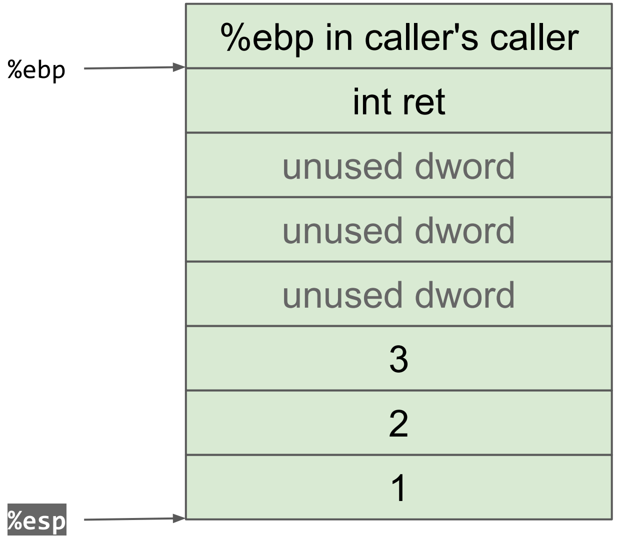
调用约定 cdecl（x86，32位）

```
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
  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
```

%eip

栈(Stack)

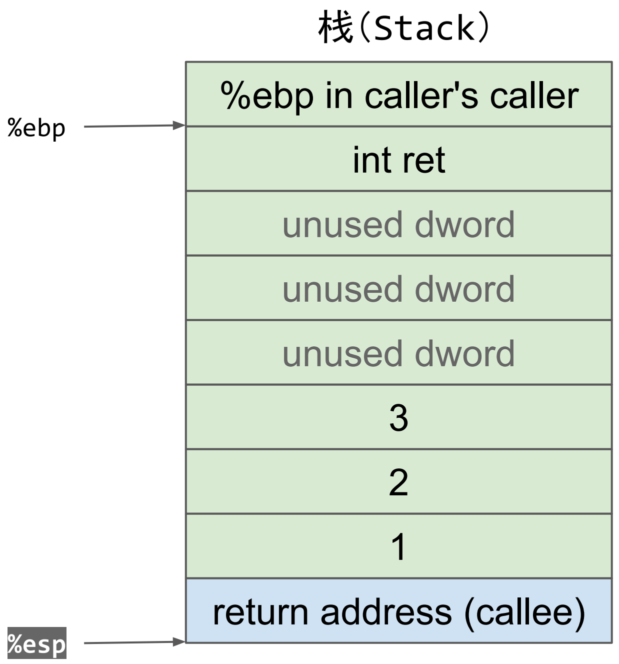


往栈上push传入callee()的参数

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）

长亭科技 CHAITIN.CN

```
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
```

栈（Stack）

%ebp in caller's caller
int ret
unused dword
unused dword
unused dword
3
2
1
return address (callee)
%ebp in caller

%esp，%ebp →

进入callee()，同样保存%ebp和%esp

调用约定 cdecl（x86，32位）

长亭科技 CHAITIN.CN

```
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
```

栈（Stack）

%ebp in caller's caller
int ret
unused dword
unused dword
unused dword
3
2
1
return address (callee)
%ebp in caller

%eax = 1 + 2 + 3

%esp, %ebp →

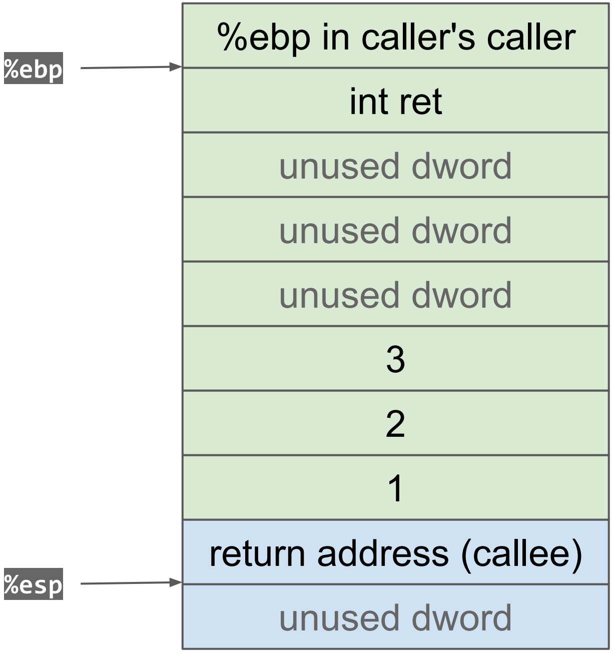
通过%ebp找到参数相加，结果存入%eax

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
```

栈（Stack）



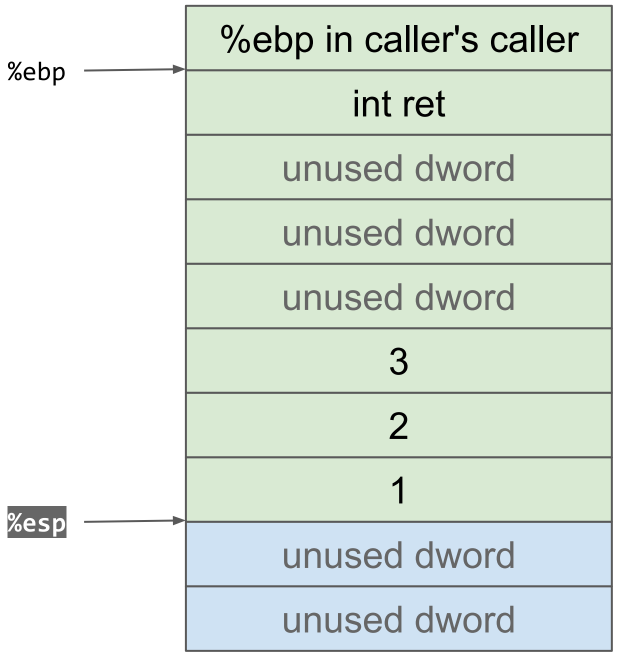
通过栈上保存的值恢复%ebp

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
```

栈(Stack)



通过栈上的返回地址，返回caller函数

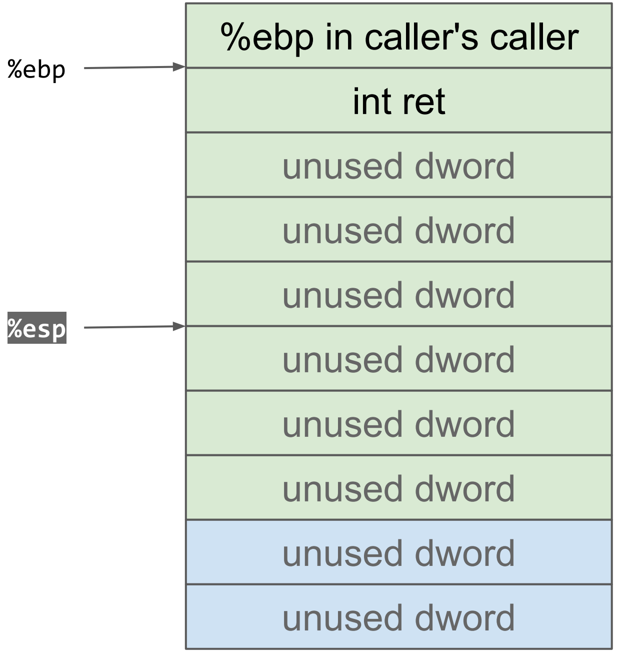
调用约定 cdecl（x86，32位）

长亭科技 CHAITIN.CN

```
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
%eip  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
```

栈（Stack）



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)

栈（Stack）

	%ebp in caller's caller
%ebp →	ret = callee(1, 2, 3) + 4
	unused dword
	unused dword
	unused dword
%esp →	unused dword
	unused dword
	unused dword
	unused dword
	unused dword

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）

长亭科技 CHAITIN.CN

```
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
```

栈（Stack）

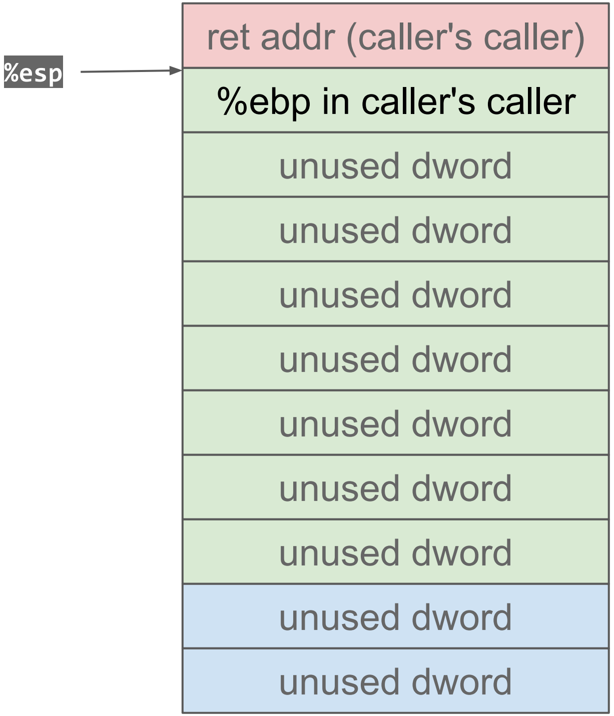
	栈（Stack）
%esp, %ebp →	%ebp in caller's caller
	unused dword
	unused dword
	unused dword
	unused dword
	unused dword
	unused dword
	unused dword
	unused dword
	unused dword

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
```

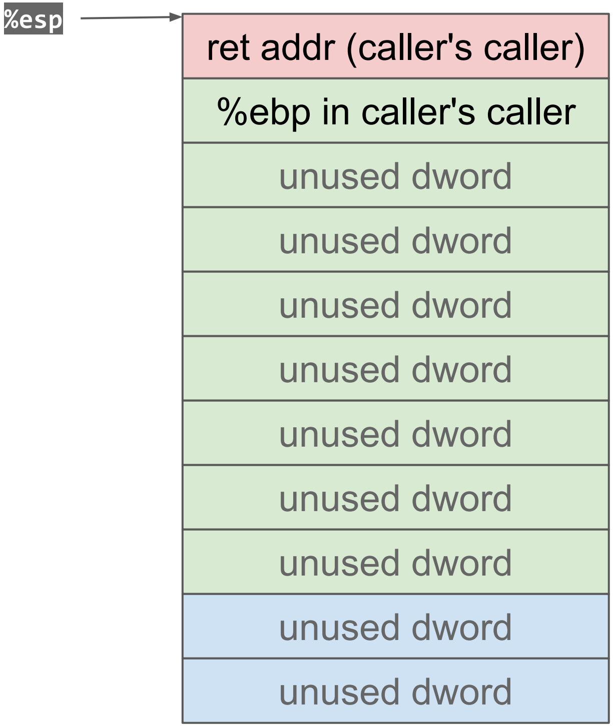


通过栈上保存的值恢复%ebp

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
```

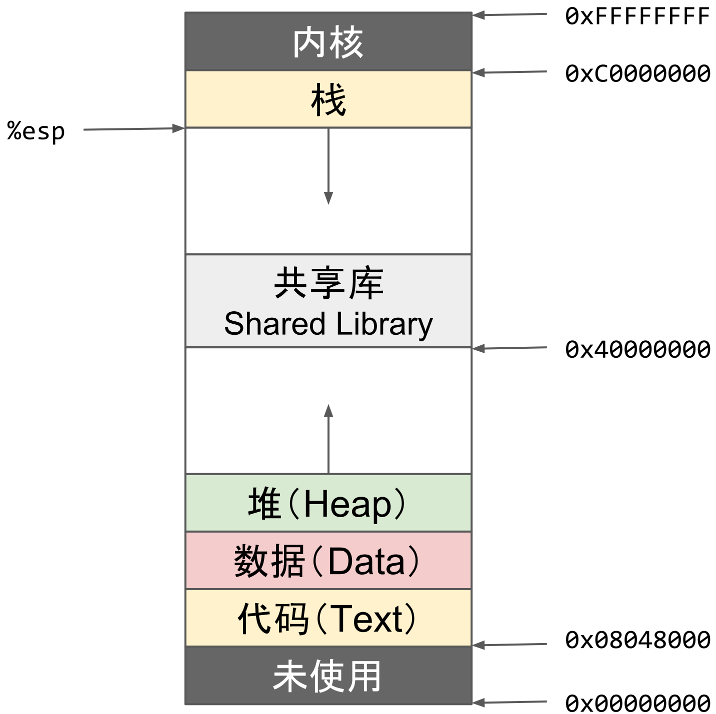


caller()执行完毕，返回上层

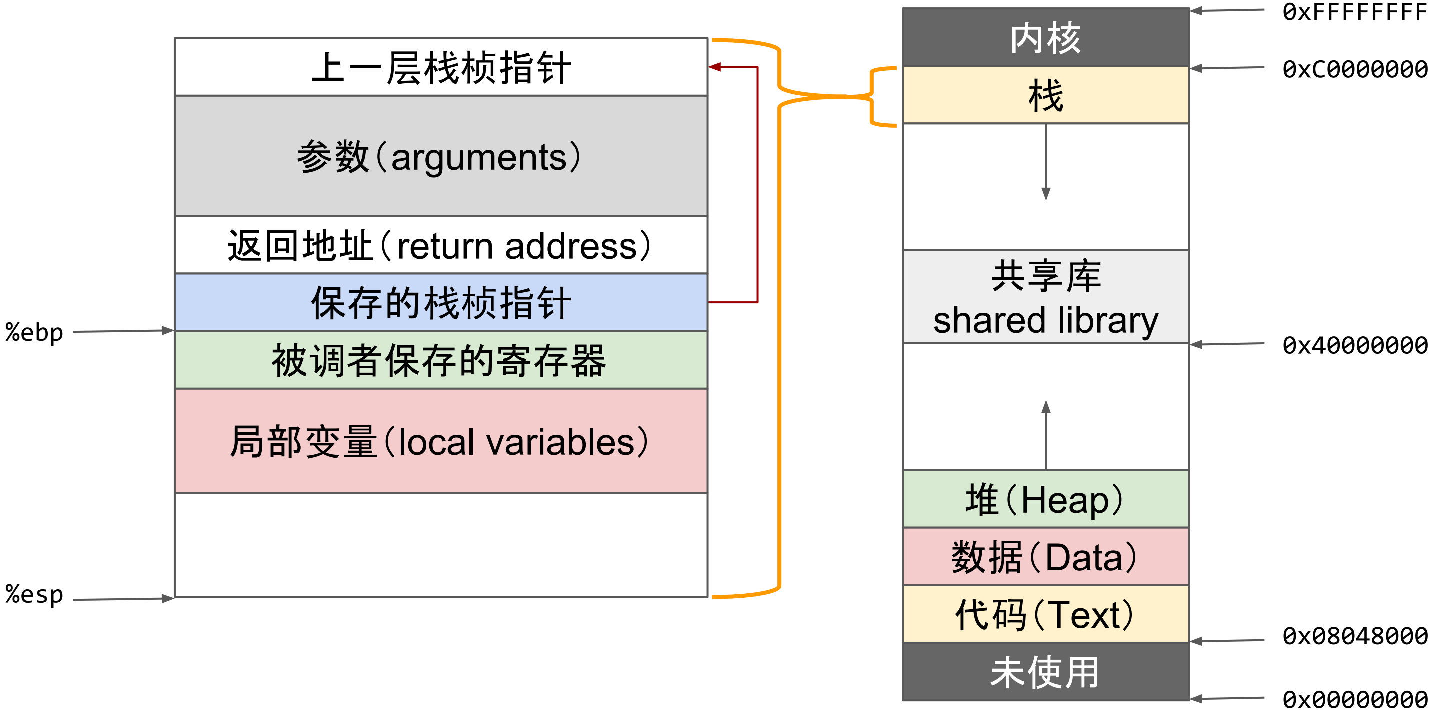
调用约定 cdecl

- 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)



ELF文件格式

- ELF：Executable and Linkable Format
- 一种Linux下常用的可执行文件、对象、共享库的标准文件格式
- 还有许多其他可执行文件格式：PE、Mach-O、COFF、COM...
- 内核中处理ELF相关代码参考：fs/binfmt_elf.c
- ELF中的数据按照Segment和Section两个概念进行划分

Segment与Section

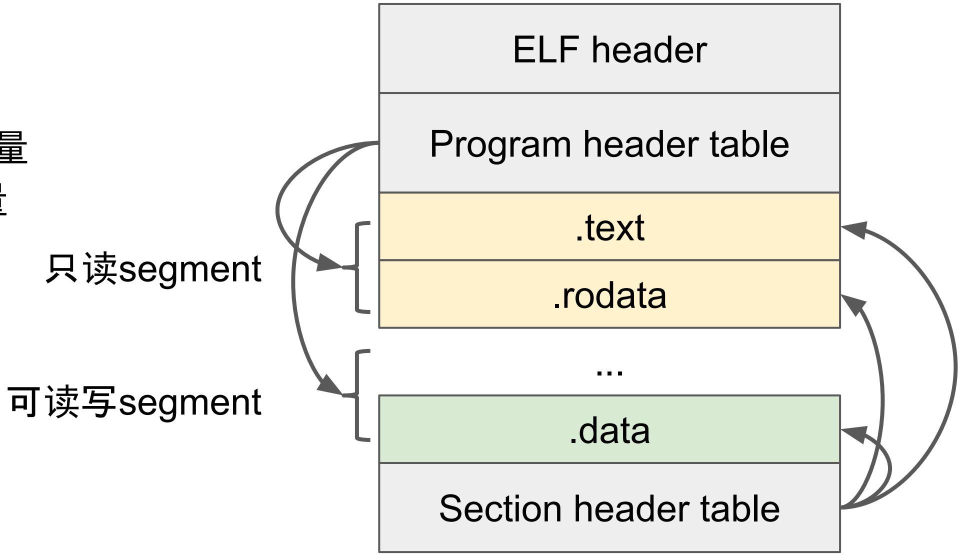
- Segment
  - 用于告诉内核，在执行ELF文件时应该如何映射内存
  - 每个Segment主要包含加载地址、文件中的范围、内存权限、对齐方式等信息
  - 是运行时必须提供的信息
- Section
  - 用于告诉链接器，ELF中每个部分是什么，哪里是代码，哪里是只读数据，哪里是重定位信息
  - 每个Section主要包含Section类型、文件中的位置、大小等信息
  - 链接器依赖Section信息将不同的对象文件的代码、数据信息合并，并修复互相引用
- Segment与Section的关系
  - 相同权限的Section会放入同一个Segment，例如.text和.rodata section
  - 一个Segment包含许多Section，一个Section可以属于多个Segment

ELF文件类型

- 可执行文件（ET_EXEC）
  - 可直接运行的程序，必 须包含segment
- 对象文件（ET_REL，*.o）
  - 需要与其他对象文件链接，必须包含section
- 动态库（ET_DYN，*.so）
  - 与其他对象文件/可执行文件链接
  - 必须同时包含segment和section

ELF文件格式

- ELF Header
  - 架构、ABI版本等基础信息
  - program header table的位置和数量
  - section header table的位置和数量
- Program header table
  - 每个表项定义了一个segment
  - 每个segment可包含多个section
- Section header table
  - 每个表项定义了一个section



查看ELF Header

长亭科技 CHAITIN.CN

```
$ readelf -h ropasaurusrex
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048340
  Start of program headers:          52 (bytes into file)
  Start of section headers:          1828 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         7
  Size of section headers:           40 (bytes)
  Number of section headers:         28
  Section header string table index: 27
```

查看Program Header

长亭科技 CHAITIN.CN

```
$ readelf -l ropasaurusrex
Elf file type is EXEC (Executable file)
Entry point 0x8048340
There are 7 program headers, starting at offset 52
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x000e0 0x000e0 R E 0x4
  INTERP         0x000114 0x08048114 0x08048114 0x00013 0x00013 R   0x1
  LOAD           0x000000 0x08048000 0x08048000 0x0051c 0x0051c R E 0x1000
  LOAD           0x00051c 0x0804951c 0x0804951c 0x0010c 0x00114 RW  0x1000
  DYNAMIC        0x000530 0x08049530 0x08049530 0x000d0 0x000d0 RW  0x4
  NOTE           0x000128 0x08048128 0x08048128 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.ABI-tag .note.gnu.build-id .hash .gnu.hash .dynsym .dynstr .gnu.version
.gnu.version_r .rel.dyn .rel.plt .init .plt .text .fini .rodata .eh_frame
   03     .ctors .dtors .jcr .dynamic .got .got.plt .data .bss
   04     .dynamic
   05     .note.ABI-tag .note.gnu.build-id
```

每个Segment映射后的内存权限

每个Segment映射到的虚拟地址

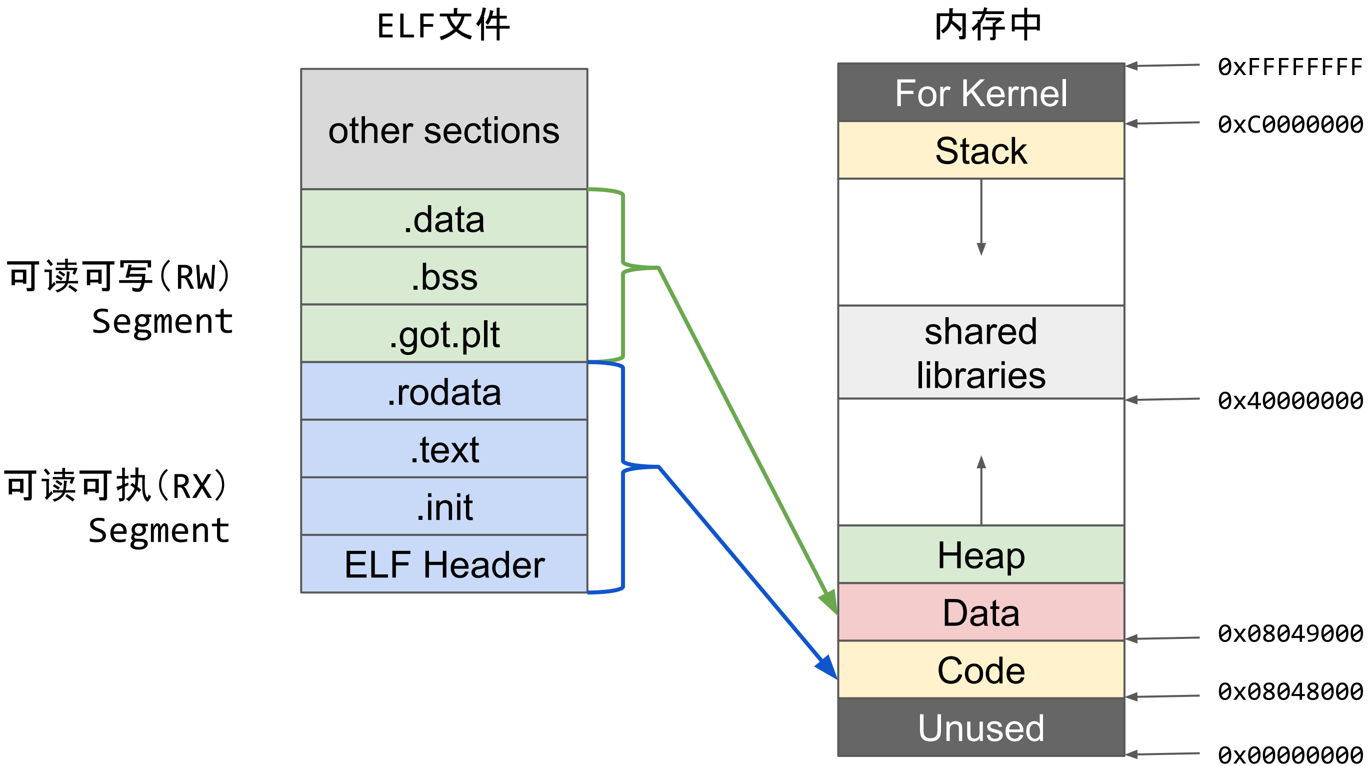
每个Segment包含哪些Section，
最左边代表Segment编号

查看Section Header

```
$ readelf -S ropasaurusrex
There are 28 section headers, starting at offset 0x724:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048114 000114 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048128 000128 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048148 000148 000024 00   A  0   0  4
  [ 4] .hash             HASH            0804816c 00016c 00002c 04   A  6   0  4
  [ 5] .gnu.hash         GNU_HASH        08048198 000198 000020 04   A  6   0  4
  [ 6] .dynsym           DYNSYM          080481b8 0001b8 000060 10   A  7   1  4
  [ 7] .dynstr           STRTAB          08048218 000218 000050 00   A  0   0  1
  [ 8] .gnu.version      VERSYM          08048268 000268 00000c 02   A  6   0  2
  [ 9] .gnu.version_r    VERNEED         08048274 000274 000020 00   A  7   1  4
  [10] .rel.dyn          REL             08048294 000294 000008 08   A  6   0  4
  [11] .rel.plt          REL             0804829c 00029c 000020 08   A  6  13  4
  [12] .init             PROGBITS        080482bc 0002bc 000030 00  AX  0   0  4
  [13] .plt              PROGBITS        080482ec 0002ec 000050 04  AX  0   0  4
  [14] .text             PROGBITS        08048340 000340 0001ac 00  AX  0   0 16
  ...
```

内存映射



ELF文件
内存中
可读可写(RW)
Segment
可读可执(RX)
Segment
other sections
.data
.bss
.got.plt
.rodata
.text
.init
ELF Header
For Kernel
Stack
shared
libraries
Heap
Data
Code
Unused
0xFFFFFFFF
0xC0000000
0x40000000
0x08049000
0x08048000
0x00000000

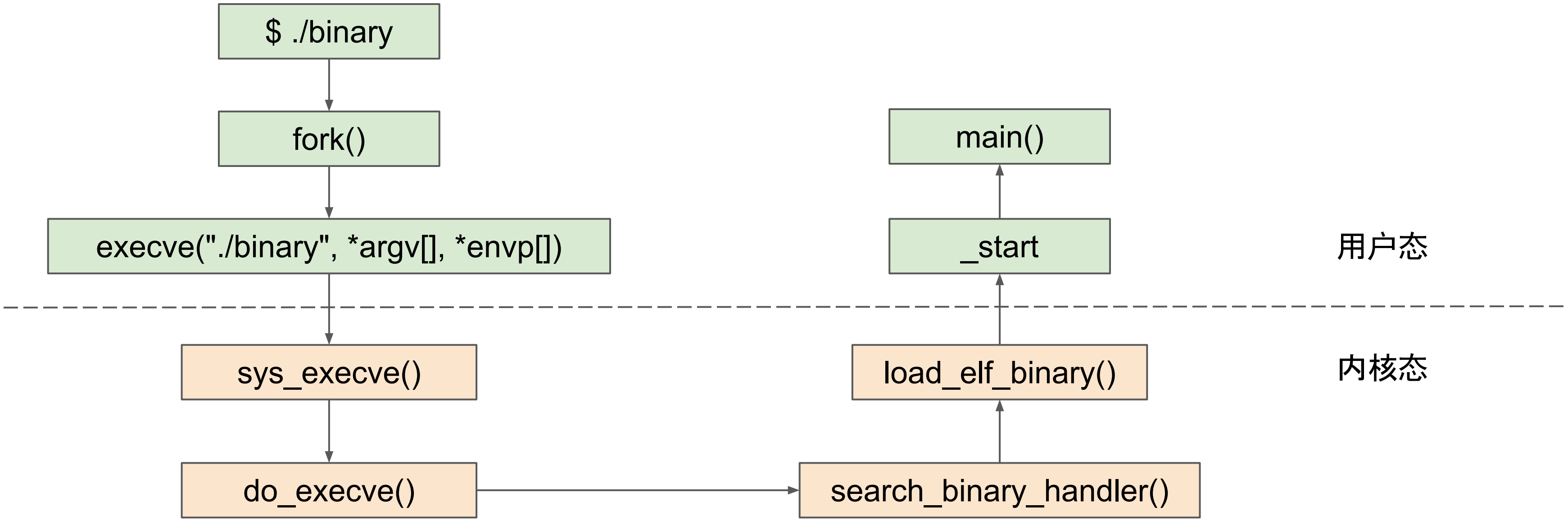
通过/proc/[pid]/maps查看内存映射情况

长亭科技
CHAITIN.CN

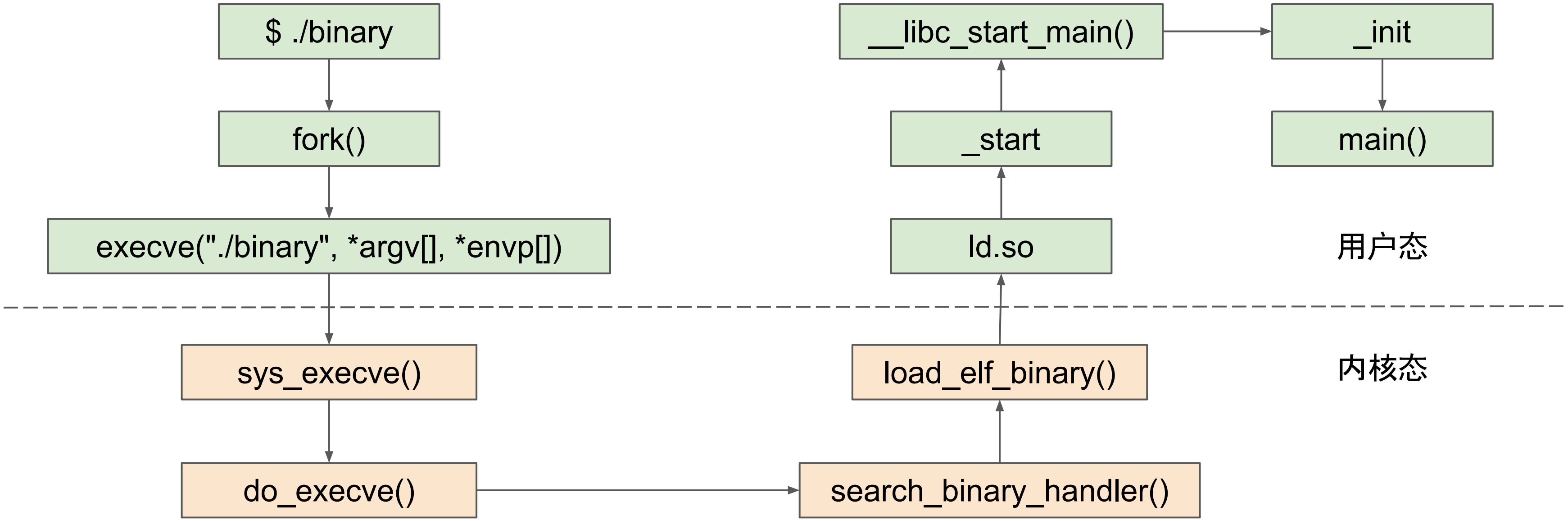
```
$ cat /proc/$(pidof ropasaurusrex)/maps
08048000-08049000 r-xp 00000000 00:25 19711621     /tmp/ropasaurusrex
08049000-0804a000 rw-p 00000000 00:25 19711621     /tmp/ropasaurusrex
f752f000-f76e8000 r-xp 00000000 08:02 17827410     /usr/lib32/libc-2.25.so
f76e8000-f76e9000 ---p 001b9000 08:02 17827410     /usr/lib32/libc-2.25.so
f76e9000-f76eb000 r--p 001b9000 08:02 17827410     /usr/lib32/libc-2.25.so
f76eb000-f76ec000 rw-p 001bb000 08:02 17827410     /usr/lib32/libc-2.25.so
f76ec000-f76ef000 rw-p 00000000 00:00 0
f7720000-f7722000 rw-p 00000000 00:00 0
f7722000-f7724000 r--p 00000000 00:00 0            [vvar]
f7724000-f7726000 r-xp 00000000 00:00 0            [vdso]
f7726000-f7748000 r-xp 00000000 08:02 17827411     /usr/lib32/ld-2.25.so
f7749000-f774a000 r--p 00022000 08:02 17827411     /usr/lib32/ld-2.25.so
f774a000-f774b000 rw-p 00023000 08:02 17827411     /usr/lib32/ld-2.25.so
ff8d6000-ff8f7000 rw-p 00000000 00:00 0            [stack]
```

可读可执行Segment
可读可写Segment

静态链接的程序的启动过程



动态链接的程序的启动过程



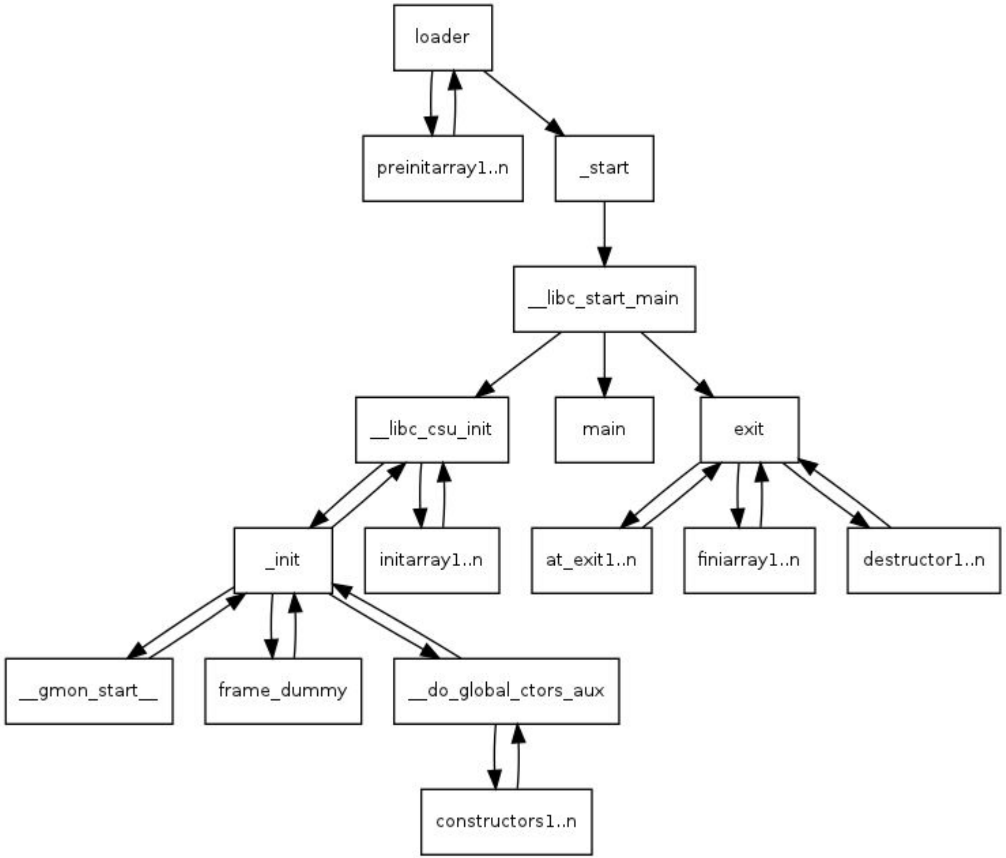
程序是如何启动的？

- sys_execve()
  - 检查参数和环境变量
- do_execve()
  - 解析ELF头，填充二进制格式相关参数（linux_binprm结构题）
- search_binary_handler()
  - 搜索已注册的二进制格式列表，找到正确的格式
- load_elf_binary()
  - 解析program header
  - 从.interp节（section）中找到装载器ld.so的路径
  - 映射内存段（segment）
  - 修改sys_execve的返回值为ld.so或静态链接ELF的入口地址

ld.so & _start & __libc_start_main

- ld.so
  - 负责加载所有共享库
  - 初始化GOT表
- _start
  - 为__libc_start_main传递环境变量和.init/.fini/main函数
- __libc_start_main
  - 调用 .init
  - 调用 main
  - 调用 .fini
  - 调用 exit

ELF启动过程流程图


loader
preinitarray1..n
_start
__libc_start_main
__libc_csu_init
main
exit
_init
initarray1..n
at_exit1..n
finiarray1..n
destructor1..n
__gmon_start__
frame_dummy
__do_global_ctors_aux
constructors1..n