# Hello World:
# A Brief Introduction to Capture-The-Flag

Zhuo Zhang (zhan3299@purdue.edu)

PURDUE
UNIVERSITY

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

# About Me

- Zhuo Zhang 张倬
- Ph.D. Student at Purdue CS
- 原上海交通大学 0ops CTF Team核心成员（2014-2018）

# Selected CTF Experience

- 参与比赛：
  - 1st place at the 40th IEEE S&P Celebration Scavenger Hunt
  - 4th place (a*0*e) at DEFCON CTF 2018
  - 3rd place (a*0*e) at DEFCON CTF 2017
  - 首届拟态防御国际精英邀请赛第二名（0ops）
  - 首届强网杯精英赛第三名（0ops，国内队伍第一名）

- 参与命题：
  - 0CTF/TCTF 2017 (大陆唯一DEFCON CTF外卡赛)
  - 0CTF/TCTF 2016 (大陆唯一DEFCON CTF外卡赛)
  - 2016年全国大学生信息安全竞赛-技能赛

# Content

- CTF基本信息介绍

- CTF解题模式题目类型讲解
  - Reverse 逆向分析
  - Pwnable 漏洞利用
  - Crypto 密码学
  - Web 网络安全
  - Misc 杂项

- 学习方法与资料

# Capture The Flag

- 全球范围内，最流行的一种信息安全竞赛形式，起源于1996年的美国黑客大会 **DEFCON CTF**
- 对以往黑客们通过互相发起真实攻击进行技术对抗的一种替代
- 全程可控，不干扰真实网络
- 题目种类多样而有趣，重点考察综合能力

# CTF 比赛形式

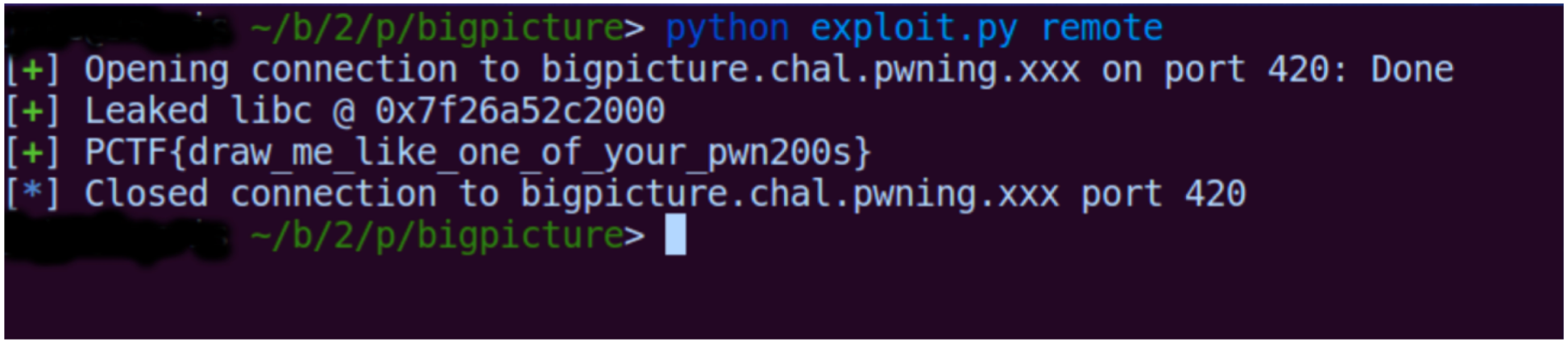
- Jeopardy 解题模式
  - 初赛常见，考察面广
  - 方便远程在线参加
  - 每解出一题，获得一定的分数
  - 时长12小时到48小时不等
  - https://ctfzone.sjtu.cn/

# CTF 比赛形式

- Jeopardy 解题模式
  - 初赛常见，考察面广
  - 方便远程在线参加
  - 每解出一题，获得一定的分数
  - 时长12小时到48小时不等
  - https://ctfzone.sjtu.cn/
- Attack-Defence 攻防模式
  - 现场决赛 （3-8人）
  - 维护己方服务，攻击他方服务
  - 攻击者得分，失守者失分
  - 获得他人服务器机密数据（flag）即视为攻击成功

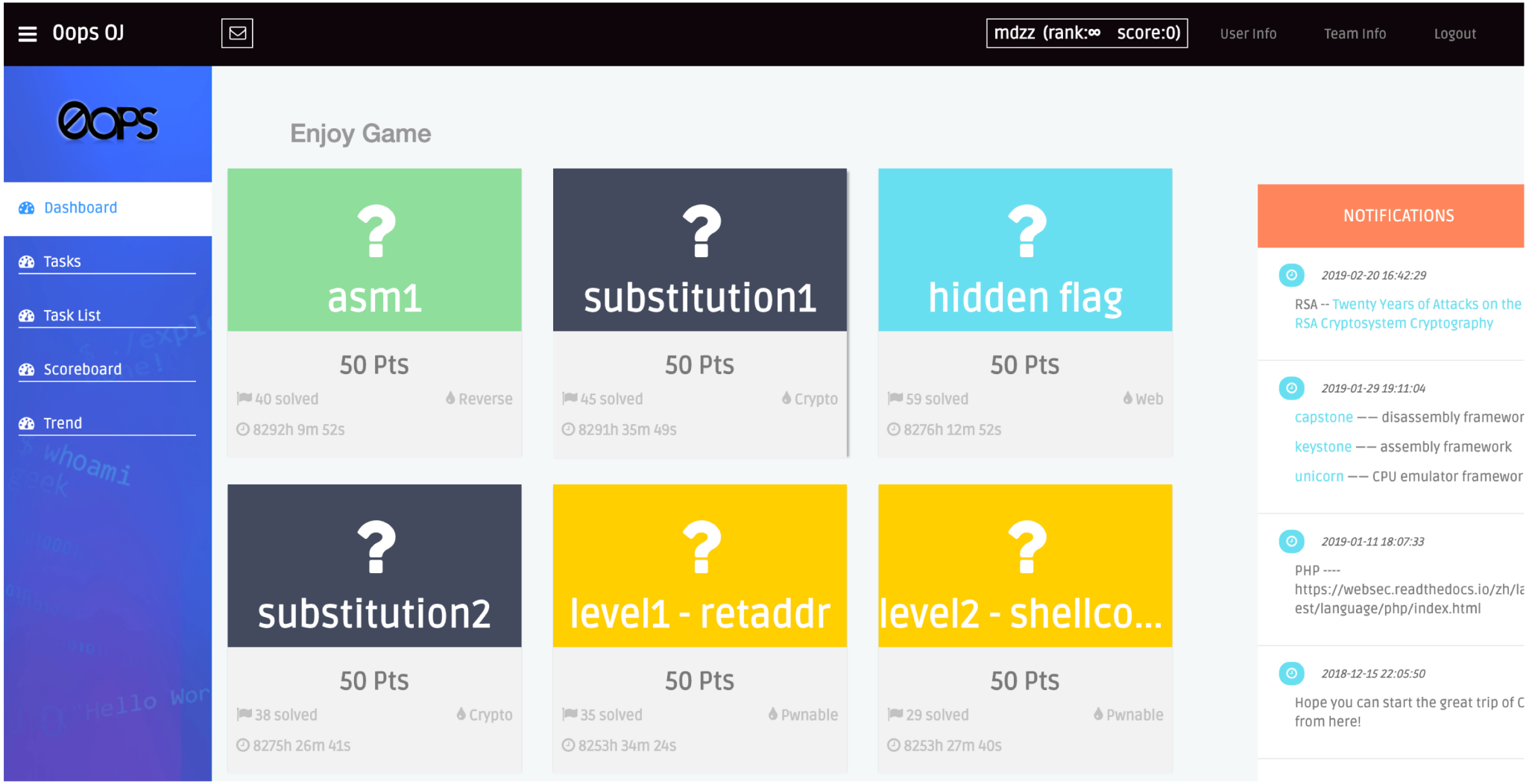
# CTF 比赛形式

- Jeopardy 解题模式
  - 初赛常见，考察面广
  - 方便远程在线参加
  - 每解出一题，获得一定的分数
  - 时长12小时到48小时不等
  - https://ctfzone.sjtu.cn/
- Attack-Defence 攻防模式
  - 现场决赛 （3-8人）
  - 维护己方服务，攻击他方服务
  - 攻击者得分，失守者失分
  - 获得他人服务器机密数据（flag）即视为攻击成果

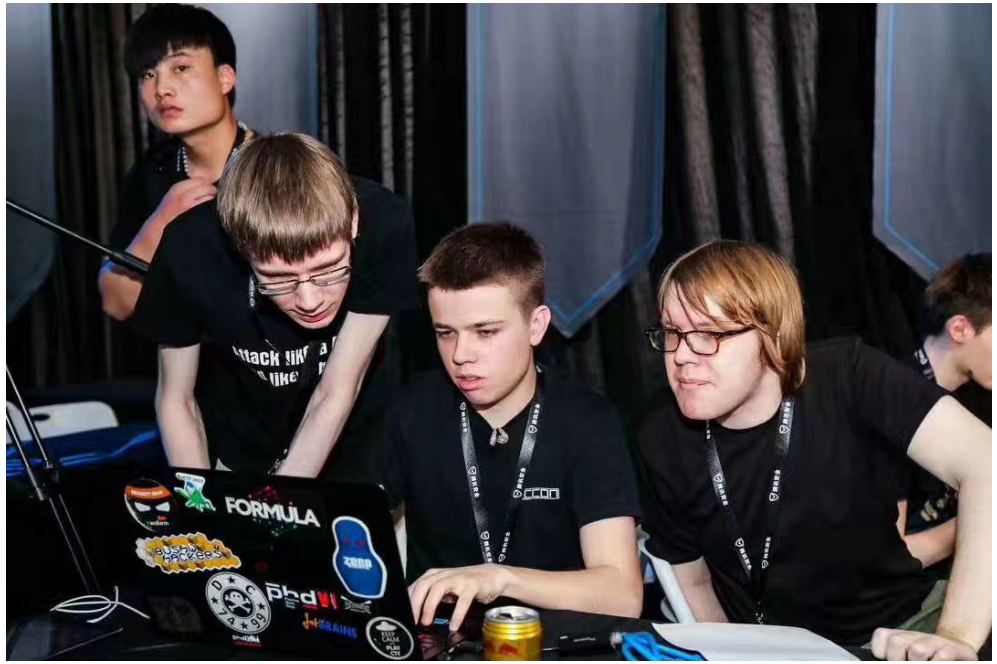



**核心目标：获取一串称为 “flag” 的特定字符串**

# CTF 比赛形式


0ops OJ
mdzz (rank:∞ score:0)
User Info
Team Info
Logout
Dashboard
Tasks
Task List
Scoreboard
Trend
Enjoy Game
asm1
50 Pts
40 solved
Reverse
substitution1
50 Pts
45 solved
Crypto
hidden flag
50 Pts
59 solved
Web
substitution2
50 Pts
38 solved
Crypto
level1 - retaddr
50 Pts
35 solved
Pwnable
level2 - shellco...
50 Pts
29 solved
Pwnable
NOTIFICATIONS
2019-02-20 16:42:29
RSA -- Twenty Years of Attacks on the RSA Cryptosystem Cryptography
2019-01-29 19:11:04
capstone —— disassembly framewor
keystone —— assembly framework
unicorn —— CPU emulator framewor
2019-01-11 18:07:33
PHP ----
https://websec.readthedocs.io/zh/la
est/language/php/index.html
2018-12-15 22:05:50
Hope you can start the great trip of C
from here!

# CTF 比赛形式




# CTF 必备基础知识

- Python – CTF届通用语言
  - 易用：脚本语言，方便使用
  - 易学：语法简单，类自然语言
  - 通用：第三方库丰富，使用者众多

# CTF 必备基础知识

- Python – CTF届通用语言
  - 易用：脚本语言，方便使用
  - 易学：语法简单，类自然语言
  - 通用：第三方库丰富，使用者众多

- Linux – 开源操作系统
  - 系统特性：万物皆文件，权限配置简明高效
  - 基本操作：基本命令，bash语法
  - 设计艺术：任务分而治之

# Jeopardy 解题模式

- Reverse 逆向分析
- Pwnable 漏洞利用
- Crypto 密码学
- Web 网络安全
- Misc 杂项

# Jeopardy 解题模式

- **Reverse 逆向分析**
- **Pwnable 漏洞利用**
- **Crypto 密码学**
- Web 网络安全
- Misc 杂项

# **Reverse** 逆向分析

- 题型简介
  - 提供一个二进制可执行文件（e.g., exe, elf）
  - 程序含有**生成flag**或**检查flag**的功能逻辑
  - 需要通过二进制程序分析破解程序，理解程序逻辑 （easyelf1）

# **Reverse** 逆向分析

- 题型简介
  - 提供一个二进制可执行文件（e.g., exe, elf）
  - 程序含有**生成flag**或**检查flag**的功能逻辑
  - 需要通过二进制程序分析破解程序，理解程序逻辑
- 常用工具
  - 静态分析：IDA Pro, NSF Ghidra, radare2
  - 动态分析：GDB(peda), qemu

# **Reverse** 逆向分析

- 题型简介
  - 提供一个二进制可执行文件（e.g., exe, elf）
  - 程序含有**生成flag**或**检查flag**的功能逻辑
  - 需要通过二进制程序分析破解程序，理解程序逻辑
- 常用工具
  - 静态分析：IDA Pro, NSF Ghidra, radare2
  - 动态分析：GDB, qemu
- 逆向分析的核心
  - 对汇编语言的理解：x86/x64, ARM, mips等（样例讲解：asm1.s）

# asm1.s

asm1

Assembly is the basic of binary.
Please tell me the return value of main function. The flag is "Oops{`decimal return value`}"

asm1.s

Please input your flag

Submit

# asm1.s

```
0       .file   "asm1.c"
1       .intel_syntax noprefix
2       .text
3       .globl
4       .type   main, @function
```

汇编头信息，不影响分析，可以跳过

# asm1.s

```
05 main:
06 .LFB0
07     .cfi_startproc
08     push   rbp
09     .cfi_def_cfa_offset 16
10     .cfi_offset 6, -16
11     mov rbp, rsp
12     .cfi_def_cfa_register 6
13     push   r12
14     push   rbx
15     .cfi_offset 12, -24
16     .cfi_offset 3, -32
```

# asm1.s

```
05 main:
06 .LFB0
07     .cfi_startproc
08     push   rbp
09     .cfi_def_cfa_offset 16
10     .cfi_offset 6, -16
11     mov rbp, rsp
12     .cfi_def_cfa_register 6
13     push   r12
14     push   rbx
15     .cfi_offset 12, -24
16     .cfi_offset 3, -32
```

Main函数起始地址

# asm1.s

```
05 main:
06 .LFB0
07     .cfi_startproc
08     push   rbp
09     .cfi_def_cfa_offset 16
10     .cfi_offset 6, -16
11     mov rbp, rsp
12     .cfi_def_cfa_register 6
13     push   r12
14     push   rbx
15     .cfi_offset 12, -24
16     .cfi_offset 3, -32
```

CFI 相关变量，不影响分析
(Call Frame Instruction)

# asm1.s

```
05 main:
06 .LFB0
07     .cfi_startproc
08     push   rbp
09     .cfi_def_cfa_offset 16
10     .cfi_offset 6, -16
11     mov rbp, rsp
12     .cfi_def_cfa_register 6
13     push   r12
14     push   rbx
15     .cfi_offset 12, -24
16     .cfi_offset 3, -32
```

更新栈空间

# asm1.s

```
05 main:
06 .LFB0
07     .cfi_startproc
08     push   rbp
09     .cfi_def_cfa_offset 16
10     .cfi_offset 6, -16
11     mov rbp, rsp
12     .cfi_def_cfa_register 6
13     push   r12
14     push   rbx
15     .cfi_offset 12, -24
16     .cfi_offset 3, -32
```

保存callee-saved registers

# asm1.s

```
05 main:
06 .LFB0
07     .cfi_startproc
08     push   rbp
09     .cfi_def_cfa_offset 16
10     .cfi_offset 6, -16
11     mov rbp, rsp
12     .cfi_def_cfa_register 6
13     push   r12
14     push   rbx
15     .cfi_offset 12, -24
16     .cfi_offset 3, -32
```

标准函数开头

# asm1.s

```
17      mov r12d, 0
18      mov ebx, 0
19      jmp .L2
20  .L3:
21      add r12d, ebx
22      add ebx, 2
23  .L2:
24      cmp ebx, 2017
25      jbe .L3
```

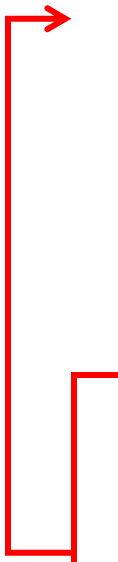
初始化r12d和ebx为0

# asm1.s

```
17      mov r12d, 0
18      mov ebx, 0
19      jmp .L2
20  .L3:
21      add r12d, ebx
22      add ebx, 2
23  .L2:
24      cmp ebx, 2017
25      jbe .L3
```

# asm1.s

```
17      mov r12d, 0
18      mov ebx, 0
19      jmp .L2
20  .L3:
21      add r12d, ebx
22      add ebx, 2
23  .L2:
24      cmp ebx, 2017
25      jbe .L3
```



比较ebx和2017的大小，小于等于则跳转至L3

1. 这是一个循环
2. 循环控制变量是ebx

# asm1.s

```
17      mov r12d, 0
18      mov ebx, 0
19      jmp .L2
20  .L3:
21      add r12d, ebx
22      add ebx, 2
23  .L2:
24      cmp ebx, 2017
25      jbe .L3
```

把ebx的值加入r12d，
并且ebx += 2

1. r12d是sum
2. ebx步长为2

# asm1.s

```
17      mov r12d, 0
18      mov ebx, 0
19      jmp .L2
20  .L3:
21      add r12d, ebx
22      add ebx, 2
23  .L2:
24      cmp ebx, 2017
25      jbe .L3
```

r12d = 0 + 2 + … + 2012 + 2014 + 2016

# asm1.s

```
26      mov eax, r12d
27      pop rbx
28      pop r12
29      pop rbp
30      .cfi_def_cfa 7, 8
31      ret
```

eax是main返回值

# asm1.s

```
26      mov eax, r12d
27      pop rbx
28      pop r12
29      pop rbp
30      .cfi_def_cfa 7, 8
31      ret
```

恢复callee-saved registers

# asm1.s

```
26      mov eax, r12d
27      pop rbx
28      pop r12
29      pop rbp
30      .cfi_def_cfa 7, 8
31      ret
```

返回__libc_start_main

# asm1.s

```
26      mov eax, r12d
27      pop rbx
28      pop r12
29      pop rbp
30      .cfi_def_cfa 7, 8
31      ret
```

**Flag = 0ops{0 + 2 + … + 2012 + 2014 + 2016}**

# Reverse 逆向分析

- 题型简介
  - 提供一个二进制可执行文件（e.g., exe, elf）
  - 程序含有**生成flag**或**检查flag**的功能逻辑
  - 需要通过二进制程序分析破解程序，理解程序逻辑
- 常用工具
  - 静态分析：IDA Pro, NSF Ghidra, radare2
  - 动态分析：GDB, qemu
- 逆向分析的核心
  - 对汇编语言的理解：x86/x64, ARM, mips等（样例讲解：asm1.s）
  - 对程序底层行为的理解：C语言基础 （样例讲解： easyelf1）

# **Reverse** 逆向分析

- 题型简介
  - 提供一个二进制可执行文件（e.g., exe, elf）
  - 程序含有**生成flag**或**检查flag**的功能逻辑
  - 需要通过二进制程序分析破解程序，理解程序逻辑
- 常用工具
  - 静态分析：IDA Pro, NSF Ghidra, radare2
  - 动态分析：GDB, qemu
- 逆向分析的核心
  - 对汇编语言的理解：x86/x64, ARM, mips等（样例讲解：asm1.s）
  - 对程序底层行为的理解：C语言基础 （样例讲解： easyelf1）

Static and Dynamic analysis

# **Reverse** 逆向分析

- 难度加深
  - **程序混淆**：困难的题目会被刻意混淆或加壳，这些混淆方式有可能是已有的方法（e.g., ollvm），也可能是出题人自己设计的方法
  - **文件混淆**：有些题目会在文件格式上做手脚，使得常规程序无法正确分析
  - **大型文件**：有些题目的文件体量很大，只需要弄清楚和flag相关的逻辑即可

# Reverse 逆向分析

- 难度加深
  - **程序混淆**：困难的题目会被刻意混淆或加壳，这些混淆方式有可能是已有的方法（e.g., ollvm），也可能是出题人自己设计的方法
  - **文件混淆**：有些题目会在文件格式上做手脚，使得常规程序无法正确分析
  - **大型文件**：有些题目的文件体量很大，只需要弄清楚和flag相关的逻辑即可
- 解决办法：
  - 深入理解文件格式（e.g., elf header, eh_frame）
  - 在编译原理和汇编语言的基础上自底向上人工分析
  - 多多练习，熟能生巧

# Reverse 逆向分析

- 难度加深
  - **程序混淆**：困难的题目会被刻意混淆或加壳，这些混淆方式有可能是已有的方法（e.g., ollvm），也可能是出题人自己设计的方法
  - **文件混淆**：有些题目会在文件格式上做手脚，使得常规程序无法正确分析
  - **大型文件**：有些题目的文件体量很大，只需要弄清楚和flag相关的逻辑即可
- 解决办法：
  - 深入理解文件格式（e.g., elf header, eh_frame）
  - 在编译原理和汇编语言的基础上自底向上人工分析
  - 多多练习，熟能生巧
- 练习平台： http://reversing.kr/

# Pwnable 漏洞利用

- 题型简介
  - 提供一个二进制可执行文件（通常为elf），作为一个运行在远程服务器上的服务
  - 通过逆向分析**挖掘程序漏洞**，再利用漏洞**进行攻击**以获得远程服务器的操作权限，最终获得flag (retaddr)

# Pwnable 漏洞利用

- 题型简介
  - 提供一个二进制可执行文件（通常为elf），作为一个运行在远程服务器上的服务
  - 通过逆向分析**挖掘程序漏洞**，再利用漏洞**进行攻击**以获得远程服务器的操作权限，最终获得flag
- 常用工具
  - 逆向工具：见前文，逆向分析是漏洞利用的基础
  - 漏洞利用：pwntools, ROPgarget, Onegadget, etc.

# Pwnable 漏洞利用

- 漏洞利用的核心
  - 需要非常熟悉编译原理及程序底层行为
  - 需要非常熟悉各种漏洞形式及其相应的攻击方式
    - 栈溢出
    - 堆溢出
    - 格式化字符串攻击
    - Use After Free
    - 变量未初始化
    - ……
  - 更困难的题目会涉及linux kernel、依赖库glibc的实现细节等

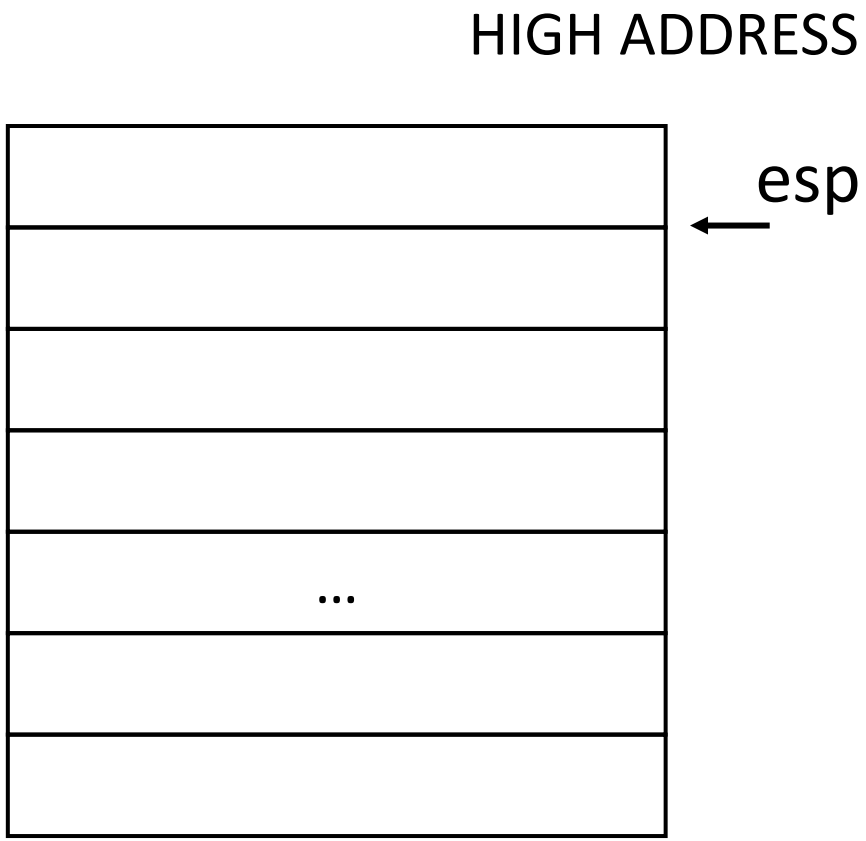
# Pwnable 漏洞利用

- 漏洞利用的核心
  - 需要非常熟悉编译原理及程序底层行为
  - 需要非常熟悉各种漏洞形式及其相应的攻击方式
    - **栈溢出**（样例讲解：retaddr）
    - 堆溢出
    - 格式化字符串攻击
    - Use After Free
    - 变量未初始化
    - ……
  - 更困难的题目会涉及linux kernel、依赖库glibc的实现细节等

# 基础知识

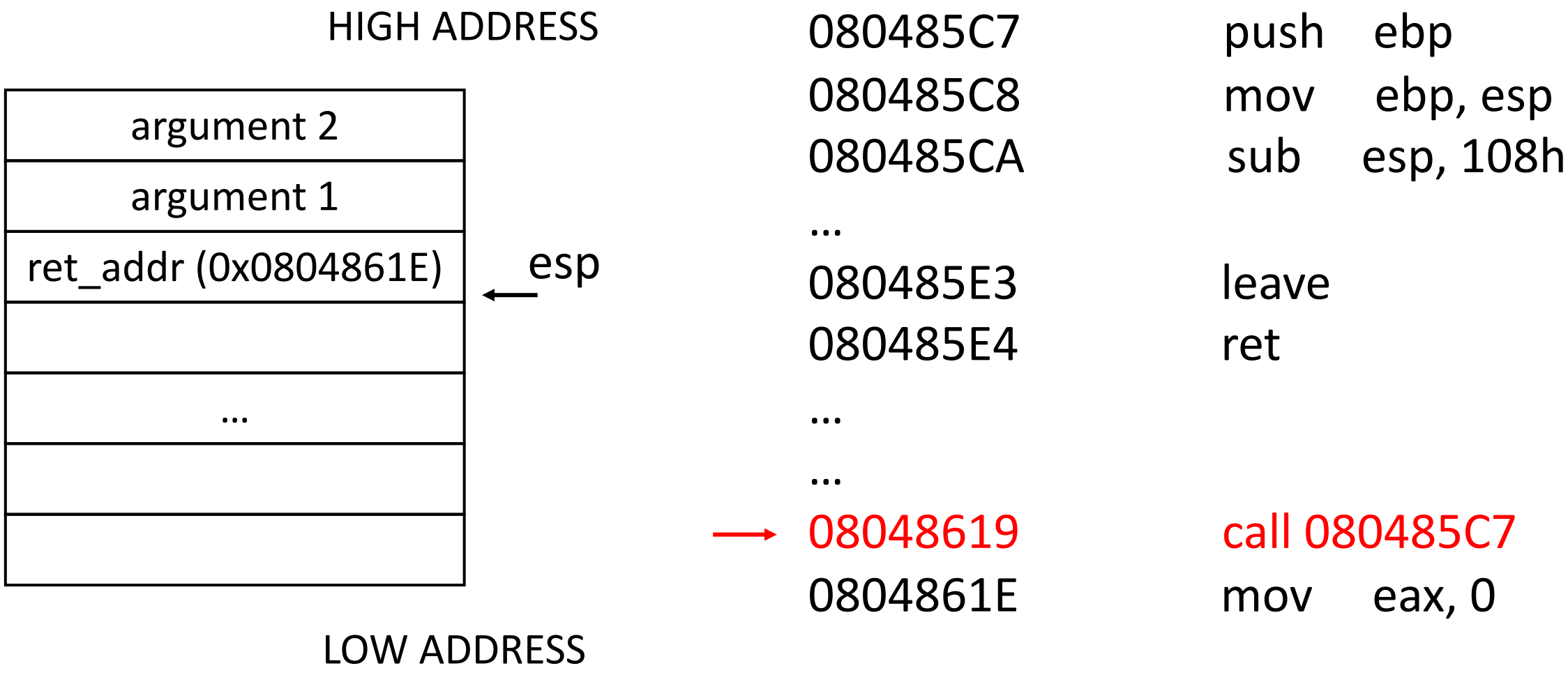
- 了解x86、x64的寄存器器的基本作用
  - rsp/esp: 栈顶指针
  - rbp/ebp: 栈底指针*
- Calling convention
  - X86: 栈传参
  - X64:  寄存器传参（rdi, rsi, rdx, rcx, r8, r9）+ 栈传参
- 栈布局

# 栈布局

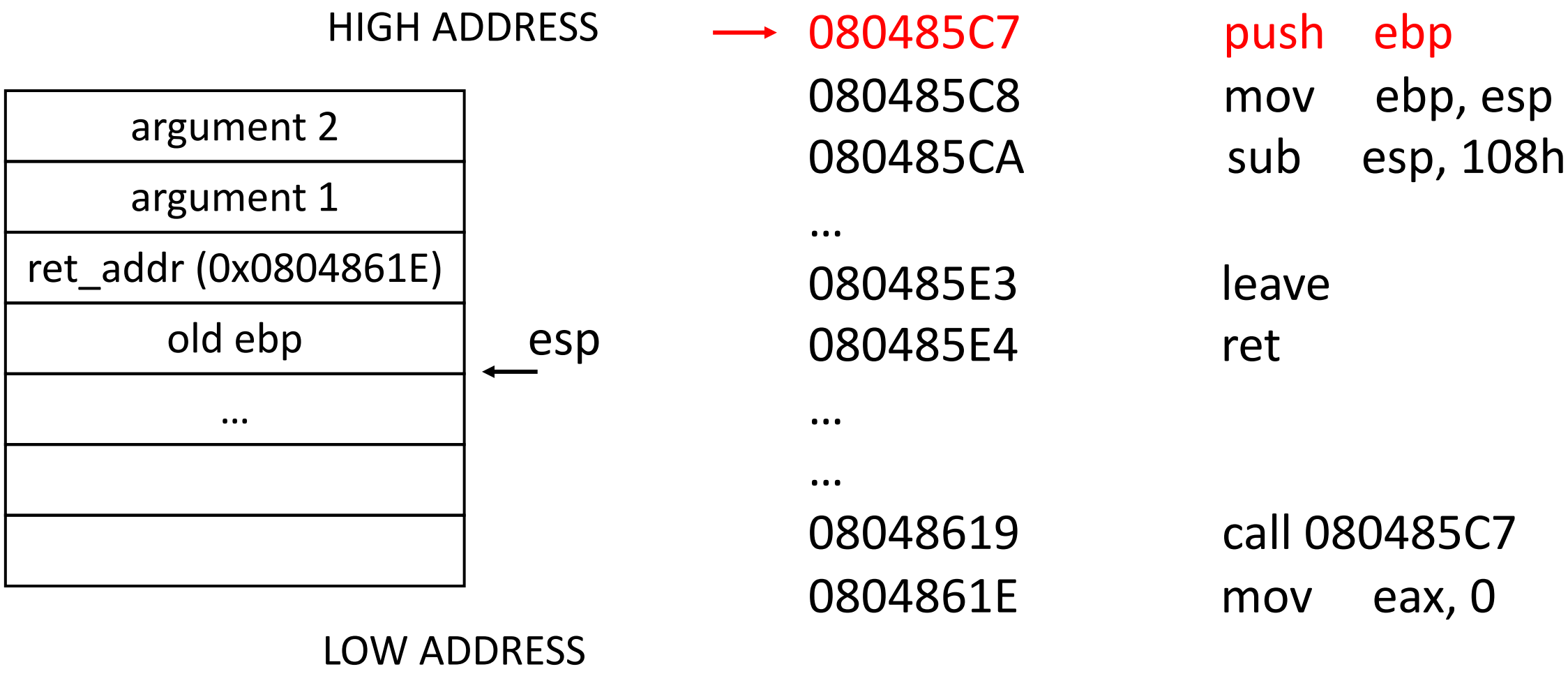



```
080485C7        push    ebp
080485C8        mov     ebp, esp
080485CA        sub     esp, 108h
…
080485E3        leave
080485E4        ret
…
…
08048619        call 080485C7
0804861E        mov     eax, 0
```

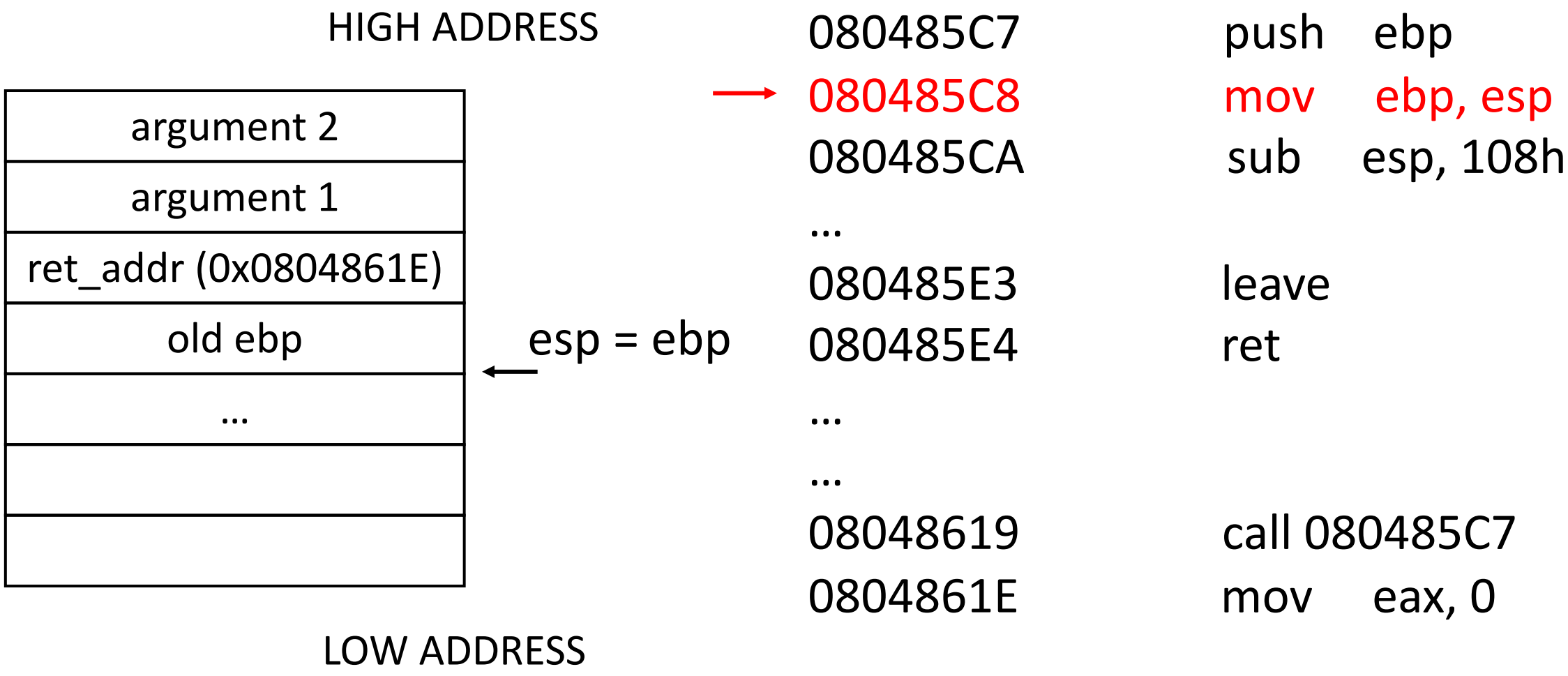
# 栈布局


HIGH ADDRESS
argument 2
argument 1
ret_addr (0x0804861E)
esp
…
LOW ADDRESS
080485C7 push ebp
080485C8 mov ebp, esp
080485CA sub esp, 108h
…
080485E3 leave
080485E4 ret
…
…
08048619 call 080485C7
0804861E mov eax, 0

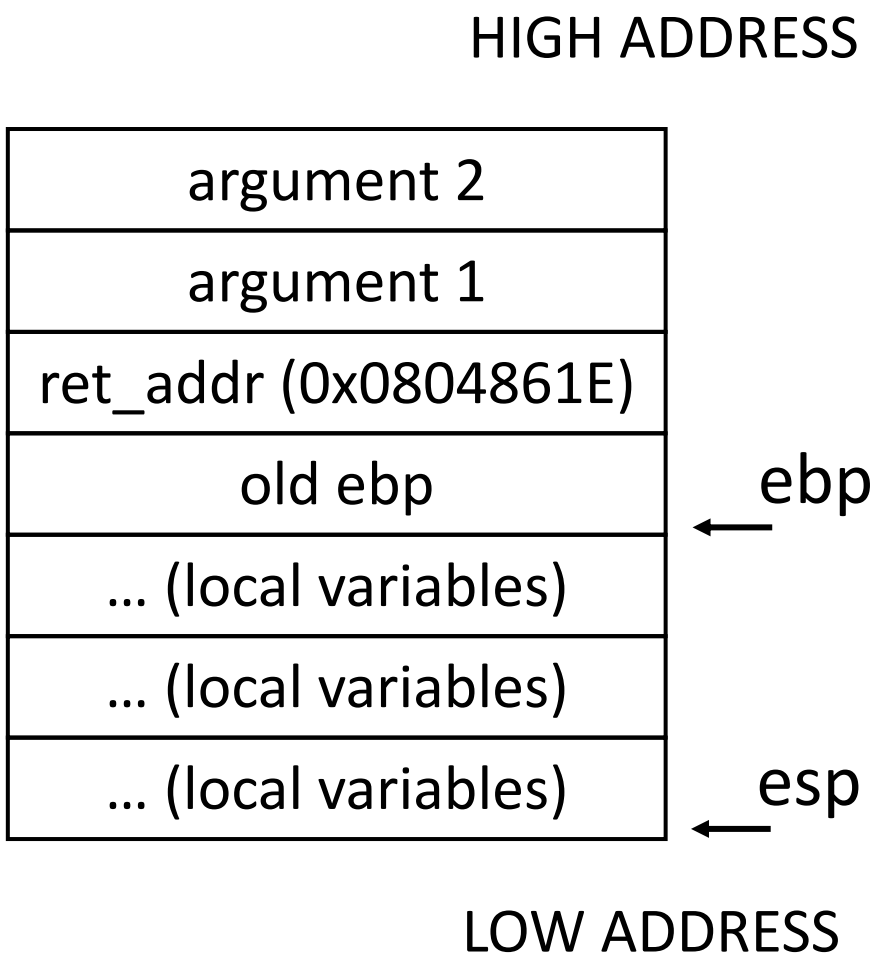
# 栈布局


HIGH ADDRESS
argument 2
argument 1
ret_addr (0x0804861E)
old ebp
esp
…
LOW ADDRESS
080485C7 push ebp
080485C8 mov ebp, esp
080485CA sub esp, 108h
…
080485E3 leave
080485E4 ret
…
…
08048619 call 080485C7
0804861E mov eax, 0

# 栈布局


HIGH ADDRESS
argument 2
argument 1
ret_addr (0x0804861E)
old ebp
…
esp = ebp
LOW ADDRESS
080485C7 push ebp
080485C8 mov ebp, esp
080485CA sub esp, 108h
…
080485E3 leave
080485E4 ret
…
…
08048619 call 080485C7
0804861E mov eax, 0

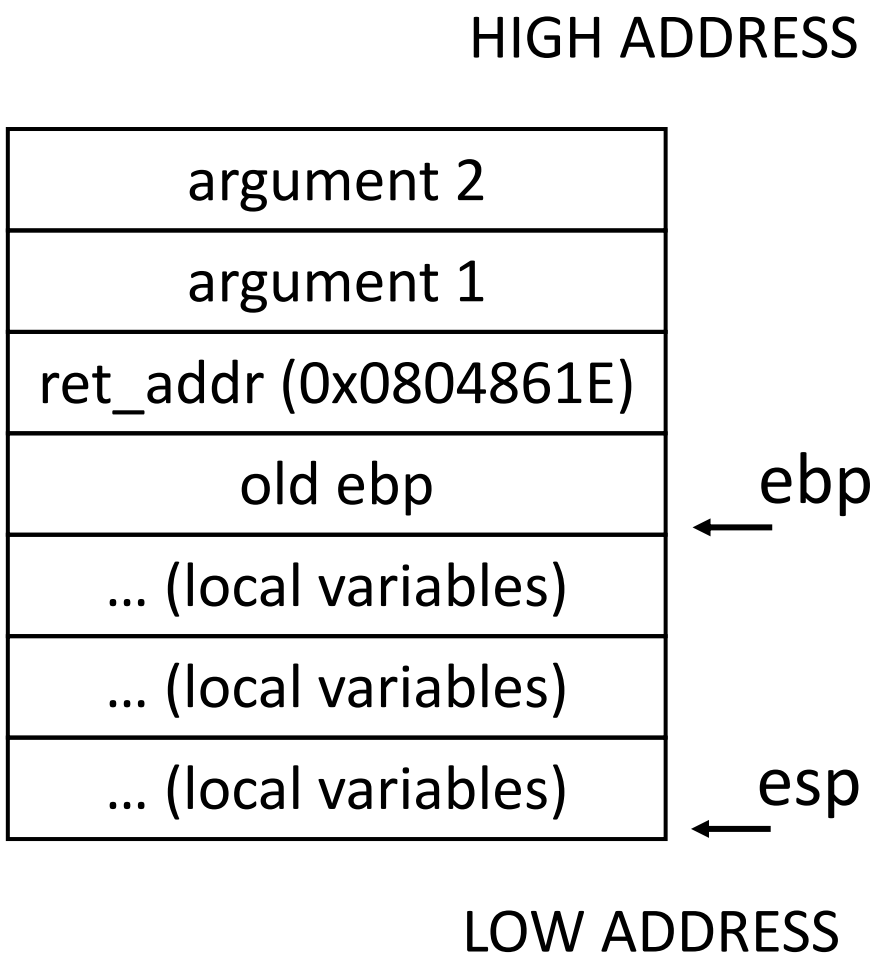
# 栈布局




```
080485C7    push   ebp
080485C8    mov    ebp, esp
080485CA    sub    esp, 108h
…
080485E3    leave
080485E4    ret
…
…
08048619    call 080485C7
0804861E    mov    eax, 0
```

# 栈布局

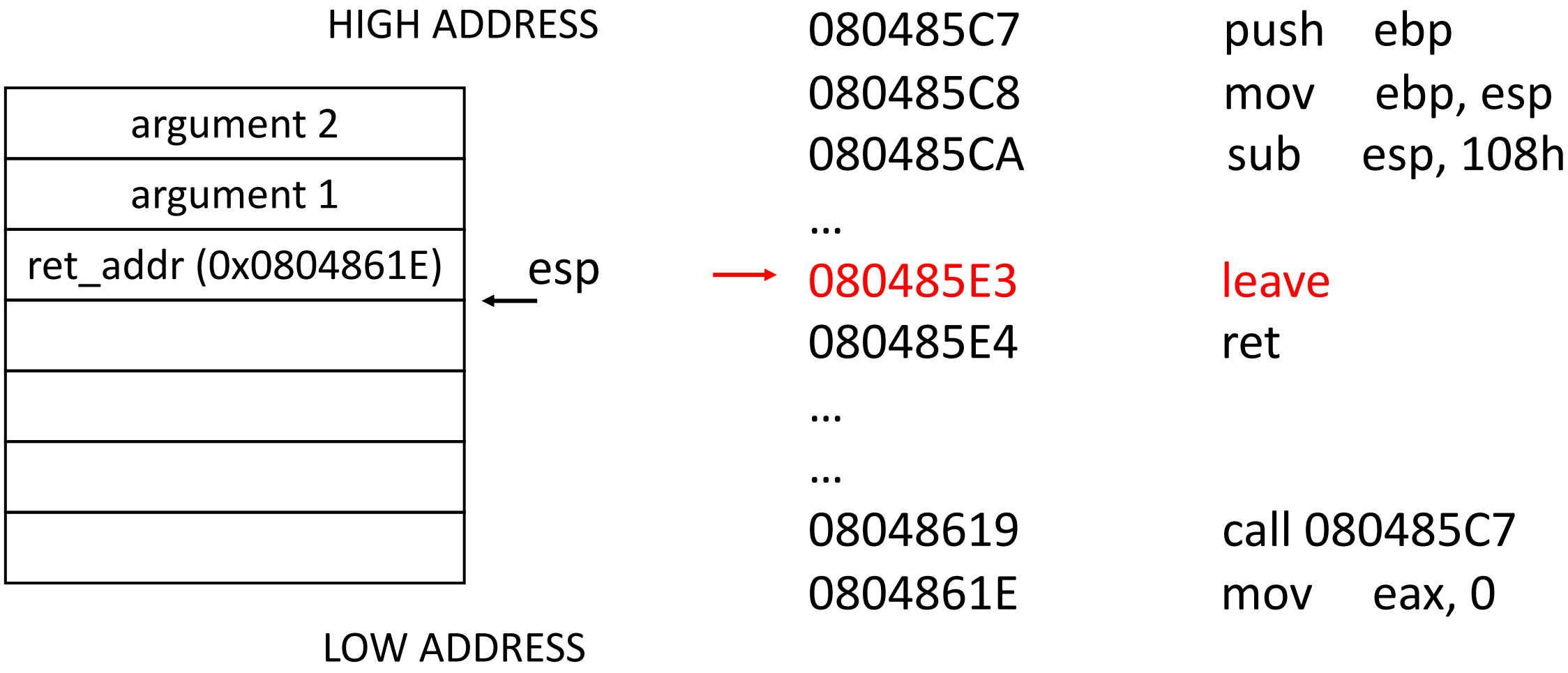
leave → mov esp, ebp
pop ebp




```
080485C7    push   ebp
080485C8    mov    ebp, esp
080485CA    sub    esp, 108h
…
080485E3    leave
080485E4    ret
…
…
08048619    call 080485C7
0804861E    mov    eax, 0
```

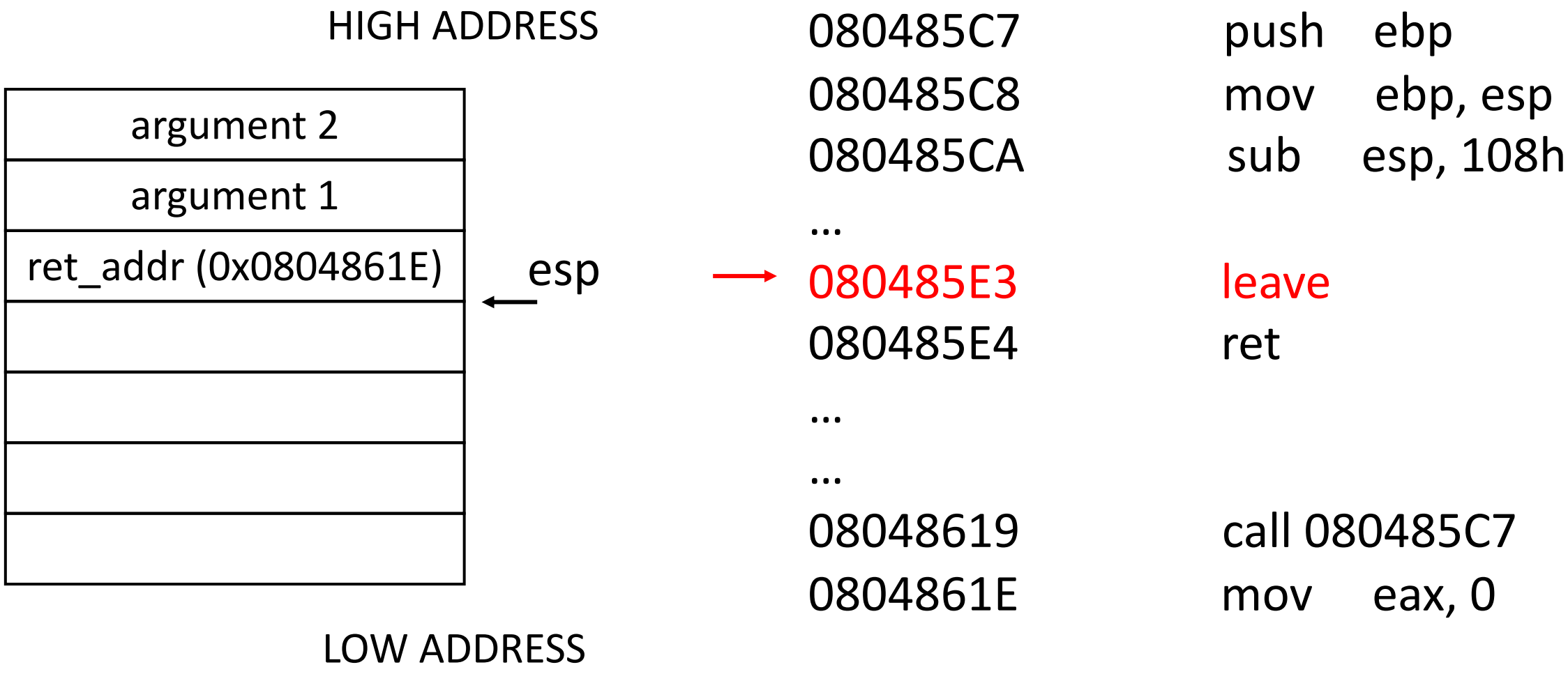
# 栈布局

leave → mov esp, ebp
pop ebp



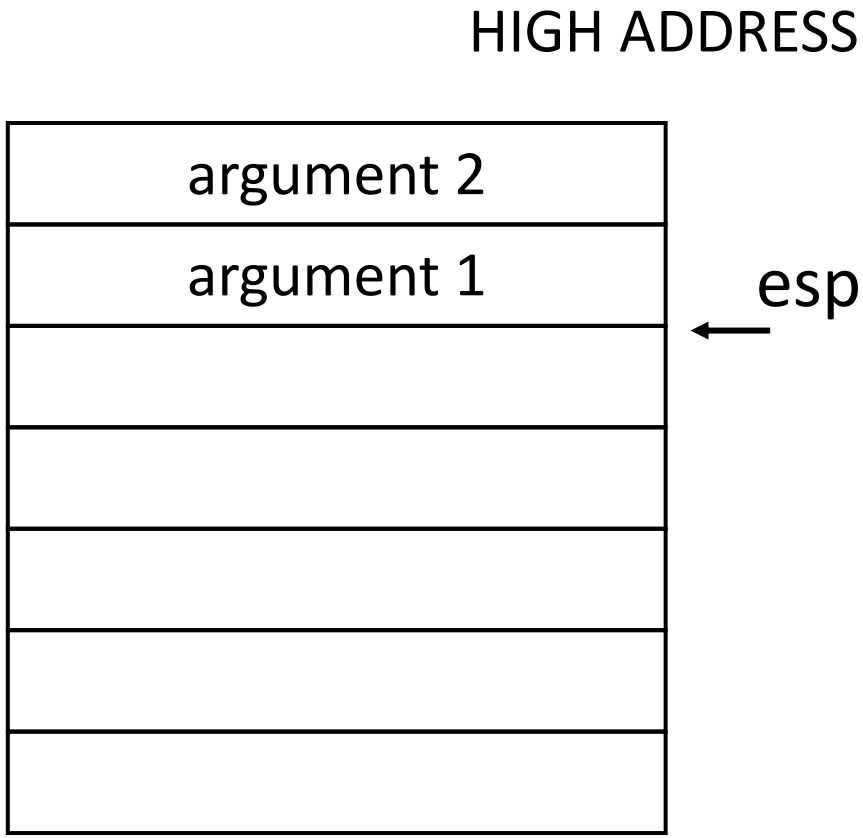
# 栈布局

ret → pop eip



# 栈布局

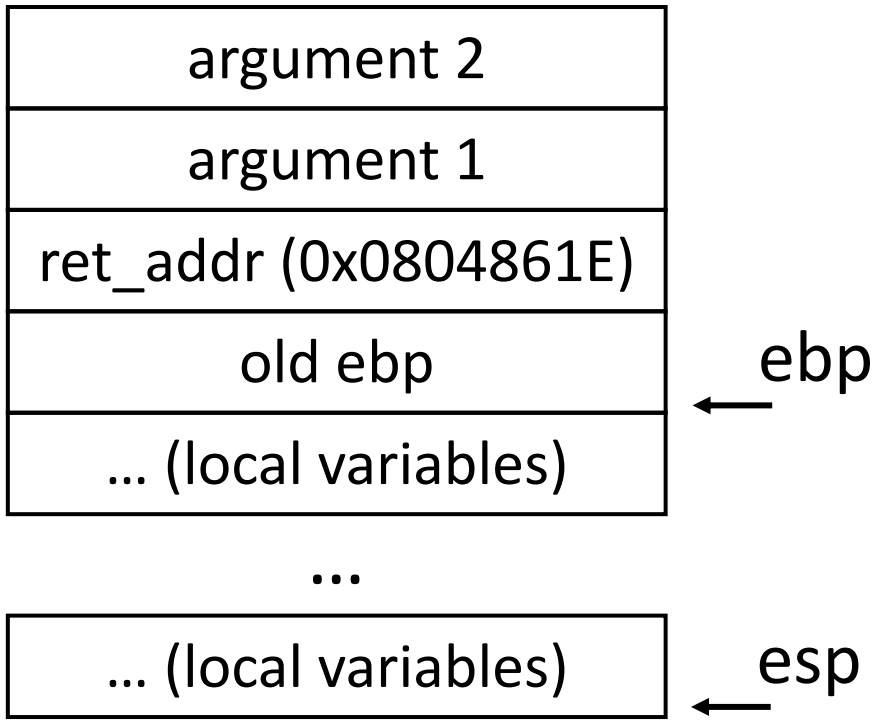
ret → pop eip




```
080485C7        push    ebp
080485C8        mov     ebp, esp
080485CA        sub     esp, 108h
...
080485E3        leave
080485E4        ret
...
...
08048619        call 080485C7
0804861E        mov     eax, 0
```

# retaddr

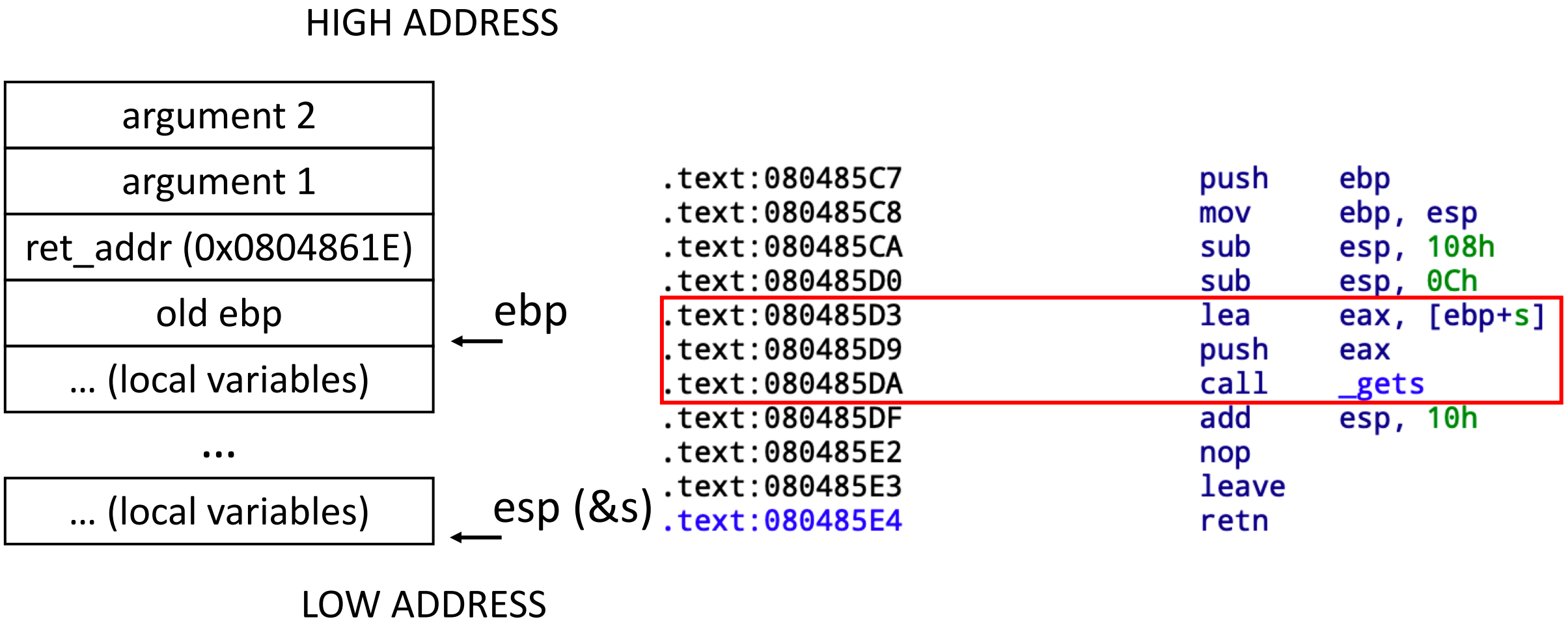
HIGH ADDRESS



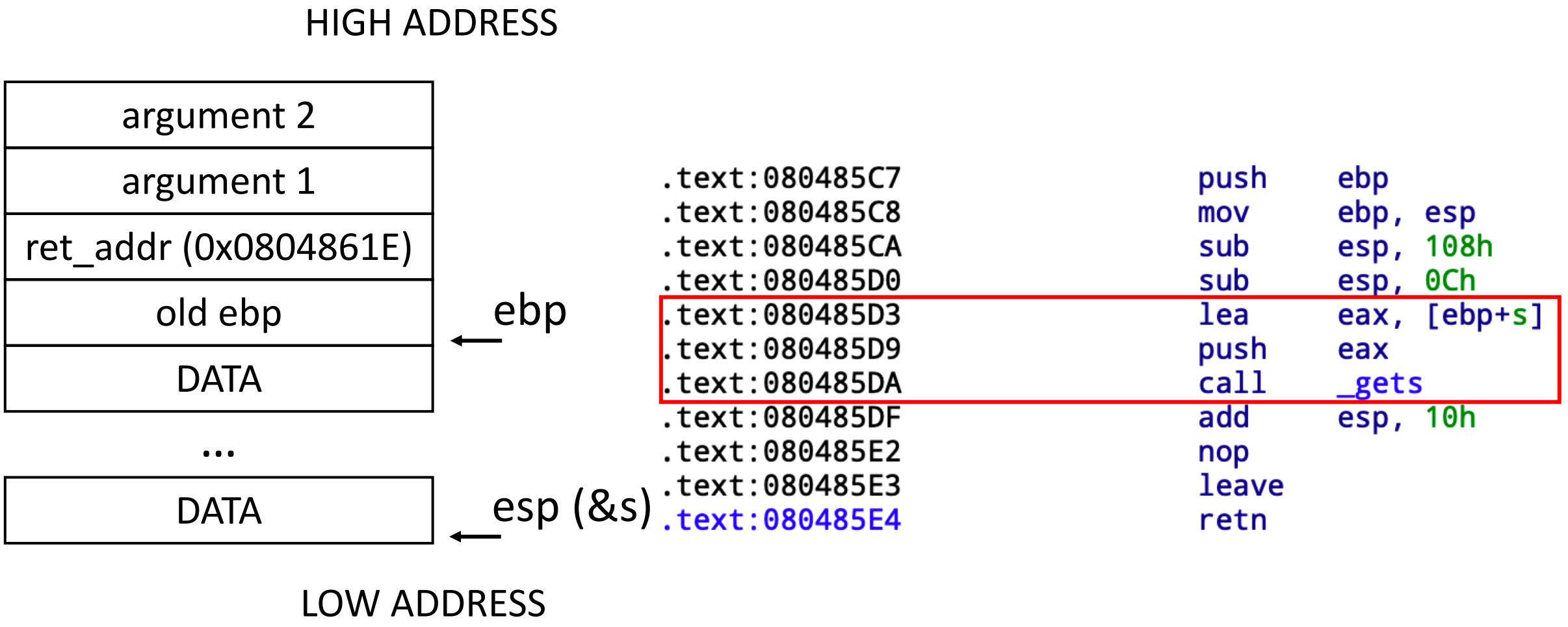

```
.text:080485C7    push    ebp
.text:080485C8    mov     ebp, esp
.text:080485CA    sub     esp, 108h
.text:080485D0    sub     esp, 0Ch
.text:080485D3    lea     eax, [ebp+s]
.text:080485D9    push    eax
.text:080485DA    call    _gets
.text:080485DF    add     esp, 10h
.text:080485E2    nop
.text:080485E3    leave
.text:080485E4    retn
```

LOW ADDRESS

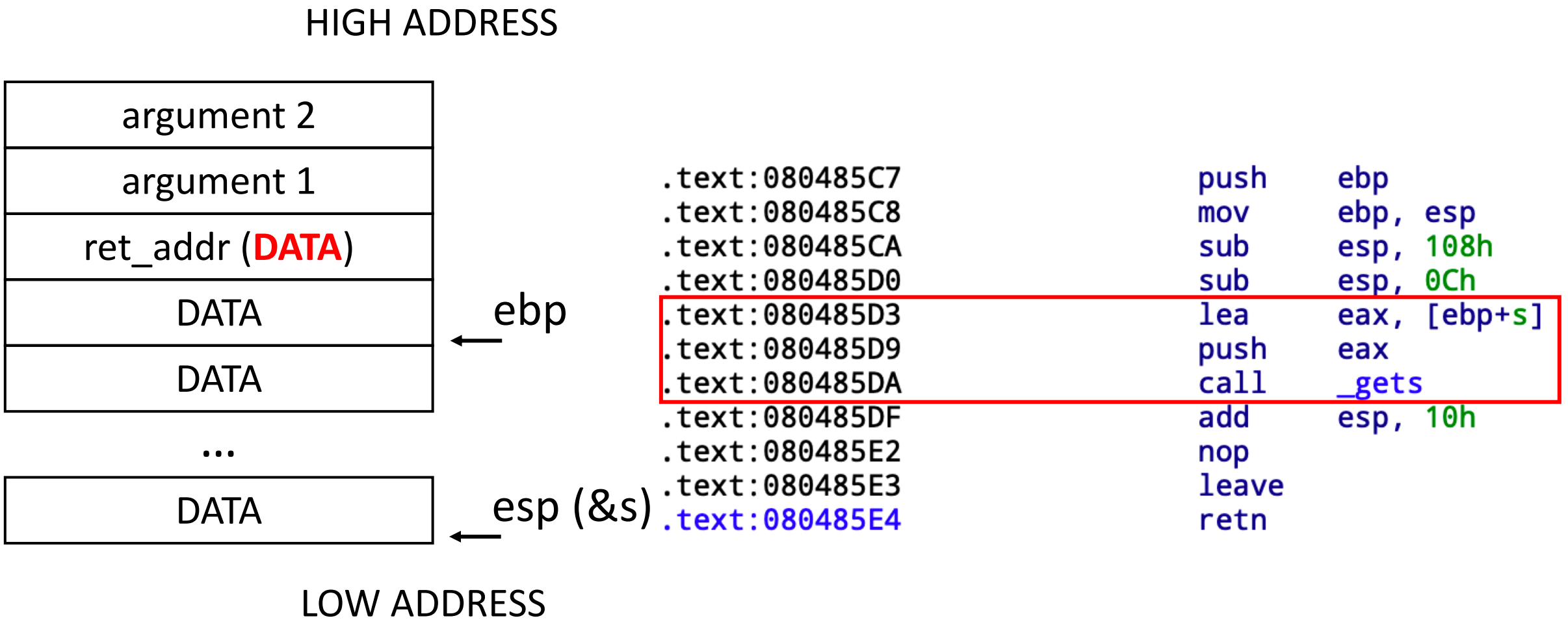
# retaddr


HIGH ADDRESS
argument 2
argument 1
ret_addr (0x0804861E)
old ebp
... (local variables)
...
... (local variables)
ebp
esp (&s)
.text:080485C7 push ebp
.text:080485C8 mov ebp, esp
.text:080485CA sub esp, 108h
.text:080485D0 sub esp, 0Ch
.text:080485D3 lea eax, [ebp+s]
.text:080485D9 push eax
.text:080485DA call _gets
.text:080485DF add esp, 10h
.text:080485E2 nop
.text:080485E3 leave
.text:080485E4 retn
LOW ADDRESS

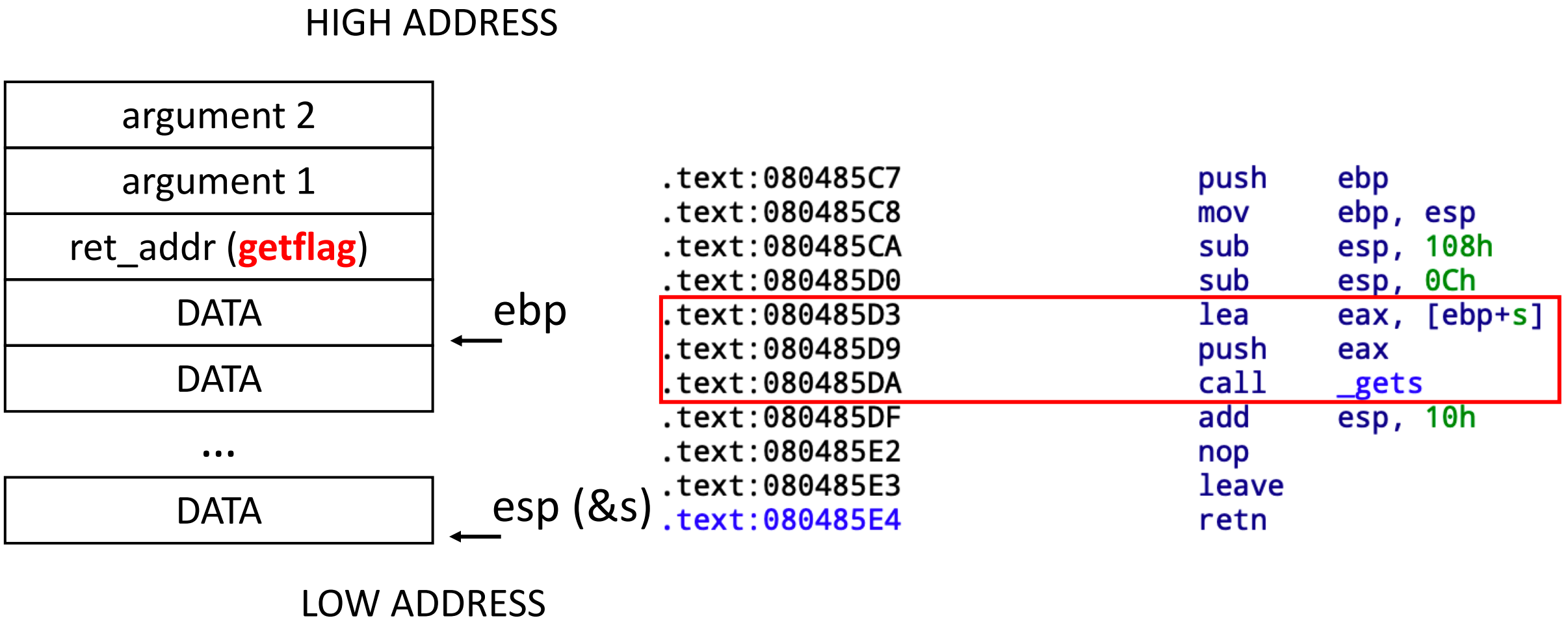
# retaddr


HIGH ADDRESS
argument 2
argument 1
ret_addr (0x0804861E)
old ebp
DATA
...
DATA
ebp
esp (&s)
.text:080485C7 push ebp
.text:080485C8 mov ebp, esp
.text:080485CA sub esp, 108h
.text:080485D0 sub esp, 0Ch
.text:080485D3 lea eax, [ebp+s]
.text:080485D9 push eax
.text:080485DA call _gets
.text:080485DF add esp, 10h
.text:080485E2 nop
.text:080485E3 leave
.text:080485E4 retn
LOW ADDRESS

# retaddr


HIGH ADDRESS
argument 2
argument 1
ret_addr (DATA)
DATA
DATA
...
DATA
ebp
esp (&s)
.text:080485C7 push ebp
.text:080485C8 mov ebp, esp
.text:080485CA sub esp, 108h
.text:080485D0 sub esp, 0Ch
.text:080485D3 lea eax, [ebp+s]
.text:080485D9 push eax
.text:080485DA call _gets
.text:080485DF add esp, 10h
.text:080485E2 nop
.text:080485E3 leave
.text:080485E4 retn
LOW ADDRESS

# retaddr


HIGH ADDRESS
argument 2
argument 1
ret_addr (getflag)
DATA
DATA
...
DATA
ebp
esp (&s)
.text:080485C7 push ebp
.text:080485C8 mov ebp, esp
.text:080485CA sub esp, 108h
.text:080485D0 sub esp, 0Ch
.text:080485D3 lea eax, [ebp+s]
.text:080485D9 push eax
.text:080485DA call _gets
.text:080485DF add esp, 10h
.text:080485E2 nop
.text:080485E3 leave
.text:080485E4 retn
LOW ADDRESS

# retaddr


HIGH ADDRE
argument 2
argument 1
ret_addr (getflag)
DATA
DATA
...
DATA
ebp
esp (&s)
LOW ADDRESS
TRY IT
.text:080485DF
.text:080485E2
.text:080485E3
.text:080485E4
push ebp
mov ebp, esp
sub esp, 108h
sub esp, 0Ch
lea eax, [ebp+s]
push eax
call _gets
add esp, 10h
nop
leave
retn

# retaddr: advanced attach

我们是否可以直接获得远程服务器的控制权限呢？

Remote Code Execution

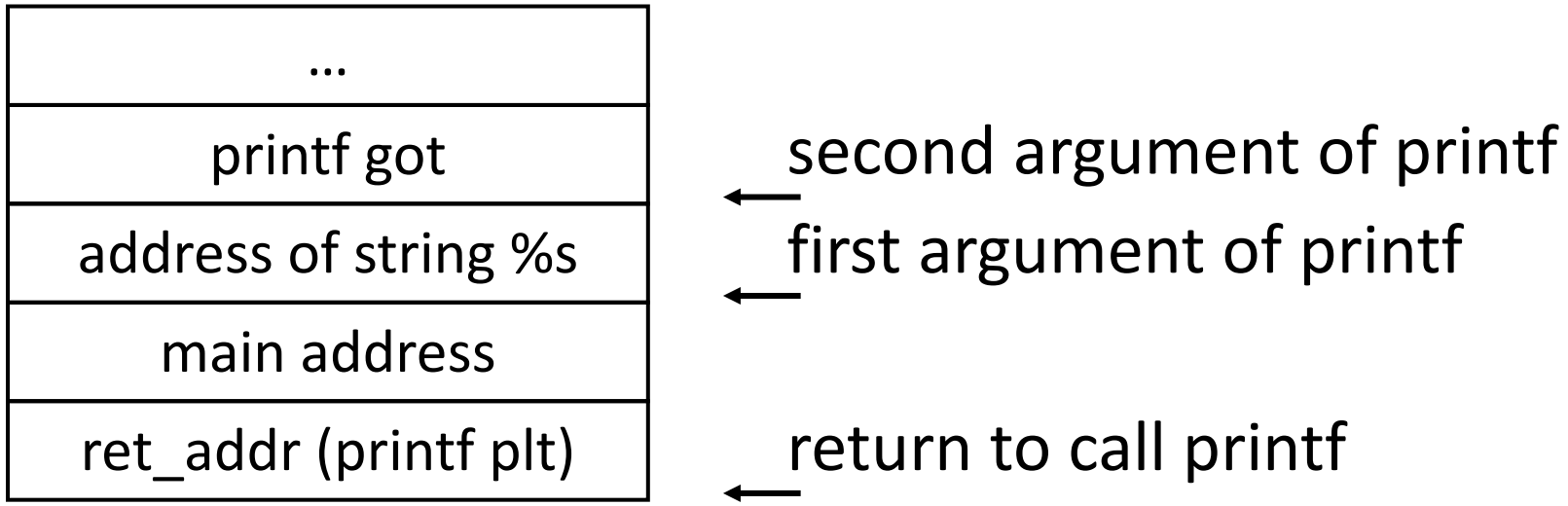
Challenge:

ASLR: 内存地址随机化
NX: 可写内存不可执行

Yes, We can!

# retaddr: advanced attach

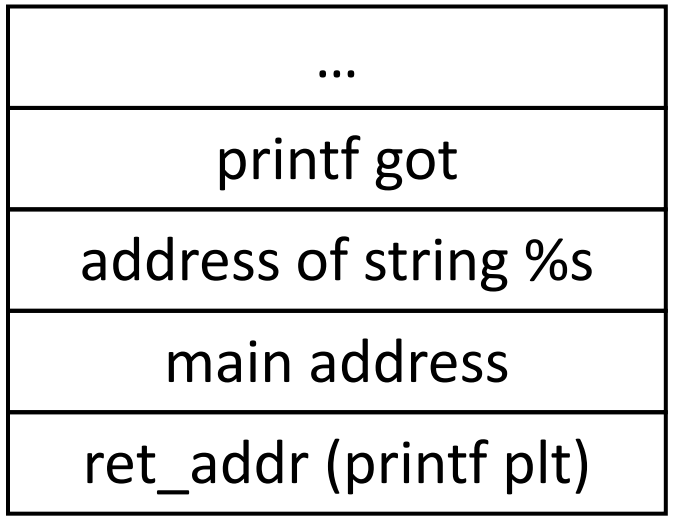
ASLR: Info Leak
NX: ROP




printf("%s", &printf's got)

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP




second argument of printf (real address of printf)

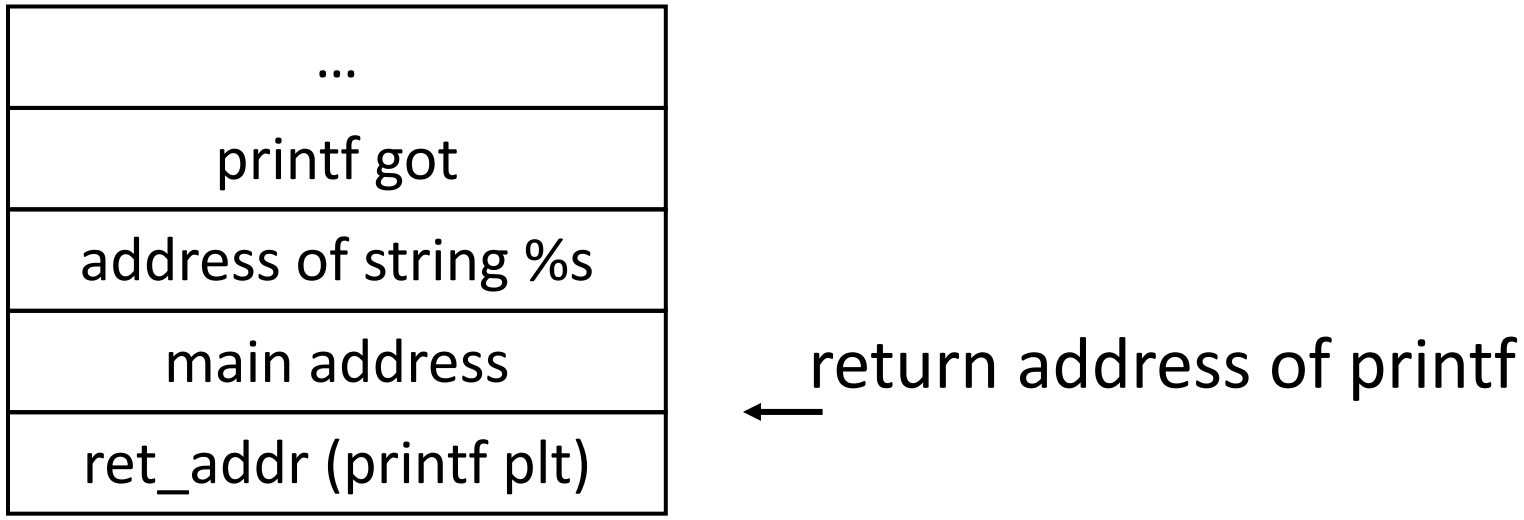
first argument of printf

return to call printf

printf("%s", &printf's got): INFO LEAK

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP

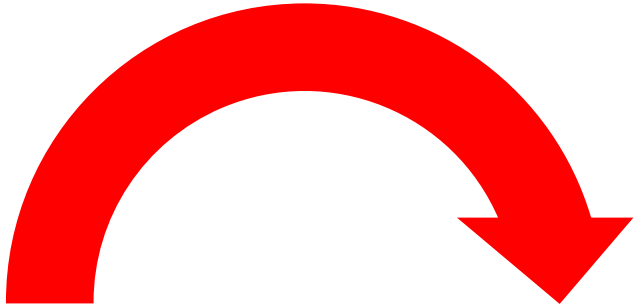



printf("%s", &printf's got) → main

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP

| … |
| --- |
| printf got |
| address of string %s |
| main address |
| ret_addr (printf plt) |



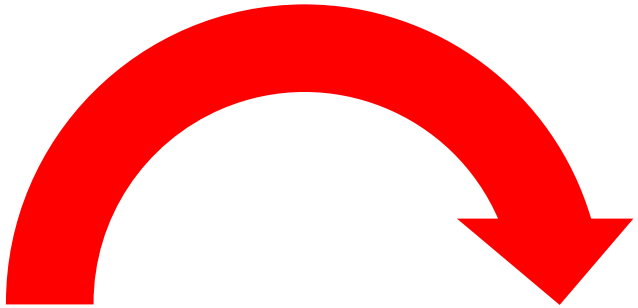
**STACK OVERFLOW AGAIN**

printf(“%s”, &printf’s got) → main

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP

| … |
| --- |
| gets got |
| address of string %s |
| main address |
| ret_addr (printf plt) |



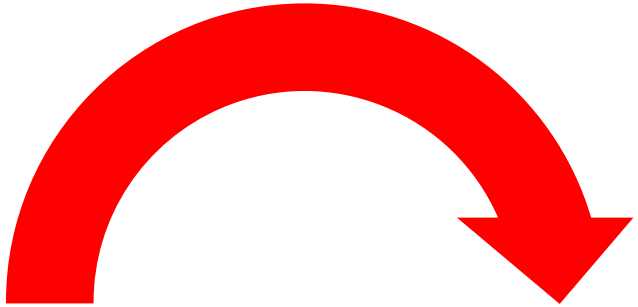
**STACK OVERFLOW AGAIN**

printf(“%s”, &printf’s got) → main → printf(“%s”, &gets’s got) → main

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP

| … |
| --- |
| XXX got |
| address of string %s |
| main address |
| ret_addr (printf plt) |



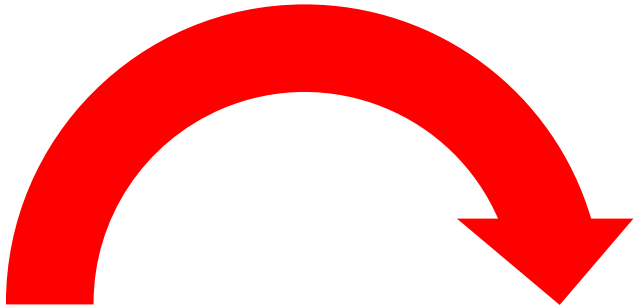
**ENOUGH INFORMATION TO IDNENTIFY LIBC VERSION**

printf(“%s”, &printf’s got) → main → printf(“%s”, &gets’s got) → main → … →

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP

| … |
| --- |
| XXX got |
| address of string %s |
| main address |
| ret_addr (printf plt) |



1. system function
2. “/bin/sh” string

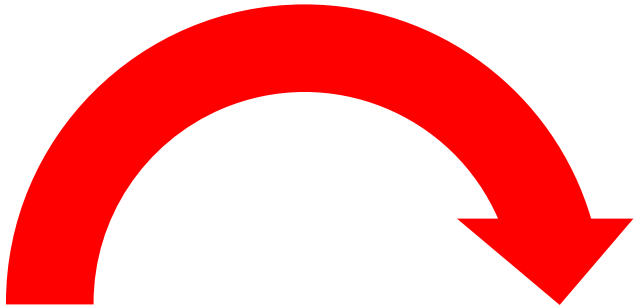
**ENOUGH INFORMATION TO IDNENTIFY LIBC VERSION**

printf(“%s”, &printf’s got) → main → printf(“%s”, &gets’s got) → main → … →

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP

| … |
| --- |
| … |
| string /bin/sh address |
| main address |
| ret_addr (system addr) |



1. system function
2. “/bin/sh” string

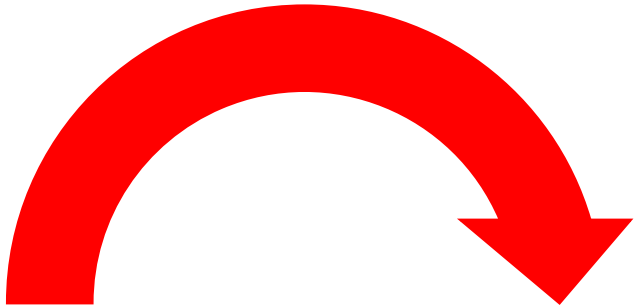
**ENOUGH INFORMATION TO IDNENTIFY LIBC VERSION**

printf(“%s”, &printf’s got) → main → printf(“%s”, &gets’s got) → main → … →

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP

| … |
| --- |
| … |
| string /bin/sh address |
| main address |
| ret_addr (system addr) |



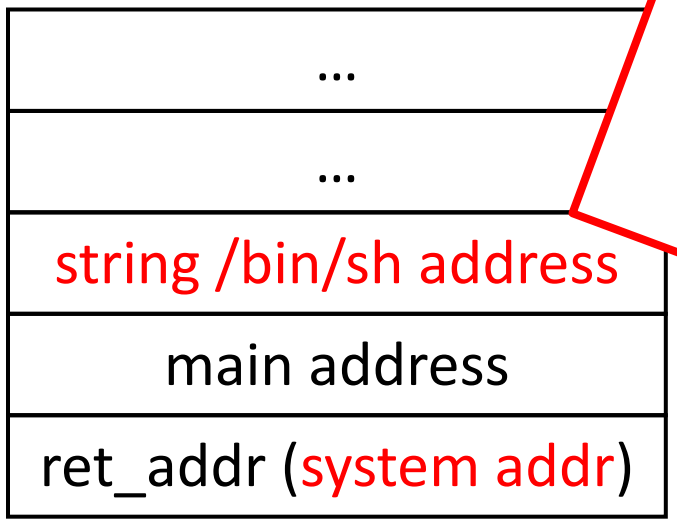
1. system function
2. “/bin/sh” string

**ENOUGH INFORMATION TO IDNENTIFY LIBC VERSION**

printf(“%s”, &printf’s got) → main → printf(“%s”, &gets’s got) → main → … → main → **system(“/bin/sh”)**

# retaddr: advanced attach

ASLR: Info Leak
NX: ROP





1. system function
2. "/bin/sh" string

**ENOUG TION**
**TO IDNENTIFY VERSION**

printf("%s", &printf's got) → main → printf("%s", &gets's got) → main → ... → main → **system("/bin/sh")**

# Pwnable 漏洞利用

- 漏洞利用的核心
  - 需要非常熟悉编译原理及程序底层行为
  - 需要非常熟悉各种漏洞形式及其相应的攻击方式
    - 栈溢出
    - 堆溢出
    - 格式化字符串攻击
    - Use After Free
    - 变量未初始化
    - ……
  - 更困难的题目会涉及linux kernel、依赖库glibc的实现细节等
- 漏洞利用所涉及的技巧过多，需要积累

# Pwnable 漏洞利用

- 漏洞利用的核心
  - 需要非常熟悉编译原理及程序底层行为
  - 需要非常熟悉各种漏洞形式及其相应的攻击方式
    - 栈溢出
    - 堆溢出
    - 格式化字符串攻击
    - Use After Free
    - 变量未初始化
    - ……
  - 更困难的题目会涉及linux kernel、依赖库glibc的实现细节等
- 漏洞利用所涉及的技巧过多，需要积累
- 练习平台：http://pwnable.kr/　　https://pwnable.tw/

# Crypto 密码学

- 题型简介
  - 提供一个存在漏洞的加密算法或加密协议，以及对应的密文或者使用该协议的远端服务 （substitution2）
  - 通过分析和破解加密算法，从密文中恢复出flag或者入侵远端服务器获得flag

# Crypto 密码学

- 题型简介
  - 提供一个存在漏洞的加密算法或加密协议，以及对应的密文或者使用该协议的远端服务
  - 通过分析和破解加密算法，从密文中恢复出flag或者入侵远端服务器获得flag
- 常用工具
  - **Sagemath**：开源数学软件，类似Matlab
  - Python的第三方库： pycrypto, gmpy2, etc.
  - 各种小工具：yafu、attackrsa, xortool, etc.

# **Crypto** 密码学

- 密码学的本质：**数学题**（样题讲解：easyrsa）

# RSA

- Two big prime numbers: ***p & q***
- Get ***n = p * q***
- Get ***phi(n) = (p - 1) * (q - 1)*** 欧拉函数
- Get e, s.t. ***gcd(e, phi(n)) = 1***
- Get d, s.t. ***e*d % phi(n) = 1***
- Public Key: ***(n, e)***
- Private Key: ***(n, d)***

# RSA

- Public Key: ***(n, e)***
- Private Key: ***(n, d)***
- Message: ***m***

- ***Encrypt: c = m^e % n***
- ***Decrypt: m = c^d % n***

- ***欧拉定理：m^(c*d) % n = m***

# Crypto 密码学

- 密码学的本质：**数学题**（样题讲解：easyrsa）
- 需要对高等数学有一定的了解：如有限域、循环群、线性代数、椭圆曲线、线性码等等
- 比较简单的题目往往可以通过现有工具解出，github上各种密码学破解工具极多，要善于查找，避免重复造轮子
- 有的困难题目需要查找相关密码学论文以寻求解法
- 练习平台： https://id0-rsa.pub/

# Web 网络安全

- 题型简介
  - 提供一个存在漏洞的网站
  - 通过寻找并利用网站中的漏洞以获得服务器权限，获得flag
- 常用工具
  - ？？？
- 题目设计知识面极广，技巧极多
  - 前端框架：jQuery, node.js, angularjs, etc.
  - 数据库系统：oracle, mssql, mysql, sqlite, mongdb, etc.
  - 后端语言：pho, python, java, etc.
  - Web Server：apache, nginx
  - WebAssembly
- 练习平台：http://www.wechall.net/

# 学习方法及资料

- **以赛代练，赛后总结（writeup）**
  - http://ctftime.org/（国际赛事集合）
  - https://www.xctf.org.cn/ （国内CTF赛事）
- 以往比赛资料
  - https://github.com/ctfs
- 单项练习（见前文）

CTF工具及资料多种多样，在阅读他人writeup的时候记得总结，高效使用**国内外**搜索引擎

# Thank You!