

# 软件安全—软件漏洞机理与防护

## V3 典型软件漏洞机理分析

---

彭国军 傅建明 教授

[guojpeng@whu.edu.cn](mailto:guojpeng@whu.edu.cn)

[jmfuwhu@126.com](mailto:jmfuwhu@126.com)

武汉大学国家网络安全学院

# 课程提纲

---

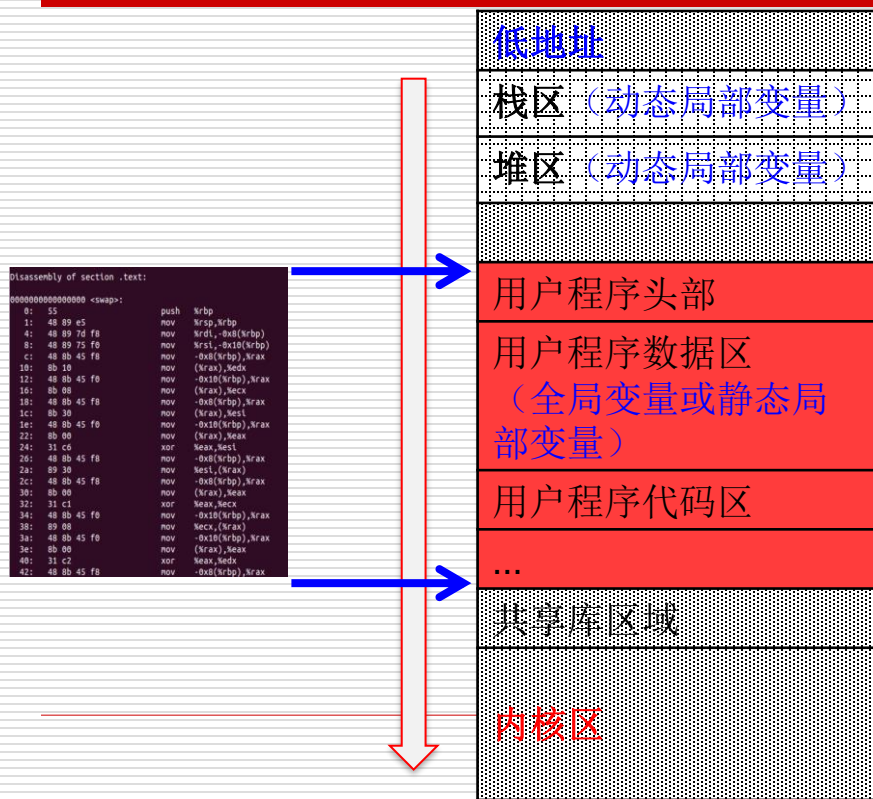
- 3.1 函数与栈帧
- 3.2 栈溢出机理
- 3.3 栈溢出利用
- 3.4 格式化字符串漏洞
- 3.5 整数溢出漏洞

# 3.1 函数与栈帧

---

- 程序由主函数和子函数构成
  - 函数之间进行嵌套调用
  - 函数在运行过程中，其自身数据及必要的中间数据放在哪里？
    - 内存缓冲区（通过变量进行引用）
      - 全局变量
        - 在函数外定义，在整个程序中有效
      - 局部变量
        - 在函数内定义，在函数内部有效

# 关键数据在内存中的存储位置



## 两个问题

### □ 数据访问:

- 动态分配空间的变量位置如何确定?

### □ 流程维护:

- 子函数执行完毕后如何返回当前函数

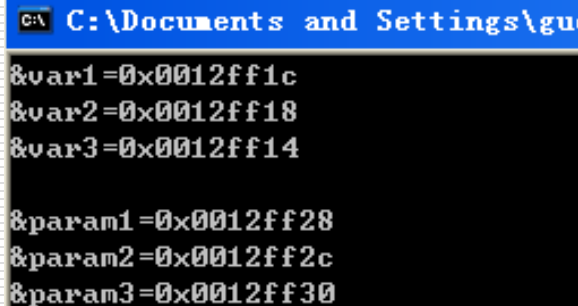
关键: 栈区

# 函数的调用机制

```
#include <stdio.h>
void __stdcall func(int param1,int param2,int param3)
{
    int var1=param1;
    int var2=param2;
    int var3=param3;
    //输出var1-3所在的位置
    printf("&var1=0x%08x\n",&var1);
    printf("&var2=0x%08x\n",&var2);
    printf("&var3=0x%08x\n\n",&var3);

    printf("&param1=0x%08x\n",&param1);
    printf("&param2=0x%08x\n",&param2);
    printf("&param3=0x%08x\n\n",&param3);
    //输出param1-3所在的位置
    return;
}

int main()
{
    func(1,2,3);
    return 0;
}
```



C:\Documents and Settings\gu...  
&var1=0x0012ff1c  
&var2=0x0012ff18  
&var3=0x0012ff14  
  
&param1=0x0012ff28  
&param2=0x0012ff2c  
&param3=0x0012ff30

# 函数的栈帧

## 栈帧构建过程

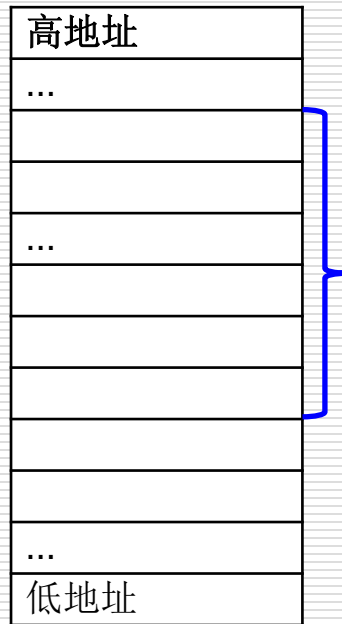
- 栈帧位于栈区，每个函数在运行之初**创建并开始维护**自己的栈帧
  - 栈底-EBP：每个函数固定
  - 栈顶-ESP：函数执行过程中动态变化
  - 局部变量空间分配：  
**ESP=ESP-XX**
  - 局部变量位置定位：**EBP-XX**

ESP-> EBP->

EBP-4->

EBP-8->

## 栈区示意图



# 函数的调用过程

```
22: func(1,2,3);
00401108 push 3
0040110A push 2
0040110C push 1
0040110E call 004010020
23: return 0;
00401113 xor eax,eax
```

## 1.子函数调用

```
3: void __stdcall
func(int param1,int param2,int param3)
4: {
00401020 push ebp
00401021 mov ebp,esp
00401023 sub esp,4Ch
00401026 push ebx
00401027 push esi
00401028 push edi
00401029 lea edi,[ebp-4Ch]
0040102C mov ecx,13h
00401031 mov eax,0CCCCCCCCh
00401036 rep stos dword ptr [edi]
5: int var1=param1;
00401038 mov eax,dword ptr [ebp+8]
0040103B mov dword ptr [ebp-4],eax
```

## 2.创建栈帧

## 3.分配局部变量

```
...
15: printf("&param3=0x%08x\n\n",&param3);
0040109F lea edx,[ebp+10h]
004010A2 push edx
004010A3 push offset string "&param3=0x%08x\n\n" (00422fb4)
004010A8 call printf (00401140)
004010AD add esp,8
16: //输出param1-3所在的位置
17: return;
18: }
004010B0 pop edi
004010B1 pop esi
004010B2 pop ebx
004010B3 add esp,4Ch
004010B6 cmp ebp,esp
004010B8 call __chkesp (004011c0)
004010BD mov esp,ebp
004010BF pop ebp
004010C0 ret 0Ch
```

## 4.销毁栈帧

## 5.返回到调用者

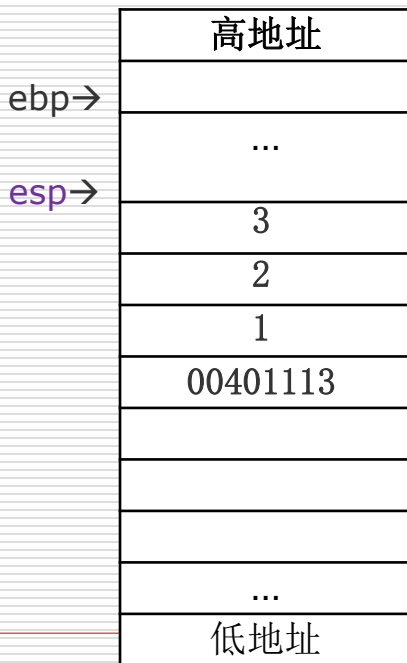
# 函数的调用过程

## 子函数调用

```
22:      func(1,2,3);  
00401108  push    3  
0040110A  push    2  
0040110C  push    1  
0040110E  call    004010020  
23:      return 0;  
00401113  xor     eax,eax
```

- **push指令：**
  - **数据压栈指令**
  - $esp=esp-4$ , **数据**->[esp]      子函数返回地址→
- **Call指令：**
  - $esp=esp-4$
  - **下一条指令地址**->[esp]
  - **Call目标地址**→EIP

栈区示意图



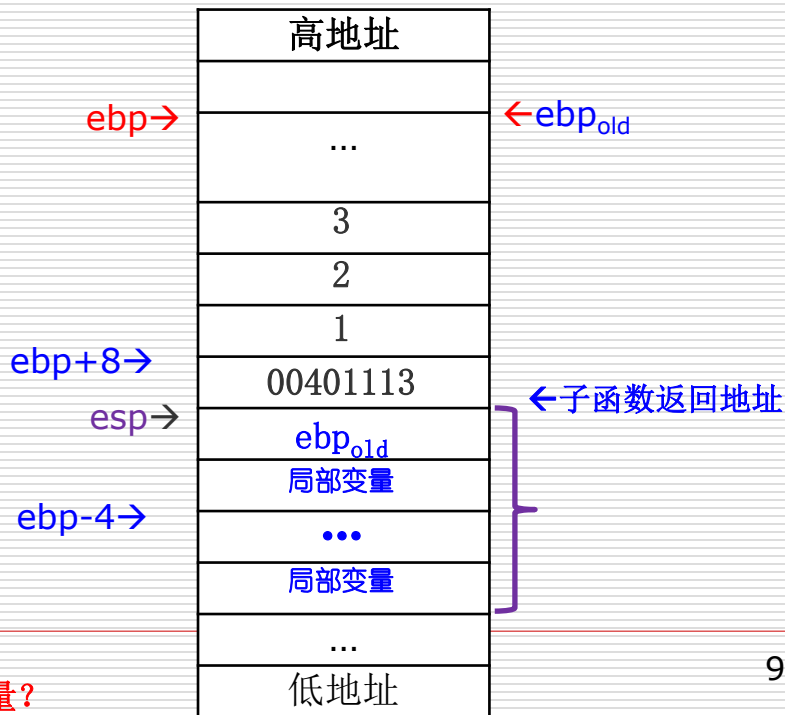


# 函数的调用过程

## 栈帧创建，局部变量空间分配

```
3: void __stdcall
func(int param1,int param2,int param3)
4: {
00401020 push    ebp
00401021 mov     ebp,esp
00401023 sub     esp,4Ch
00401026 push    ebx
00401027 push    esi
00401028 push    edi
00401029 lea     edi,[ebp-4Ch]
0040102C mov     ecx,13h
00401031 mov     eax,0CCCCCCCCh
00401036 rep stos dword ptr [edi]
5:   int var1=param1;
00401038 mov     eax,dword ptr [ebp+8]
0040103B mov     dword ptr [ebp-4],eax
```

## 栈区示意图



如何引用变量：外部变量，内部定义的局部变量？

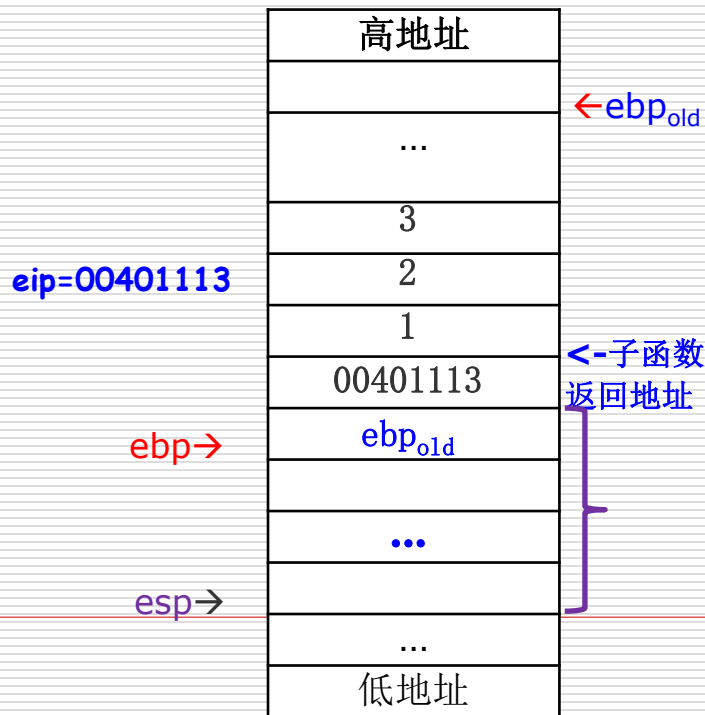
# 函数的调用过程

## 栈帧销毁与子函数返回

```
...  
004010BD  mov     esp,ebp  
004010BF  pop     ebp  
004010C0  ret     0Ch
```

- 出栈指令: `pop ebp`  
(1/2) `[esp]->ebp` : `ebp-old -> ebp`  
(2/2) `esp=esp+4`
- 返回指令: `ret 0Ch`  
(1/3) `[esp]->eip` : `eip=00401113`  
(2/3) `esp=esp+4`  
(2/3) `esp=esp+0Ch` //平衡调用者堆栈

栈区示意图

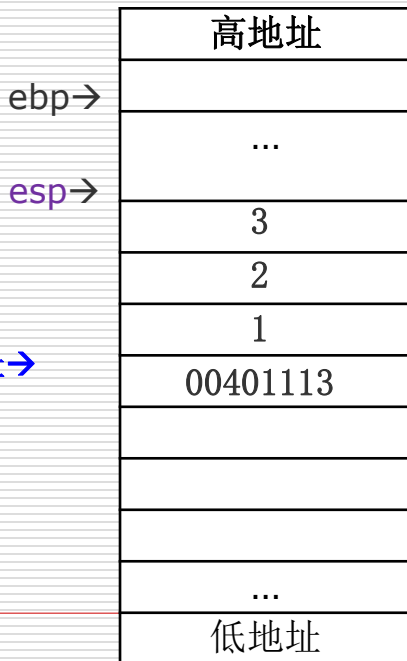


# 函数的调用过程-小结

## 子函数调用与返回

```
22:      func(1,2,3);
00401108  push      3
0040110A  push      2
0040110C  push      1
0040110E  call      004010020
23:      return 0;
00401113  xor       eax,eax 子函数返回地址→
```

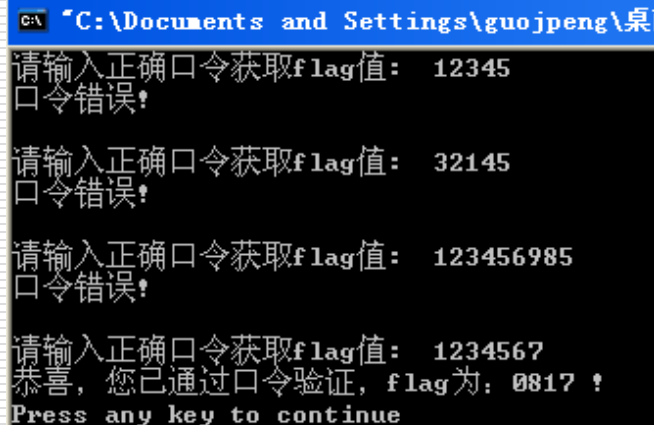
栈区示意图



## 3.2 栈溢出机理

```
#include <stdio.h>
#define PASSWORD "1234567"
int verify_password(char *password)
{
    int authenticated;
    char buffer[8]; // 局部变量
    authenticated = strcmp(password, PASSWORD);
    strcpy(buffer, password); // 拷贝用户输入的password到缓冲区buffer!
    return authenticated; // 返回验证值
}

main()
{
    int valid_flag = 0; // 全局变量初始化
    char passwd[1024]; // 构建缓冲区, 存储用户输入的口令字符串
    while(1)
    {
        printf("请输入正确口令获取flag值: ");
        scanf("%s", passwd);
        valid_flag = verify_password(passwd);
        if(valid_flag)
        {
            printf("口令错误!\n\n");
        }
        else
        {
            printf("恭喜, 您已通过口令验证, flag为: 0817 !\n");
            // 告知用户flag信息
        }
        break;
    }
}
```



```
C:\C:\Documents and Settings\guojpeng\桌面
请输入正确口令获取flag值: 12345
口令错误!

请输入正确口令获取flag值: 32145
口令错误!

请输入正确口令获取flag值: 123456985
口令错误!

请输入正确口令获取flag值: 1234567
恭喜, 您已通过口令验证, flag为: 0817 !
Press any key to continue
```



```
C:\C:\Documents and Settings\guojpeng\桌面\溢出代码范例
请输入正确口令获取flag值: 87654321
恭喜, 您已通过口令验证, flag为: 0817 !
Press any key to continue
```

# Strcpy函数的功能与缺陷

□ **Strcpy** (**buffer**, **passwd**)

□ **功能：** 将passwd指向的字符串（以“\0”作为结尾），全部拷贝到buffer指向的区域

□ **缺陷：** 即使passwd字符串长度大于buffer实际长度8

高地址
...
&passwd
函数返回地址
ebp-old
authenticated
buffer[4-7]
buffer[0-3]
...
低地址

# 继续溢出之后

## 子函数返回

```
0040106C  mov
esp,ebp
0040106E  pop    ebp
0040106F  ret    4
```

EBP=0x31323334

EIP=0x35363738

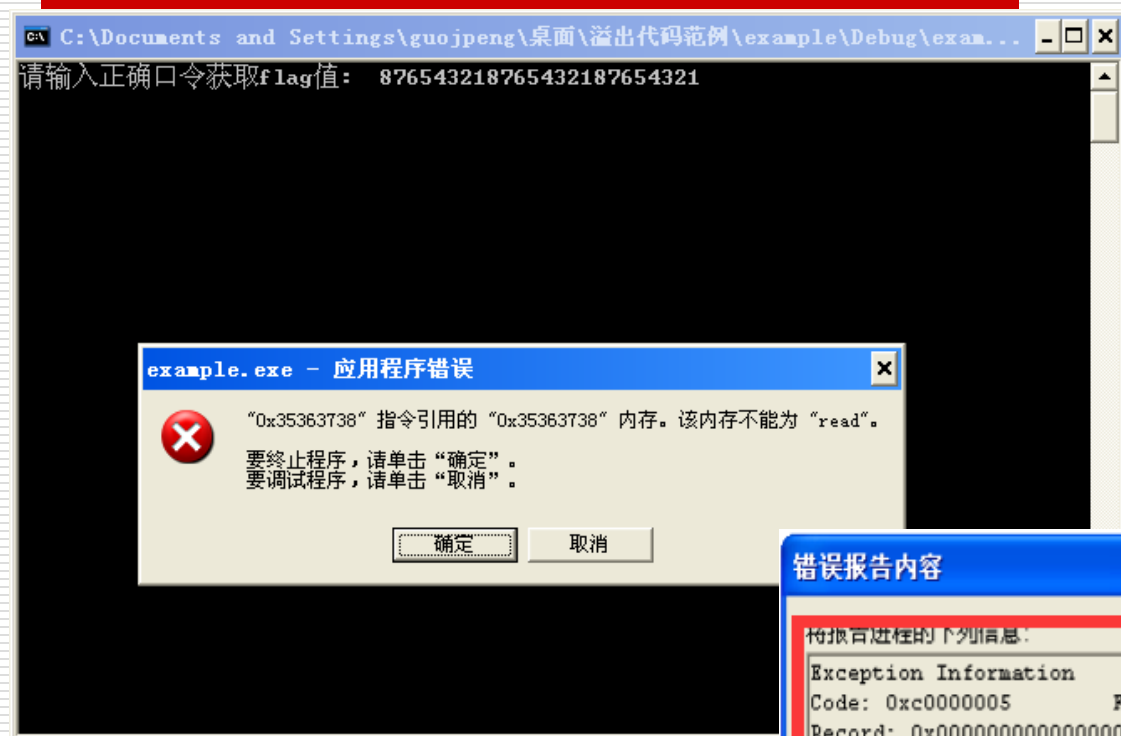
输入:  
876543218765432187654321

ebp

ebp→

esp→

高地址
...
&passwd: 0x34,0x33,0x32,0x31
正常函数返回地址:[0x35363738]
0x38,0x37,0x36,0x35
ebp-old: 0x34,0x33,0x32,0x31
Authenticated: 0x38,0x37,0x36,0x35
buff[4-7]: 0x34,0x33,0x32,0x31
buff[0-3]: 0x38,0x37,0x36,0x35
...
低地址



#### 错误报告内容

将报告进程的下列信息:

Exception Information

Code: 0xc0000005

Flags: 0x00000000

Record: 0x0000000000000000

Address: 0x0000000035363738

# 缓冲区溢出的根本原因

---

□ 没有内嵌支持的边界保护

□ 程序员安全编程技巧和意识



## 3.3 栈溢出攻击

-如何精心设计输入内容覆盖邻接变量

输入为**1234568**,

则**PASSWD**实际为:

0x31,0x32,0x33,0x34,0x35,0x36,0x38,0x00

高地址
...
&passwd: 0x12FB7C
正常函数返回地址:0x4010EB
ebp-old: 0x12FF80
Authenticated: 0x01,0x00,0x00,0x00
buffer[4-7]: 0x35,0x36,0x38,0x00
buffer[0-3]: 0x31,0x32,0x33,0x34
...
低地址

输入为**87654321**,

则**PASSWD**实际为:

0x38,0x37,0x36,0x35,0x34,0x33,0x32,0x31,0x00

高地址
...
&passwd: 0x12FB7C
正常函数返回地址:0x4010EB
ebp-old: 0x12FF80
Authenticated: 0x00,0x00,0x00,0x00
buffer[4-7]: 0x34,0x33,0x32,0x31
buffer[0-3]: 0x38,0x37,0x36,0x35
...
低地址

溢出前后【strcpy函数执行后】

## 3.3 栈溢出攻击

-如何精心设计输入内容执行指定代码

高地址
...
&passwd
函数返回地址
ebp_old
authenticated
buffer[4-7]
buffer[0-3]
...
低地址

正常栈帧结构

高地址
黑客可能输入什么？
...
低地址

黑客输入字符串导致的异常栈帧