

《漏洞利用及渗透测试基础》实验报告

姓名：齐明杰 学号：2113997 班级：信安2班

实验名称：

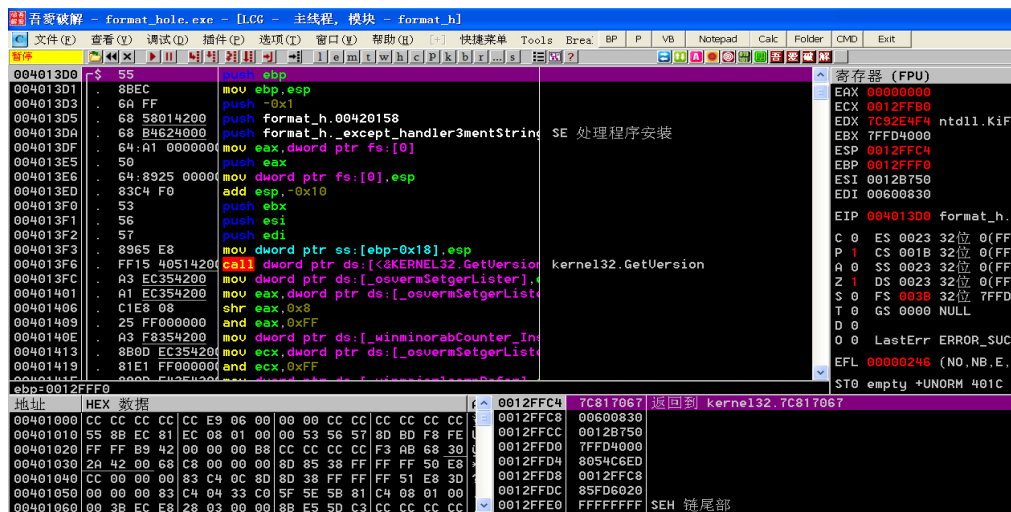
格式化字符串漏洞-任意地址的数据获取

实验要求：

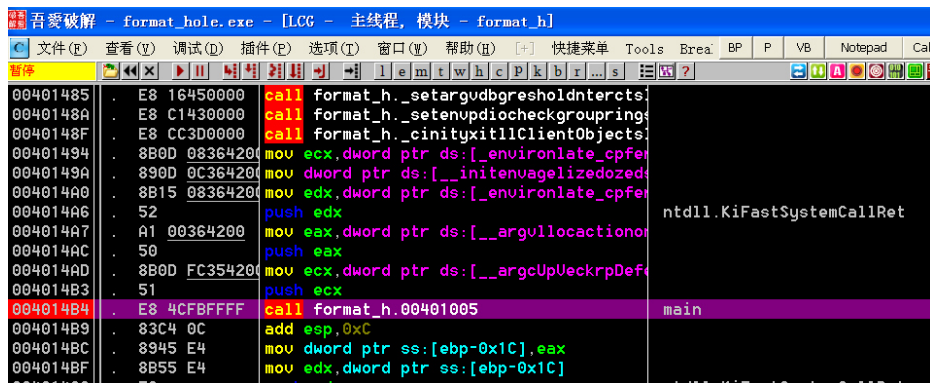
以第四章示例 4-7 代码，完成任意地址的数据获取，观察 Release 模式和 Debug 模式的差异，并进行总结

实验过程：

1. 输入示例代码，在 DEBUG 模式下编译生成 exe，拖入 ollydbg 运行如下图：



向下翻找到调用 main 函数的地址：



2. 单步跟入 main 函数中，发现栈帧的切换以及栈内初始化了很大一块区域：

sub esp, 0x108 为局部变量赋予的空间超出了数组 str 所需的空间(0xc8)。
 接下来三个入栈: push ebx, push esi, push edi 保存了这些寄存器的状态。

然后进行了栈区内容的初始化(赋值为 0xCCCCCCCC), 其中, ecx 作为计数器使用, 再执行 rep stos 语句循环对栈进行赋值:

```

lea edi, dword ptr ss:[ebp-0x108]
mov ecx, 0x42
mov eax, 0xCCCCCCCC
rep stos dword ptr es:[edi]

```

然后调用函数 fgets(str, 200, stdin), 三个 push 对应三个参数:

```

0040102E . 68 302A4200 push offset format_h._iobAllocHookId_lev
00401033 . 68 C8000000 push 0xC8
00401038 . 8D85 38FFFFFF lea eax, dword ptr ss:[ebp-0xC8]
0040103E . 50 push eax
0040103F . E8 CC000000 call format_h.fgetstIDeportModerruprings
00401044 . 83C4 0C add esp, 0xC

```

此时我们输入 AAAA%x%x%x%x, 如下图所示:



接下来是对 printf(str) 的调用, lea 语句获取 str 数组的起始地址, 然后将其入栈:

```

00401047 . 8D8D 38FFFFFF lea ecx, dword ptr ss:[ebp-0xC8]
0040104D . 51 push ecx
0040104E . E8 3D000000 call format_h.printfugdbgresholdntercts

```

值得注意的是, 在调用 call printf 前, 栈区的状态如下:

第一行是输入的格式化串, 第二到四行的值分别对应之前 push ebx, push esi, push edi 的结果, 后面连续的 0xCCCCCCCC 是前面赋值的结果

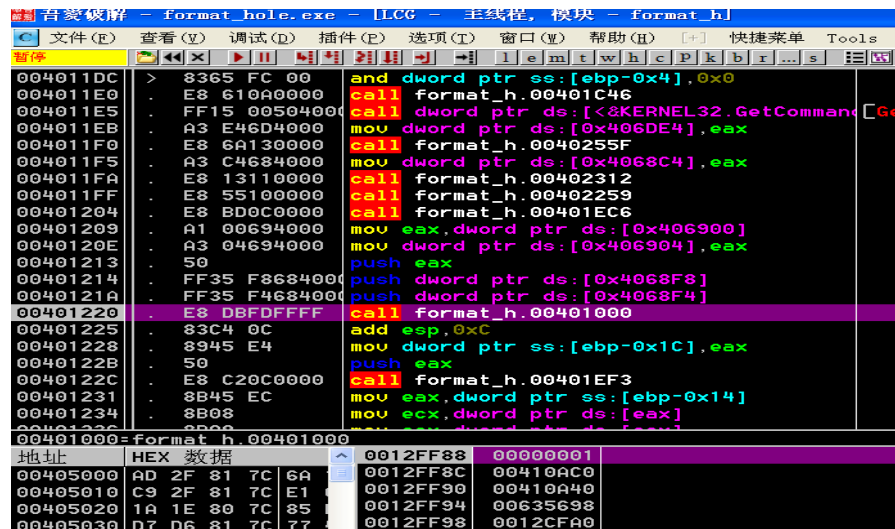
因此，printf(“AAAA%X%X%X%X”)此时会往栈内高地址继续读取四个“参数”，依次输出，分别为 00635698, 0012CFA0, 7FFDE000, CCCCCC 因此最后的输出结果为 AAAA63569812CFA07FFDE000CCCCCCC，如下图：

```

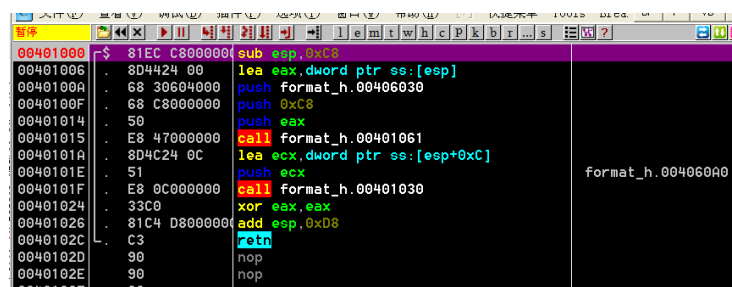
C:\Program Files\Microsoft Visual
AAAA%X%X%X%X
AAAA63569812CFA07FFDE000CCCCCCC

```

3. Vc6 切换到 release 模式，编译后拖入 od 运行，单步跟找到 main 函数的入口：



F7 跟入 main 函数中，反汇编代码如下：



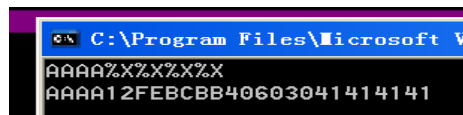
和 DEBUG 模式下反汇编代码对比可见，release 模式没有 push ebp, mov ebp, esp 的语句，同时 sub esp, 0xc8 正好为 str 数组的大小，没有空间的多余，而 DEBUG 模式下为 sub esp, 0x108，申请了大量的栈空间。

同时，release 模式下也没有对寄存器状态的保存(与之对应，DEBUG 模式下有 push ebx, push esi, push edi)，也没有利用 rep stos 语句来对栈区进行统一赋值。之后 3 个 push 和一个 call 对应了调用函数 fgets(str, 200, stdin)。

我们输入 AAAA%X%X%X%X，紧接着将调用 printf(str)，在 call printf 语句前栈区的状态如下图：

0040101A	804C24 0C	lea ecx,dword ptr ss:[esp+0xC]	
0040101E	51	push ecx	
0040101F	E8 0C000000	call format_h.00401030	
00401024	33C0	xor eax,eax	
00401026	81C4 D8000000	add esp,0xD8	
0040102C	C3	retn	
0040102D	90	nop	
0040102E	90	nop	
0040102F	90	nop	
00401030	53	push ebx	
00401031	56	push esi	
00401032	BE 50604000	mov esi,format_h.00406050	
00401037	57	push edi	
00401038	56	push esi	
00401039	E8 5B020000	call format_h.00401299	
0040103F	8BFA	mov edi,ebx	
00401030=format_h.00401030			
地址	HEX 数据	0012FEAC	0012FEB0 ASCII "AAAA%X%X%X\n"
00405000	AD 2F 81 7C 6A	0012FEB0	0012FEB0 ASCII "AAAA%X%X%X\n"
00405010	C9 2F 81 7C E1	0012FEB4	000000BB
00405020	1A 1E 80 7C 85	0012FEB8	00406030 UNICODE "溃"
00405030	D7 D6 81 7C 77	0012FEB8	41414141
00405040	98 2F 81 7C 88	0012FEC0	58255825
00405050	0D FF 92 7C A5	0012FEC4	58255825
00405060	12 18 80 7C E1	0012FEC8	0000000A

在 release 模式下，此时栈内第一行为 printf() 的参数 str，第二、三、四行为 fgets() 的三个参数，第五、六、七行存储了输入(AAAA%X%X%X)，因此，逻辑上跟 debug 模式一样，即 printf 此时会往栈内高地址继续读取四个“参数”，依次输出，但栈的内容却不一样，因此最终输出的结果有所区别：AAAA12FEB0BB406030414141，如下图所示：



通过以上分析，若把 AAAA 换成地址，第四个%X 换成%s，则可以完成对任意地址数据的获取

最后执行了 add esp, 0xD8, 这也是 release 模式和 debug 的一大区别，即 debug 在每次 call 后均会 add esp，而 release 将堆栈平衡进行了统一处理。

DEBUG 模式和 RELEASE 模式的差异总结:

- 1、RELEASE 模式下 main 函数不执行严格的栈帧转换(即 push ebp, mov ebp, esp)，也不对栈空间进行统一初始化(即 rep stos 指令)，也不通过 push 保存寄存器原来的值。
- 2、DEBUG 模式 main 函数一开始 sub esp 会分配更大的栈空间，char str[200]是从靠近 EBP 的地址分配空间，因此在 DEBUG 模式下如果要读到 str 的地址，需要很多的格式化字符
- 3、RELEASE 模式下并没有严格的栈帧分配，在 main 函数的 retn 语句前，才有 add esp 的处理

心得体会:

通过本次实验，我进一步掌握了 ollydbg 的使用方法，对软件调试更加熟悉。

同时理解了格式化字符串漏洞引起的问题，以及该漏洞的利用。

也了解了 DEBUG 和 RELEASE 模式下编译出 exe 的区别