

XCTF-csaw2013reversing2

原创

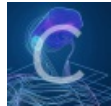
cr4ke3 于 2019-12-22 21:03:57 发布 458 收藏 1

分类专栏: [XCTF](#)

版权声明: 本文为博主原创文章, 遵循 [CC 4.0 BY-SA](#) 版权协议, 转载请附上原文出处链接和本声明。

本文链接: https://blog.csdn.net/weixin_43784056/article/details/103655968

版权



[XCTF 专栏收录该内容](#)

5 篇文章 0 订阅

订阅专栏

三种做法

1、ida静态分析修改指令

main函数反编译的代码

```
int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
    int v3; // ecx
    CHAR *lpMem; // [esp+8h] [ebp-Ch]
    HANDLE hHeap; // [esp+10h] [ebp-4h]

    hHeap = HeapCreate(0x40000u, 0, 0);
    lpMem = (CHAR *)HeapAlloc(hHeap, 8u, MaxCount + 1);
    memcpy_s(lpMem, MaxCount, &unk_409B10, MaxCount);
    if ( sub_40102A() || IsDebuggerPresent() ) // IsDebuggerPresent() 为检测动态调试的函数
    {
        debugbreak(); // 中断函数, 汇编语言为int 3
        sub_401000(v3 + 4, lpMem); // 生成flag的函数
        ExitProcess(0xFFFFFFFF); // 退出程序
    }
    MessageBoxA(0, lpMem + 1, "Flag", 2u); // 弹框输出flag
    HeapFree(hHeap, 0, lpMem);
    HeapDestroy(hHeap);
    ExitProcess(0);
}
```

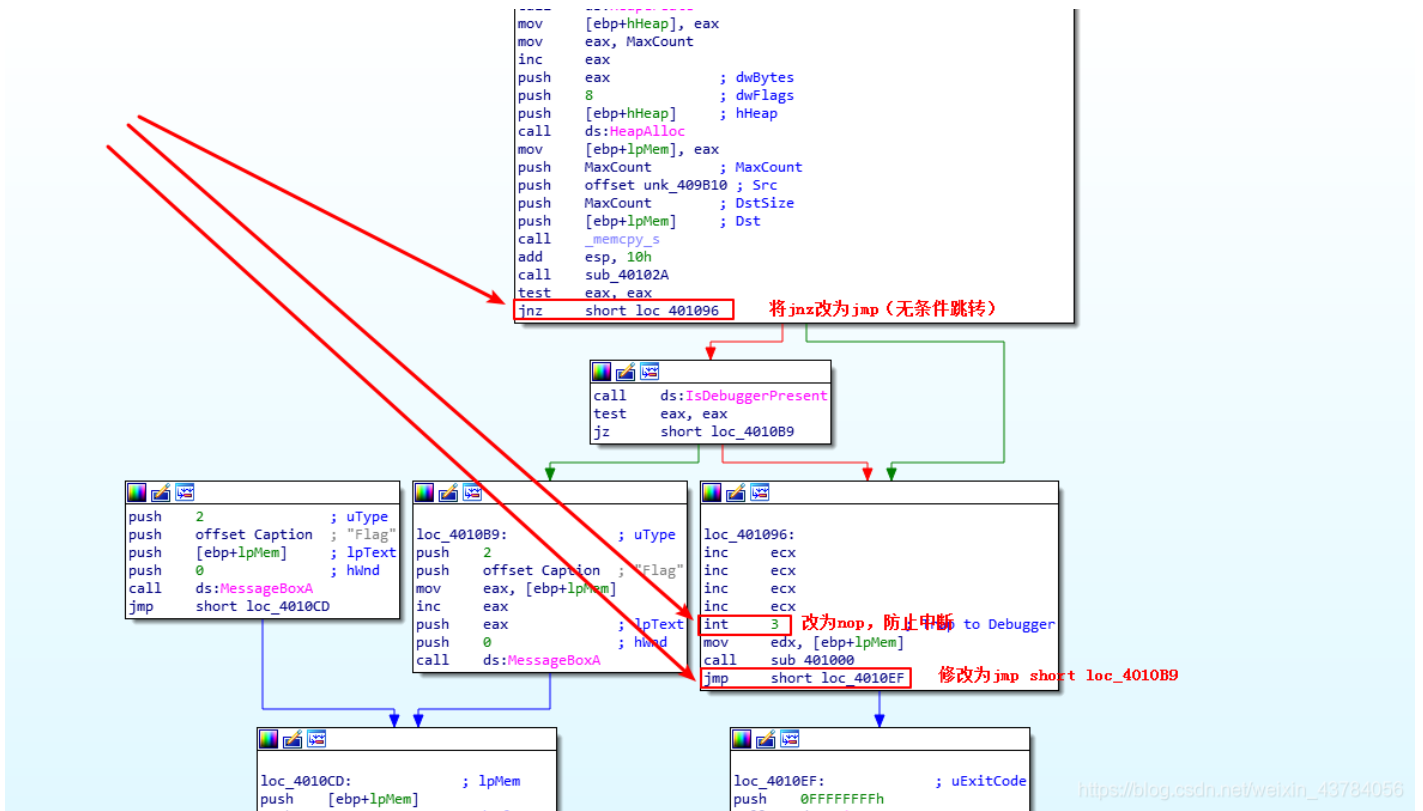
https://blog.csdn.net/weixin_43784056

由于运行之后的是乱码, 所以可以猜测生成flag的函数没有执行, 所以需要跳到生成flag的函数执行, 但是前面的中断函数不能执行, 需要nop掉, 并且后面退出程序的函数不能执行, 需要跳到弹框函数继续执行。(修改的路径和文件名不要有中文, 我用ida修改的时候踩了坑, 大家可以试一试)

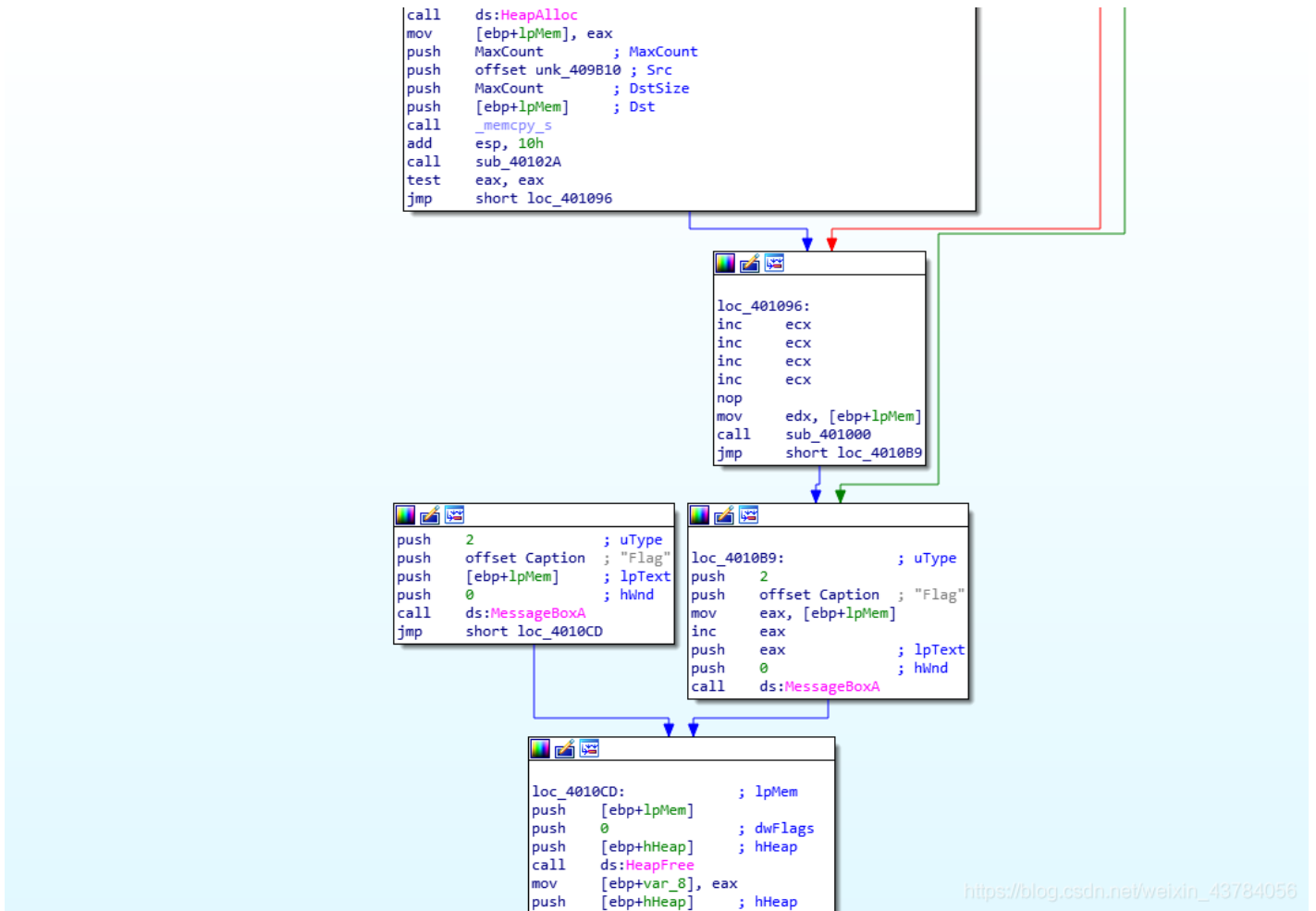
ida修改代码的方法:

- 1、鼠标停留在要修改的汇编代码上, 然后点击Edit > Patch program > Assemble (中文: 编辑 > 修补程序 > 汇编)
- 2、修改完成后: Edit > Patch program > Apply pathes to input file > OK (中文: 编辑 > 修补程序 > 修补程序应用到输入文件 > 确定)

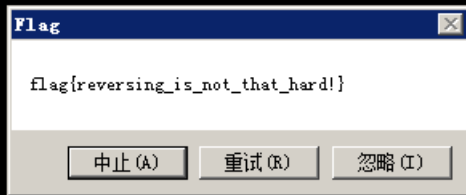
修改之前的汇编代码



修改之后的汇编代码



修改完成之后，直接运行文件，得到flag



https://blog.csdn.net/weixin_43784056

2、ollydbg动态调试，nop大法

将文件导入ollydbg后，直接右键 > 中文搜索引擎 > 智能搜索，找到Flag

地址	反汇编	文本字符串
01341001	mov esi, dword ptr ds:[0x1349B36]	华梯
01341052	mov eax, dword ptr ds:[0x1349B34]	\$
01341067	push dword ptr ds:[0x1349B34]	\$
01341072	push dword ptr ds:[0x1349B34]	\$
013410A7	push 03eef104.01347820	Flag
013410B8	push 03eef104.01347820	Flag
01341130	call 03eef104.01346837	Critical CPU selection
01341A45	push 03eef104.01346154	mscoree.dll
01341A54	push 03eef104.01346144	CorExitProcess
01341DBB	push 03eef104.01346E2C	Runtime Error!\n\nProgram:
01341DFC	push 03eef104.01346AFC	<program name unknown>
01341E3D	push 03eef104.01346AF4	...
01341E52	push 03eef104.01346AEC	\n\n
01341E83	push 03eef104.01346AA0	Microsoft Visual C++ Runtime Library
0134276B	push 03eef104.01346C00	KERNEL32.DLL
013429D8	push 03eef104.01346C00	KERNEL32.DLL
013429F9	push 03eef104.01346C3C	FlsAlloc
01342A01	push 03eef104.01346C30	FlsGetValue
01342A0E	push 03eef104.01346C24	FlsSetValue
01342A1B	push 03eef104.01346C1C	FlsFree
01343344	mov ecx, dword ptr ds:[0x1346BF4]	\t
01343791	push 03eef104.01346CAC	USER32.DLL
013437AC	push 03eef104.01346CA0	MessageBoxW
013437C5	push 03eef104.01346C90	GetActiveWindow
013437D5	push 03eef104.01346C7C	GetLastActivePopup
013437E5	push 03eef104.01346C60	GetObjectInformationW
013437FE	push 03eef104.01346C48	GetProcessWindowStation
013440FE	lea esi, dword ptr ds:[ecx+0x1349760]	?

https://blog.csdn.net/weixin_43784056

双击之后向上找到IsDebuggerPresent函数，点击这句汇编，F4让程序运行到此处

0134107B	-	E8 88000000	call 03eef104.01341108	
01341080	-	83C4 10	add esp,0x10	
01341083	-	E8 A2FFFFFF	call 03eef104.0134102A	
01341088	-	85C0	test eax,eax	
0134108A	-	75 0A	jnz short 03eef104.01341096	
0134108C	-	FF15 1460340	call dword ptr ds:[&KERNEL32.IsDebuggerPresent]	IsDebuggerPresent
01341092	-	85C0	test eax,eax	
01341094	-	74 23	je short 03eef104.013410B9	
01341096	>	41	inc ecx	
01341097	-	41	inc ecx	
01341098	-	41	inc ecx	
01341099	-	41	inc ecx	
0134109A	-	CC	int3	
0134109B	-	8B55 F4	mov edx,dword ptr ss:[ebp-0xC]	
0134109E	-	E8 5DFFFFFF	call 03eef104.01341000	
013410A3	-	EB 4A	jmp short 03eef104.013410EF	
013410A5	-	6A 02	push 0x2	
013410A7	-	68 20783401	push 03eef104.01347820	Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410AC	-	FF75 F4	push dword ptr ss:[ebp-0xC]	Flag
013410AF	-	6A 00	push 0x0	Text = "惶惇苳请拖井衷瓊屹擎贊擬轴競轴竟髯梯"
013410B1	-	FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410B7	-	EB 14	jmp short 03eef104.013410CD	
013410B9	>	6A 02	push 0x2	Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410BB	-	68 20783401	push 03eef104.01347820	Flag
013410C0	-	8B45 F4	mov eax,dword ptr ss:[ebp-0xC]	
013410C3	-	40	inc eax	
013410C4	-	50	push eax	Text = NULL
013410C5	-	6A 00	push 0x0	hOwner = NULL
013410C7	-	FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410CD	>	FF75 F4	push dword ptr ss:[ebp-0xC]	Memory = 00B907D0
013410D0	-	6A 00	push 0x0	Flags = 0
013410D2	-	FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410D5	-	FF15 0860340	call dword ptr ds:[&KERNEL32.HeapFree]	HeapFree
013410DB	-	8945 F8	mov dword ptr ss:[ebp-0x8],eax	
013410E1	-	FF15 0C60340	call dword ptr ds:[&KERNEL32.HeapDestroy]	HeapDestroy
013410E7	-	6A 00	push 0x0	hHeap = 00B90000
013410E9	-	FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	kernel32.ExitProcess
013410EF	>	6A FF	push -0x1	ExitCode = 0xFFFFFFFF
013410F1	-	FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	ExitProcess
013410F7	-	C9	leave	

F8两次，发现一个跳转，根据之前ida的分析，这应该就是那个if语句的判断，跳过的中间部分就是生成flag的函数，所以我们把这个跳转nop掉

0134107B	-	E8 88000000	call 03eef104.01341108	
01341080	-	83C4 10	add esp,0x10	
01341083	-	E8 A2FFFFFF	call 03eef104.0134102A	
01341088	-	85C0	test eax,eax	
0134108A	-	75 0A	jnz short 03eef104.01341096	
0134108C	-	FF15 1460340	call dword ptr ds:[&KERNEL32.IsDebuggerPresent]	
01341092	-	85C0	test eax,eax	
01341094	-	98	nop	
01341095	>	98	nop	
01341096	>	41	inc ecx	
01341097	-	41	inc ecx	
01341098	-	41	inc ecx	
01341099	-	41	inc ecx	
0134109A	-	CC	int3	
0134109B	-	8B55 F4	mov edx,dword ptr ss:[ebp-0xC]	
0134109E	-	E8 5DFFFFFF	call 03eef104.01341000	
013410A3	-	EB 4A	jmp short 03eef104.013410EF	
013410A5	-	6A 02	push 0x2	
013410A7	-	68 20783401	push 03eef104.01347820	Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410AC	-	FF75 F4	push dword ptr ss:[ebp-0xC]	Flag
013410AF	-	6A 00	push 0x0	Text = "惶惇苳请拖井衷瓊屹擎贊擬轴競轴竟髯梯"
013410B1	-	FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410B7	-	EB 14	jmp short 03eef104.013410CD	
013410B9	>	6A 02	push 0x2	Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410BB	-	68 20783401	push 03eef104.01347820	Flag
013410C0	-	8B45 F4	mov eax,dword ptr ss:[ebp-0xC]	
013410C3	-	40	inc eax	
013410C4	-	50	push eax	Text = NULL
013410C5	-	6A 00	push 0x0	hOwner = NULL
013410C7	-	FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410CD	>	FF75 F4	push dword ptr ss:[ebp-0xC]	Memory = 00B907D0
013410D0	-	6A 00	push 0x0	Flags = 0
013410D2	-	FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410D5	-	FF15 0860340	call dword ptr ds:[&KERNEL32.HeapFree]	HeapFree
013410DB	-	8945 F8	mov dword ptr ss:[ebp-0x8],eax	
013410DE	-	FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410E1	-	FF15 0C60340	call dword ptr ds:[&KERNEL32.HeapDestroy]	HeapDestroy
013410E7	-	6A 00	push 0x0	
013410E9	-	FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	kernel32.ExitProcess
013410EF	>	6A FF	push -0x1	ExitCode = 0xFFFFFFFF
013410F1	-	FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	ExitProcess
013410F7	-	C9	leave	

继续F8执行，执行到int 3，这是中断语句，所以也nop掉

0134107B	- E8 88000000	call 03eef104.01341108	
01341080	- 83C4 10	add esp,0x10	
01341083	- E8 A2FFFFFF	call 03eef104.0134102A	
01341088	- 85C0	test eax,eax	
0134108A	~ 75 0A	inc short 03eef104.01341096	
0134108C	- FF15 1460340	call dword ptr ds:[&KERNEL32.IsDebuggerPresent]	IsDebuggerPresent
01341092	- 85C0	test eax,eax	
01341094	90	nop	
01341095	90	nop	
01341096	> 41	inc ecx	
01341097	- 41	inc ecx	
01341098	- 41	inc ecx	
01341099	- 41	inc ecx	
0134109A	90	nop	
0134109B	- 8B55 F4	mov edx,dword ptr ss:[ebp-0xC]	
0134109E	- E8 5DFFFFFF	call 03eef104.01341000	
013410A3	~ EB 4A	jmp short 03eef104.013410EF	
013410A5	- 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410A7	- 68 20783401	push 03eef104.01347820	Flag
013410AC	- FF75 F4	push dword ptr ss:[ebp-0xC]	Text = "惶怍苙请拖井夷琮屹擎焚概轴兢轴竟霖梯"
013410AF	- 6A 00	push 0x0	hOwner = NULL
013410B1	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410B7	~ EB 14	jmp short 03eef104.013410CD	
013410B9	> 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410BB	- 68 20783401	push 03eef104.01347820	Flag
013410C0	- 8B45 F4	mov eax,dword ptr ss:[ebp-0xC]	
013410C3	- 40	inc eax	
013410C4	- 50	push eax	Text = NULL
013410C5	- 6A 00	push 0x0	hOwner = NULL
013410C7	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410CD	> FF75 F4	push dword ptr ss:[ebp-0xC]	Memory = 00B907D0
013410D0	- 6A 00	push 0x0	Flags = 0
013410D2	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410D5	- FF15 0860340	call dword ptr ds:[&KERNEL32.HeapFree]	HeapFree
013410DB	- 8945 F8	mov dword ptr ss:[ebp-0x8],eax	
013410DE	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410E1	- FF15 0C60340	call dword ptr ds:[&KERNEL32.HeapDestroy]	HeapDestroy
013410E7	- 6A 00	push 0x0	
013410E9	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	kernel32.ExitProcess
013410EF	> 6A FF	push -0x1	ExitCode = 0xFFFFFFFF
013410F1	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	ExitProcess
013410F7	- C9	leave	

https://blog.csdn.net/weixin_43784056

F8执行完生成flag的函数后，后面有一个大跳转，跳到退出程序的函数

0134107B	- E8 88000000	call 03eef104.01341108	
01341080	- 83C4 10	add esp,0x10	
01341083	- E8 A2FFFFFF	call 03eef104.0134102A	
01341088	- 85C0	test eax,eax	
0134108A	~ 75 0A	inc short 03eef104.01341096	
0134108C	- FF15 1460340	call dword ptr ds:[&KERNEL32.IsDebuggerPresent]	IsDebuggerPresent
01341092	- 85C0	test eax,eax	
01341094	90	nop	
01341095	90	nop	
01341096	> 41	inc ecx	
01341097	- 41	inc ecx	
01341098	- 41	inc ecx	
01341099	- 41	inc ecx	
0134109A	90	nop	
0134109B	- 8B55 F4	mov edx,dword ptr ss:[ebp-0xC]	
0134109E	- E8 5DFFFFFF	call 03eef104.01341000	
013410A3	~ EB 4A	jmp short 03eef104.013410EF	
013410A5	- 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410A7	- 68 20783401	push 03eef104.01347820	Flag
013410AC	- FF75 F4	push dword ptr ss:[ebp-0xC]	Text = ""
013410AF	- 6A 00	push 0x0	hOwner = NULL
013410B1	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410B7	~ EB 14	jmp short 03eef104.013410CD	
013410B9	> 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410BB	- 68 20783401	push 03eef104.01347820	Flag
013410C0	- 8B45 F4	mov eax,dword ptr ss:[ebp-0xC]	
013410C3	- 40	inc eax	
013410C4	- 50	push eax	Text = 00000009 ???
013410C5	- 6A 00	push 0x0	hOwner = NULL
013410C7	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410CD	> FF75 F4	push dword ptr ss:[ebp-0xC]	Memory = 00B907D0
013410D0	- 6A 00	push 0x0	Flags = 0
013410D2	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410D5	- FF15 0860340	call dword ptr ds:[&KERNEL32.HeapFree]	HeapFree
013410DB	- 8945 F8	mov dword ptr ss:[ebp-0x8],eax	
013410DE	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410E1	- FF15 0C60340	call dword ptr ds:[&KERNEL32.HeapDestroy]	HeapDestroy
013410E7	- 6A 00	push 0x0	
013410E9	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	kernel32.ExitProcess
013410EF	> 6A FF	push -0x1	ExitCode = 0xFFFFFFFF
013410F1	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	ExitProcess
013410F7	- C9	leave	

https://blog.csdn.net/weixin_43784056

所以我们将这个跳转也给nop掉，继续F8，执行完一个MessageBoxA（弹框）函数后，发现程序此时处于Running状态，弹出一个什么也没有的框，其实这是另外一个弹框函数，真正输出flag的弹框函数是后面那个，在我们之前那个ida的修改之后的汇编图也可以发现，确实是有个没有被调用的弹框函数，所以我们之前可以那个nop掉的跳转改为跳转到下面那个弹框函数，但既然说了是nop大法，就nop到底

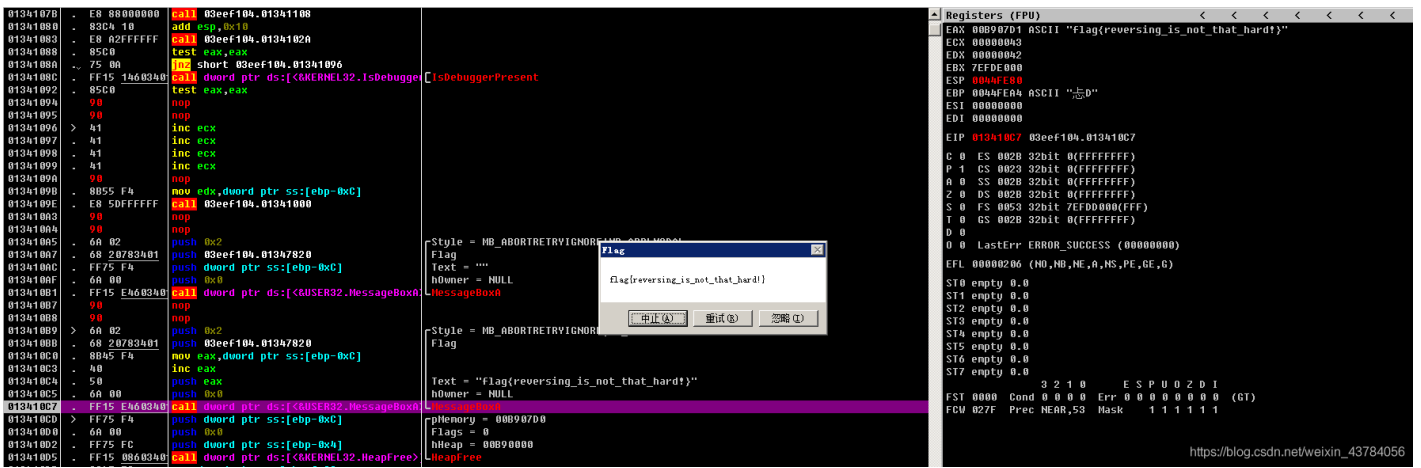
0134107B	- E8 88000000	call 03eef104.01341108	
01341080	- 83C4 10	add esp,0x10	
01341083	- E8 A2FFFFFF	call 03eef104.0134102A	
01341088	- 85C0	test eax,eax	
0134108A	- 75 0A	jnz short 03eef104.01341096	
0134108C	- FF15 1460340	call dword ptr ds:[&KERNEL32.IsDebuggerPresent]	[IsDebuggerPresent
01341092	- 85C0	test eax,eax	
01341094	- 90	nop	
01341095	- 90	nop	
01341096	> 41	inc ecx	
01341097	- 41	inc ecx	
01341098	- 41	inc ecx	
01341099	- 41	inc ecx	
0134109A	- 90	nop	
0134109B	- 8B55 F4	mov edx,dword ptr ss:[ebp-0xC]	
0134109E	- E8 5DFFFFFF	call 03eef104.01341000	
013410A3	- 90	nop	
013410A4	- 90	nop	
013410A5	- 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410A7	- 68 20783401	push 03eef104.01347820	Flag
013410AC	- FF75 F4	push dword ptr ss:[ebp-0xC]	Text = 00000002 ???
013410AF	- 6A 00	push 0x0	hOwner = NULL
013410B1	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410B7	- EB 14	jmp short 03eef104.013410CD	
013410B9	> 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410BB	- 68 20783401	push 03eef104.01347820	Flag
013410C0	- 8B45 F4	mov eax,dword ptr ss:[ebp-0xC]	
013410C3	- 40	inc eax	
013410C4	- 50	push eax	Text = NULL
013410C5	- 6A 00	push 0x0	hOwner = NULL
013410C7	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410CD	> FF75 F4	push dword ptr ss:[ebp-0xC]	hMemory = 00000002
013410D0	- 6A 00	push 0x0	Flags = 0
013410D2	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = NULL
013410D5	- FF15 0860340	call dword ptr ds:[&KERNEL32.HeapFree]	HeapFree
013410DB	- 8945 F8	mov dword ptr ss:[ebp-0x8],eax	
013410DE	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = NULL
013410E1	- FF15 0C60340	call dword ptr ds:[&KERNEL32.HeapDestroy]	HeapDestroy
013410E7	- 6A 00	push 0x0	
013410E9	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	kernel32.ExitProcess
013410EF	> 6A FF	push -0x1	ExitCode = 0xFFFFFFFF
013410F1	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	ExitProcess



点击中止之后，发现又要执行一个跳转，跳过了我们真正的弹框函数

0134107B	- E8 88000000	call 03eef104.01341108	
01341080	- 83C4 10	add esp,0x10	
01341083	- E8 A2FFFFFF	call 03eef104.0134102A	
01341088	- 85C0	test eax,eax	
0134108A	- 75 0A	jnz short 03eef104.01341096	
0134108C	- FF15 1460340	call dword ptr ds:[&KERNEL32.IsDebuggerPresent]	[IsDebuggerPresent
01341092	- 85C0	test eax,eax	
01341094	- 90	nop	
01341095	- 90	nop	
01341096	> 41	inc ecx	
01341097	- 41	inc ecx	
01341098	- 41	inc ecx	
01341099	- 41	inc ecx	
0134109A	- 90	nop	
0134109B	- 8B55 F4	mov edx,dword ptr ss:[ebp-0xC]	
0134109E	- E8 5DFFFFFF	call 03eef104.01341000	
013410A3	- 90	nop	
013410A4	- 90	nop	
013410A5	- 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410A7	- 68 20783401	push 03eef104.01347820	Flag
013410AC	- FF75 F4	push dword ptr ss:[ebp-0xC]	Text = ""
013410AF	- 6A 00	push 0x0	hOwner = NULL
013410B1	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410B7	- EB 14	jmp short 03eef104.013410CD	
013410B9	> 6A 02	push 0x2	-Style = MB_ABORTRETRYIGNORE MB_APPLMODAL
013410BB	- 68 20783401	push 03eef104.01347820	Flag
013410C0	- 8B45 F4	mov eax,dword ptr ss:[ebp-0xC]	
013410C3	- 40	inc eax	
013410C4	- 50	push eax	Text = 00000003 ???
013410C5	- 6A 00	push 0x0	hOwner = NULL
013410C7	- FF15 E460340	call dword ptr ds:[&USER32.MessageBoxA]	MessageBoxA
013410CD	> FF75 F4	push dword ptr ss:[ebp-0xC]	hMemory = 00B907D0
013410D0	- 6A 00	push 0x0	Flags = 0
013410D2	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410D5	- FF15 0860340	call dword ptr ds:[&KERNEL32.HeapFree]	HeapFree
013410DB	- 8945 F8	mov dword ptr ss:[ebp-0x8],eax	
013410DE	- FF75 FC	push dword ptr ss:[ebp-0x4]	hHeap = 00B90000
013410E1	- FF15 0C60340	call dword ptr ds:[&KERNEL32.HeapDestroy]	HeapDestroy
013410E7	- 6A 00	push 0x0	
013410E9	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	kernel32.ExitProcess
013410EF	> 6A FF	push -0x1	ExitCode = 0xFFFFFFFF
013410F1	- FF15 0060340	call dword ptr ds:[&KERNEL32.ExitProcess]	ExitProcess

将这个跳转nop掉，接着F8，就可以看到flag了



3、分析代码写脚本

main函数代码

```

int __cdecl __noreturn main(int argc, const char **argv, const char **envp)
{
    int v3; // ecx
    CHAR *lpMem; // [esp+8h] [ebp-Ch]
    HANDLE hHeap; // [esp+10h] [ebp-4h]

    hHeap = HeapCreate(0x40000u, 0, 0);
    lpMem = (CHAR *)HeapAlloc(hHeap, 8u, MaxCount + 1);
    memcpy_s(lpMem, MaxCount, &unk_409B10, MaxCount);
    if ( sub_40102A() || IsDebuggerPresent() )
    {
        __debugbreak();
        sub_401000(v3 + 4, lpMem);
        ExitProcess(0xFFFFFFFF);
    }
    MessageBoxA(0, lpMem + 1, "Flag", 2u);
    HeapFree(hHeap, 0, lpMem);
    HeapDestroy(hHeap);
    ExitProcess(0);
}

```

关键函数sub_401000的两个参数，v3后面没有用到，向上找lpMem的赋值语句，memcpy_s，将unk_409B10地址的值给了它，双击查看

```

.data:00409B0C          db  0
.data:00409B0D          db  0
.data:00409B0E          db  0
.data:00409B0F          db  0
.data:00409B10 unk_409B10 db 0BBh          ; DATA XREF: _main+33f0
.data:00409B11          db 0CCh
.data:00409B12          db 0A0h
.data:00409B13          db 0BCh
.data:00409B14          db 0DCh
.data:00409B15          db 0D1h
.data:00409B16          db 0BEh
.data:00409B17          db 0B8h
.data:00409B18          db 0CDh
.data:00409B19          db 0CFh
.data:00409B1A          db 0BEh
.data:00409B1B          db 0AEh
.data:00409B1C          db 0D2h
.data:00409B1D          db 0C4h
.data:00409B1E          db 0ABh
.data:00409B1F          db 82h
.data:00409B20          db 0D2h
.data:00409B21          db 0D9h
.data:00409B22          db 93h
.data:00409B23          db 0B3h
.data:00409B24          db 0D4h
.data:00409B25          db 0DEh
.data:00409B26          db 93h
.data:00409B27          db 0A9h
.data:00409B28          db 0D3h
.data:00409B29          db 0CBh
.data:00409B2A          db 0B8h
.data:00409B2B          db 82h
.data:00409B2C          db 0D3h
.data:00409B2D          db 0CBh
.data:00409B2E          db 0BEh
.data:00409B2F          db 0B9h
.data:00409B30          db 9Ah
.data:00409B31          db 0D7h
.data:00409B32          db 0CCh
.data:00409B33          db 0DDh
.data:00409B34 ; rsize_t MaxCount
.data:00409B34 MaxCount dd 24h          ; DATA XREF: _main+18f0

```

https://blog.csdn.net/weixin_43784056

进入sub_401000函数内部，代码

```

unsigned int __fastcall sub_401000(int a1, int a2)
{
    int v2; // esi
    unsigned int v3; // eax
    unsigned int v4; // ecx
    unsigned int result; // eax

    v2 = dword_409B38;
    v3 = a2 + 1 + strlen((const char *)(a2 + 1)) + 1;
    v4 = 0;
    result = ((v3 - (a2 + 2)) >> 2) + 1;
    if ( result )
    {
        do
            *(_DWORD *)(a2 + 4 * v4++) ^= v2;
        while ( v4 < result );
    }
    return result;
}

```

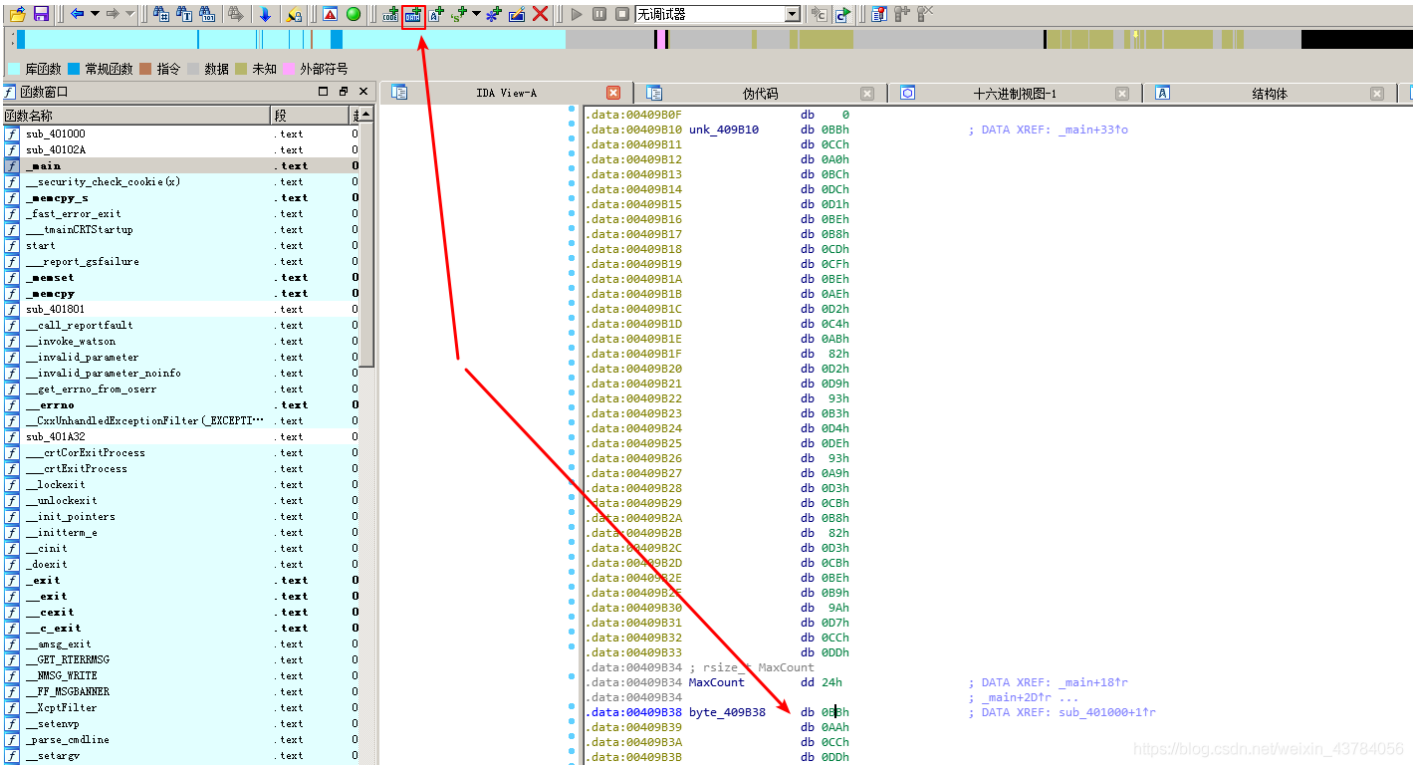
a2也就是lpMem，发现后面的异或语句有v2，向上找v2的赋值语句，找到v2 = dword_409B38，双击dword_409B38，找到内容


```

.data:00409B34 MaxCount      dd 24h                ; DATA XREF: _main+181r
.data:00409B34              ; _main+2D1r ...
.data:00409B38 dword_409B38  dd 0DDCCAABh         ; DATA XREF: sub_401000+11r
.data:00409B3C              align 10h
.data:00409B40 ; char *dword_409B40
.data:00409B40 dword_409B40  dd 0                  ; DATA XREF: __tmainCRTStartup+B51w
.data:00409B40              ; __setenvp+F1r ...

```

这里是四个字节显示的，又由于小端存储，所以顺序是颠倒的，我们可以将其转换成一个字节查看



然后根据源码写脚本，写的有点不太明白，记录一下

代码参照文章：<https://www.cnblogs.com/DirWang/p/11420740.html>

```

x=[0xbb,0xaa,0xcc,0xdd]
y=[0xBB,0xCC,0xA0,0xBC,0xDC,0xD1,0xBE,0xB8,0xCD,0xCF,0xBE,0xAE,0xD2,0xC4,0xAB,0x82,0xD2,0xD9,0x93,0xB3,0xD4]
i=0
z=[]
while i<len(y):
    t=chr(y[i]^x[i%4])
    z.append(t)
    i+=1
print(z)
print(''.join(z))

```

```

['\x00', 'f', 'l', 'a', 'g', '{', 'r', 'e', 'v', 'e', 'r', 's', 'i', 'n', 'g', '_', 'i', 's', '_', 'n', 'o', 't', '_', 't', 'h', 'a', 't', '_', 'h', 'a', 'r', 'd', '!', '}',
'\x00', '\x00']
flag{reversing_is_not_that_hard!}

```

这里就可以知道为什么调用第一个弹窗会输出空白，因为第一个弹窗函数，是直接第一个字符输出的，但是第一个字符解码后为'\0'，直接截断，所以会输出空白，第二个弹窗是从lpMem+1开始输出的