Skype v5.9.0.123 and Below Remote Default Unauthenticated Off-By-One

Table of Contents

Updates

- 6/10/12: initial version
- 6/11/12: changes reflecting a better understanding of some consequences of the vulnerability

Summary

An off-by-one (or off-by-two in certain cases) vulnerability exists in Skype command 56, which is processed by default by Skype clients when routed via a client's Supernode (using command 53). This vulnerability results in the overwrite of:

- either the least significant byte(s) of a pointer that will later be processed;
- or of heap meta-data.

RCE is likely but was not demonstrated (I would guess exploitability 1 if someone already has a Skype library, 3 otherwise).

I will not provide a PoC, however it shouldn't be a problem for Skype people to reproduce the bug with the example down below.

Vulnerability

Command 56, designed to be sent by a node's Supernode, requires a DWORD array object (type 6) of id 33. This array is formatted into a string and logged before being further processed. Unfortunately the formatting of the string doesn't account for worst case scenarios where one or two bytes could overflow before the buffer could be extended.

To start with, here is the organization of the stack variables of the function processing command 56 in Skype 5.9.0.123:

```
.text:0076A3C0                  xxxCommand56    proc near               ; DATA XREF: .text:010B6C44↓o
.text:0076A3C0
.text:0076A3C0                  Index           = dword ptr -64h
.text:0076A3C0                  pDynamicBuffer  = dword ptr -60h
.text:0076A3C0                  pDwords         = dword ptr -5Ch
.text:0076A3C0                  this            = dword ptr -58h
.text:0076A3C0                  nDwords         = dword ptr -54h
.text:0076A3C0                  pStackBuffer    = byte ptr -50h
.text:0076A3C0                  pBuffer         = dword ptr -10h
.text:0076A3C0                  BufferLength    = dword ptr -0Ch
.text:0076A3C0                  BufferSize      = dword ptr -8
.text:0076A3C0                  pObjectSet      = dword ptr  8
```

Illustration 1: Stack layout

Then here is the breakdown of the processing of the DWORD array in that same function:

```
.text:0076A40F                 ObjectExists:                           ; CODE XREF: xxxCommand56+1E↑j
.text:0076A40F 8B 58 0C                        mov     ebx, [eax+0Ch]
.text:0076A412 8B 40 08                        mov     eax, [eax+8]
.text:0076A415 C1 EB 02                        shr     ebx, 2
.text:0076A418 89 44 24 14                     mov     [esp+70h+pDwords], eax
.text:0076A41C 33 C0                           xor     eax, eax
.text:0076A41E 8D 74 24 20                     lea     esi, [esp+70h+pStackBuffer]
.text:0076A422 89 5C 24 1C                     mov     [esp+70h+nDwords], ebx
.text:0076A426 89 74 24 60                     mov     [esp+70h+pBuffer], esi
.text:0076A42A 89 44 24 64                     mov     [esp+70h+BufferLength], eax ; length of the string in the buffer
.text:0076A42E C7 44 24 68 40 00+              mov     [esp+70h+BufferSize], 64 ; size of the buffer
.text:0076A436 88 44 24 20                     mov     [esp+70h+pStackBuffer], al
.text:0076A43A 89 44 24 0C                     mov     [esp+70h+Index], eax
.text:0076A43E 85 DB                           test    ebx, ebx
.text:0076A440 0F 84 1D 01 00 00               jz      ForLoopExit
.text:0076A446
.text:0076A446                 ForLoop:                                ; CODE XREF: xxxCommand56+19D↓j
.text:0076A446 8B 4C 24 14                     mov     ecx, [esp+70h+pDwords]
.text:0076A44A 8B 54 24 0C                     mov     edx, [esp+70h+Index]
.text:0076A44E 8B 3C 91                        mov     edi, [ecx+edx*4]
.text:0076A451 85 C0                           test    eax, eax        ; BufferLength==0?
.text:0076A453 74 72                           jz      short AppendNumber
.text:0076A455
.text:0076A455                 AppendComma:
.text:0076A455 8B 4C 24 68                     mov     ecx, [esp+70h+BufferSize]
.text:0076A459 3B C1                           cmp     eax, ecx        ; BufferLength<BufferSize?
.text:0076A45B 72 53                           jb      short DoNotResizeBuffer_0
.text:0076A45D 8D 50 01                        lea     edx, [eax+1]
.text:0076A460 3B D1                           cmp     edx, ecx        ; BufferLength+1<=BufferSize?
.text:0076A462 76 4C                           jbe     short DoNotResizeBuffer_0
.text:0076A464 83 C0 40                        add     eax, 64
.text:0076A467 8D 4C 24 20                     lea     ecx, [esp+70h+pStackBuffer]
.text:0076A46B 89 44 24 68                     mov     [esp+70h+BufferSize], eax ; BufferSize=BufferLength+1+63
.text:0076A46F 3B F1                           cmp     esi, ecx        ; pBuffer==pStackBuffer?
.text:0076A471 75 06                           jnz     short loc_76A479
.text:0076A473 33 F6                           xor     esi, esi
.text:0076A475 89 74 24 60                     mov     [esp+70h+pBuffer], esi ; pBuffer=NULL
.text:0076A479
.text:0076A479                 loc_76A479:                             ; CODE XREF: xxxCommand56+B1↑j
.text:0076A479 89 74 24 10                     mov     [esp+70h+pDynamicBuffer], esi
.text:0076A47D 50                              push    eax             ; uBytes
.text:0076A47E 8D 74 24 14                     lea     esi, [esp+74h+pDynamicBuffer]
.text:0076A482 E8 B9 1C E6 FF                  call    xxxResizePtr    ; esi=&ptr,arg_0=size
.text:0076A487 8B 74 24 14                     mov     esi, [esp+74h+pDynamicBuffer]
.text:0076A48B 83 C4 04                        add     esp, 4
.text:0076A48E 83 7C 24 60 00                  cmp     [esp+70h+pBuffer], 0 ; pBuffer==NULL?
.text:0076A493 75 13                           jnz     short DoNotCopyStackBuffer_0
.text:0076A495 8B 54 24 64                     mov     edx, [esp+70h+BufferLength]
.text:0076A499 52                              push    edx
.text:0076A49A 8D 44 24 24                     lea     eax, [esp+74h+pStackBuffer]
.text:0076A49E 50                              push    eax
.text:0076A49F 56                              push    esi
.text:0076A4A0 E8 CB 6E 79 00                  call    _memcpy_0       ; memcpy(pDynamicBuffer,pStackBuffer,BufferLength)
.text:0076A4A5 83 C4 0C                        add     esp, 0Ch
.text:0076A4A8
.text:0076A4A8                 DoNotCopyStackBuffer_0:                 ; CODE XREF: xxxCommand56+D3↑j
.text:0076A4A8 8B 44 24 64                     mov     eax, [esp+70h+BufferLength]
.text:0076A4AC 89 74 24 60                     mov     [esp+70h+pBuffer], esi ; pBuffer=pDynamicBuffer
.text:0076A4B0
.text:0076A4B0                 DoNotResizeBuffer_0:                    ; CODE XREF: xxxCommand56+9B↑j
.text:0076A4B0                                                         ; xxxCommand56+A2↑j
.text:0076A4B0 8D 0C 06                        lea     ecx, [esi+eax]
.text:0076A4B3 40                              inc     eax
.text:0076A4B4 89 44 24 64                     mov     [esp+70h+BufferLength], eax ; BufferLength+=1
.text:0076A4B8 85 C9                           test    ecx, ecx
.text:0076A4BA 74 0B                           jz      short AppendNumber
.text:0076A4BC C6 01 2C                        mov     byte ptr [ecx], ',' ; pBuffer[BufferLength]=','
.text:0076A4BF 8B 44 24 64                     mov     eax, [esp+70h+BufferLength]
.text:0076A4C3 8B 74 24 60                     mov     esi, [esp+70h+pBuffer]
```

Illustration 2: Part 1 of the formatting

First, it initializes the main pointer (pBuffer) to a 64 byte stack based buffer (pStackBuffer) and the buffer size accordingly. It then enters a loop. If it isn't the first iteration, it will extend (and copy) the buffer if necessary and append a comma.

```
.text:0076A4C7                  AppendNumber:                           ; CODE XREF: xxxCommand56+93↑j
.text:0076A4C7                                                          ; xxxCommand56+FA↑j
.text:0076A4C7 8D 48 0A                         lea     ecx, [eax+10]
.text:0076A4CA 3B C8                            cmp     ecx, eax
.text:0076A4CC 76 56                            jbe     short DoNotResizeBuffer_1
.text:0076A4CE 2B C8                            sub     ecx, eax        ; Huh?
.text:0076A4D0 03 C8                            add     ecx, eax
.text:0076A4D2 3B 4C 24 68                      cmp     ecx, [esp+70h+BufferSize] ; BufferLength+10<=BufferSize?
.text:0076A4D6 76 4C                            jbe     short DoNotResizeBuffer_1
.text:0076A4D8 8D 41 3F                         lea     eax, [ecx+63]
.text:0076A4DB 8D 4C 24 20                      lea     ecx, [esp+70h+pStackBuffer]
.text:0076A4DF 89 44 24 68                      mov     [esp+70h+BufferSize], eax ; BufferSize=BufferLength+10+63
.text:0076A4E3 3B F1                            cmp     esi, ecx        ; pBuffer==StackBuffer?
.text:0076A4E5 75 06                            jnz     short loc_76A4ED
.text:0076A4E7 33 F6                            xor     esi, esi
.text:0076A4E9 89 74 24 60                      mov     [esp+70h+pBuffer], esi ; pBuffer=NULL
.text:0076A4ED
.text:0076A4ED                  loc_76A4ED:                             ; CODE XREF: xxxCommand56+125↑j
.text:0076A4ED 89 74 24 10                      mov     [esp+70h+pDynamicBuffer], esi
.text:0076A4F1 50                               push    eax
.text:0076A4F2 8D 74 24 14                      lea     esi, [esp+74h+pDynamicBuffer]
.text:0076A4F6 E8 45 1C E6 FF                   call    xxxResizePtr    ; esi=&ptr,arg_0=size
.text:0076A4FB 8B 74 24 14                      mov     esi, [esp+74h+pDynamicBuffer]
.text:0076A4FF 83 C4 04                         add     esp, 4
.text:0076A502 83 7C 24 60 00                   cmp     [esp+70h+pBuffer], 0 ; pBuffer==NULL?
.text:0076A507 75 13                            jnz     short DoNotCopyStackBuffer_1
.text:0076A509 8B 54 24 64                      mov     edx, [esp+70h+BufferLength]
.text:0076A50D 52                               push    edx
.text:0076A50E 8D 44 24 24                      lea     eax, [esp+74h+pStackBuffer]
.text:0076A512 50                               push    eax
.text:0076A513 56                               push    esi
.text:0076A514 E8 57 6E 79 00                   call    _memcpy_0       ; memcpy(pDynamicBuffer,pStackBuffer,BufferLength)
.text:0076A519 83 C4 0C                         add     esp, 0Ch
.text:0076A51C
.text:0076A51C                  DoNotCopyStackBuffer_1:                 ; CODE XREF: xxxCommand56+147↑j
.text:0076A51C 8B 44 24 64                      mov     eax, [esp+70h+BufferLength]
.text:0076A520 89 74 24 60                      mov     [esp+70h+pBuffer], esi ; pBuffer=pDynamicBuffer
.text:0076A524
.text:0076A524                  DoNotResizeBuffer_1:                    ; CODE XREF: xxxCommand56+10C↑j
.text:0076A524                                                          ; xxxCommand56+116↑j
.text:0076A524 57                               push    edi
.text:0076A525 03 F0                            add     esi, eax
.text:0076A527 68 F8 0D FD 00                   push    offset aD_0     ; "%d"
.text:0076A52C 56                               push    esi
.text:0076A52D E8 33 45 79 00                   call    _sprintf        ; sprintf(&pBuffer[BufferLength],"%d",pDwords[Index])
.text:0076A532 8B 74 24 6C                      mov     esi, [esp+7Ch+pBuffer]
.text:0076A536 8B 44 24 70                      mov     eax, [esp+7Ch+BufferLength]
.text:0076A53A 8D 0C 06                         lea     ecx, [esi+eax]
.text:0076A53D 83 C4 0C                         add     esp, 0Ch
.text:0076A540 8D 79 01                         lea     edi, [ecx+1]
.text:0076A543
.text:0076A543                  loc_76A543:                             ; CODE XREF: xxxCommand56+188↓j
.text:0076A543 8A 11                            mov     dl, [ecx]
.text:0076A545 41                               inc     ecx
.text:0076A546 84 D2                            test    dl, dl
.text:0076A548 75 F9                            jnz     short loc_76A543
.text:0076A54A 2B CF                            sub     ecx, edi
.text:0076A54C 03 C1                            add     eax, ecx
.text:0076A54E 8B 4C 24 0C                      mov     ecx, [esp+70h+Index]
.text:0076A552 41                               inc     ecx
.text:0076A553 89 44 24 64                      mov     [esp+70h+BufferLength], eax ; BufferLength+=strlen(pBuffer+BufferLength)
.text:0076A557 89 4C 24 0C                      mov     [esp+70h+Index], ecx ; Index+=1
.text:0076A55B 3B CB                            cmp     ecx, ebx        ; Index<nDwords?
.text:0076A55D 0F 82 E3 FE FF FF                jb      ForLoop
.text:0076A563
.text:0076A563                  ForLoopExit:                            ; CODE XREF: xxxCommand56+80↑j
```

Illustration 3: Part 2 of the formatting

It then makes sure that there is enough room to fit **10** additional bytes, extends (and copies) the buffer if necessary and formats the number into the buffer with “%d”. The issue here is that “%d” can be **11** bytes if positive (including the terminating null bytes) and **12** if negative due to the leading unary minus.

If we trigger the formatting of a **10** digit positive number while **10** bytes away from the end of the buffer, we will overflow the buffer by **one** byte (*the trailing null*). If we do the same with a **10** digit negative number, we overflow the buffer by **two** bytes (*one digit followed by a null byte*).

Consequences

Given that pBuffer is located after pStackBuffer, an overflow of the buffer, while in the stack, will overwrite the LSB of pBuffer. This will introduce an inconsistency since pBuffer will be != pStackBuffer and as such will be considered as a heap buffer. Depending on the DWORD array, it might end up being passed to LocalFree at the end of the function, or xxxResizePtr (which will do a LocalReAlloc) or actually reused to receive additional formatted numbers – overwriting data on the stack.

The LocalReAlloc of a stack based address is an interesting path as it can return the initial pointer itself (which depends on the content of the 8 bytes preceding the pointer, interpreted as a heap chunk header), at least on Windows XP. This means that further sprintf() will be done on the stack, overwriting stack variables.

If the off-by-one happens after the buffer has already shifted to the heap, the first two bytes of the next chunk metadata could be corrupted.

Example

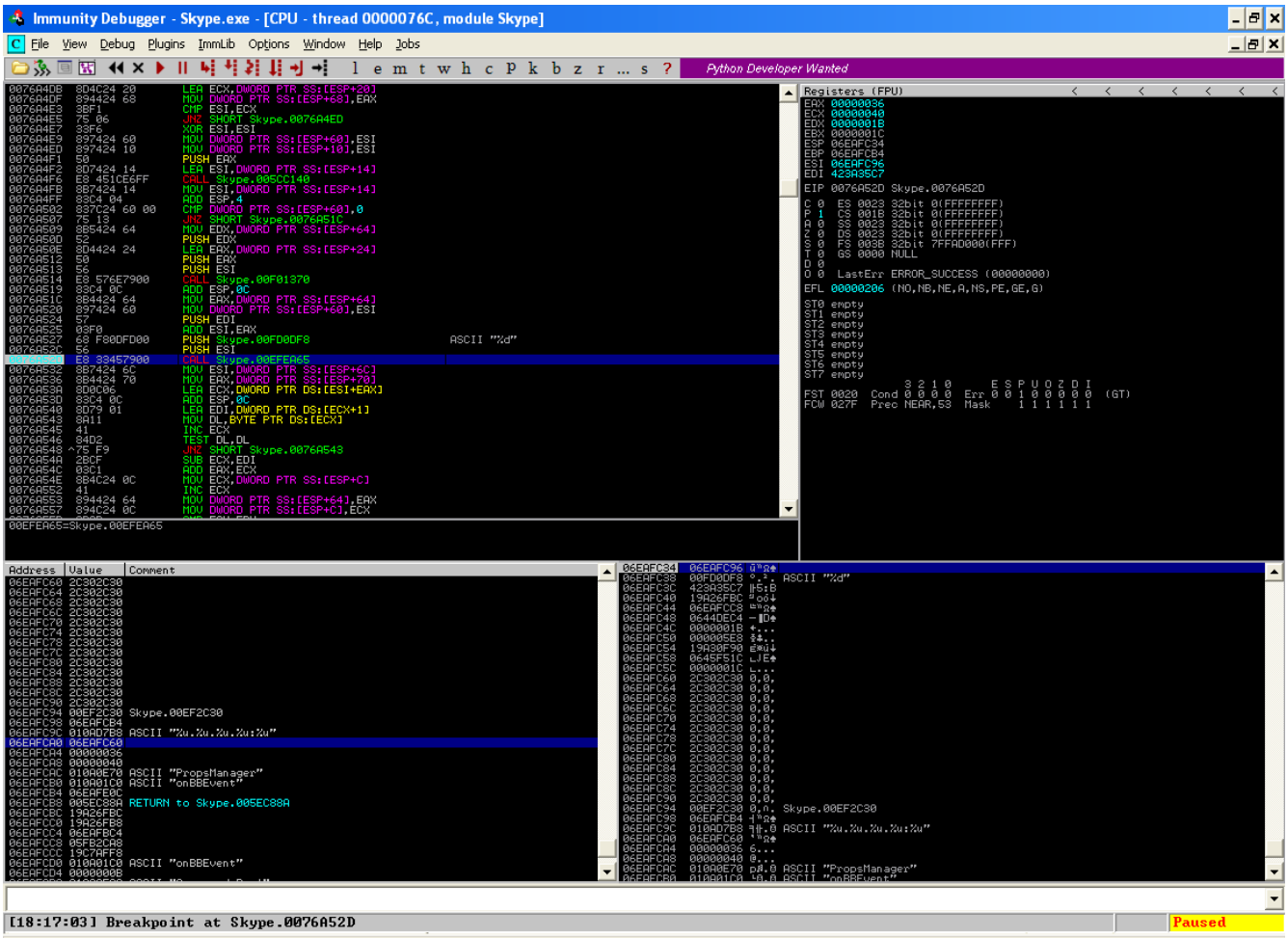

Illustration 4: Off-By-One

In the example above, the stack buffer is located at 0x6eafc60. We already formatted in (64-10)/2 times 0, and the breakpoint is on the sprintf() where we are going to format 0x423a35c7 (1111111111) in. This will overwrite the LSB of pBuffer with a null byte. The program will later attempt to free or resize 0x6eafc00, which is really a stack variable.

Extending the previous example, sending [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1111111111,1111111111,1111111111] as an object of type 6 and id 33 in command 56 will trigger “bad things” as shown in the screenshot below, where an ObjectSet pointer was overwritten with 0x31313131 (a register – EDI – saved on the stack was overwritten and later restored):

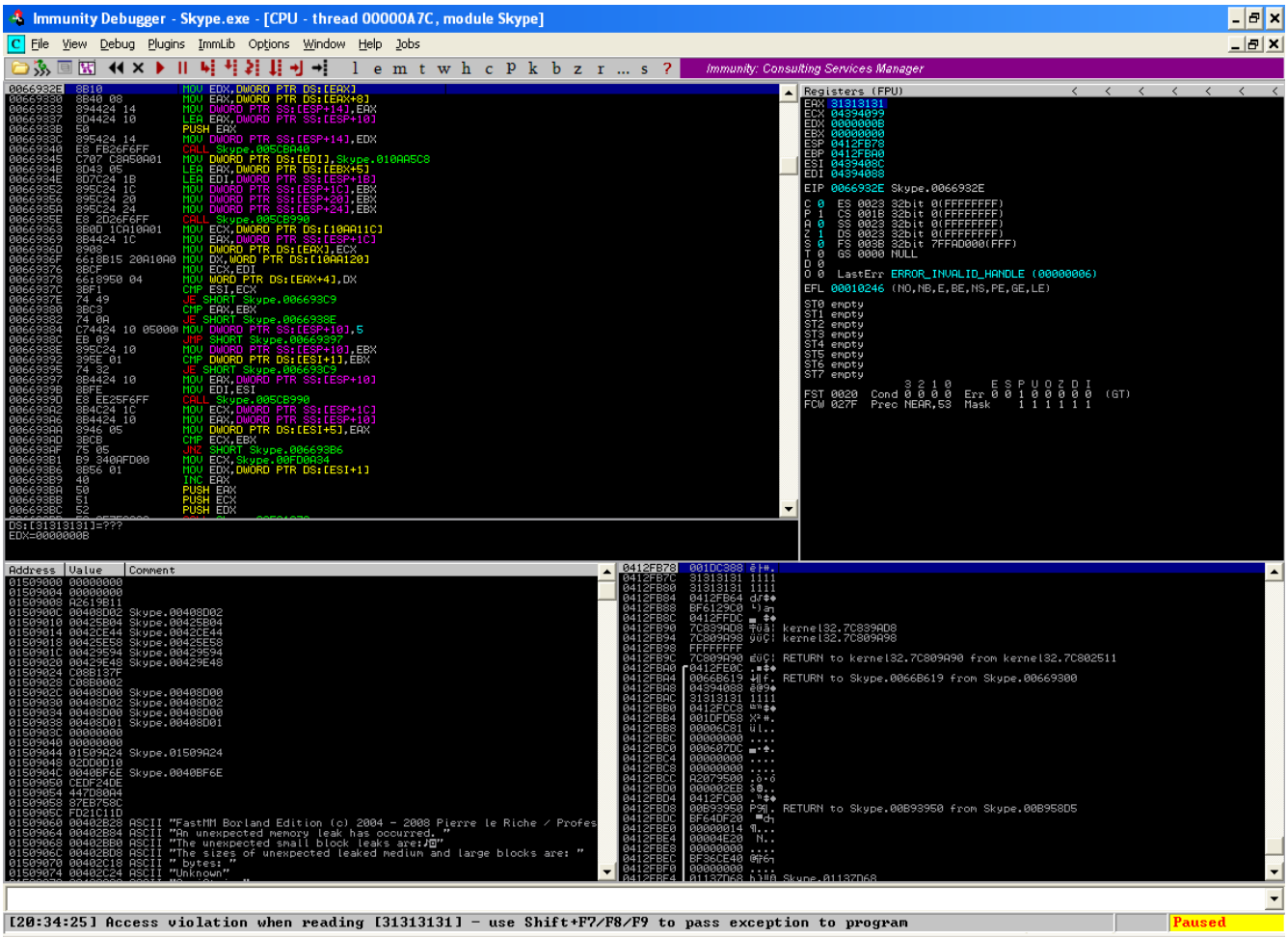

Illustration 5: ObjectSet overwrite