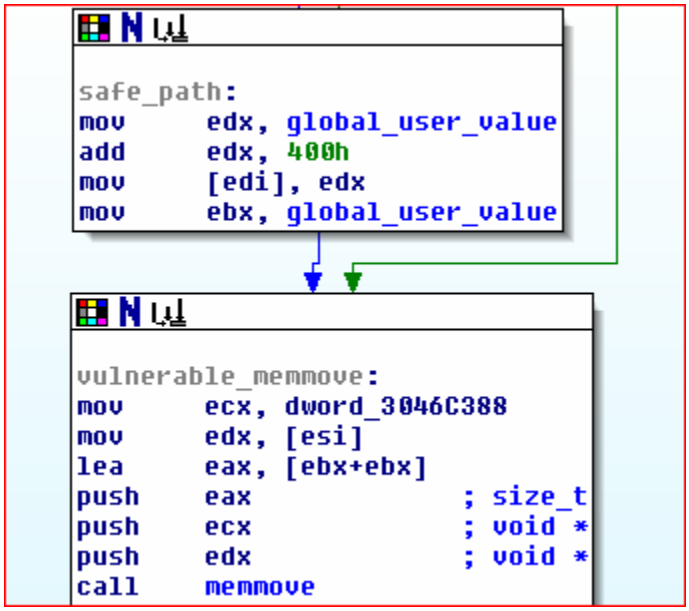
Analysis of MS-07-036
Multiple Excel Vulnerabilities

I. Introduction

This research was conducted using the Office 2003 Excel Viewer application and the corresponding security patch for MS-07-036 - Vulnerabilities in Microsoft Excel Could Allow Remote Code Execution. The point is to reverse engineer the patch to verify the effectiveness of changes made to the code, and also to present the potential vulnerability points in the un-patched software so that similar bugs are easy to spot in the future. It will also be useful to know if other versions of Excel (in particular Office 2002 SP-2) that aren't mentioned in Microsoft's advisory are affected, due to end-of-life or other reason.

II. The Vulnerability (well, one of them)

A vulnerability exists because the code fails to check the size of the third parameter to **memmove**, which allows a specially crafted Excel file to cause an on-read access violation past the source buffer or an on-write access violation past the destination buffer. In the un-patched version of excel.exe for testing (11.0.6412.0) a sub routine exists at 0x30282490. Inside, there is a call to **memmove**, but more importantly, there are two paths of execution that lead to that **memmove**:

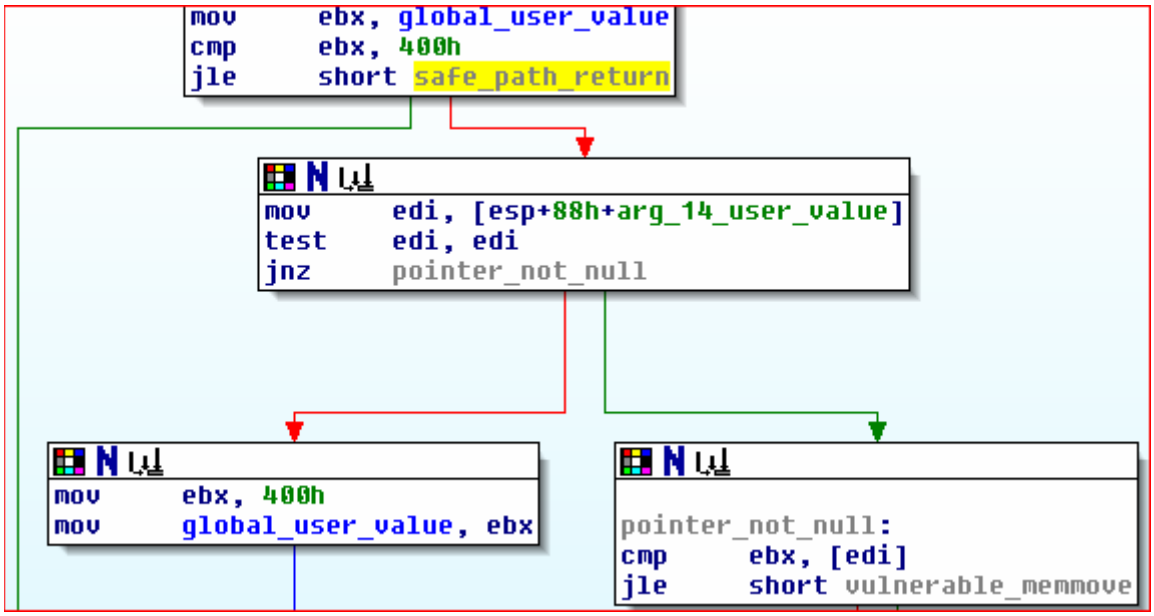


You can see, there are blue and green arrows that both point to the location named **vulnerable_memmove**. Also, the third parameter, indicated by **size_t**, is produced by adding **[ebx+ebx]**. So, the point is to understand where **ebx** comes from, and the impact that it may have on **memmove** if it isn't checked properly.

In the code block labeled **safe_path**, **ebx** is obtained from **global_user_value**, which is properly handled in code that precedes the **safe_path** block. Unfortunately, in the other path (the green arrow), things are handled less carefully.

Take a look at the un-safe path below. First, if **global_user_value** is less than or equal to **400h**, we branch to **safe_path_return** and no harm can be done. For the example, let's assume that **global_user_value** is **40000000h**. In this case, we don't branch to **safe_path_return**, but instead we encounter the next check.

At the next check, there is validation to see if the 6th argument to the function (**arg_14_user_value**) is zero (via the **test** instruction). At this point, we can't tell if the 6th argument is a pointer (to any data type) or just a 32 bit number. However looking a little further at the **pointer_not_null** location, which is reached as long as **arg_14_user_value** is not zero, we do see **[edi]** being dereferenced, indicating that yes, the 6th argument is a pointer. Furthermore, it's being dereferenced in the context of a compare instruction where the other operand is an integer, so we know it's a pointer to numeric data and not to a character buffer or something.

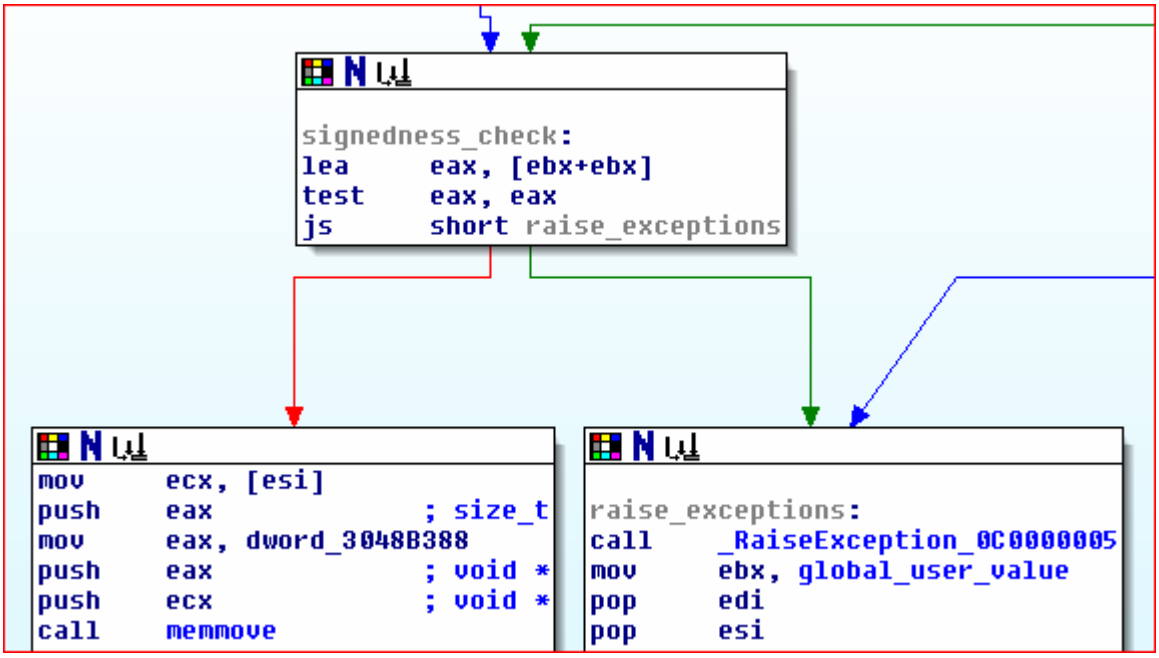


Note that if the **arg_14_user_value** is zero (NULL), then we branch to the left and will not reach the vulnerable **memmove**. As long as the **global_user_value** (remember our example is **40000000h**) is greater than the value pointed to by **arg_14_user_value**, then execution will continue at **vulnerable_memmove**, where the value of **ebx** is doubled and passed as the number of bytes to copy into the destination buffer with **memmove**.

Our example would try to move 16 gigabytes worth of data (**40000000h + 40000000h**) into the destination buffer, because the size parameter to **memmove** is unsigned. This would either cause an on-read access violation by tripping off the end of the source page, or it would cause an on-write access violation by spilling into an invalid memory location past the buffer.

III. The (Implemented) Solution

Now, take a look at the same function from the patched excel.exe (11.0.8142.0). The sub routine is at 0x302A1490 in this code. Immediately it's apparent that there is now only one direct path to the **memmove**, and it includes a signed-ness check _after_ the addition of **ebx** and _before_ the usage of the resulting value as the size parameter to **memmove**. If the result is signed (via **test/js**, which modifies the signed flag if the high order bit is set) then we branch to the **raise_exception** location. I named this location because the first thing it does is call a wrapper function around the Kernel32.dll RaiseException() export with exception code 0x0C0000005 (Access Violation).

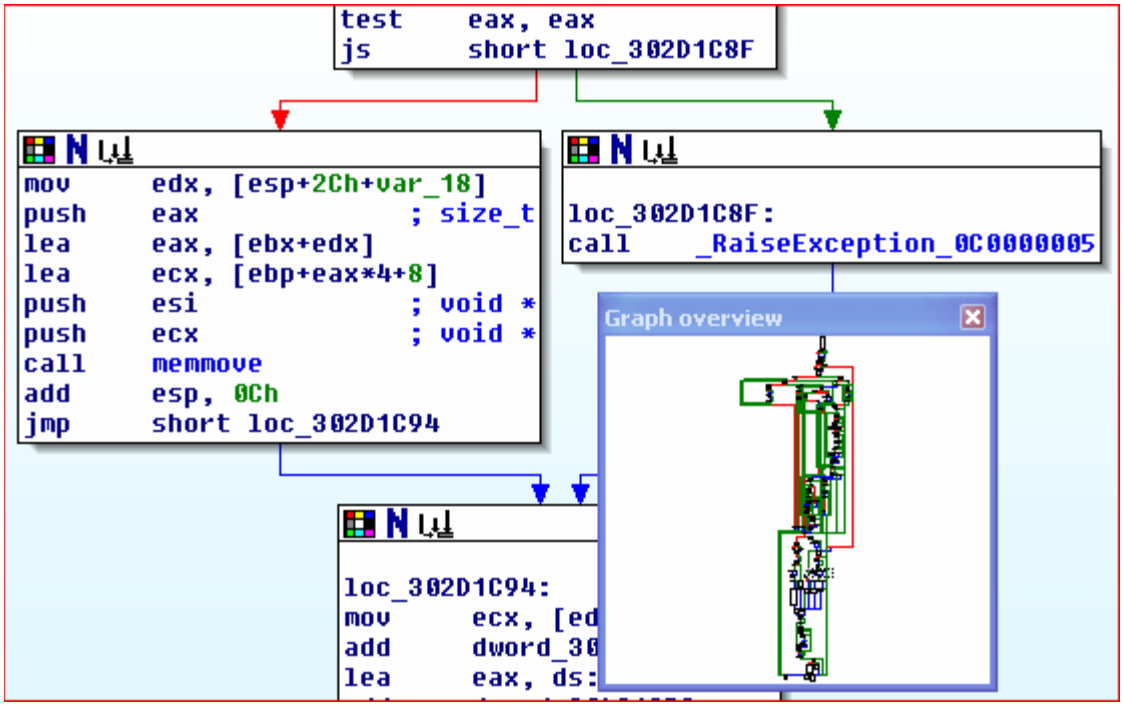


IV. Additional Vulnerability Points

The function described above is not the only one modified by the MS-07-036 patch, in fact there are several. Upon inspection, the same type of problem is being fixed by introducing a signed-ness check and raising an exception if one is encountered. This permits the assumption that every new call to RaiseException() occurs around a vulnerable call to memmove. I manually reviewed a few and it is an accurate assumption.

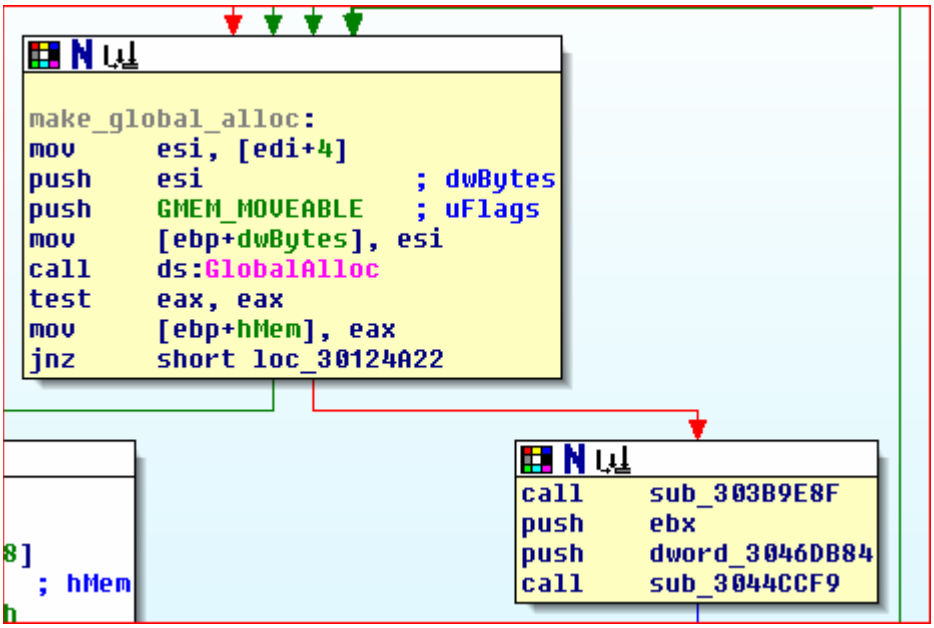
In the un-patched excel.exe, there are three calls to RaiseException(), while, in the patched version there are eleven, potentially indicating eight different vulnerability points in the older versions of Excel.

The fix is apparent even in much larger functions:

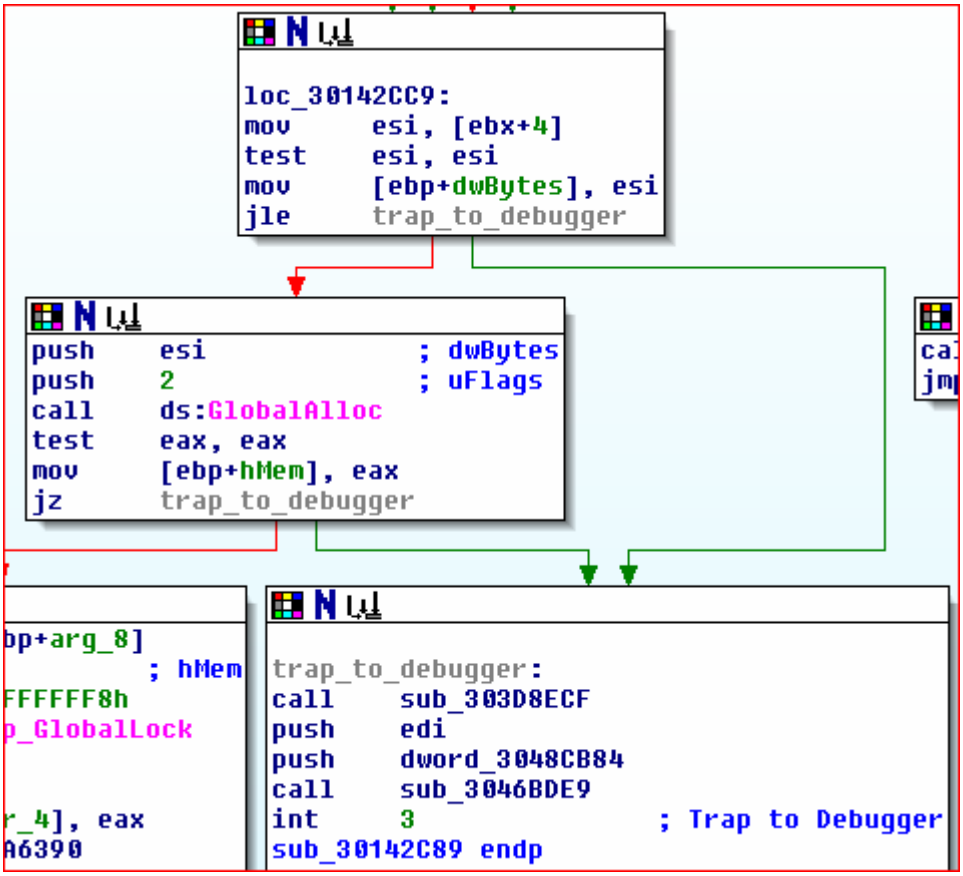


Some other potential memory corruption issues appear to have been fixed by MS-07-036. In the un-patched version, a parameter is passed to a sub routine (0x30124970 un-patched, 0x 30142C89 patched) which appears to be a pointer to a structure that begins with two 16-bit members followed by a 32-bit member. The 32-bit member is passed to GlobalAlloc() without being checked, and more importantly, the code doesn't seem to halt or handle exceptions efficiently after the handling of this call.

In the example from the un-patched function below, **edi** points to the structure base. I don't bother to reverse this binary far enough to know if the value of **[edi+4]** is checked prior to being processed in the point below, because the patch (yet to be shown) tells me that most likely it isn't (or else there would be no need to add a check in the patch).



Let's now take a look at the patched function. Below, note that **ebx** takes the place of **edi** in the previous function and that any values above 0x7FFFFFFF cause an int 3.



In most cases, unless a JIT debugger is set, this should invoke Dr. Watson, essentially halting the program. In the un-patched version however, execution proceeds into multiple other functions and it appears the parent function is allowed to return to its calling function. It seems with this change that the patch just crashes the program immediately upon encountering an unexpected value.

This introduction of “trap to debugger” interrupts after checking a value before a heap allocation call is also fairly consistent throughout the patched and un-patched binaries – it occurs in multiple functions.

In other functions of the code, it is apparent that a potential off-by-one overflow was fixed. The value **101h** is passed to **MSO_6877** (Ordinal export #6877 from mso.dll), which is essentially the size parameter for another **memmove** operation.

```
.text:302ED51B                  test    [ebp+arg_4], 4
.text:302ED51F                  jz      short loc_302ED563
.text:302ED521                  push    esi
.text:302ED522                  mov     esi, [ebp+arg_0]
.text:302ED525                  push    edi
.text:302ED526                  mov     edi, [ebp+arg_10]
.text:302ED529                  push    101h
.text:302ED52E                  lea     eax, [esi+108Eh]
.text:302ED534                  push    eax
.text:302ED535                  push    dword ptr [edi+4]
.text:302ED538                  call    MSO_6877
```

In the patched version, there are some changes, including the size being reduced to **100h**. We can also see some compiler optimization that moved the address of **MSO_6877** into **esi** register instead of calling it directly. This is because in the fixed version, there is more than one call to **MSO_6877** in the same function and **call esi** only requires two bytes worth of instructions. It also changed in the sense that the source and destination buffers (**var_404** and **var_204**) are now local stack variables, whereas before they were either heap buffers or local buffers to a calling function.

```
.text:303F23C9                  mov     esi, MSO_6877
.text:303F23CF                  mov     edi, 100h
.text:303F23D4                  push    edi
.text:303F23D5                  lea     eax, [ebp+var_404]
.text:303F23DB                  push    eax
.text:303F23DC                  lea     eax, [ebp+var_204]
.text:303F23E2                  push    eax
.text:303F23E3                  call    esi ; MSO_6877
```

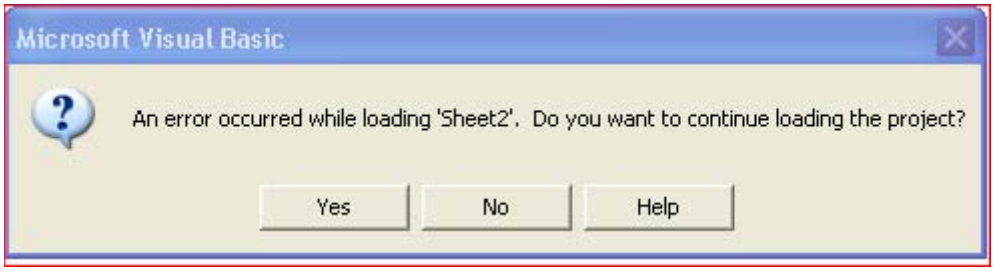
The size of the two buffers is shown below in the stack view of the patched function. They are each 512 bytes. These are wide character (Unicode) buffers and 100h is the maximum number of wide characters (2 bytes each on most systems) that **memmove** should write to the destination buffer. **101h** as in the un-patched version could result in **202h** (514 bytes) – or otherwise, an off-by-one error, which in this case actually leads to the ability to overwrite two bytes instead of one due to the size of wide characters.

```
-00000404 var_404         db 512 dup(?)
-00000204 var_204         db 512 dup(?)
-00000004 var_4           dd ?
+00000000  s              db 4 dup(?)
+00000004  r              db 4 dup(?)
```

V. Published Excel Proof-Of-Concept

There was a published proof-of-concept on milw0rm dated 2007-06-27 that produces an on-read access violation in Office 2002 SP-2 Excel. This is one of the versions not mentioned to be affected by the vulnerabilities presented in MS-07-036 (perhaps because it is just end-of-life). Either way, as a result, there is no patch available for Office 2002 SP-2 Excel, to address the POC or any of the other vulnerabilities described above.

In the POC, there is simply a NULL pointer deferenced within VBE6.DLL, which presumably exists due to invalid values within the embedded Visual Basic project. The vulnerability is triggered even if macros are disabled; however it will not be triggered if a user chooses 'No' upon receiving the error message:



If Excel is allowed to repair the proof-of-concept file by removing the Visual Basic and ActiveX projects, then the POC is disarmed.

VI. Conclusion

Although it probably goes unsaid, this proves that Office 2002 SP-2 Excel is vulnerable to at least one of the potential code execution vulnerabilities for which there are no vendor-provided patches. The Office 2003 releases are also confirmed vulnerable (to one or more) of the exploitable conditions, which of course only solidifies the vendor advisory – however in the process of confirming, we had some fun and documented a few good methods of spotting similar bugs.