MWR LABS

Security Advisory

Microsoft Office Protected-View Out-Of-Bound Array Access

2017-11-23

Software	Microsoft Office
Affected Versions	Microsoft Excel 2010, 2013, 2016 (x86 and x64)
CVE Reference	CVE-2017-8692 (Uniscribe Remote Code Execution Vulnerability)
Author	Yong Chuan Koh (@yongchuank)
Severity	Important
Vendor	Microsoft Corporation
Vendor Response	Fixed on 12 Sept 2017

Description:

Microsoft Office is a suite of desktop applications consisting of Word, Excel, Powerpoint, Outlook and various other productivity applications. Among these, Word, Excel and Powerpoint implemented the Protected-View sandbox technology as a defence-in-depth exploit mitigation.

An out-of-bound array access was discovered while the Excel broker parsed an attacker controlled Protected-View Inter-Process Communication (IPC) message from the sandbox process.

Impact:

A successful exploitation would allow an attacker to elevate his privileges from AppContainer to Medium, thereby breaking out of the Protected-View sandbox.

MWR LABS

Security Advisory

Cause:

The vulnerability existed because as the broker process looped through an array of SCRIPT_ITEM objects, it dereferences the current (N) and next (N+1) SCRIPT_ITEM objects to calculate the difference of iCharPos value between these two objects. However, if N is the last SCRIPT_ITEM object, then an out-of-bound dereference for the N+1 object would occur.

Interim Workaround:

Avoid opening Microsoft Office Excel files from untrusted sources.

Solution:

Users should apply the September security updates from Microsoft.

Technical details

The following analysis is based on EXCEL 16.0.4266.1001.

In Protected-View mode, the EXCEL broker receives and services IPC messages from the Excel sandbox, distinguished by a message-tag. One of these messages has the 0x071200 tag, which the sandbox uses to request the broker to input a specified string into the formula bar as it is isolated from the AppContainer. Subsequently, the Excel broker would process the formula-bar string with gdi32full!ScriptItemize().

This POC formula-bar string is sent to Excel broker with the 0x071200 IPC message:

```
?0S^'&1:bCWX4[tcY%=D~W@vJ}MpMr<ijSar<#9<OtrX_S7j\ldH"?qF>!uMnO>(q-j(@-
g?Mcav)MzM_<m+T[zA46ykI#V5\2Kj|42
```

Upon receipt, Excel would transform the string into an array of SCRIPT_ITEM objects with the following sequence of calls to gdi32full!ScriptItemize().

```
ScriptItemize (
   -  pwcInChars  =
      L"?0S^'&1:bCWX4[tcY%=D~W@vJ}MpMr<ijSar<#9<OtrX_S7j\ldH"?qF>!uMnO>(q-j(@-
      g?Mcav)MzM_<m+T[zA46ykI#V5\2Kj|42"
   -  cInChars = 0x00000021
   -  cMaxItems = 0x00000022
```

```
    -  psControl = &(0x00800009)
    -  psState = &(0x0001)
    -  pItems
    -  pcItems
)
ScriptItemize (
    -  pwcInChars  = L"Sar<#9<OtrX_S7j\ldH"?qF>!uMnO>(q-j(@-
       g?Mcav)MzM_<m+T[zA46ykI#V5\2Kj|42"
    -  cInChars = 0x00000021
    -  cMaxItems = 0x00000022
    -  psControl = &(0x00800009)
    -  psState = &(0x0001)
    -  pItems
    -  pcItems
)
ScriptItemize (
    -  pwcInChars  = L"j(@-g?Mcav)MzM_<m+T[zA46ykI#V5\2Kj|42"
    -  cInChars = 0x00000004
    -  cMaxItems = 0x00000005
    -  psControl = &(0x00800009)
    -  psState = &(0x0001)
    -  pItems
    -  pcItems
)
```

As the out-of-bound dereference occurred in the last call to gdi32full!ScriptItemize() on the *pItems* output buffer, we next examine how it was allocated. After some reversing, the buffer was found to be allocated in Mso99Lwin32client!sub_B6899(), together with an assignment of the *cMaxItems* parameter. The snippet below shows the relevant blocks of this allocation.

Figure 1: Snippet in Mso99Lwin32client!sub_B6899(), to allocate pItems buffer

In the above snippet, at Mso99Lwin32client!000B6962 with a *cInChars* value of 4, the *cMaxItems* parameter is determined by the following:

- *cMaxChars* = *cInChars* > 2 ? (*cInChars*+1) : 2

Next, in the code-block Excel allocates a buffer (*pBuffer*) of 58h bytes consisting of these 2 sub-buffers:

- Size of Unknown-SubBuffer = *cInChars* * 4 * 3
  = 30h bytes
- Size of pItems-SubBuffer = (*cMaxItems*) * sizeof(SCRIPT_ITEM)
  = (*cMaxItems*) * 8
  = 28h bytes

Eventually a pointer to *pItems* is obtained from *pBuffer*:

- *pItems* = *pBuffer* + (*cInChars* * 0xC)
  = *pBuffer* + (*cInChars* * 3 * 4)

Finally, back in gdi32full!ScriptItemize(), it was observed that it called gdi32full!ScriptTokenize() to write *cInChars* number of SCRIPT_ITEM structures retrieved from the *pwcInChars* parameter to the *pItems* buffer. The last SCRIPT_ITEM (ie: *cMaxItems*-th) object is then used to "summarize" the number of tokenized characters with the flag 3.

- *pItems* [*cInChars*] = *pItems* [*cMaxItems*-1]
  = SCRIPT_ITEM {
    int iCharPos = *cInChars*,
    SCRIPT_ANALYSIS a {
    WORD wdWord1 = 0,
    SCRIPT_STATE s = 3 //SCRIPT_STATE.fEngineReserved
    }
  }

The snippet below shows the code-blocks where the last SCRIPT_ITEM object is written:

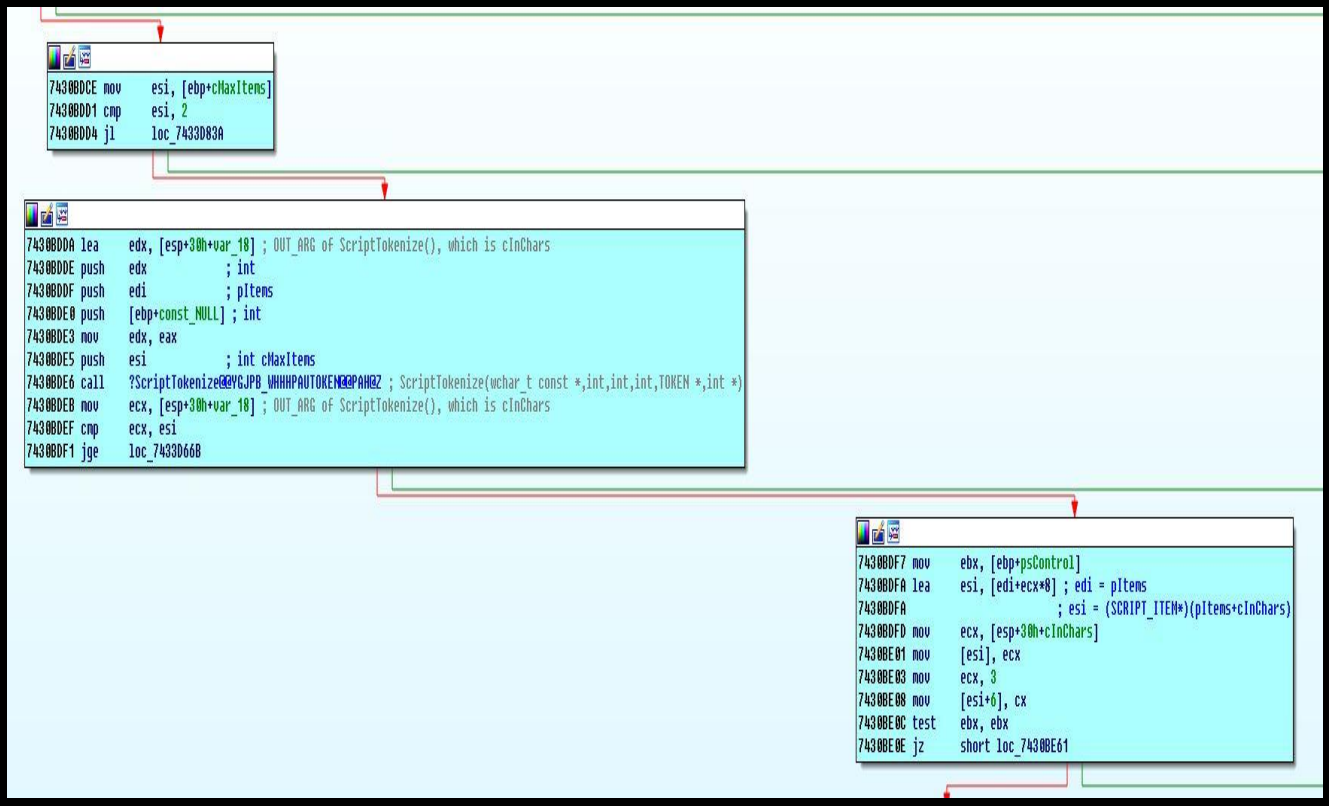

Figure 2: Snippet in gdi32full!ScriptItemizeCommon(), to populate pItems buffer

The populated *pItems* buffer is then parsed into a loop in gdi32full!FindMatchingPair(), which is represented by the following pseudo-code:

```
int __stdcall FindMatchingPair(SCRIPT_ITEM **ppItems,
                               SCRIPT_ITEM *pItems_LastItem, ...)
{
. . .
SCRIPT_ITEM* var_10;
if (*ppItems < pItems_LastItem) {
      do {
            var_10 = *ppItems;
            int iCharPosOfCurrScriptItem = (var_10 + 0)->iCharPos;
            int iCharPosOfNextScriptItem = (var_10 + 1)->iCharPos; //OOB Deref
            int edx = iCharPosOfNextScriptItem - iCharPosOfCurrScriptItem;
```

```
            if (var_10->a.eScript > 0x114)          { ... }
            else if (var_10->a.eScript == 0x114)    { ... }
            else                                    { ... }
            (*ppItems)++;
      } while (*ppItems <= pItems_LastItem);
}
. . .
}
```

In this do-while loop, the *pItems* pointer is incremented until the last SCRIPT_ITEM object. However in the loop, the next SCRIPT_ITEM object is also dereferenced. Therefore this causes an out-of-bound dereference when *pItem* points to the last SCRIPT_ITEM object.

The following windbg output show below demonstrates the crash:

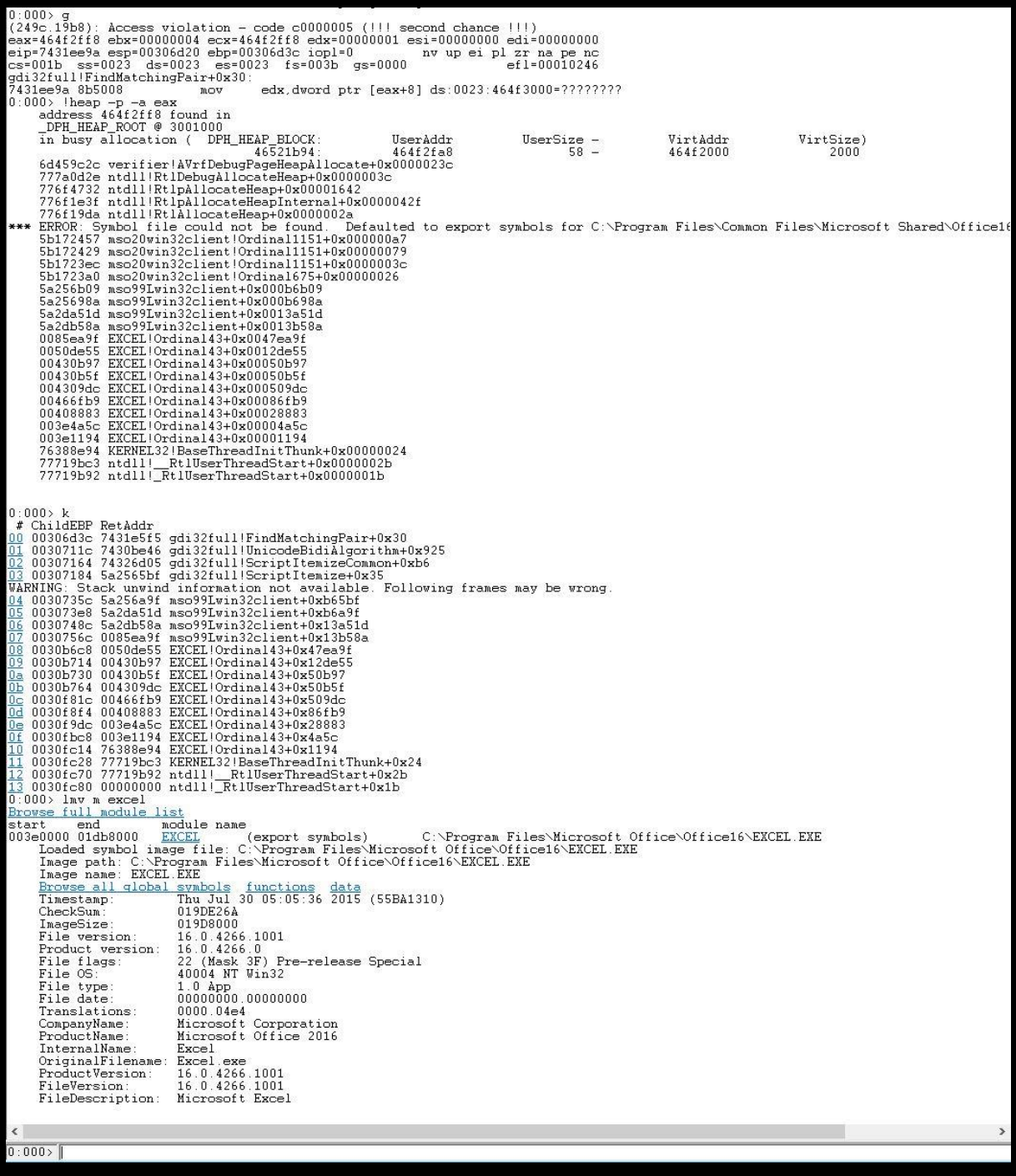

Figure 3: windbg output of Out-of-Bound SCRIPT_ITEM buffer dereference

In conclusion, the root-cause of this bug is the insufficiently-sized *pItems* buffer that Excel (or Mso99Lwin32client) allocates for ScriptItemize(). This is probably due to the developers' oversight when

reading the documentation for the ScriptItemize() function. From MSDN[1], the *cMaxItems* and *pItems* parameters are described as such:

> ...
>
> *cInChars* [in]
>
> Number of characters in *pwcInChars* to itemize.
>
> *cMaxItems* [in]
>
> Maximum number of **SCRIPT_ITEM** structures defining items to process.
>
> ...
>
> *pItems* [out]
>
> Pointer to a buffer in which the function retrieves **SCRIPT_ITEM** structures representing the items that have been processed. The buffer should be `(cMaxItems + 1) * sizeof(SCRIPT_ITEM)` bytes in length. It is invalid to call this function with a buffer to hold less than two **SCRIPT_ITEM** structures. The function always adds a terminal item to the item analysis array so that the length of the item with zero-based index "i" is always available as:
>
> `pItems[i+1].iCharPos - pItems[i].iCharPos;`

The developers probably noted that since *cMaxItems* is the maximum number of SCRIPT_ITEM structures to process, they allocated the *pItems* buffer to only *cMaxItems* structures where in fact, it should have *cMaxItems*+1.

On 12 Sept 2017, Microsoft identified this vulnerability as a “Uniscribe Remote Code Execution Vulnerability”, and listed Windows 8, Windows 10, Windows Server 2012 and Windows Server 2016 as affected products. So this vulnerability should have been patched in gdi32.dll (at the root-cause) instead of Excel.

[1] ScriptItemize function: https://msdn.microsoft.com/en-us/library/windows/desktop/dd368556(v=vs.85).aspx

Detailed Timeline

Date	Summary
2017-05-22	MWR Labs reported vulnerability and POC to MSRC
2017-05-22	MSRC acknowledged and opened case 38823
2017-05-23	MSRC responded that the team could not reproduce the issue
2017-05-23	MWR Labs sent crash dump to MSRC
2017-08-04	MSRC responded that this will be patched in September 2017
2017-09-12	MSRC assigned CVE-2017-8692 and released patch for this vulnerability
2017-11-23	MWR Labs released advisory