

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.

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)
- pItem
- pcItem
)
ScriptItemize (
- pwcInChars = L"5ar<#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)
- pItem
- pcItem
)
ScriptItemize (
- pwcInChars = L"j (@-g?Mcav)MzM_<m+T[zA46ykI#V5\2Kj|42"
- cInChars = 0x00000004
- cMaxItems = 0x00000005
- psControl = &(0x00800009)
- psState = &(0x0001)
- pItem
- pcItem
)
```

As the out-of-bound dereference occurred in the last call to `gdi32full!ScriptItemize()` on the *pItem* 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_B68990, 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 pltems-SubBuffer = $(cMaxItems) * \text{sizeof}(\text{SCRIPT_ITEM})$
= $(cMaxItems) * 8$
= 28h bytes

Eventually a pointer to `pltems` is obtained from `pBuffer`:

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

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

- ```
pltems[cInChars] = pltems[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:

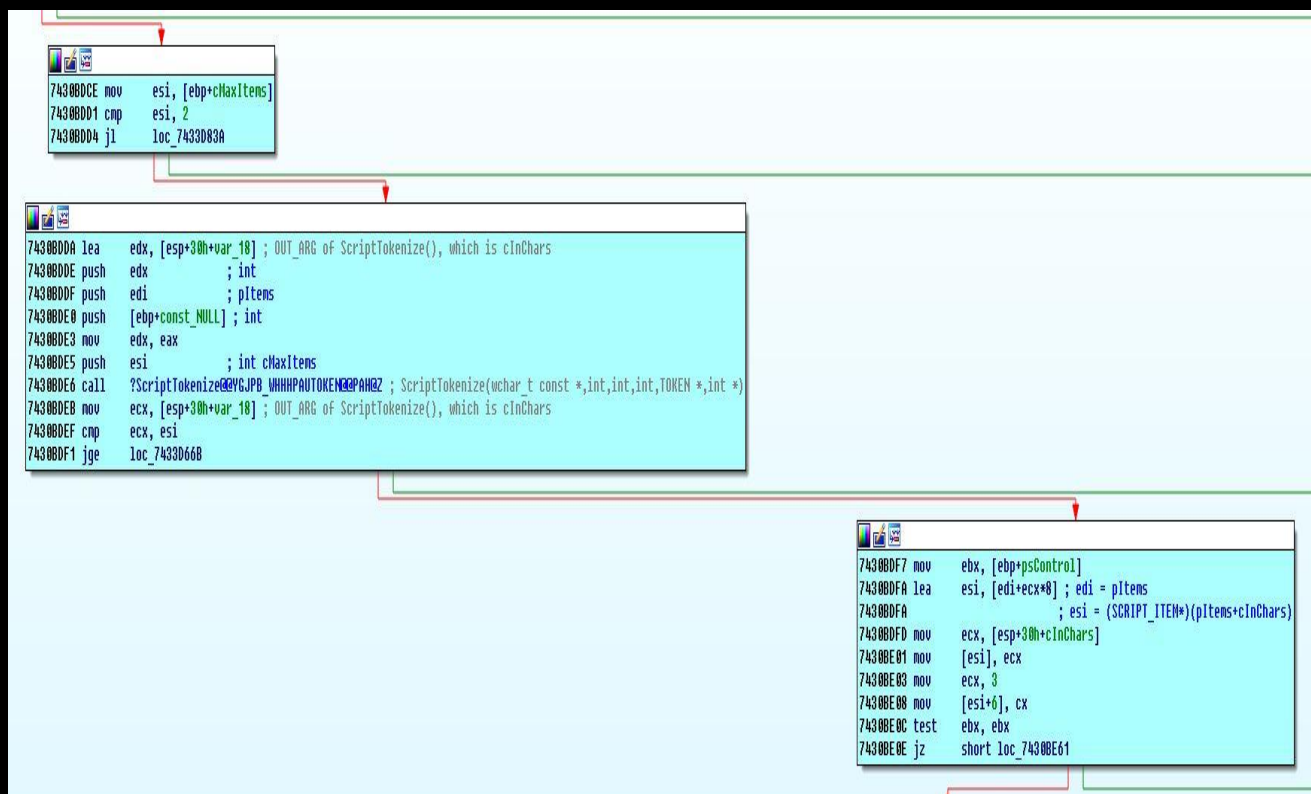


Figure 2: Snippet in `gdi32full!ScriptItemizeCommon()`, to populate `pItems` buffer

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;
 } while (1);
 }
}

```

```
 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:



```

0:000> g
(249c.19b8): Access violation - code c0000005 (!!! second chance !!!)
eax=464f2ff8 ebx=00000004 ecx=464f2ff8 edx=00000001 esi=00000000 edi=00000000
eip=7431ee9a esp=00306d20 ebp=00306d3c iopl=0 nv up ei pl zr na pe nc
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00010246
gdi32full!FindMatchingPair+0x30:
7431ee9a 8b5008 mov edx,dword ptr [eax+8] ds:0023:464f3000=????????
0:000> !heap -p -a eax
address 464f2ff8 found in
_DPH_HEAP_ROOT @ 3001000
in busy allocation (DPH_HEAP_BLOCK: UserAddr UserSize - VirtAddr VirtSize)
6d459c2c verifier!AVrfDebugPageHeapAllocate+0x0000023c 46521b94: 464f2fa8 58 - 464f2000 2000
777a0d2e ntdll!RtlDebugAllocateHeap+0x0000003c
776f4732 ntdll!RtlpAllocateHeap+0x00001642
776f1e3f ntdll!RtlpAllocateHeapInternal+0x0000042f
776f19da ntdll!RtlAllocateHeap+0x0000002a
*** ERROR: Symbol file could not be found. Defaulted to export symbols for C:\Program Files\Common Files\Microsoft Shared\Office16
5b172457 mso20win32client!Ordinal1151+0x000000a7
5b172429 mso20win32client!Ordinal1151+0x00000079
5b1723ec mso20win32client!Ordinal1151+0x0000003c
5b1723a0 mso20win32client!Ordinal1675+0x00000026
5a256b09 mso99Lwin32client+0x000b6b09
5a25698a mso99Lwin32client+0x000b698a
5a2da51d mso99Lwin32client+0x0013a51d
5a2db58a mso99Lwin32client+0x0013b58a
0085ea9f EXCEL!Ordinal43+0x0047ea9f
0050de55 EXCEL!Ordinal43+0x0012de55
00430b97 EXCEL!Ordinal43+0x00050b97
00430b5f EXCEL!Ordinal43+0x00050b5f
004309dc EXCEL!Ordinal43+0x000509dc
00466fb9 EXCEL!Ordinal43+0x00086fb9
00408883 EXCEL!Ordinal43+0x00028883
003e4a5c EXCEL!Ordinal43+0x00004a5c
003e1194 EXCEL!Ordinal43+0x00001194
76388e94 KERNEL32!BaseThreadInitThunk+0x00000024
77719bc3 ntdll!_RtlUserThreadStart+0x0000002b
77719b92 ntdll!_RtlUserThreadStart+0x0000001b

0:000> k
ChildEBP RetAddr
00 00306d3c 7431ee9a gdi32full!FindMatchingPair+0x30
01 0030711c 7430be46 gdi32full!UnicodeBidiAlgorithm+0x925
02 00307164 74326d05 gdi32full!ScriptItemizeCommon+0xb6
03 00307184 5a256b09 gdi32full!ScriptItemize+0x35
WARNING: Stack unwind information not available. Following frames may be wrong.
04 0030735c 5a256a9f mso99Lwin32client+0xb65bf
05 003073e8 5a2da51d mso99Lwin32client+0xb6a9f
06 0030748c 5a2db58a mso99Lwin32client+0x13a51d
07 0030756c 0085ea9f mso99Lwin32client+0x13b58a
08 0030b6c8 0050de55 EXCEL!Ordinal43+0x47ea9f
09 0030b714 00430b97 EXCEL!Ordinal43+0x12de55
0a 0030b730 00430b5f EXCEL!Ordinal43+0x50b97
0b 0030b764 004309dc EXCEL!Ordinal43+0x50b5f
0c 0030f81c 00466fb9 EXCEL!Ordinal43+0x509dc
0d 0030f8f4 00408883 EXCEL!Ordinal43+0x86fb9
0e 0030f9dc 003e4a5c EXCEL!Ordinal43+0x28883
0f 0030fbc8 003e1194 EXCEL!Ordinal43+0x4a5c
10 0030fc14 76388e94 EXCEL!Ordinal43+0x1194
11 0030fc28 77719bc3 KERNEL32!BaseThreadInitThunk+0x24
12 0030fc70 77719b92 ntdll!_RtlUserThreadStart+0x2b
13 0030fc80 00000000 ntdll!_RtlUserThreadStart+0x1b
0:000> lmv n excel
Browse full module list
start end module name
003e0000 01db8000 EXCEL (export symbols) C:\Program Files\Microsoft Office\Office16\EXCEL.EXE
Loaded symbol image file: C:\Program Files\Microsoft Office\Office16\EXCEL.EXE
Image path: C:\Program Files\Microsoft Office\Office16\EXCEL.EXE
Image name: EXCEL.EXE
Browse all global symbols functions data
Timestamp: Thu Jul 30 05:05:36 2015 (55BA1310)
Checksum: 019DE26A
ImageSize: 019D8000
File version: 16.0.4266.1001
Product version: 16.0.4266.0
File flags: 22 (Mask 3F) Pre-release Special
File OS: 40004 NT Win32
File type: 1.0 App
File date: 00000000.00000000
Translations: 0000.04e4
CompanyName: Microsoft Corporation
ProductName: Microsoft Office 2016
InternalName: Excel
OriginalFilename: Excel.exe
ProductVersion: 16.0.4266.1001
FileVersion: 16.0.4266.1001
FileDescription: Microsoft Excel

```

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<sup>1</sup>, the *cMaxItems* and *pItems* parameters are described as such:

```
...
cInChars [in]
 Number of characters in pwclnChars 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.

---

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