Attacking Interoperability: An OLE Edition

- Haifei Li (haifei.li@intel.com)
- Bing Sun (bing.sun@intel.com)
About Us: Haifei

• Security Researcher at Intel Security (formerly McAfee)
  • Previously: Microsoft, Fortinet

• Work on several questions (for good purposes):
  1) How to find vulnerabilities
  2) How to exploit them
  
  At McAfee my interests have been extended to a 3rd question:
  3) How to detect the effect by answering the 1st and 2nd.

  Work on research-backed projects aiming at detecting the most stealthy exploits or zero-days (e.g., the Advanced Exploit Detection System)

About Us: Bing

• Security Research Manager of IPS security research team at Intel Security Group (formerly McAfee)

• Focus:
  1) Advanced vulnerability exploitation and detection
  2) Rootkits techniques and detection
  3) Firmware security
  4) Virtualization security

Declaration

- Even though we are going to talk about OLE, for Object Linking and Embedding, we will cover only Embedding in this presentation.
  - Due to the length of our presentation
  - This is a really big area
Agenda

- What Is OLE?
- Historical Zero Days Involving OLE
- OLE Internals
- Attack Surface
- Conclusion
What Is OLE?

- Object Linking and Embedding
  - Based on Component Object Model (COM)

- It serves the majority of interoperability on Office/WordPad
  - Working with default/third-party applications to provide rich documentation features to Office/WordPad users
What Is OLE in Our Lives, Really?

- Embedding a document in another document

To Employees: Benefits Enrollment and Payroll Set-up
ACTION REQUIRED

<table>
<thead>
<tr>
<th>PAYROLL SETUP</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHAT YOU HAVE TO DO</td>
</tr>
<tr>
<td>---------------------</td>
</tr>
<tr>
<td>Read</td>
</tr>
<tr>
<td>A/R</td>
</tr>
</tbody>
</table>

- By double-clicking on the “Checklist” document readers will be able to open another document
- Very convenient for Office users
Agenda

- What Is OLE?
- Historical Zero Days Involving OLE
- OLE Internals
- Attack Surface
- Conclusion
OLE-related Zero Days in History

- Almost all previous critical Office/WordPad zero days actually involve OLE

- **CVE-2014-4114/6352** (a.k.a. “Sandworm” zero day)
  - Reported in October 2014. Logic fault, really serious
  - 2 OLE objects found in the original sample
  - *Microsoft failed to fix it in the initial patch*
OLE-related Zero Days in History

- CVE-2014-1761
  - Reported in March 2014 by Google, highly targeted attack
  - RTF format-handling fault, not a vulnerability in OLE object, but leverages OLE mechanism to load a non-ASLR module, “MSCOMCTL.OCX”, to bypass ASLR

\objh749{\*\objclass MSComctlLib.ImageComboCtrl.2}{\*\objdata
OLE-related Zero Days in History

- CVE-2013-3906
  - Detected and reported by us in October 2013
  - Microsoft Graphics Component fault, not a vulnerability in OLE object, but leverages ActiveX/OLE mechanism to perform a heap spray in Office
OLE-related Zero Days in History

- CVE-2012-0158 / CVE-2010-3333
  - Years-old vulnerabilities inMSCOMCTL.OCX
  - Classic OLE vulnerabilities
  - Still see samples in the wild today. :P

- Just in: A similar zero-day attack in MSCOMCTL.OCX (CVE-2015-2424)
  - Disclosed on July 15 by iSIGHT Partners
A Short Summary

- OLE objects not only produce critical zero-day vulnerabilities, but also help greatly on Office/WordPad vulnerability exploitation
  - Loading non-ASLR modules
  - Heap-spray in Office process
  - ...

- Bug class through memory corruption to logic bugs
Agenda

- What Is OLE?
- Historical Zero Days Involving OLE
- OLE Internals
- Attack Surface
- Conclusion
Previous Related Work

- There is barely no previous research focusing on OLE internals, but we will mention two:
  - “Attacking Interoperability”
    - by Mark Dowd, Ryan Smith, and David Dewey in 2009
    - *We named our presentation in honor of the great work done in this paper*

- Parvez Anwar’s blog site has some work related to Office/OLE
  - [https://www.greyhathacker.net](https://www.greyhathacker.net)
OLE Is a Subset of COM

OLE objects are COM objects that expose specific Interfaces. Must have:

IPersistStorage
IOleObject
To explain the OLE internals, first we need to understand what happens when a user opens a document containing OLE objects.

- User performs action on the OLE object (e.g., clicking, double-clicking)
- “Verb” is performed automatically by Office features (e.g., PowerPoint animation)
OLE Initialization

- Initializing/loading an OLE object can be done simply via the `ole32!OleLoad()` API

```c
HRESULT OleLoad(
    _In_  LPSTORAGE pStg,
    _In_  REFIID riid,
    _In_  LPOLECLIENTSITE pClientSite,
    _Out_ LPVOID *ppvObj
);
```

The **OleLoad** function performs the following steps:

- If necessary, performs an automatic conversion of the object (see the **OleDoAutoConvert** function).
- Gets the CLSID from the open storage object by calling the **IStorage::Stat** method.
- Calls the **CoCreateInstance** function to create an instance of the handler. If the handler code is not available, the default handler is used (see the **OleCreateDefaultHandler** function).
- Calls the **IOleObject::SetClientSite** method with the `pClientSite` parameter to inform the object of its client site.
- Calls the **QueryInterface** method for the **IPersistStorage** interface. If successful, the **IPersistStorage::Load** method is invoked for the object.
- Queries and returns the interface identified by the `riid` parameter.
OLE Initialization

- We focus on the two major steps
  - Step 1: calling CoCreateInstance to initialize the OLE object
  - Step 2: calling IPersistStorage to initialize the OLE object’s initial status (data)

- Next let’s analyze the two steps in detail
Step 1: CoCreateInstance

ole32!wCreateObject+0x101:
75b41553 e8b37feff call ole32!CoCreateInstance (75b29d0b)
0018de38 0018de98 00000000 00000403 64c0c954
0:000> k
75b3f2af ole32!wCreateObject+0x101
75b3f1d4 ole32!OleLoadWithoutBinding+0x9c
632c4eb4 ole32!OleLoad+0x37
0:000> db poi(esp)
0018de98 02 26 02 00 00 00 00 00-c0 00 00 00 00 00 00 46
0:000> db poi(esp+4*3)
64c0c954 12 01 00 00 00 00 00 00-c0 00 00 00 00 00 00 00

CoCreateInstance(CLSID,
    NULL,
    CLSCTX_INPROC_SERVER | CLSCTX_INPROC_HANDLER | CLSCTX_NO_CODE_DOWNLOAD,
    IID(IOleObject))
Where Does CLSID Come From?

- The CLSID comes from the document, indicating which OLE object the user wants to initialize.

- Because Office/WordPad supports a couple of document file types, locating the CLSID varies:
  - Office Open-XML format (.docx, .xlsx, .pptx, .ppsx, etc)
  - RTF (.rtf)
  - Office Binary format (.doc, .xls, .ppt, pps, etc)
  - Office even supports HTML format

- We are going to give examples in the Open-XML format and RTF.
For Open-XML Format, the CLSID is read from the "OLESS" binary data file.
CLSID in RTF

- For RTF, it uses the outdated OLE 1.0 format to define an OLE object

- Specifying the CLSID is done via specifying the corresponding **ProgID**, in “\objdata” RTF control word*
  - **ProgID** will be “translated” to CLSID at runtime via `CLSIDFromProgID`

```rtf
{\rtf1\object\objocx}{\*\objdata
01050000       //OLEVersion
02000000       //FormatID, EmbeddedObject

08000000
5061636b61676500 //ProgID "Package"

00000000
00000000
D4290000
```

*If the ProgID is invalid, and the following native data follows the OLESS format, the CLSID will be read from the OLESS native data*
Step 2: IPersistStorage::Load

The container calls the “Load()” method on the OLE object’s IPersistStorage interface to initialize its initial status.

```plaintext
; __int32 __stdcall CPackage::Load(CPackage *this, LPSTORAGE pStg)
?Load@CPackage@@UAGJPAUIStorage@@@Z proc near

var_1C= dword ptr -1Ch
NumberOfBytesWritten= dword ptr -18h
pclsid= CLSID ptr -14h
var_4= dword ptr -4
this= dword ptr 8
pStg= dword ptr 0Ch

mov edi, edi
push ebp
mov ebp, esp
sub esp, 1Ch
```
Step 2: IPersistStorage::Load

- IID: 0000010a-0000-0000-C000-000000000046

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandsOffStorage</td>
<td>Instructs the object to release all storage objects that have been passed to it by its container and to enter HandsOff mode.</td>
</tr>
<tr>
<td>InitNew</td>
<td>Initializes a new storage object.</td>
</tr>
<tr>
<td>IsDirty</td>
<td>Determines whether an object has changed since it was last saved to its current storage.</td>
</tr>
<tr>
<td><strong>Load</strong></td>
<td>Loads an object from its existing storage.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves an object, and any nested objects that it contains, into the specified storage object. The object enters NoScribble mode.</td>
</tr>
<tr>
<td>SaveCompleted</td>
<td>Notifies the object that it can write to its storage object.</td>
</tr>
</tbody>
</table>

```c
HRESULT Load(
    [in] IStorage *pStg
);```

Load the initial “status” for the OLE object when it’s being initialized.
Storage Data

- It really depends on the OLE object for handling the IStorage - loading its initial status
  - As the code for implementing the IPersistStorage interface sits in the OLE provider (OLE object)

- The **Storage Data** (represented in the “IStorage” parameter) is stored in document file
  - Like the “CLSID” field, it’s also from the document file (which the attacker supplies)
  - But there are differences
    - OLE container (Office/WordPad) reads the CLSID in order to instantiate the OLE object
    - OLE container reads the Storage Data and passes it to the OLE object, which is responsible for processing the data
Storage Data in Office Open-XML

- Represented in **OLESS data file**
- The following example shows the Storage Data for Flash Player OLE object
  - CLSID: D27CDB6E-AE6D-11CF-96B8-444553540000
  - Read Storage Data from **OLESS data file** (oleObject1.bin)
  - Read from the “Contents” section
Storage Data in RTF

- Represented in OLE1 Native Data

```plaintext
\{
\*\objdata
01050000     // OLE version
02000000      // Format ID, EmbeddedObject
1B000000      // ProgID
4D53436F6D63746C4C69622E4C697374566965774374726C2E3200
00000000
00000000
000E0000      // OLE1 Native Data (length + data)
D0CF11E0A1B11AE1000000000000000000000000000000000003E00030
1000000200000001000000FEFFFFFFFF00000000000000000000000000000000
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```


A Short Break

- We have explained the two key steps in OLE Initialization
- Next, let’s take a look at the “Verb” action

User opens the document

OLE object Initialized

CoCreateInstance

IPersistStorage::Load

“Verb” action performed

- User performs action on the OLE object (e.g., clicking, double-clicking)
  or
- “Verb” performed automatically by Office features (e.g., PowerPoint animation)
OLE “Verb” Action

- In essence, performing “verb” action is just calling the IOleObject::DoVerb on the OLE object

- IOleObject
  - IID: 00000112-0000-0000-C000-000000000046
  - 24 methods on this Interface

- There are a few parameters for this IOleObject::DoVerb method, but we need to focus only on the first one: the “iVerb,” which under certain scenarios can be controlled by the attacker
  - For example, via PowerPoint Show files (.ppsx, .pps)
IOleObject::DoVerb

packager!CPackage::DoVerb:
731e580c 8bff mov edi,edi
0:000> dd esp
0031c89c 660651c6 0054ec80 FFFFFFFD 00000000

HRESULT DoVerb(
[in] LONG iVerb,
[in] LPMSG lpmsg,
[in] IOleClientSite *pActiveS
[in] LONG lindex,
[in] HWND hwndParen
[in] LPCRECT lprcPosRe
);

- <p:cmd type="verb" cmd="-3">
  - <p:cBhvr>
    - <p:cTn id="10" dur="1000" fill="hold">
      - <p:stCondLst>
        <p:cond delay="0"/>
      </p:stCondLst>
    </p:cTn>
  - <p:tgtEl>
    <p:spTgt spid="4"/>
  </p:tgtEl>
</p:cBhvr>
</p:cmd>
Agenda

- What Is OLE?
- Historical Zero Days Involving OLE
- OLE Internals
- Attack Surface
- Conclusion
So, what may an attacker possibly perform in a document-based attack via OLE?

We need to understand what data an attacker may supply from documents

Is the attacker able to supply the CLSID for CoCreateInstance during OLE Initialization?
  
  Answer: Yes (explained)

Is the attacker able to supply the Storage used in IPersistStorage::Load() during OLE Initialization?
  
  Answer: Yes (explained)

Is the attacker able to supply the "verb" id during OLE "Verb" Action?
  
  Answer: Yes (explained)
Attack I - IPersistStorage::Load

- It’s the most obvious one
  - You want to parse some data; I give you the crafted data
  - Sometimes it will result in memory corruptions; sometimes it may be a logic bug

- In fact, most of the previously disclosed OLE vulnerabilities were actually in the IPersistStorage::Load() function

- Let’s give some examples
CVE-2012-0158

- Lots of previous analysis has shown this, in MSCOMCTL.OCX

```assembly
mov    ecx, [ebx]
push   esi
push   edi
push   eax
push   ebx
call   dword ptr [ecx+0Ch] ; read the large length 0x8282
mov    esi, eax
test   esi, esi
jl     short loc_275C87EF
```

- But, where does the routine really come from?
Tracing back, we arrive here

```assembly
.text:276008D9 sub_276008D9 proc near ; DATA XREF: .text:275903E0↓
.text:276008D9
.text:276008D9
.text:276008D9
.text:276008D9 arg_0 = dword ptr 8
.text:276008D9 arg_4 = dword ptr 0Ch
.text:276008D9
.text:276008DA push ebp
.text:276008DB mov ebp, esp
.text:276008DC mov eax, [ebp+arg_4]
.text:276008DF lea edx, [ebp+arg_4]
.text:276008E2 push edx
.text:276008E3 push 0
.text:276008E5 mov ecx, [eax]
.text:276008E7 push 10h
.text:276008E9 push 0
.text:276008EB push offset aContents ; "Contents"
.text:276008F0 push eax
.text:276008F1 call dword ptr [ecx+10h] ; opening the stream named "CONTENTS"
.text:276008F4 test eax, eax
.text:276008F6 j1 short loc_27600916
.text:276008F8 mov eax, [ebp+arg_0]
.text:276008FB push esi
.text:276008FC push [ebp+arg_4]
.text:276008FF add eax, 0FFFFFFFFh
.text:27600902 mov ecx, [eax]
.text:27600904 push eax
.text:27600905 call dword ptr [ecx+14h] ; call to 275B66DE
```

What is the function sub_276008D9 really?
After some REing, we realize this is exactly the “IPersistStorage::Load” method

Indeed, the stack-based overflow exists in the IPersistStorage::Load method

```
.text:275906C0  IPersistStorage_vtable dd offset IPersistStorage__QueryInterface
.text:275906C0                      ; DATA XREF: sub_27586000;
.text:275906C0                      ; sub_2759453E+50↓0
.text:275906C4                      dd offset IPersistStorage__AddRef
.text:275906C8                      dd offset IPersistStorage__Release
.text:275906CC                      dd offset IPersistStorage__GetRunningClass
.text:275906D0                      dd offset IPersistStorage__IsDirty
.text:275906D4                      dd offset IPersistStorage__InitNew
.text:275906D8                      dd offset IPersistStorage__Load ; 0x276008D9
.text:275906DC                      dd offset IPersistStorage__Save
.text:275906E0                      dd offset IPersistStorage__SaveCompleted
.text:275906E4                      dd offset IPersistStorage__HandsOffStorage
```
“Package” Temp File Dropping

- Reported in McAfee Labs blog in July 2014
  - Still unpatched!
- Recently, James Forshaw leveraged the “feature” in the exploitation of an NTLM Reflection EoP vulnerability he discovered: [https://code.google.com/p/google-security-research/issues/detail?id=325](https://code.google.com/p/google-security-research/issues/detail?id=325)

- The issue also exists in the “IPersistStorage::Load” function
“Package” Temp File Dropping

0:000> r
packager!C Package::EmbedReadFromStream+0x2c6:
733c404d call packager!CopyStreamToFile (733c6974)
0:000> du poi(esp+4)
04fdc008 "C:\Users\ADMINI~1\AppData\Local\"
04fdc048 "Temp\dwmapi.dll"
0:000> k
733c4aaa packager!C Package::EmbedReadFromStream+0x2c6
733c627e packager!C Package::PackageReadFromStream+0x6b
7749eb44 packager!C Package::Load+0x10d
Attack II: IOleObject::DoVerb

- This is the "iVerb" param for the IOleObject::DoVerb

```c
HRESULT DoVerb(
    [in] LONG iVerb,
    [in] LPMSG lpmsg,
    [in] IOleClientSite *pActiveSite,
    [in] LONG lindex,
    [in] HWND hwndParent,
    [in] LPCRECT lprcPosRect
);
```

- The value of the "iVerb" can be defined in some place the attacker can control. For example: PowerPoint Show)

```xml
- <p:cmd type="verb" cmd="-3">
  - <p:cBhvr>
    - <p:cTn id="10" dur="1000" fill="hold">
      - <p:stCondLst>
        <p:cond delay="0"/>
      </p:stCondLst>
    </p:cTn>
  - <p:tgtEl>
    <p:spTgt spid="4"/>
  </p:tgtEl>
</p:cBhvr>
</p:cmd>
```
Attack II: IOleObject::DoVerb

- The attacker can supply the “iVerb” value and call the “IOleObject::DoVerb” method automatically
  - For example, via the PowerPoint Show “Animations” feature

- Different values will result in different actions. For example:
  - You give value 0, it performs predefined action 0, maybe opening the object
  - You give value -1, it performs predefined action -1, maybe doing something else
OLE objects can choose not to implement their own `IOleObject` but use the default/standard interface
- Thus resulting in some standard “verb” actions
- See next

However, there are also a number of OLE objects that chose to implement their own `IOleObject`
- An action the developer implemented but that may be abused by bad guys
- Usually logic issues
# Standard “Verb” Actions


<table>
<thead>
<tr>
<th>Value</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>The default action for the object.</td>
</tr>
<tr>
<td>-1</td>
<td>Activates the object for editing. If the application that created the object supports in-place activation, the object is activated within the OLE container control.</td>
</tr>
<tr>
<td>-2</td>
<td>Opens the object in a separate application window. If the application that created the object supports in-place activation, the object is activated in its own window.</td>
</tr>
<tr>
<td>-3</td>
<td>For embedded objects, hides the application that created the object.</td>
</tr>
<tr>
<td>-4</td>
<td>If the object supports in-place activation, activates the object for in-place activation and shows any user interface tools. If the object doesn’t support in-place activation, the object doesn’t activate, and an error occurs.</td>
</tr>
<tr>
<td>-5</td>
<td>If the user moves the focus to the OLE container control, creates a window for the object and prepares the object to be edited. An error occurs if the object doesn’t support activation on a single mouse click.</td>
</tr>
<tr>
<td>-6</td>
<td>Used when the object is activated for editing to discard all record of changes that the object’s application can undo.</td>
</tr>
</tbody>
</table>
The Sandworm Zero Day

The “Sandworm” zero-day attack (CVE-2014-4114) was the first ever exploit targeting this “IOleObject::DoVerb” vector.
When “verb” is 3
Performing “context-menu” actions!

```cpp
v20 = (a1 - 8);
if ( v20 >= 0 )
{
    hMenu = CreatePopupMenu();
    if ( hMenu )
    {
        v23 = CPackage::GetContextMenu(&v21);
        if ( v23 >= 0 )
        {
            mii.cbSize = 48;
            mii.fMask = 2;
            if ( GetMenuItemInfoW(hMenu, v_iVerb - 2, 1, &mii) ) // position = 3 -2 = 1
                // means the 2nd item on the menu.
                
                if ( *(a1 + 48) == 3 )
                    v23 = CPackage::CreateTempFile(0);
                if ( v23 >= 0 )
                {
                    v16 = mii.wID - 2;
                    v13 = 36;
                    v14 = 0;
                    v15 = 0;
                    v17 = 0;
                    v18 = 0;
                    v19 = 1;
                    v23 = (*(v21 + 16))(v21, &v13); // CDefFolderMenu::InvokeCommand
                    // Do the real job: “clicking” the 2nd item on the menu.
                }
                else
                {
                    v23 = 0x40181u;
                }
        }
    }
    DestroyMenu(hMenu);
```
The Sandworm Zero Day

- What could possibly be wrong?
- The “context-menu” options for different file types are different
- The file content as well as the filename (file type) are controlled via “IPersistStorage::Load”
  - Remember our “Package” Temp File Dropping case study? They are the same!
  - So, this neat zero-day actually leveraged two attack vectors
- For example, installing an .inf
  - Pwned! Logic bug!
So, we have discussed two important attack vectors for OLE: IPersistStorage::Load and IOleObject::DoVerb

Are there any more?
  Definitely

Let’s review the very first step of loading an OLE object
  Calling the \texttt{CoCreateInstance} trying to initialize the OLE objects, the OLE object is specified by CLSID, which is provided in the document file

What does \texttt{CoCreateInstance} do? The following:
\begin{verbatim}
CoGetClassObject(rclsid, dwClsContext, NULL, IID_IClassFactory, &pCF);
hresult = pCF->CreateInstance(pUnkOuter, riid, ppvObj)
pCF->Release();
\end{verbatim}

\texttt{CoGetClassObject} needs to first load the DLL associated with the CLSID into the process
What Is “CLSID-Associated” DLL?

- A DLL has an associated CLSID in your Windows Registry
  - HKEY_CLASSES_ROOT\CLSID
  - The “InprocServer32” key specifies where the DLL (“server”) is
Attack III: CLSID-Associated DLL Loading

- What could possibly be wrong here?
  - From an attacker’s perspective?

- As we’ve discussed, OLE objects are a *subset* of COM objects, which is another *subset* of CLSID-associated objects
  - Many COM objects registered in the OS are not OLE objects
    - *Several hundreds* vs. *several thousands*
    - Sometimes even a DLL that has a CLSID associated in the Windows Registry is not necessarily a COM

- But, **CoCreateInstance** will still load the CLSID-associated DLL in the process
  - Regardless whether it is an “OLE DLL”
  - The loaded DLL won’t be unloaded, even if it’s determined later not to be an “OLE DLL”
This is a *design* problem in the process of initializing OLE objects on Windows, in our opinion.

- Without loading the DLL first, you won’t be able to know whether the COM exposes the interface you want!

Let’s compare it with its well-known “sister” feature: the ActiveX Controls in Internet Explorer.

- Unlike OLE, IE11 loading an ActiveX Control (say, in IE) will first result in checking the “preapproved” list
  - HKLM\Software\Microsoft\Windows\CurrentVersion\Ext\PreApproved
- So, if the ActiveX CLSID is not in the list, the DLL won’t be really loaded into the IE process
  - No problem for ActiveX in IE
Consequences

- What bad things might happen due to the problem we discussed?
  - We can load any DLL into the process as long as the DLL is associated with a CLSID
  - Considering the attack is launched via a document

- There are quite a few

- Note: Loading OLE DLL may also have the same problems. But, being able to load every CLSID-associated DLL increases the attack surface *significantly*
Consequence 1: Non-ASLR DLL

- Loading non-ASLR DLL in container process
  - Namely, Word, PowerPoint, Excel, WordPad
  - Thus used to bypass ASLR for exploitation

- Note, not only the CLSID-associated DLL may be non-ASLR, but sometimes the CLSID-associated DLL could also link to other non-ASLR DLLs (so loaded as well)

- Does not work on Office 2013 and later because they enabled “Force ASLR”
  - Still works on Office <= 2010 and WordPad 😊
Example 1: otkloadr.WRAssembly.1

- Trying to load the “COM object” identified by ProgID: otkloadr.WRAssembly.1

```plaintext
{\rtf1\object\objocx\*/\objdata
01050000
02000000
16000000   //otkloadr.WRAssembly.1
6f746b6c6f6164722e5752417373656d626c792e3100
00000000
00000000
01000000
41
01050000
00000000
}}}

- It’s not even a COM!
Example 1: otkloadr.WRAssembly.1

- Will load “C:\Program Files\Microsoft Office\Office14\ADDINS\OTKLOADR.DLL,” which will result in loading linked non-ASLR MSVCR71.DLL in the same directory

- Disclosed by Parvez Anwar in June 2014 at http://www.greyhathacker.net/?p=770, already fixed by Microsoft
Example 2: mscormmc.dll

- This non-ASLR DLL is on the default Windows 7
  - C:\Windows\Microsoft.NET\Framework\v1.0.3705\mscormmc.dll

- A couple CLSIDs are associated on this DLL, for example:
  - {18BA7139-D98B-43C2-94DA-2604E34E175D}

- Then make an Office document or RTF containing an OLE object with the CLSID. You will get the non-ASLR DLL loaded into the process

- Still works! Finding non-ASLR DLL made easy; found this in just a few minutes
Consequence 2: Memory Corruption

- Sometimes, loading an “unprepared” DLL is enough to trigger a memory corruption

- Example: Microsoft Office Uninitialized Memory Use Vulnerability (CVE-2015-1770)
  - CLSID: CDDBCC7C-BE18-4A58-9CBF-D62A012272CE
  - Associated DLL: C:\Program Files\Microsoft Office\Office15\OSF.DLL
  - Just trying to load the CLSID-associated DLL will give you a crash (exploitable)!
  - The OSF.DLL is certainly not designed for you to load as OLE or ActiveX Control
  - Discovered by Yong Chuan Koh of MWR Labs, more details at
Consequence 3: DLL-Preloading

- There’s another attack scenario that hides in the deep.
- Note, this is about document-based attacking.

- The **current working directory** is something the attacker can control.

- I shouldn’t have to explain a DLL-Preloading attack.

---

22. Description. Detail a list of deliverables including documentation.


Microsoft Office contains a module that is vulnerable to DLL hijacking upon referenced from a crafted WebDAV or SMB share containing an Office file.
**DLL-Preloading Example: OLE Loading**

- **CVE-2015-2369** is a good example we reported, fixed just in July Patch Tuesday
- Minimal PoC in less than one tweet (140 bytes) 😊
  ```
  \rt\object\objocx\objdata
  01050000002000000014000000574D444D434553502E574D444D43
  4553502E31000000000000000000000041010500000000000
  ```
- CLSID-associated DLL
  - ProgID: WMDMCESP.WMDMCESP.1
  - CLSID: `{067B4B81-B1EC-489f-B111-940EBDC44EBA}`
  - DLL: `%systemroot%\System32\cewmdm.dll`
- Will result in loading a DLL named “rapi.dll” from the current working directory
- Demo!
Demo
Summary of Attacking Vectors

Based on the time-flow of a victim opening the document, the attack vectors are:

I. Various types of attacks may occur during the “CLSID-associated DLL Loading” process—the very first step of “OLE Object Initialization”
   - Non-ASLR DLL loading
   - Memory Corruption
   - DLL preloading
   - ... 

II. Various types of vulnerabilities may exist in the “IPersistStorage::Load” routine, another step of the “OLE Object Initialization”
   - A lot of zero-day attacks focus on this area

III. “Verb” action attack via “IOleObject::DoVerb”
   - Usually logic bugs, more dangerous
Every Step Attacked

User opens the document

- User performs action on the OLE object (e.g., clicking, double-clicking)
- “Verb” performed automatically by Office features (e.g., PowerPoint animation)

OLE object Initialized

“Verb” action performed

OLE DLL Loading (CoCreateInstance)

OLE Data Initialization (IPersistStorage::Load)
Summary of Attack Surface

- The OLE mechanism offers a huge attack surface
- Unlike ActiveX, an OLE object is not restricted by security enhancement features like “Pre-Approved List,” Safe For Scripting (SFS), or Safe For Initialization (SFI)
- Being able to load any* CLSID-associated DLL makes the attack surface even much bigger
  - Hundreds of OLE objects on default Windows
  - Thousands of CLSID-associated DLLs on default Windows
- Don’t forget it’s an open area!
  - The more apps installed, the bigger the surface becomes
  - It’s possible one day we’ll see a document-based attack targeting specific users having specific software installed on the system

*Note that the OLE-loading process honors the IE/Office Killbits, so if a CLSID is killbitted, the associated DLL will not be loaded.
Agenda

- What Is OLE?
- Historical Zero Days Involving OLE
- OLE Internals
- Attack Surface
- Conclusion
Conclusion

- The OLE mechanism serves the majority of Microsoft’s documentation interoperability with other components

- A huge attack surface offered
  - New ActiveX?
  - Even though it’s not scriptable, it can do much more than we expected

- What to expect next after the preso?
  - Many OLE-related vulnerabilities will probably be discovered
  - Probably more zero-day attacks targeting Office/WordPad
  - Detection and defense need to be improved*, for both sandboxing and static approaches
  - An OLE-specific detection method is on the way

*We have reported some new evasion tech recently (https://blogs.mcafee.com/mcafee-labs/threat-actors-use-encrypted-office-binary-format-evade-detection), suggesting the difficulties on detecting Office-based attack correctly.
Conclusion

- To vendor (Microsoft)
  - The questionable “OLE Loading” mechanism needs to be revisited, maybe redesigned
    - You can't just load every CLSID-associated DLL into the Office/WordPad process
  - A large-scale internal pentest on the default OS is needed
    - New attacking vectors produce many new vulnerabilities
- Training third-party vendors
  - Just like what you have done before for ActiveX
Major References

Thank You!

haifei.li@intel.com
bing.sun@intel.com

• We’d like to especially thank researcher James Forshaw, who helped peer-review the presentation
• Thanks to Chong Xu, Stanley Zhu, and Dan Sommer of Intel Security and Xiaoning Li of Intel Labs