Leveraging Excel DDE for lateral movement via DCOM

By Philip Tsukerman

Archived: 2026-04-05 19:56:32 UTC

DDE, or **Dynamic Data Exchange**, is a legacy interprocess communication mechanism that's been part of some Windows applications since as early as 1987. DDE enables applications to request items made available by other programs, such as cells in a Microsoft Excel spreadsheet, and be notified of any changes within these items.

Read a blog by Philip on a vulnerability with Excel 4.0 Macros.

The DDE mechanism has appeared in recent discussions on how to carry out macro-less code execution in Office documents. This technique works by making Excel (or other MS Office applications) evaluate an expression ("*=cmd|' /C calc'!A0*", for example) that requires data to be transmitted via DDE from another application. This allows an attacker to specify an arbitrary command line as the DDE server to be run, thus performing arbitrary code execution.

https://sensepost.com/blog/2017/macro-less-code-exec-in-msword/This functionality is implemented in the background using a method called **DDEInitiate**, which is exposed through COM. This made me wonder if the DDE functionality in Office applications could be used remotely through DCOM in a manner similar to the techniques described by Matt Nelson of SpecterOps (here and here) by trying to call this method through DCOM.

A quick look at the methods exposed by the **Excel.Application** object shows promise.

```
PS C:\Users\philip> $excel = [activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application"))
PS C:\Users\philip> $excel|Get-Member -Name DDE*


   TypeName: Microsoft.Office.Interop.Excel.ApplicationClass

Name              MemberType Definition
----              ---------- ----------
DDEExecute        Method     void DDEExecute(int Channel, string String), void _Application.DDEExecute(int Channel, string String)
DDEInitiate       Method     int DDEInitiate(string App, string Topic), int _Application.DDEInitiate(string App, string Topic)
DDEPoke           Method     void DDEPoke(int Channel, System.Object Item, System.Object Data), void _Application.DDEPoke(int Channel, System.Object Item, System.Object Data)
DDERequest        Method     System.Object DDERequest(int Channel, string Item), System.Object _Application.DDERequest(int Channel, string Item)
DDETerminate      Method     void DDETerminate(int Channel), void _Application.DDETerminate(int Channel)
DDEAppReturnCode Property   int DDEAppReturnCode {get;}


PS C:\Users\philip> _
```

Indeed, the DDEInitiate method exists and is even documented by MSDN.

Application.DDEInitiate Method (Excel)

office 365 dev account | Last Updated: 6/12/2017 | 1 Contributor 

Opens a DDE channel to an application.

Syntax

expression . **DDEInitiate**(***App*** , ***Topic***)

expression A variable that represents an **Application** object.

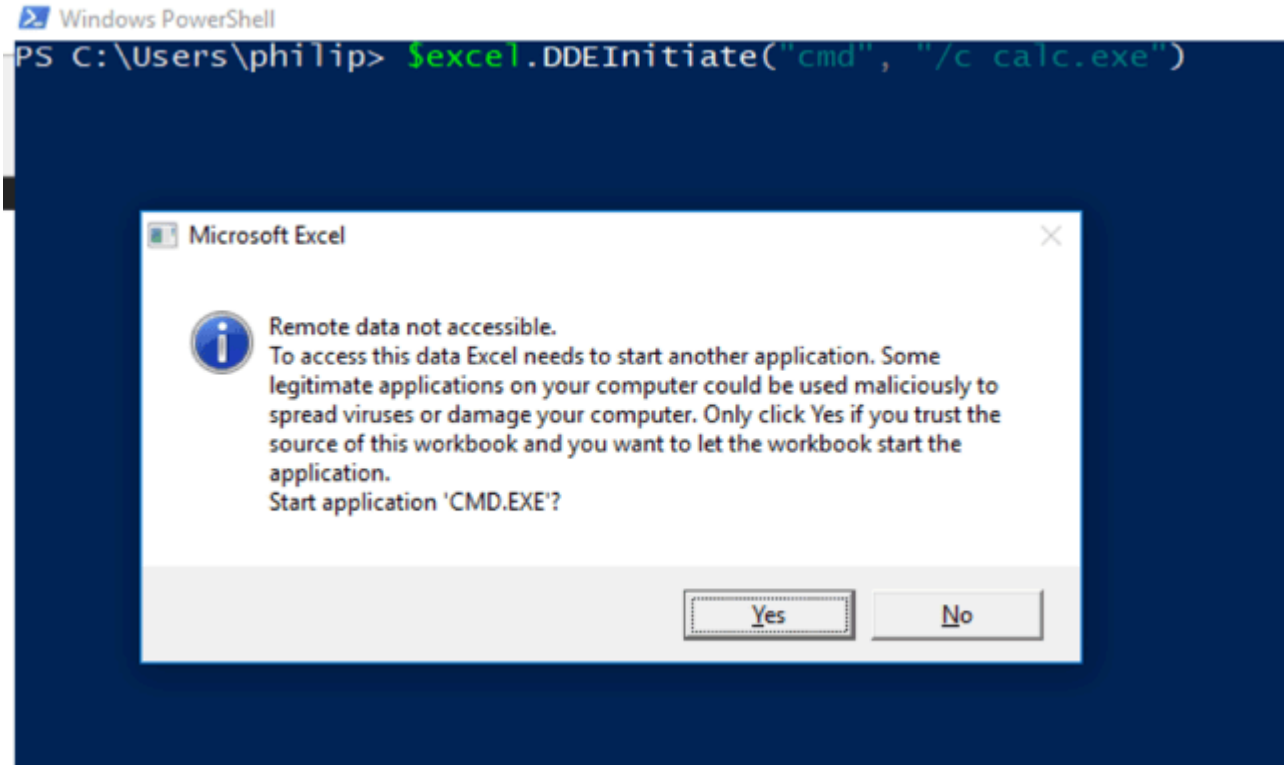
Parameters

Name	Required/Optional	Data Type	Description
App	Required	**String**	The application name.
Topic	Required	**String**	Describes something in the application to which you're opening a channel ? usually a document of that application.

We may supply "*cmd*" as the **App** parameter, while the **Topic** parameter will be any chosen command line arguments.

This method is not without its quirks; it limits the **App** parameter to eight characters (no directly calling PowerShell for you). 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.

Now let's try to actually run the method.



Initially, things didn’t go as I hoped. Clicking "*yes*" indeed spawned a shell that ran calc, but this isn't much of a lateral movement method if it requires a user on the victim machine to interact with this extremely suspicious alert (especially if they didn't even open Excel). We need to look at the DCOM object again for assistance.

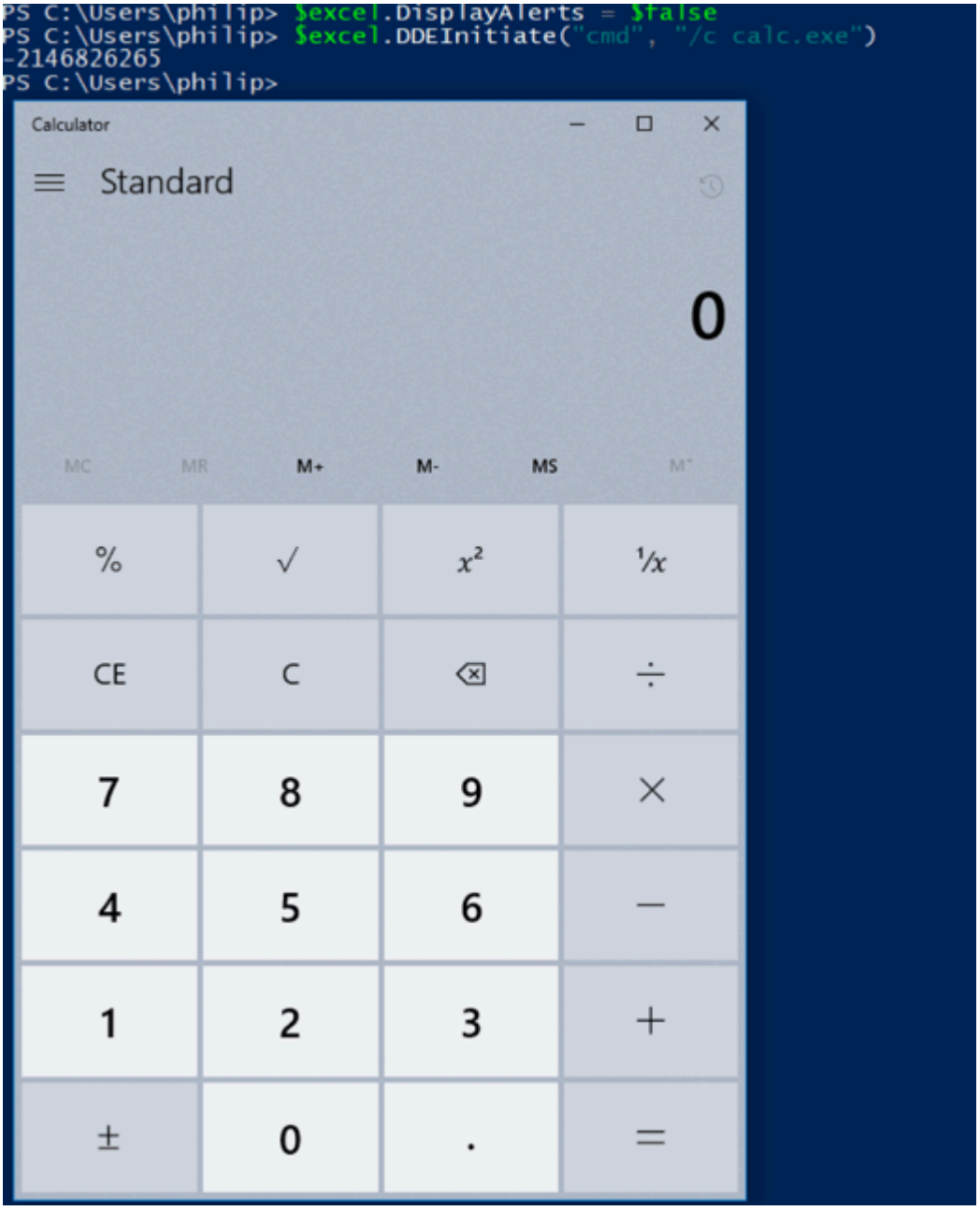
```
PS C:\Users\philip> $excel|Get-Member -Name *alert*

   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;}

PS C:\Users\philip> _
```

The **DisplayAlerts** property looks very promising, and playing with it gives good results.



It seems the **DisplayAlerts** property controls the alert presented by **DDEInitiate**.

Sadly, while some other Office applications, including MS Word, do expose the **DDEInitiate** method via DCOM, I have not been able to get it to work on anything but Excel.

While all of this was done on a single machine, getting this technique to work remotely can be done by simply replacing

```
PS C:\Users\philip> $excel = [activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application"))
```

with

```
PS C:\Users\philip> $excel = [activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Application", "XXX.XXX.XXX.XXX"))
```

Now all we need to do is replace the command line with our favorite PowerShell download cradle, and we're good to go with our new lateral movement technique.

Create a closed-loop, strategic security process for your defense. Read how to create a closed-loop security process with MITRE ATT&CK.

About the Author

Philip Tsukerman

Philip Tsukerman is a researcher Cybereason Innovation Labs.
