COM Hijacking Techniques

nccgroup

Who am I?

- David Tulis
- Offensive security practitioner and researcher
- Currently living in Philadelphia
- Senior Security Consultant with NCC Group
  - Red team operations lead
  - Windows and Active Directory pentesting
  - Social engineering, physical security

Twitter: @kafkaesqu3

nccgroup

Talk agenda

- COM fundamental concepts
- Real-world use cases
- The insecure COM object loading process
- Identifying hijack events on a system
- Hijack demos
- Techniques for stabilizing COM hijack
- Practical usage in offensive operations
- Takeaways for red/blue teams

nccgroup

Why am I doing a talk just about COM hijacking

- COM is deeply ingrained in all versions of Windows OS
- The interface can be used in unexpected ways
- An adversary likes to use things in unexpected ways
- COM hijack abuse is a method to have another process run untrusted code
  - Host persistence
  - Defence evasion
  - Lateral movement
- T1122 on the MITRE ATT&CK framework
  - MITRE has recorded 6 different threat groups abusing COM hijacking
- Minimal public information about COM hijacking abuse

nccgroup

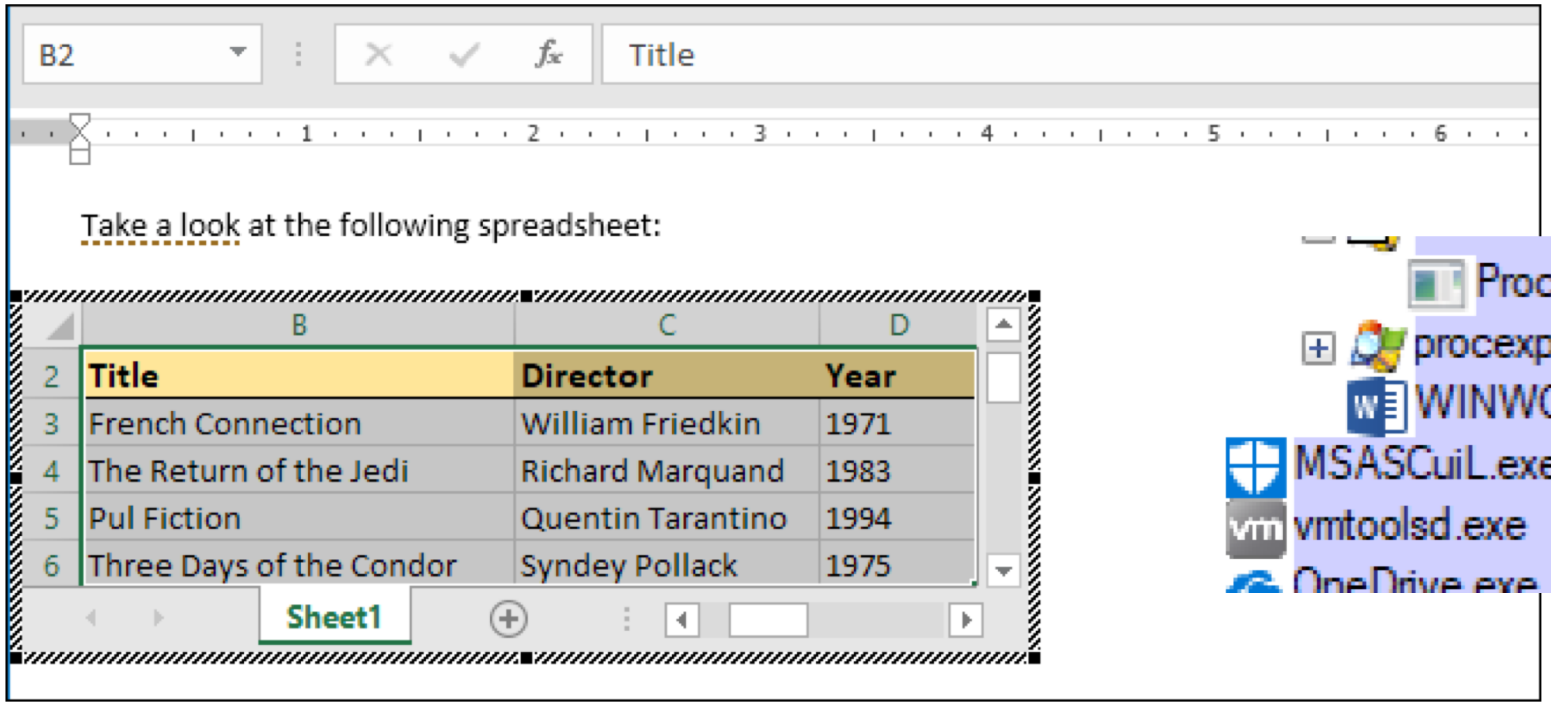
WTF is COM?

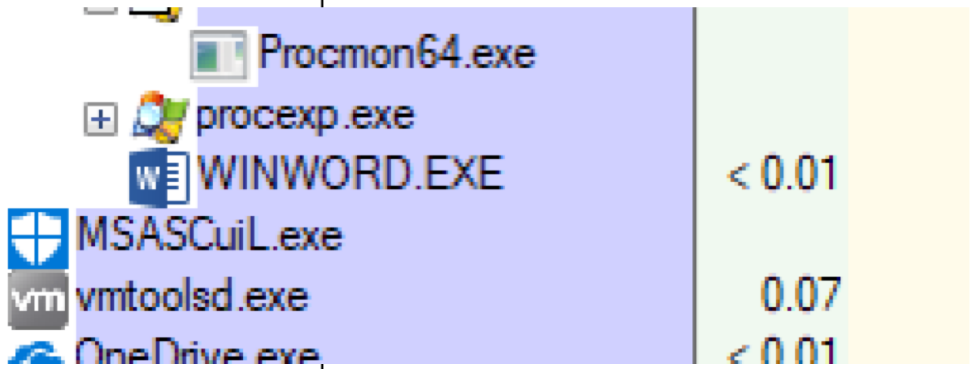
- COM = “Component Object Model”
- Released in 1992 in Windows 3.1
- COM lies at the core of many Windows OS technologies and frameworks
  - OLE, ActiveX, COM+, DCOM, Windows shell, DirectX, UMDF, WinRT
- COM is a standard to facilitate program interoperability, code reuse, and interprocess communications

nccgroup

COM example #1

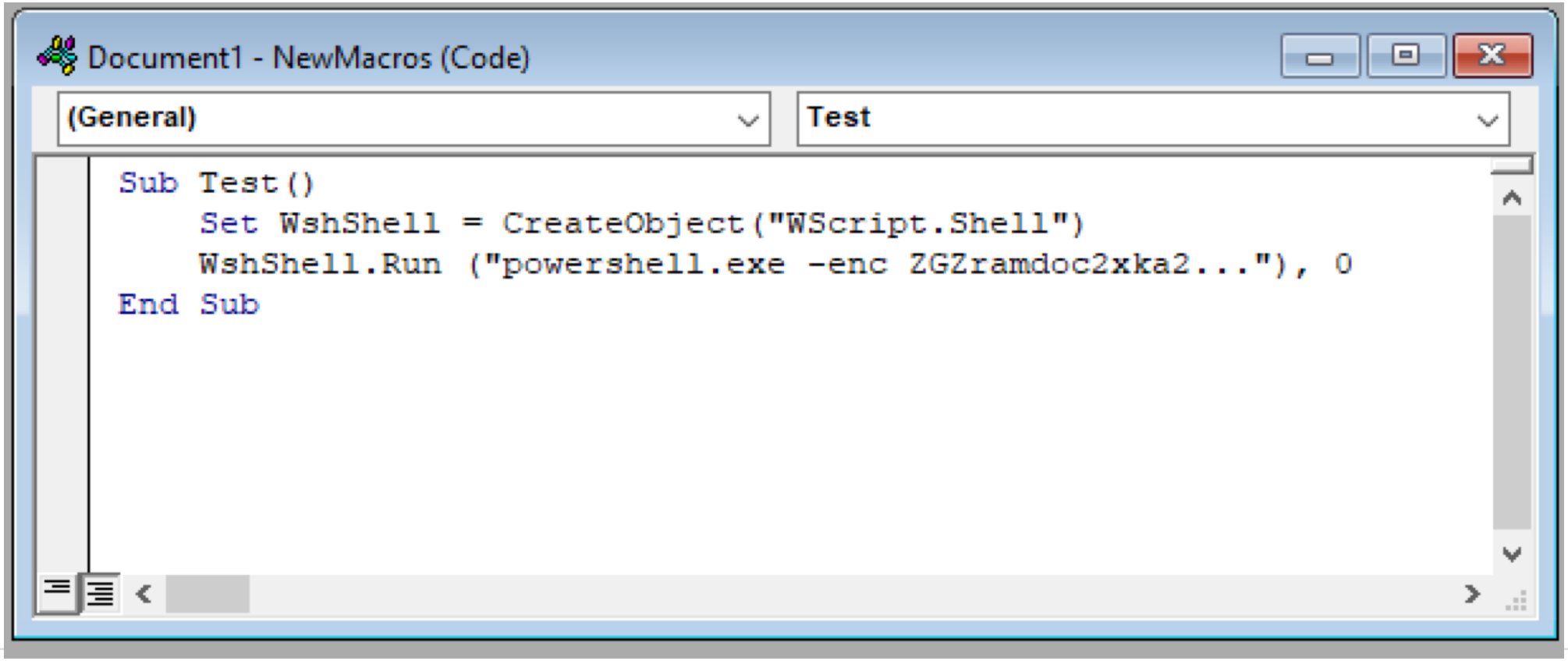
- COM was originally created by Microsoft engineers as a framework document linking and object sharing
- Through COM, you can get full access to the Excel API, in Word, without creating an Excel process





COM example #2

- Does this code look familiar?
- Creating and executing an arbitrary command in VBA, using Wscript.Shell COM object



nccgroup

Why COM?

Applications need a way to share objects, code.

- Statically linking .lib = developers must compile against your library, wasted space from duplicate libraries **(BAD)**
- Dynamic linking .dll = no versioning control, “DLL hell” **(BAD)**
- COM = **GOOD**

COM core principle: “separate the implementation from the interface”

COM libraries can can be called from any process, in any language
COM libraries can be modified without breaking the binary-level interface

nccgroup

COM server components

- COM library == COM Server
  - A COM server contains the library's implementation (code and classes), and an interface for accessing the implementation
- Other applications that want to interact with the server are the COM clients
- COM servers can be hosted in process (a DLL) or out of process (an EXE)
- COM classes are given names
  - ProgID: Friendly name (LibraryName.ClassName.Version)
  - CLSID: Unique name {5d58708c-5603-4ee0-81de-e88ec9ece235}
- COM interfaces are also given names an interface identifier (IID), also a GUID
- All COM interfaces expose 3 methods:
  - QueryInterface - retrieve references to the interfaces an object implements
  - AddRef/Release - increment/decrement object reference counter

nccgroup

COM client programming

- COM clients can be written in any language
  - .NET: COM Interops
  - VBA: CreateObject
  - Powershell: New-Object -ComObject
  - Python: pywin32
  - Ruby: win32ole
  - Node: win32com
  - Golang: win32com.client
- CoCreateInstance() is the Windows API call to activate an object
  - CoGetClassObject() to retrieve pointer to interface of specified CLSID
  - CreateInstance() to create an uninitialized object
- QueryInterface() to determine if a server supports a specific interface
  - COM DLLs export DllGetClassObject(), which is what QueryInterface() calls
- If you don’t know the CLSID of an object, use CLSIDFromProgID()

nccgroup

COM object registration

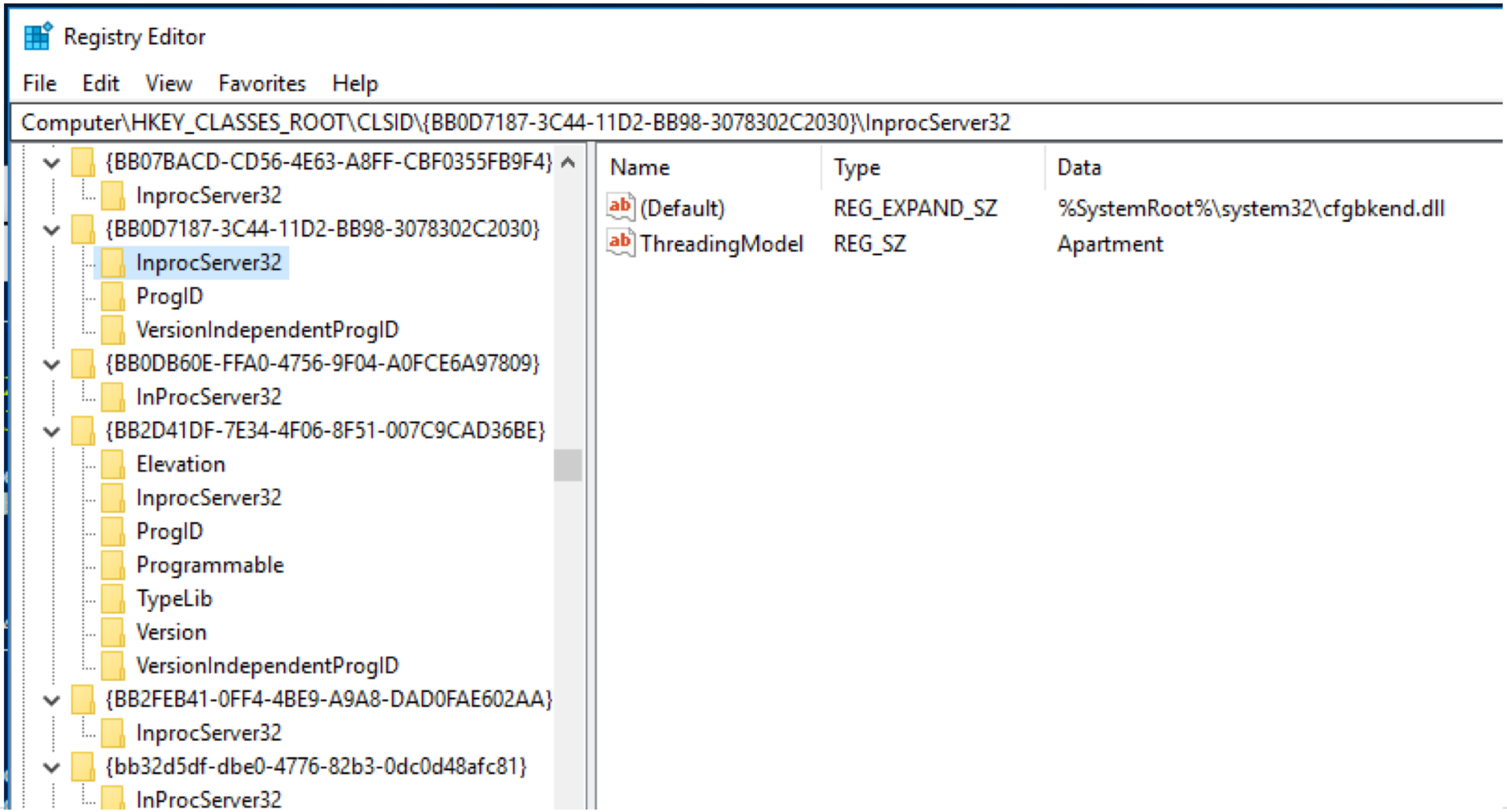
- COM objects can self-register using regsvr32.exe
- This is done through exported functions:
  - DllRegisterServer
  - DllUnregisterServer

COM object registration

Time ...	Process Name	PID	Operation	Path	Result
9:47:5...	regsvr32.exe	1968	RegQueryKey	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}	SUCCESS
9:47:5...	regsvr32.exe	1968	RegOpenKey	HKCU\Software\Classes\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}	NAME NOT FOUND
9:47:5...	regsvr32.exe	1968	RegSetValue	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\(Default)	SUCCESS
9:47:5...	regsvr32.exe	1968	RegCloseKey	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}	SUCCESS
9:47:5...	regsvr32.exe	1968	RegQueryKey	HKCU\Software\Classes	SUCCESS
9:47:5...	regsvr32.exe	1968	RegQueryKey	HKCU\Software\Classes	SUCCESS
9:47:5...	regsvr32.exe	1968	RegQueryKey	HKCU\Software\Classes	SUCCESS
9:47:5...	regsvr32.exe	1968	RegOpenKey	HKCU\Software\Classes\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\Inproc...	NAME NOT FOUND
9:47:5...	regsvr32.exe	1968	RegOpenKey	HKCR	SUCCESS
9:47:5...	regsvr32.exe	1968	RegQueryKey	HKCR	SUCCESS
9:47:5...	regsvr32.exe	1968	RegCreateKey	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\Inprocserver32	SUCCESS
9:47:5...	regsvr32.exe	1968	RegCloseKey	HKCR	SUCCESS
9:47:5...	regsvr32.exe	1968	RegQueryKey	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\Inprocserver32	SUCCESS
9:47:5...	regsvr32.exe	1968	RegQueryKey	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\Inprocserver32	SUCCESS
9:47:5...	regsvr32.exe	1968	RegOpenKey	HKCU\Software\Classes\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\Inproc...	NAME NOT FOUND
9:47:5...	regsvr32.exe	1968	RegSetValue	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\Inprocserver32\(Default)	SUCCESS
9:47:5...	regsvr32.exe	1968	RegCloseKey	HKCR\CLSID\{b4ed3329-8c6b-49aa-8315-de1734925901}\Inprocserver32	SUCCESS

nccgroup

COM object registration



COM object registration

Important subkeys:

- InprocServer/InprocServer32 - in process COM objects
- LocalServer/LocalServer32 - external COM objects, in another process
- InprocServer/LocalServer keys are for backwards compatibility for 16 bit exes and not common
- 32-bit, 64-bit processes/process servers use InprocServer32/LocalServer32

COM object registration registry hives:

- Per-user object registration: HKEY_CURRENT_USER\Software\Classes\CLSID
- System-wide registration: HKEY_LOCAL_MACHINE\Software\Classes\CLSID
- HKEY_CLASSES_ROOT (HKCR) is a virtual registry hive, showing merged view of HKCU and HKLM

nccgroup

COM object population survey

- Test box: Windows 10 Pro
  - Browsers: Chrome
  - Productivity: Microsoft Office 2016
- Total: 6557 CLSIDs
  - 6338 InprocServer32 keys
  - 219 LocalServer32 keys
  - 0 LocalServer/InprocServer keys):
- 18 keys in HKCU hive
- 6539 keys in HKLM hive
- 1679 unique COM server implementations on disk
  - 1608 in-process COM servers (.DLLs)
  - 149 out-of-process COM servers (.EXEs)
  - Misc file types (.ax, .cpl, .ocx)

nccgroup

Try this at home!

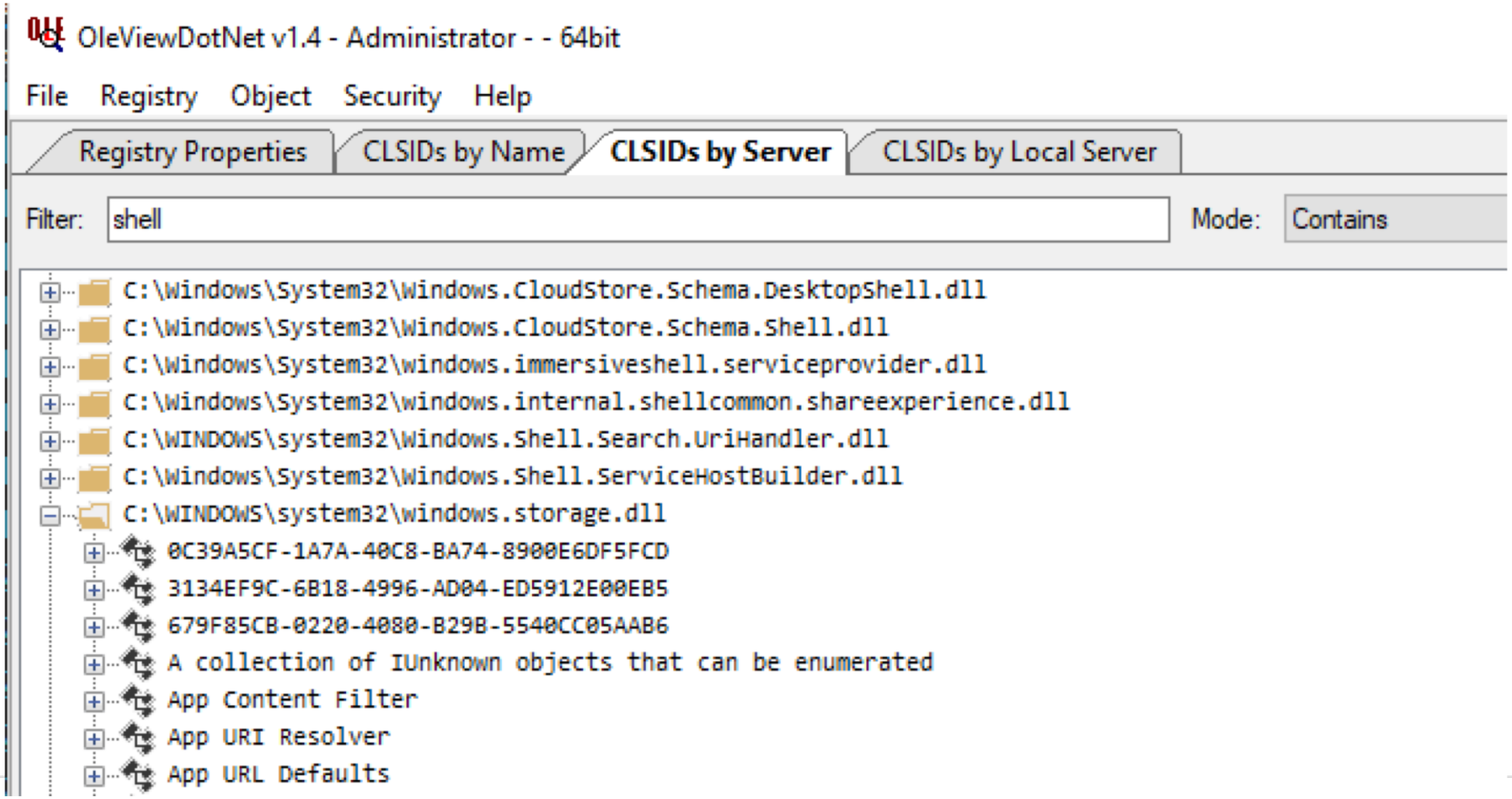
```
$HKCR_keys = Get-CLSIDRegistryKeys -RegHive HKCR
$HKCR_keys | where-object {$_ -like "*inprocserver"} | Measure-Object
$HKCR_keys | where-object {$_ -like "*inprocserver32"} | Measure-Object
$HKCR_keys | where-object {$_ -like "*localserver"} | Measure-Object
$HKCR_keys | where-object {$_ -like "*localserver32"} | Measure-Object

$HKLM_keys = Get-CLSIDRegistryKeys -RegHive HKLM
$HKLM_keys | Measure-Object
$HKCU_keys = Get-CLSIDRegistryKeys -RegHive HKCU
$HKCU_keys | Measure-Object
```

Powershell cmdlets from https://github.com/nccgroup/Accomplice/COMHijackToolkit.ps1

nccgroup

Exploring your system's COM components in detail

- Microsoft's Oleview, part of Windows SDK
- James Forshaw's OLEViewDotNet- https://github.com/tyranid/oleviewdotnet/



The vulnerability in the object resolution process

Keys are loaded from

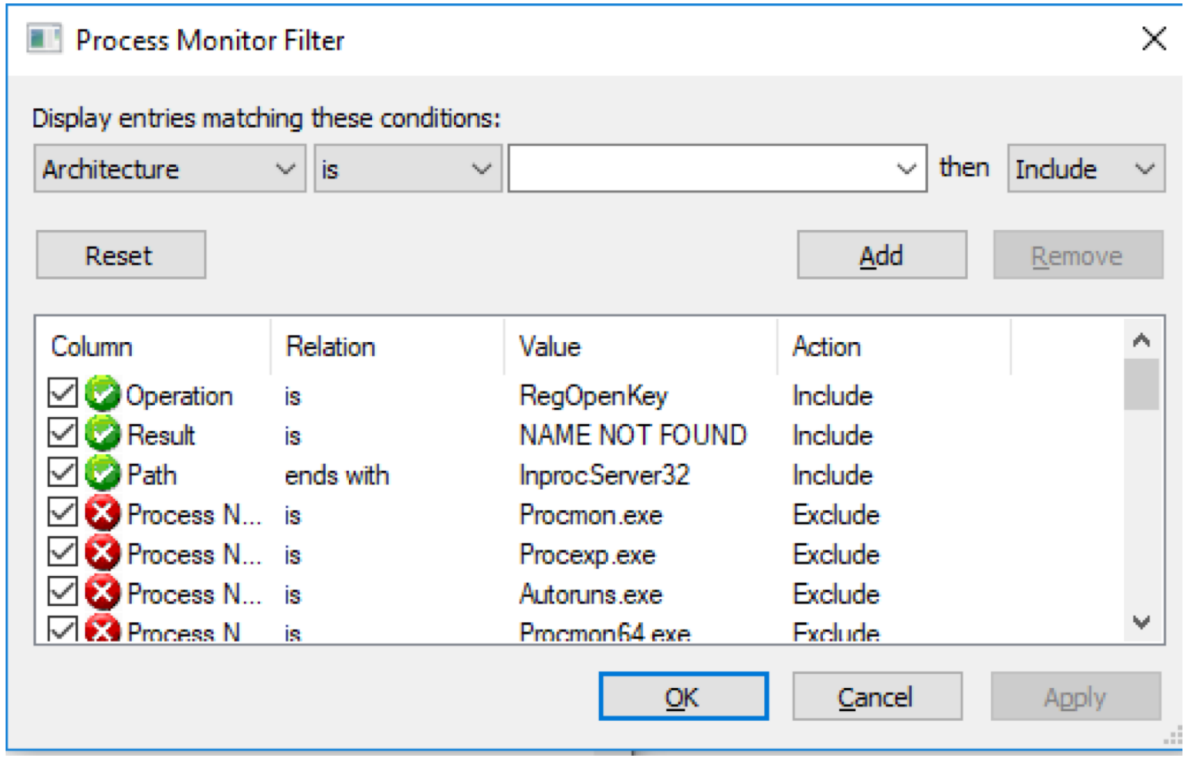
1) Per-user object registration: HKCU\Software\Classes\CLSID < -- **PRECEDENCE**
2) System-wide object registration: HKLM\Software\Classes\CLSID

- Per-user COM registration has precedence over system-wide COM object registration
  - This means keys are read from HKCU before they are read from HKLM
  - Exception to this rule: elevated process read directly from HKLM to prevent trivial privilege escalation
- As we saw above, the **vast majority of objects are registered at system-wide**
- **No special privileges are required to add keys to HKCU!**
- Even if client read keys from HKCR (HKLM+HKCU), a CLSID duplicated in HKLM and HKCU will only show the value from HKCU

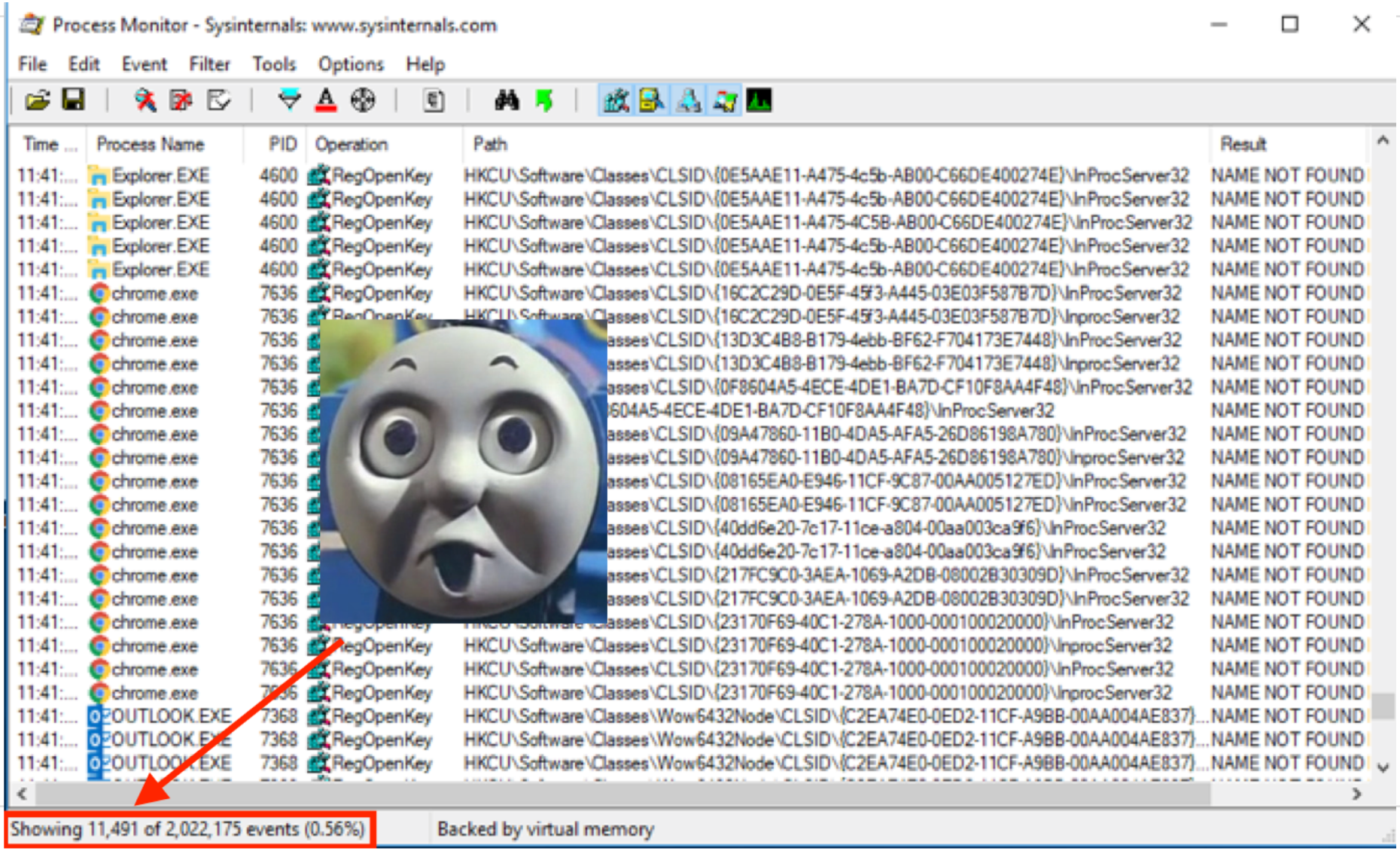
nccgroup

Finding hijackable keys

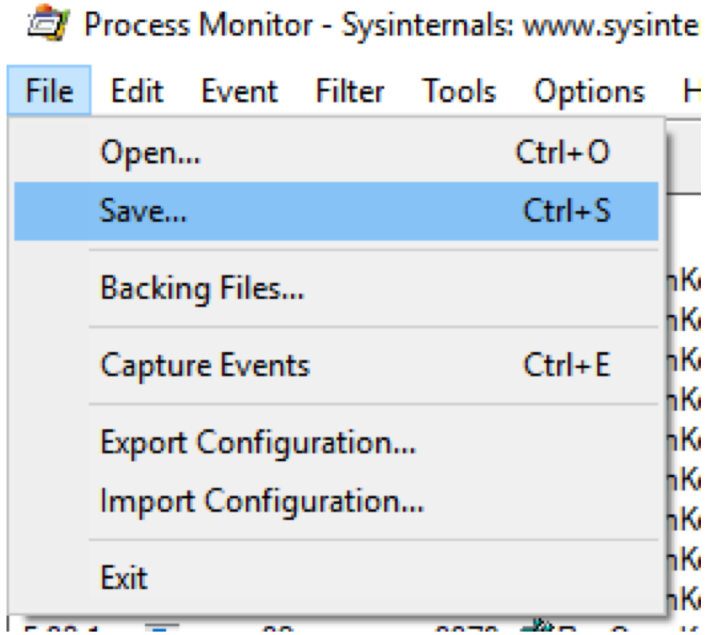
- Test box is Windows 10 Pro with Chrome and Office 2016 installed
- Because the largest number of keys are in-process, we will focus on these
- Testing strategy = Identify COM objects activated:
  - At regular intervals
  - At application launch
  - By common actions (open file, click icon, etc)
- Testing time: About 5 minutes?
- Procmon filters:
  - Operation is RegKeyOpen
  - Result is NAME NOT FOUND
  - Path ends with InprocServer32
  - Exclude if path starts with HKLM

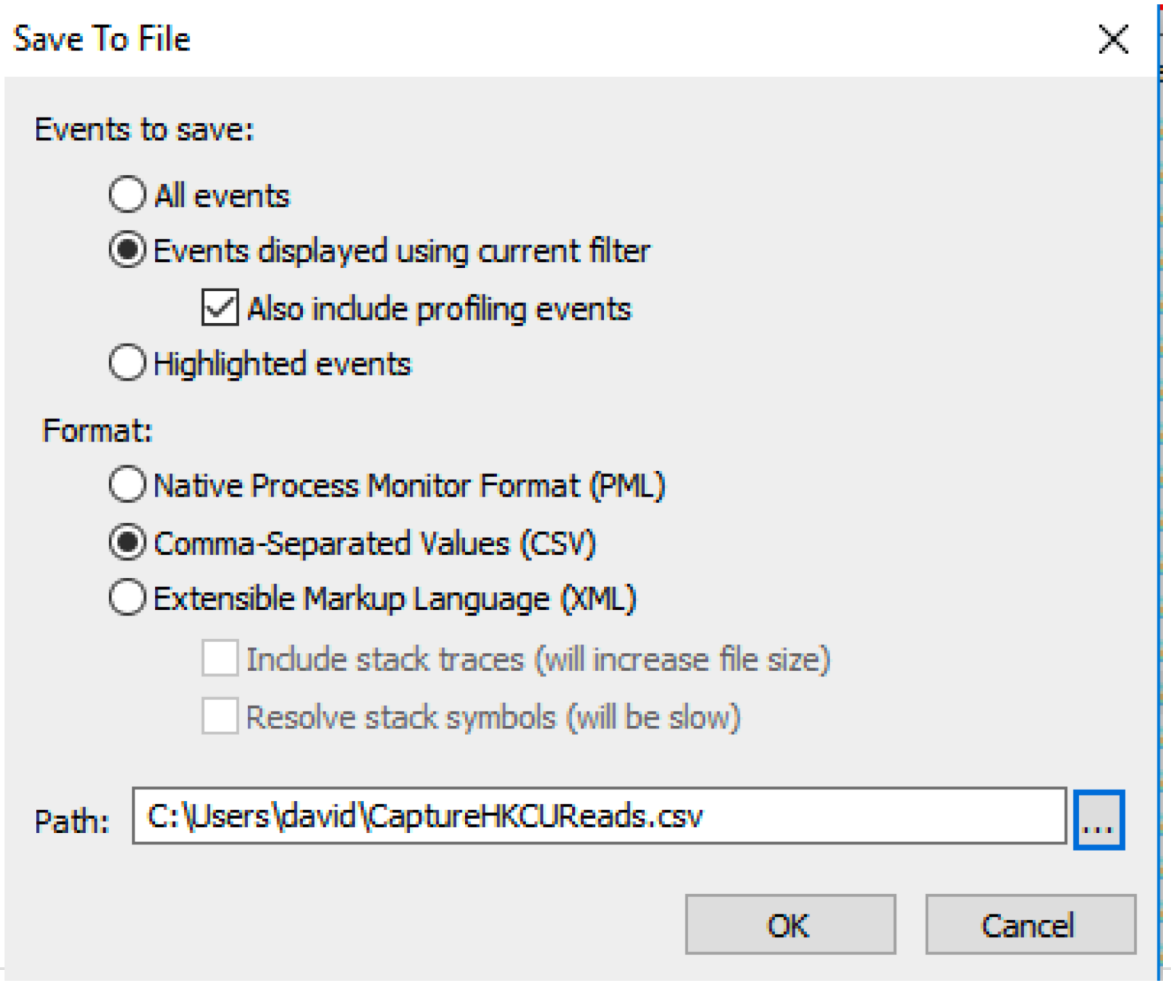


Test results



Export events from Procmon as CSV





Results

```
Extract-HijackableKeysFromProcmonCSV -CSVfile CaptureHKCUReads.CSV
```

Results: 628 unique InprocServer32 CLSID keys we can hijack:

- explorer.exe:122
- winword.exe:99
- powerpnt.exe:95
- excel.exe:90
- chrome.exe:87
- outlook.exe:85
- other: 50

cmdlet from https://github.com/nccgroup/Accomplice/COMHijackToolkit.ps1

nccgroup

Creating a target library

- InprocServer32 target must be a DLL
- DLL does not need to be a valid COM library
  - No need to implement COM registration or interface querying functions
- DLL files have an initialization function called DllMain, the perfect place for our code
  - Whenever process loads or unloads the DLL, DllMain is called
  - Whenever a process creates or exits a thread, DllMain is called
  - In the case of COM activation, calls to CoCreateInstance() or CreateInstance() to a CLSID with an in-process COM server will trigger DllMain
- Loader lock is held system-wide before entering DllMain
- Loader lock exists to address the fragility of the Windows loader
  - Can’t load any other DLLs
  - Can only call functions already mapped into memory (kernel32.dll)
  - Can’t lock anything, or wait for any locks to be released
  - Can’t wait for other processes/threads to complete

nccgroup

DllMain implementation

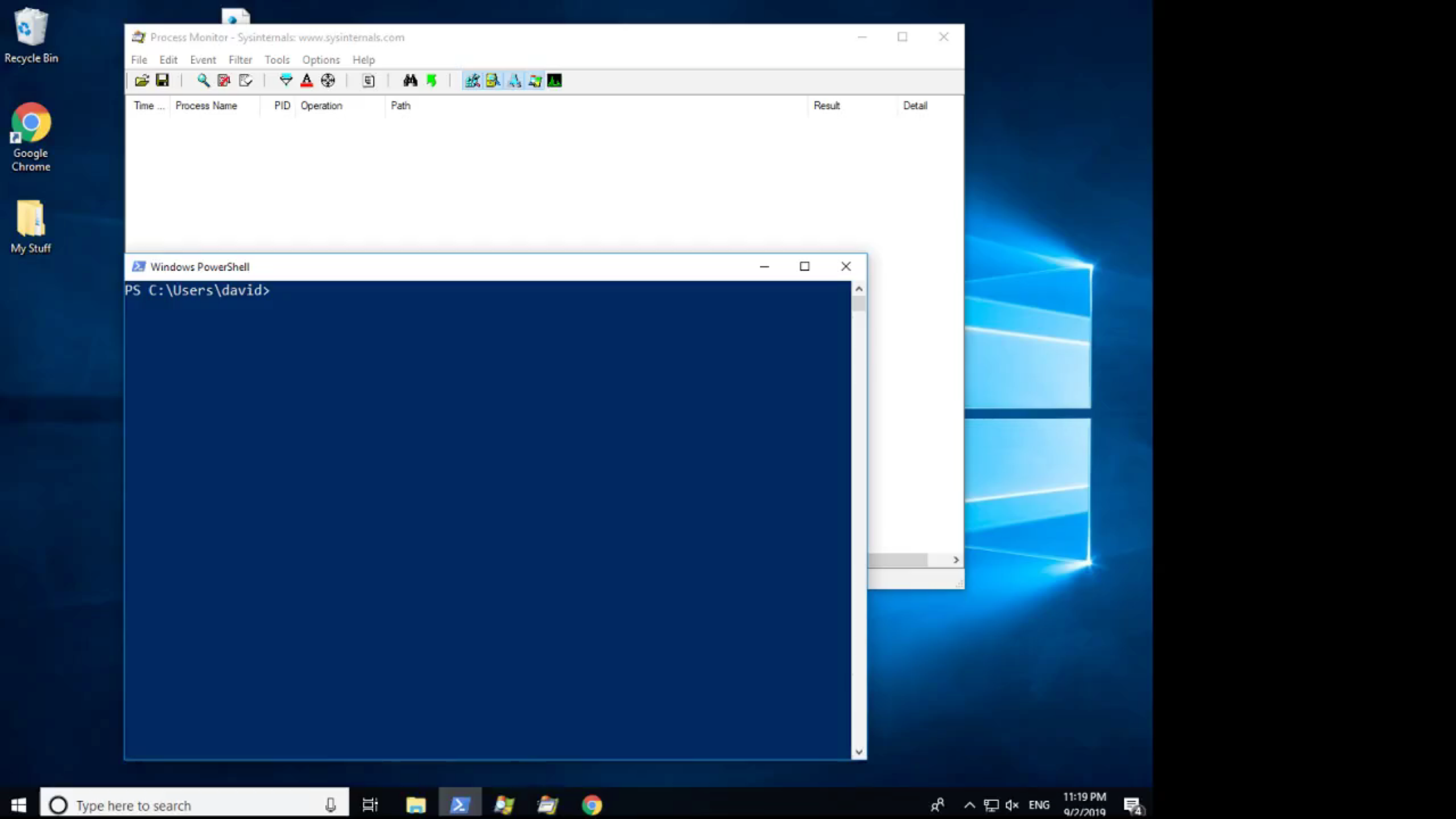
```
BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul_reason_for_call, LPVOID
lpReserved)
{
    switch (ul_reason_for_call)     {
    case DLL_PROCESS_ATTACH: {
        DoEvilStuff(NULL);
        break;
    }
    case DLL_THREAD_ATTACH:
    case DLL_THREAD_DETACH:
    case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}
```


nccgroup

Option #1: Create process

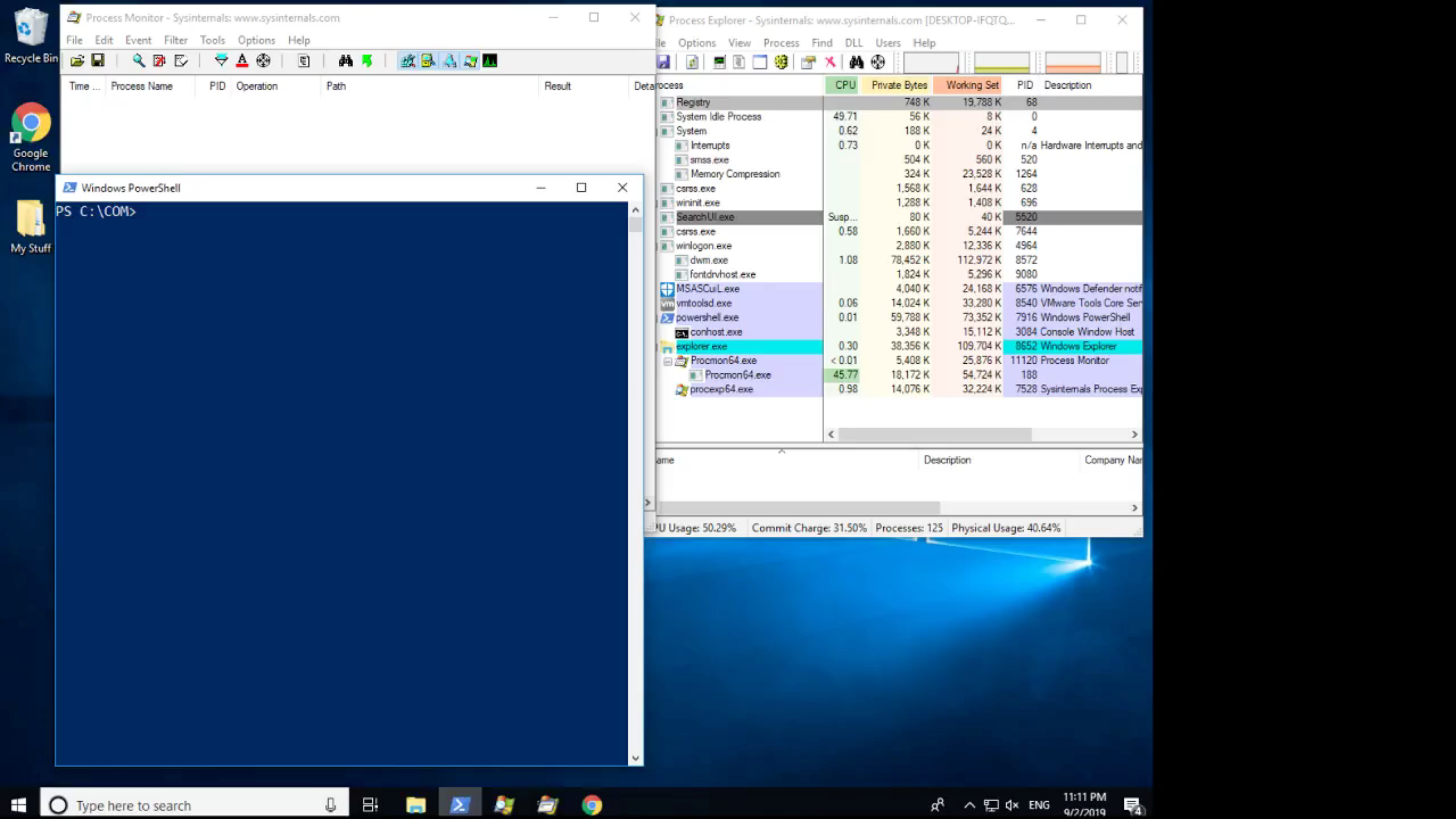
```
//spawning a new process
DWORD WINAPI DoEvilStuff(LPVOID lpParam) {
  STARTUPINFO  info = { 0 };
  PROCESS_INFORMATION processInfo;
  std::wstring cmd = L"C:\\Windows\\System32\\calc.exe";
  BOOL hR = CreateProcess(NULL, (LPWSTR)cmd.c_str(), NULL, NULL,
                          TRUE, 0, NULL, NULL, &info, &processInfo);
  if (hR == 0) {
      return 1;
  }
  return 0;
}
```


nccgroup


Recycle Bin
Google Chrome
My Stuff
Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
Time ... Process Name PID Operation Path Result Detail
Windows PowerShell
PS C:\Users\david>
Type here to search
ENG
11:19 PM
9/2/2019

Option #2: Thread injection

- Spawning processes is expensive
- We are safe to call functions in kernel32.dll while loader lock is held
  - OpenProcess()
  - VirtualAllocEx()
  - WriteProcessMemory()
  - CreateRemoteThread()
- With this recipe, we can inject shellcode into another process
- https://github.com/theevilbit/injection/blob/master/SimpleThreadInjection/SimpleThreadInjection/SimpleThreadInjection.cpp


Recycle Bin
Google Chrome
My Stuff
Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
Time ... Process Name PID Operation Path Result Detail
Windows PowerShell
PS C:\COM>
Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-IFQTQ...
Options View Process Find DLL Users Help
Process CPU Private Bytes Working Set PID Description
Registry 748 K 19,788 K 68
System Idle Process 49.71 56 K 8 K 0
System 0.62 188 K 24 K 4
Interrupts 0.73 0 K 0 K n/a Hardware Interrupts and
smss.exe 504 K 560 K 520
Memory Compression 324 K 23,528 K 1264
csrss.exe 1,568 K 1,644 K 628
wininit.exe 1,288 K 1,408 K 696
SearchUI.exe Susp... 80 K 40 K 5520
csrss.exe 0.58 1,660 K 5,244 K 7644
winlogon.exe 2,880 K 12,336 K 4964
dwm.exe 1.08 78,452 K 112,972 K 8572
fontdrvhost.exe 1,824 K 5,296 K 9080
MSASCuiL.exe 4,040 K 24,168 K 6576 Windows Defender notif
vmtoolsd.exe 0.06 14,024 K 33,280 K 8540 VMware Tools Core Ser
powershell.exe 0.01 59,788 K 73,352 K 7916 Windows PowerShell
conhost.exe 3,348 K 15,112 K 3084 Console Window Host
explorer.exe 0.30 38,356 K 109,704 K 8652 Windows Explorer
Procmon64.exe < 0.01 5,408 K 25,876 K 11120 Process Monitor
Procmon64.exe 45.77 18,172 K 54,724 K 188
procexp64.exe 0.98 14,076 K 32,224 K 7528 Sysinternals Process Ex
Description Company Na
U Usage: 50.29% Commit Charge: 31.50% Processes: 125 Physical Usage: 40.64%
Type here to search
ENG
11:11 PM
9/2/2019

Option #2: Thread injection

- **Pros**: This injection technique prevents spawning new processes
- **Cons**: Shellcode injection techniques are well known to defender
- There are lots of injection techniques, some more well known than others
  - Injection technique implementation examples https://github.com/theevilbit/injection
  - Anything that only uses kernel32.dll will work!

nccgroup

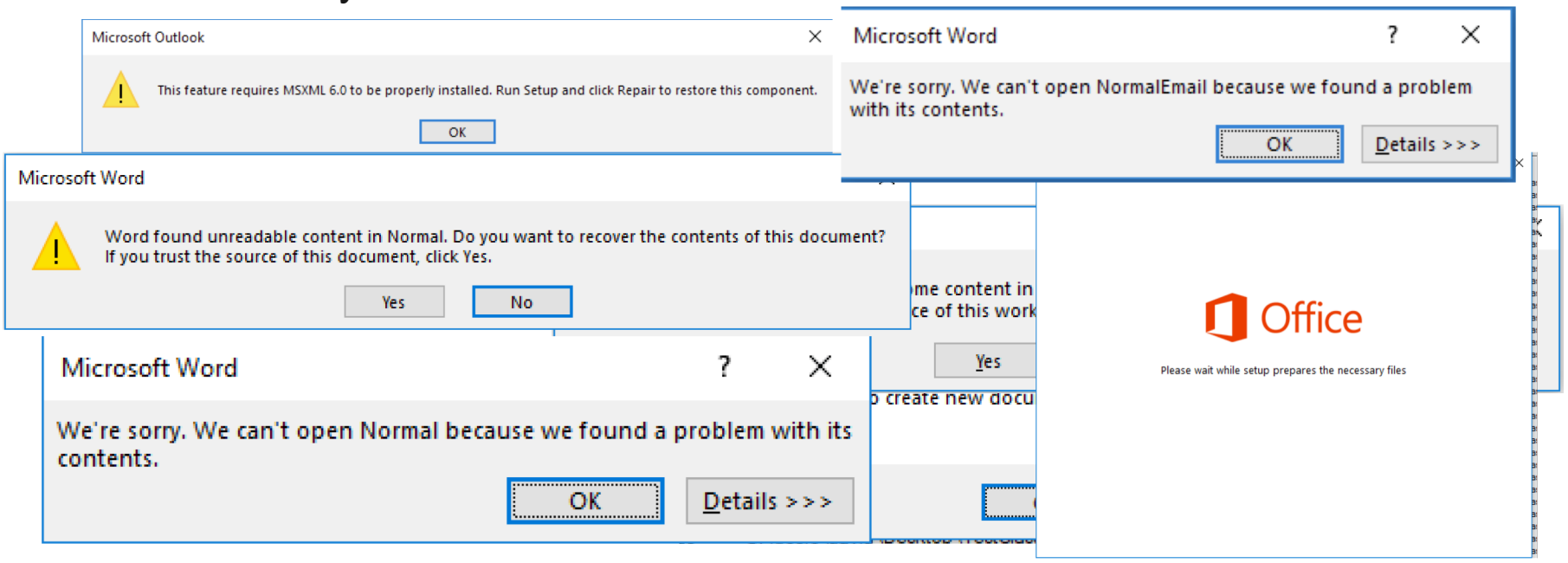
Option #3: CreateThread()

- Can we keep ourselves running in the COM client process?
- We COULD call CreateThread() inside DLLmain, but the thread won't run until DllMain exits and loader lock is released
- An alternative to running malicious code in DllMain is to run when the COM object is activated
- DllGetClassObject() is an exported function called when a COM object is activated
- There is NO loader lock inside DllGetClassObject()
- DllCanUnloadNow() – when a client calls CoFreeUnusedLibraries() to free up space, this function will be called
    - Return S_FALSE to tell clients the library is still being used and not to unload it

nccgroup

COM client post-hijack behavior

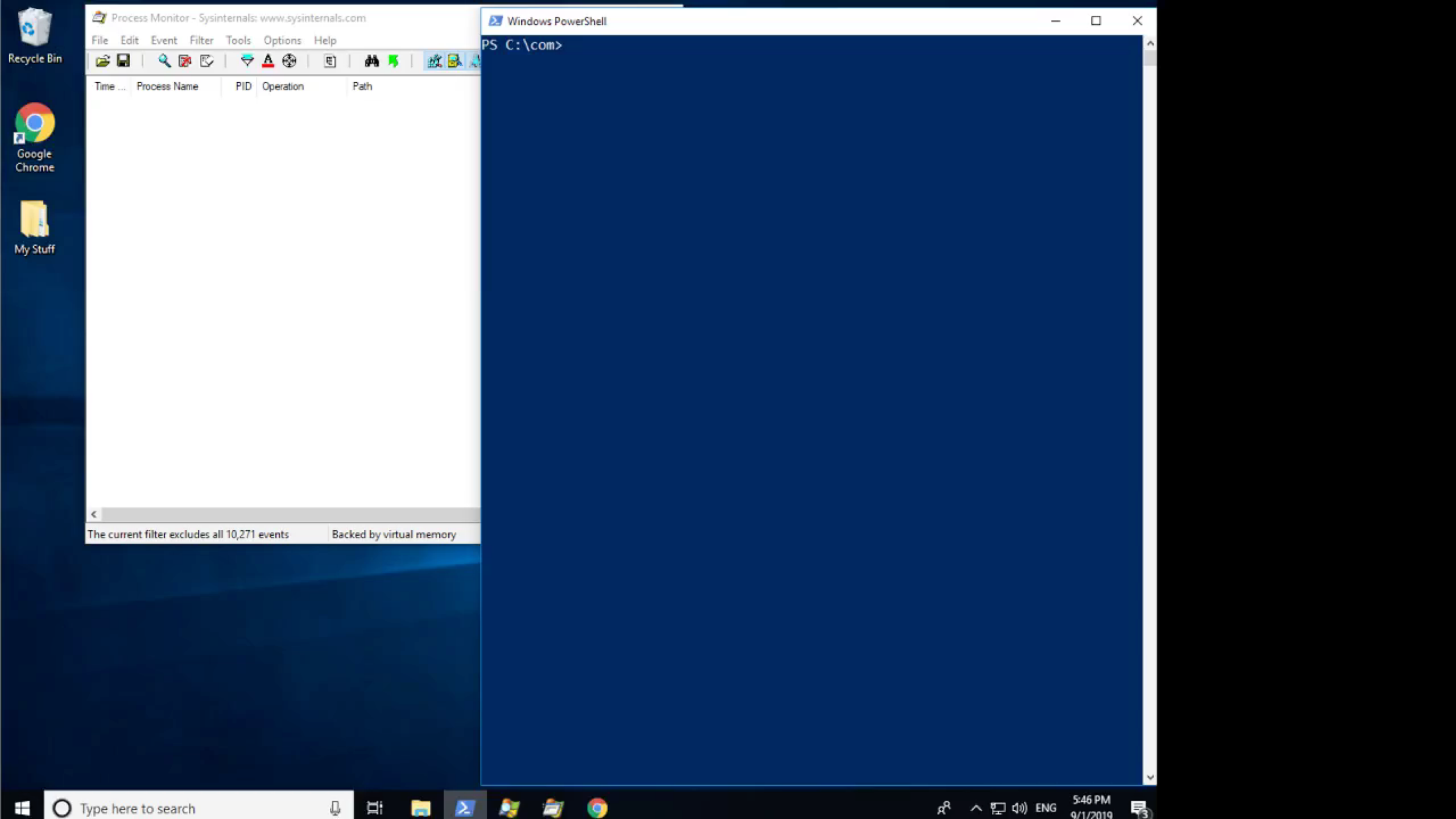
- We have no guarantee that our thread will be forcibly killed and our DLL unloaded
  - DLL name and path not what is expected
  - DLL has no exports
  - We aren't implementing the COM interfaces the client is expected
- The COM client may crash or misbehave



Measuring thread lifetimes post-hijack

- Some COM clients don’t care that no COM object was loaded, and the thread stays running
- Some COM clients kill the thread but keep DLL loaded. Others crash completely
- Some example CLSIDs you can hijack without being outright killed or unloaded (DllMain or DllGetClassObject):
  - excel.exe - {33C53A50-F456-4884-B049-85FD643ECFED}
  - explorer.exe - {A4B544A1-438D-4B41-9325-869523E2D6C7}
  - outlook.exe - {529A9E6B-6587-4F23-AB9E-9C7D683E3C50}
  - chrome.exe - {1F486A52-3CB1-48FD-8F50-B8DC300D9F9D}
- Tedious testing process, no guarantees on future stability
- Create threads both in DllMain and DllGetClassObject
  - DllGetClassObject may be called more than once
- Thread updates a file with its progress
  - Instead of creating and writing to a file, you could have this function do “whatever you want” ;)
  - Be mindful of loader lock restrictions on the thread created in DllMain
  - Thread is started when loader lock is released

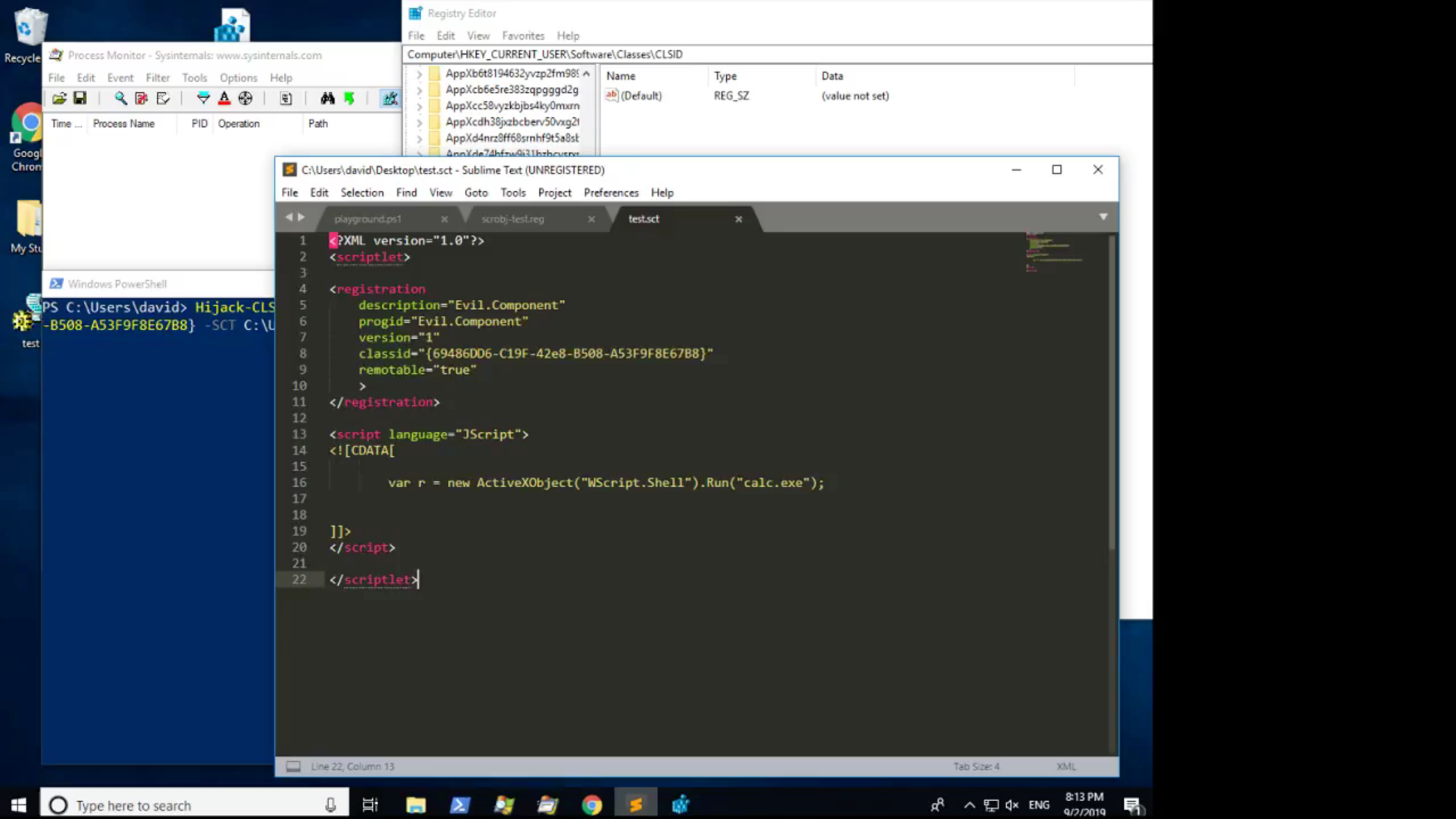
nccgroup


Recycle Bin
Google Chrome
My Stuff
Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
Time ... Process Name PID Operation Path
The current filter excludes all 10,271 events
Backed by virtual memory
Windows PowerShell
PS C:\com>
Type here to search
ENG
5:46 PM
9/1/2019

Alternative to DLLs: Scriptlets

- Maybe you aren't comfortable dropping DLLs into target environment
- COM servers can be Windows Scripting Components (.sct files)
- Technique originally documented by @subtee and @enigma0x3
  - https://www.slideshare.net/enigma0x3/windows-operating-system-archaeology
- InprocServer32 = C:\Windows\system32\scrobj.dll
- ScripletURL = Location of .sct file (local on disk or a URL)
- Blue team take note: This is NOT a common occurrence

nccgroup


Registry Editor
File Edit View Favorites Help
Computer\HKEY_CURRENT_USER\Software\Classes\CLSID
Name Type Data
(Default) REG_SZ (value not set)
Process Monitor - Sysinternals: www.sysinternals.com
File Edit Event Filter Tools Options Help
Time ... Process Name PID Operation Path
Windows PowerShell
PS C:\Users\david> Hijack-CLS
-B508-A53F9F8E67B8} -SCT C:\U
C:\Users\david\Desktop\test.sct - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
playground.ps1
scrobj-test.reg
test.sct
1 <?XML version="1.0"?>
2 <scriptlet>
3
4 <registration
5 description="Evil.Component"
6 progid="Evil.Component"
7 version="1"
8 classid="{69486DD6-C19F-42e8-B508-A53F9F8E67B8}"
9 remotable="true"
10 >
11 </registration>
12
13 <script language="JScript">
14 <![CDATA[
15
16 var r = new ActiveXObject("WScript.Shell").Run("calc.exe");
17
18
19]]>
20 </script>
21
22 </scriptlet>
Line 22, Column 13
Tab Size: 4
XML
Type here to search
ENG
8:13 PM
9/2/2019

Hijacking abandoned keys

- In some cases, there will be CLSIDs Inprocserver32 keys that point to a DLL that doesn't exist on disk
- Most COM servers live in C:\Windows\System32, which is writable by Administrators only
- Default Windows does not have any missing COM servers,
- Systems with 3rd party software (especially when software components have been uninstalled) might have more
- Example of an abandoned key left by an old version of Google Chrome (note: folder is only writable to administrators)

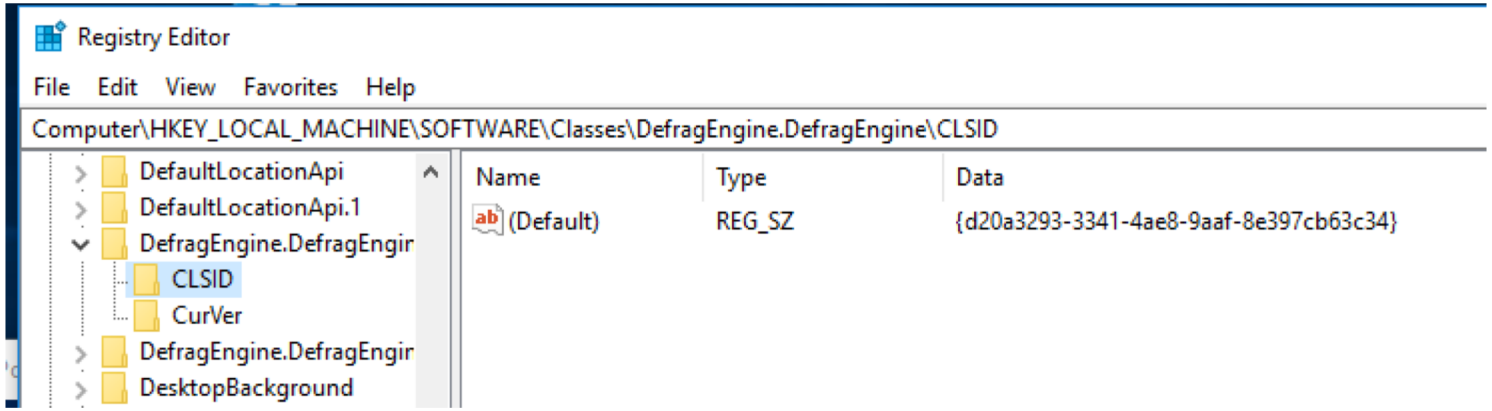
```
PS C:\Users\david> Find-MissingLibraries -CheckAccess
Write access to C:\Program Files (x86)\Google\Update: 0FD16473-86A0-4991-
B88A-D48733BF9873 -> C:\Program Files (x86)\Google\Update\1.3.33.23\psmac
hine_64.dll
```

- Technique originally documented by @bohops: https://bohops.com/2018/06/28/abusing-com-registry-structure-clsid-localserver32-inprocserver32/

nccgroup

ProgID resolution hijacking

- ProgIDs are the not-guaranteed-to-be-unique names of COM classes
- ProgIDs are resolved to their CLSIDs in the registry keys:
  - HKCU\Software\Classes
  - HKLM\Software\Classes
- A ProgID can be hijacked by adding a false ProgID to CLSID mapping in HKCU
- When a clients activates a COM object using the ProgID, the OS will resolve the ProgID by reading HKCU\Software\Classes\ProgID



TreatAs hijacking

- “TreatAs” registry key signals that a CLSID can be emulated by another CLSID
- Originally documented by @subtee and @enigma0x3 in their Windows Operating System Archaeology presentation
- TreatAs key points to another CLSID which is capable of emulating the current CLSID
- All activation requests are forwarded to the CLSID in TreatAs
- If emulated class doesn’t implement the right interfaces, there will be program instability

1) Create the malicious CLSID in HKCU, with a target COM server of choice
2) Hijack legitimate CLSID by adding TreatAs subkey, pointing to attacker’s CLSID

nccgroup

Persisting via COM hijack

COM hijacking primary use case: persistence

1) Drop SCT/DLL/EXE to disk
2) Add a registry key pointing attackers COM server, under the CLSID you want to hijack (optional if abusing an abandoned key)
    a) InprocServer32 -> DLL
    b) LocalServer32 -> EXE
    c) InprocServer32 -> Scrobl, ScriptletURL -> SCT file
    d) TreatAs/ProgID
3) Wait for CLSID to be activated

- Not a common technique
- No reboots required
- COM hijacks are invisible to Sysinternals Autoruns

nccgroup

COM proxying

Signature for DllGetClassObject:

```
HRESULT DllGetClassObject(REFCLSID rclsid, REFIID iid, LPVOID*ppv)
```

When a COM client loads the library's DllGetClassObject function:

1) Start a thread DoEvilStuff()
2) Given rclsid argument, read COM DLL location from registry
3) Load the legitimate DLL
4) Find address of DllGetClassObject in legitimate library
5) Pass arguments to DllGetClassObject in legitimate DLL
6) In DllCanUnloadNow, wait for EvilStuff() to finish before unloading

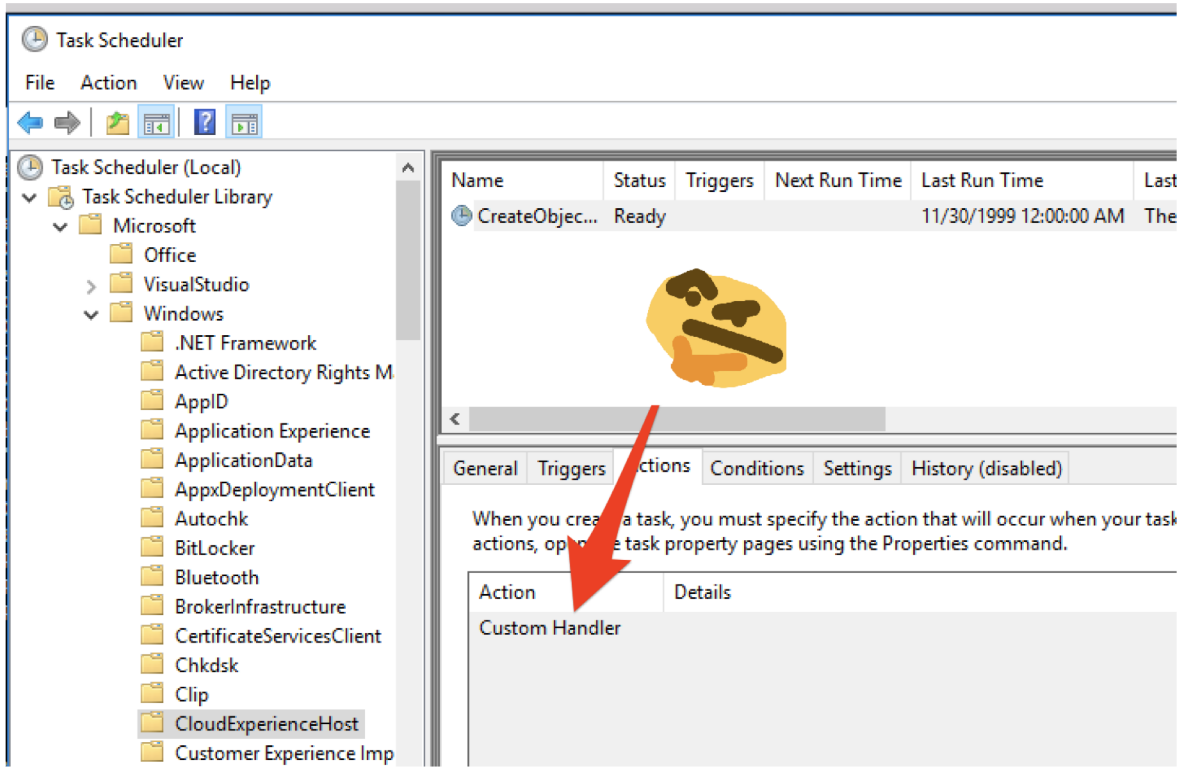
We can LoadLibrary and wait for threads, because DllGetClassObject isn't called during loader lock

Thanks Leo! https://github.com/leoloobeek/COMProxy/

nccgroup

Persisting via COM hijack of scheduled task handler

- Originally documented by Matt Nelson in 2016: https://enigma0x3.net/2016/05/25/userland-persistence-with-scheduled-tasks-and-com-handler-hijacking/



Persisting via COM hijack of scheduled task

- schtasks /query /xml /tn '\Microsoft\Windows\Shell\CreateObjectTask

https://github.com/enigma0x3/Misc-PowerShell-Stuff/blob/master/Get-ScheduledTaskComHandler.ps1

TaskName	CLSID	Dll	Logon	IsUserContext
AD RMS Rights Policy Template Management (Automated)	{CF2CF428-325B-48D3-8CA8-7633E36E5A32}	C:\Windows\system32\msdrm.dll	True	True
AD RMS Rights Policy Template Management (Manual)	{BF5CB148-7C77-4d8a-A53E-D81C70CF743C}	C:\Windows\system32\msdrm.dll	True	True
SystemTask	{58fb76b9-ac85-4e55-ac04-427593b1d060}	C:\Windows\system32\dimsjob.dll	False	True
UserTask	{58fb76b9-ac85-4e55-ac04-427593b1d060}	C:\Windows\system32\dimsjob.dll	True	True
UserTask-Roam	{58fb76b9-ac85-4e55-ac04-427593b1d060}	C:\Windows\system32\dimsjob.dll	False	True
KernelCeipTask	{e7ed314f-2816-4c26-aeb5-54a34d02404c}	C:\Windows\System32\kernelceip.dll	False	True
UsbCeip	{c27f6b1d-fe0b-45e4-9257-38799fa69bc8}	C:\Windows\System32\usbceip.dll	False	True
Scheduled	{c1f85ef8-bcc2-4606-bb39-70c523715eb3}	C:\Windows\System32\sdiagschd.dll	False	True
WinSAT	{A9A33436-678B-4C9C-A211-7CC38785E79D}	C:\Windows\system32\WinSATAPI.dll	False	True

Process injections via COM hijack

- Process injection is a defence evasion technique, where malicious code is run by other processes on the system
- Lots of documented techniques for performing this
- COM hijacking is a unique method of injecting code into another process's address space through registry manipulations only
- No calling of "suspicious" Windows APIs
- By mapping which CLSIDs are associated with which processes, hijacking particular keys will cause injection into particular processes
- Drawback: Potential "time delay" until event occurs which triggers the hijack

nccgroup

COM abuse to blend in and misdirect

- Advanced attackers want to blend in to the environment they are operating in
- These techniques do not necessarily rely on a hijack, but could be combined with other techniques to misdirect defensive actions
- Application whitelisting bypass: Invoke any CLSID:
  - `rundll32.exe -Sta {CLSID}` OR `ProgID`
  - Other programs which take CLSIDs as arguments: xwizard.exe, mmc.exe, verclsid.exe, cscript.exe, wscript.exe, openwith.exe
- Lateral movement: DCOM
  - Remote registry writes to hijack a CLSID (optional if abusing abandoned CLSIDs)
  - Drop DLL/EXE to remote host file system
  - Use DCOM to activate the object on the remote host, or wait for it to be invoked locally by the COM client
  - Original documented by @bohops: https://bohops.com/2018/04/28/abusing-dcom-for-yet-another-lateral-movement-technique/

nccgroup

Takeaways for offense

- COM hijacking is not a widely used persistence technique, and your target environments may not be prepared to detect this activity
- This persistence can be installed from userland
- Abuse of the COM interface provides offensive operators with layers of misdirection
- New styles of initial access vectors
- COM hijacks can be used for process injection
  - This violates application code integrity by loading untrusted code
  - Will not bypass Microsoft's implementation Code Integrity Guard
  - Other "hardened" applications might not prevent COM hijacks from loading untrusted code

nccgroup

Takeaways for defense

- COM hijacking is not a widely used persistence technique, and your SOC may not be prepared to detect this activity
- Software does not usually register COM objects per-user
  - Any processes writing keys to HKCU\Software\Classes\CLSID doesn't happen very often, and should be treated with suspicion
  - SwiftOnSecurity's Sysmon configuration tracks HKCU CLSID addition
  - Watch for the addition of ScriptletURL keys where the InprocServer32 key registered to scrobj.dll
- Watch for the addition of TreatAs keys
- If your users are widely given administrator privileges, an attacker could avoid HKCU and hijack objects via HKLM modifications
  - HKLM is owned by TrustedInstaller
- Check systems for key abandonment. Exploiting CLSIDs without a DLL on disk does not require any registry modifications

nccgroup

Takeaways for developers

If you are developing security sensitive or critical applications that rely on COM libraries, that run in userland, and you don't want somebody using COM hijacks to load a malicious library:

- Avoid relying on Windows to resolve the location of the COM server
- Identify the location of the library (registry, file system checks)
- LoadLibrary() to load DLL
- Directly call DllGetClassObject on the COM DLL to retrieve interface pointers

Other recommendations:

- Don't add CLSIDs for non-existent COM servers
- Don't leave CLSIDs for removed components

nccgroup

Future work

- Hijacking out of process COM servers
- Identifying other abusable registry keys
- Using COM hijacks for application hooking
  - Intercepting/modifying application behaviour
  - Access to sensitive data
- Using instrumentation frameworks such as Frida to have more visibility into COM client behavior
- Objects that give "bonus effects" when hijacked. Examples:
  - Matt Nelson: AMSI bypass via COM hijack https://enigma0x3.net/2017/07/19/bypassing-amsi-via-com-server-hijacking/
  - James Forshaw: Execution inside protected process for ring0 privileges: https://googleprojectzero.blogspot.com/2017/08/bypassing-virtualbox-process-hardening.html
  - Other userland processes relying on COM? AV/EDR, anybody?

nccgroup

Introducing your new acCOMplice

https://github.com/nccgroup/Accomplice

COMHijackToolkit.ps1

- Surveying COM objects on a system
- Hijacking objects
- Hijack cleanup

POC DLLs for demonstrated in this talk

- POCs for running payloads from DllMain
- DLL to measure the success of hijacks
- Process migrations

nccgroup

Credits!

Thank you to friends who answered my questions:

- Leo Loobeek (@leoloobeek)
- Michael Weber (@BouncyHat)

Thanks to NCC Group's wonderful research directors for their support:

- Jeff Dileo (@ChaosDatumz)
- Jennifer Fernick (@enjenneer)

Thank you to the researchers who've published research on COM abuse:

- James Forshaw (@tiraniddo)
- Casey Smith (@subtee)
- bohops (@bohops)
- Matt Nelson (@enigma0x3)
- Philip Tsukerman (@philiptsukerman)
- Ruben Boonen (@FuzzySec)

nccgroup

Questions?

Github (soon!): https://github.com/nccgroup/Accomplice

Twitter: @kafkaesqu3

nccgroup