

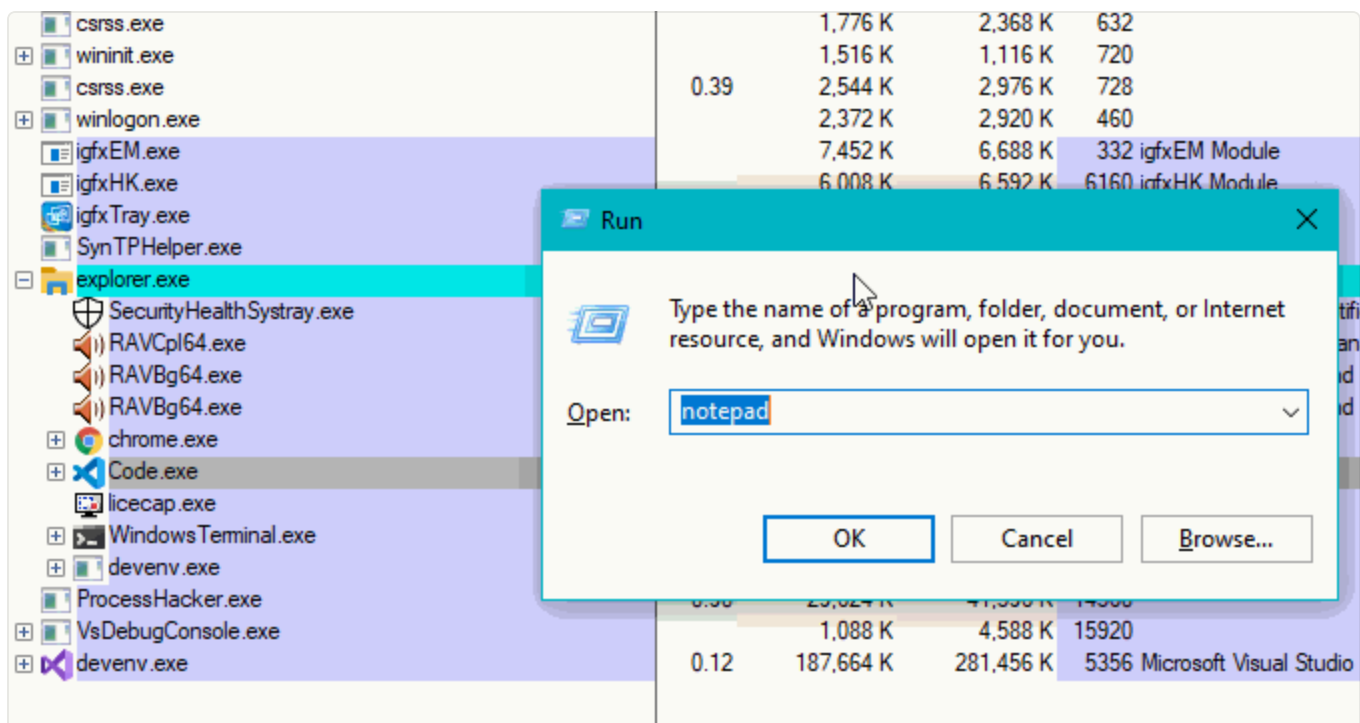


Parent Process ID (PPID) Spoofing

PPID Spoofing

PPID spoofing is a technique that allows attackers to start programs with arbitrary parent process set. This helps attackers make it look as if their programs were spawned by another process (instead of the one that would have spawned it if no spoofing was done) and it may help evade detections, that are based on parent/child process relationships.

For example, by default, most programs that an interactive user launches, will be spawned by explorer.exe:



However, with the below code, we can make it look as if the notepad.exe was spawned by igfxTray.exe (PID 6200):

```
ppid-spoofing.cpp
```

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

int main()
{
    STARTUPINFOEXA si;
    PROCESS_INFORMATION pi;
    SIZE_T attributeSize;
    ZeroMemory(&si, sizeof(STARTUPINFOEXA));

    HANDLE parentProcessHandle = OpenProcess(MAXIMUM_ALLOWED, false, 6200);

    InitializeProcThreadAttributeList(NULL, 1, 0, &attributeSize);
    si.lpAttributeList = (LPPROC_THREAD_ATTRIBUTE_LIST)HeapAlloc(GetProcessHeap(), 0, attributeSize);
    InitializeProcThreadAttributeList(si.lpAttributeList, 1, 0, &attributeSize);
    UpdateProcThreadAttribute(si.lpAttributeList, 0, PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &parentProcessHandle,
    si.StartupInfo.cb = sizeof(STARTUPINFOEXA);

    CreateProcessA(NULL, (LPSTR)"notepad", NULL, NULL, FALSE, EXTENDED_STARTUPINFO_PRESENT,
    return 0;
}
```

If we compile and run the above code, we will see the notepad pop under the spoofed parent - igfxTray.exe (PID 6200):

The screenshot displays the Windows Task Manager interface. On the left, a list of running processes is shown, including 'igfxEM.exe', 'igfxHK.exe', 'igfxTray.exe', 'SynTPHelper.exe', 'explorer.exe', 'SecurityHealthSystray.exe', 'RAVCpl64.exe', 'RAVBg64.exe', 'chrome.exe', 'Code.exe', 'loccap.exe', 'WindowsTerminal.exe', 'devenv.exe', 'notepad.exe', 'ProcessHacker.exe', 'VsDebugConsole.exe', and 'mspdbsrv.exe'. The 'igfxTray.exe' process is highlighted. On the right, a detailed view of the selected process is shown, including its PID (6200), PPID (6464), and the parent process (igfxTray Module). Below the Task Manager, a Windows PowerShell window is open, showing the command 'PS C:\labs> C:\labs\ppid-spoofing\ppid-spoofing\Debug\ppid-spoofing.exe' being executed.

PPID Spoofing Detection

For PPID spoofing detection, we can use [Event Tracing for Windows](#), and more specifically, the `Microsoft-Windows-Kernel-Process` provider.

This provider emits information about started and killed processes on the system, amongst many other things.

We can quickly check out some logs it generates by creating a trace session and subscribing to process related events (0x10 keyword):

```
logman create trace ppid-spoofing -p Microsoft-Windows-Kernel-Process 0x10 -ets
logman start ppid-spoofing
```

Let's confirm the trace session is running:

```
logman query ppid-spoofing -ets
```

```
PS C:\> logman query ppid-spoofing -ets

Name:                ppid-spoofing
Status:              Running
Root Path:           C:
Segment:             Off
Schedules:           On

Name:                ppid-spoofing\ppid-spoofing
Type:                Trace
Output Location:     C:\ppid-spoofing.etl
Append:              Off
Circular:            Off
Overwrite:           Off
Buffer Size:         8
Buffers Lost:        0
Buffers Written:     45
Buffer Flush Timer:  0
Clock Type:          Performance
File Mode:           File

Provider:
Name:                Microsoft-Windows-Kernel-Process
Provider Guid:       {22FB2CD6-0E7B-422B-A0C7-2FAD1FD0E716}
Level:               255
KeywordsAll:         0x0
KeywordsAny:         0x10 (WINEVENT_KEYWORD_PROCESS)
Properties:          64
Filter Type:         0
```

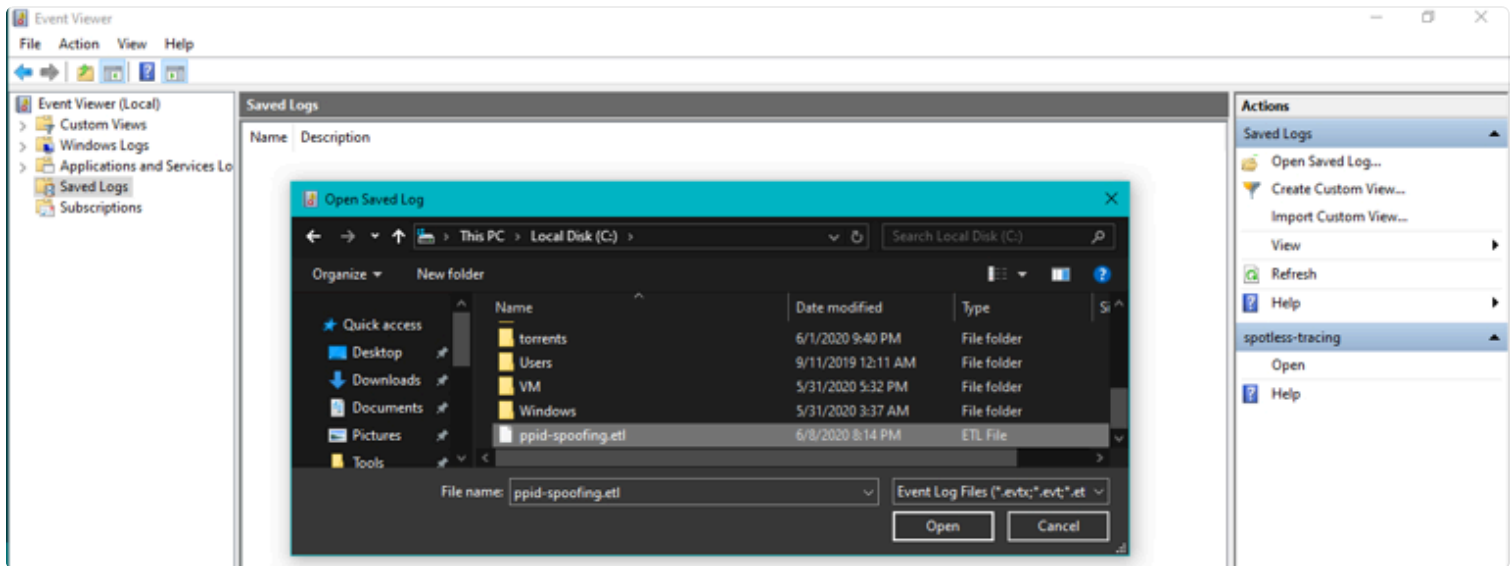
Now, let's execute our notepad.exe with a spoofed parent again and let's look at where the log files from our ETW tracing session are saved to:

```
PS C:\> logman query ppid-spoofing -ets

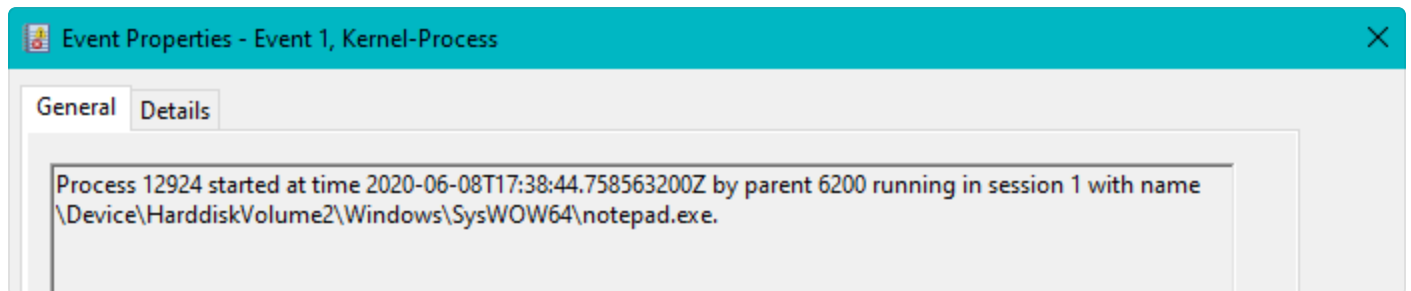
Name:                ppid-spoofing
Status:              Running
Root Path:           C:
Segment:             Off
Schedules:           On

Name:                ppid-spoofing\ppid-spoofing
Type:                Trace
Output Location:     C:\ppid-spoofing.etl
Append:              Off
Circular:            Off
Overwrite:           Off
Buffer Size:         8
```

Open the C:\ppid-spoofing.etl in Windows Event Viewer:



We can find an event with ID 1, saying that notepad was started by a process with PID 6200 (that's our spoofed PPID of the process igfxTray.exe):



If we look at the same data in an XML view (the details tab) and cross check it with our processes tree in Process Explorer, we see:

- in blue - the notepad we started with a spoofed process PID
- in red - notepad's spoofed parent process and its PID
- in black - our malicious program that started notepad with a spoofed PPID!

The screenshot displays two windows: 'Event Properties - Event 1, Kernel-Process' and 'Process Explorer - Sysinternals: www.sysinternals.com [OFFENSE\spotless]'.

Event Properties (XML View):

```
<Version>3</Version>
<Level>4</Level>
<Task>1</Task>
<Opcode>1</Opcode>
<Keywords>0x8000000000000010</Keywords>
<TimeCreated SystemTime='2020-06-08T17:38:44.757489000Z' />
<EventRecordID>1924</EventRecordID>
<Correlation />
<Execution ProcessID='11076' ThreadID='1828' ProcessorID='0'
  KernelTime='0' UserTime='1' />
<Channel />
<Computer>LT01.offense.local</Computer>
<Security />
<System />
<EventData>
  <Data Name='ProcessID'>12924</Data>
  <Data Name='ProcessSequenceNumber'>15149</Data>
  <Data Name='CreateTime'>2020-06-08T17:38:44.758563200Z</Data>
  <Data Name='ParentProcessID'>6200</Data>
  <Data Name='ParentProcessSequenceNumber'>128</Data>
  <Data Name='SessionID'>1</Data>
  <Data Name='Flags'>0</Data>
  <Data Name='ProcessTokenElevationType'>3</Data>
  <Data Name='ProcessTokenIsElevated'>0</Data>
  <Data Name='MandatoryLabel'>S-1-16-8192</Data>
  <Data Name='ImageName'>\Device\HarddiskVolume2\Windows\SysWOW64\notepad.exe</Data>
  <Data Name='ImageChecksum'>206882</Data>
  <Data Name='TimeStamp'>2478106874</Data>
  <Data Name='PackageFullName' />
  <Data Name='PackageRelativeAppId' />
</EventData>
</Event>
```

Process Explorer:

Process	CPU	Private Bytes	Working Set	PID
Registry		14,000 K	44,092 K	96
System Idle Process	71.32	60 K	8 K	0
System	0.57	200 K	2,612 K	4
csrss.exe		1,776 K	2,356 K	632
wininit.exe		1,516 K	1,116 K	720
csrss.exe	0.71	2,540 K	2,884 K	728
winlogon.exe		2,372 K	3,164 K	460
ghxEM.exe		7,452 K	6,348 K	332
ghxHK.exe		6,008 K	6,360 K	6160
ghxTr.exe		8,244 K	5,500 K	6200
notepad.exe		3,104 K	12,752 K	12924
System Helper.exe		1,060 K	1,108 K	6494
explorer.exe	3.87	92,824 K	126,716 K	6700
SecurityHealthSystray.exe		1,736 K	2,948 K	8760
RAVCpl64.exe		4,540 K	7,716 K	8872
RAVBg64.exe		5,844 K	4,940 K	8972
RAVBg64.exe		5,816 K	5,304 K	9008
chrome.exe	0.45	351,612 K	306,596 K	8632
Code.exe		34,748 K	68,132 K	8552
lsccap.exe	0.08	11,996 K	32,700 K	5480
devenv.exe	0.14	242,488 K	328,180 K	14848
Windows Terminal.exe		29,268 K	65,580 K	15788
OpenConsole.exe		2,224 K	8,996 K	14268
powershell.exe	0.02	58,364 K	72,776 K	10132
mspaint.exe	1.48	8,996 K	27,492 K	3384
mspaint.exe		8,972 K	29,148 K	7728
Windows Terminal.exe		28,816 K	63,928 K	12196
OpenConsole.exe		2,252 K	9,016 K	10920
powershell.exe		63,308 K	73,896 K	8828
ppid-spoofing.exe		756 K	3,632 K	11076
Process Hacker.exe	1.06	28,780 K	36,352 K	14360
VsDebugConsole.exe		1,088 K	4,224 K	15920
proccp64.exe	3.31	20,856 K	31,296 K	3544
mmc.exe	1.40	97,472 K	57,936 K	8084

From the above, we can conclude that when `ParentProcessId` (red, PID 6200) != `Execution Process ID` (black, PID 11076), we may be looking at a PPID spoofing.

Now that confirmed we have the required telemetry for detection, we can write a simple C# consumer to do real time PPID spoofing detection:

```
ppid-spoofing-detection.cs
```

```
# based on https://github.com/zodiacon/DotNextSP2019/blob/master/SimpleConsumer/Program.cs
using Microsoft.Diagnostics.Tracing.Parsers;
using Microsoft.Diagnostics.Tracing.Session;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;

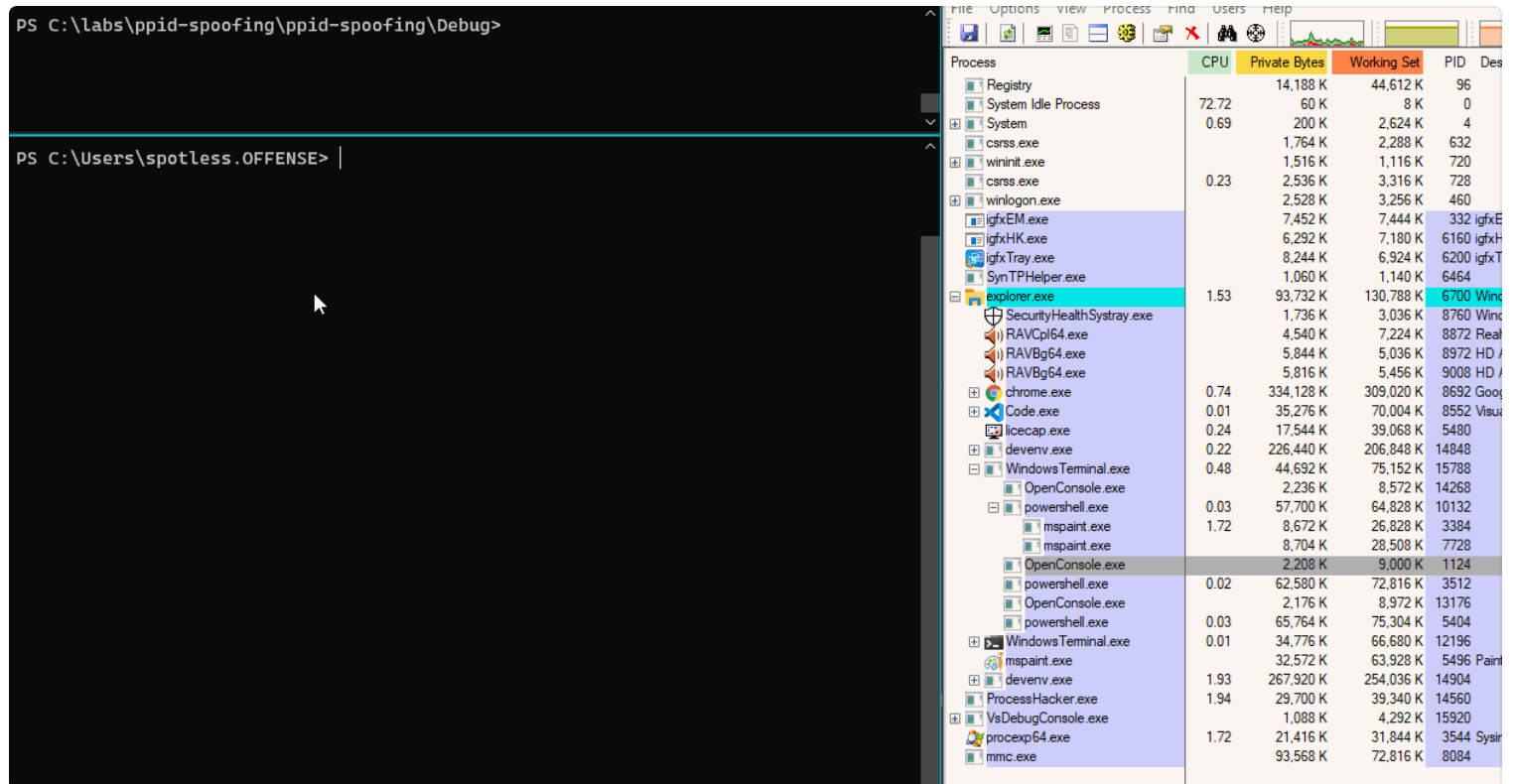
namespace PPIDSpoofingDetection
{
    static class Program
    {
        static void Main(string[] args)
        {
            using (var session = new TraceEventSession("spotless-ppid-spoofing"))
            {
                Console.CancelKeyPress += delegate {
                    session.Source.StopProcessing();
                    session.Dispose();
                };

                session.EnableProvider("Microsoft-Windows-Kernel-Process", Microsoft.Diagnostics.Tracing.Session.EnableLevel.All);
                var parser = session.Source.Dynamic;
                parser.All += e => {
                    if (e.OpcodeName == "Start" && Regex.IsMatch(e.FormattedMessage.ToLower(), "ppid"))
                    {
                        string[] messageBits = e.FormattedMessage.Replace(",", string.Empty).Split(' ');
                        int PID = int.Parse(messageBits[1]);
                        int PPID = int.Parse(messageBits[10]);
                        int realPPID = e.ProcessID;

                        // if ParentProcessId (red, PID 6200) != Execution Process ID (black, PID 6200)
                        if (PPID != realPPID)
                        {
                            // this may fail if the process is already gone.
                            string processName = Process.GetProcessById(PID).ProcessName;
                            Console.WriteLine($"{e.Timestamp} PPID Spoofing detected: {processName}");
                        }
                    }
                };
            }
            session.Source.Process();
        }
    }
}
```

```
}  
}  
}  
}
```

If we compile and run the code, and then attempt to launch notepad with a spoofed PPID again, it will get flagged:



PS C:\labs\ppid-spoofing\ppid-spoofing\Debug>

PS C:\Users\spotless.OFFENSE> |

References



Quickpost: SelectMyParent or Playing With the Windows Process Tree

Didier Stevens



Access Token Manipulation: Parent PID Spoofing, Sub-technique T1134.004 - Enterprise | MITRE ATT&CK®



Detecting Parent PID Spoofing - F-Secure Blog

F-Secure Blog



Previous

< ProcessDynamicCodePolicy: Arbitrary Code Guard (ACG)

Next

Executing C# Assemblies from Jscript and wscript with DotNetToJscript >

Last updated 3 years ago