

# FalconFriday — Direct system calls and Cobalt Strike BOFs — 0xFF14



Gijs Hollestelle · [Follow](#)

Published in FalconForce · 6 min read · Jul 23, 2021



7



*Direct system calls are a popular technique used by attackers to bypass certain EDR solutions. In this blog we deep-dive into how direct system calls could be detected based on some example Cobalt Strike BOFs that make direct system calls.*

**TL;DR for blue teams:** Attackers might use direct system calls in an attempt to bypass detection. This blog post shows a method for detecting direct system calls for opening a process using Sysmon.

**TL;DR for red teams:** Sometimes techniques used to hide malicious activities, such as making direct system calls instead of going via documented APIs, can actually make an attack less stealthy if the blue team is monitoring for these specific behaviours. Be especially cautious when using the *NtOpenProcess* direct system call as this behaviour can be detected using Sysmon with the right rules in place.



# Medium

Sign up to discover human stories that deepen your understanding of the world.

## Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

Sign up for free

## ★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

Try for 5 \$/month

BOFs arbitrary code can be executed inside the beacon process running on the target.

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

While researching some of the many BOF files available online and how they might be detected we noticed that a substantial number of them use direct system calls. One of the popular methods of making direct system calls from Beacons is the [InlineWhispers](#) tool released by Outflank. This tool allows using direct system calls from Cobalt Strike BOFs based on wrappers provided by the [SysWhispers](#) project.

The reason for using direct system calls is that this can bypass a number of EDR solutions, mostly solutions that rely on user-mode hooking to intercept calls being made. In our experience, sometimes the evasion techniques that attackers use to avoid detection can actually be used to identify malicious behaviour by searching for these techniques specifically. The question we want to address in this blog post is whether direct system calls can be detected to identify potential usage of these BOFs that rely on direct system calls.

. . .

### Differences between normal and direct system calls

A way to distinguish between a normal system call via the regular Windows APIs and a direct system call is to look at the call stack being generated. While our EDR of choice (Microsoft Defender for Endpoint) does not provide call stack traces, the excellent and free [Sysmon tool](#) does provide these for Event ID 10 (Process Access).

## Medium

Sign up to discover human stories that deepen your understanding of the world.

#### Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

#### ★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

```
);  
  
To make Medium work, we log user data. By using Medium, you agree to our Privacy Policy,  
including cookie policy.  
  
HANDLE h;  
BeaconPrintf(CALLBACK_OUTPUT, "Opening process: %d\\n",  
target_pid);  
h = kernel32$OpenProcess(MAXIMUM_ALLOWED, FALSE, target_pid);  
BeaconPrintf(CALLBACK_OUTPUT, "Handle: %x\\n", h);  
BeaconPrintf(CALLBACK_OUTPUT, "Sleeping\\n", h);  
kernel32$Sleep(5000);  
BeaconPrintf(CALLBACK_OUTPUT, "Done\\n", h);  
return;  
}
```

The second version of the BOF uses the direct system call method by directly calling NtOpenProcess via the ‘syscall’ instruction.

```
#include <windows.h>  
#include <stdio.h>  
#include <winternl.h>  
#include "beacon.h"  
#include "syscalls.h"  
  
WINBASEAPI void kernel32$Sleep(  
    DWORD dwMilliseconds  
);  
  
void go(char* args, int length) {  
    int target_pid = 6244; // hardcoded pid of an explorer.exe  
    running under same user account for ease of development  
    HANDLE h;  
  
    NTSTATUS status;  
    OBJECT_ATTRIBUTES ObjectAttributes;  
  
    InitializeObjectAttributes(&ObjectAttributes, NULL, 0, NULL,  
    NULL);  
    CLIENT_ID uPid = { 0 };  
  
    uPid.UniqueProcess = (HANDLE)(DWORD_PTR)target_pid;  
    uPid.UniqueThread = (HANDLE)0;
```

# Medium

Sign up to discover human stories that deepen your understanding of the world.

## Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

## ★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

Trace 2 — BOF using Direct NtOpenProcess System Call:

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

```
TargetProcessId: 6244
TargetImage: C:\Windows\Explorer.EXE
GrantedAccess: 0x1FFFFFF
CallTrace:
  UNKNOWN(0000022ECF78048F)
```

For reference we also included a call trace for a regular windows executable that calls *Openprocess*.

Trace 3 — Regular Windows executable calling OpenProcess API:

```
TargetProcessId: 6244
TargetImage: C:\Windows\Explorer.EXE
GrantedAccess: 0x1FFFFFF
CallTrace:
  C:\Windows\SYSTEM32\ntdll.dll+9d2e4
|C:\Windows\System32\KERNELBASE.dll+2c03e
|c:\temp\Test.exe+1e0d
|c:\temp\Test.exe+2480
|C:\Windows\System32\KERNEL32.DLL+17034
|C:\Windows\SYSTEM32\ntdll.dll+52651
```

In these call traces the ordering is top-down based on the last function called, meaning the most recently called function, which actually invoked the *NtOpenProcess* system call is at the top.

. . .

# Medium

Sign up to discover human stories that deepen your understanding of the world.

Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

- Executable -> KERNEL BASE/KERNEL 32 -> NTDLL /WIN32U/WOW64WIN -
- To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

Based on this behaviour we can build two detection rules. These rules assume Sysmon data being ingested into Azure Sentinel.

Rule 1 — *Identify direct system calls by looking if the first entry in the call stack is not NTDLL, WIN32U or WOW64WIN*

```
let ValidDlls=dynamic([
  "ntdll.dll",
  "win32u.dll",
  "wow64win.dll"
]);
Sysmon
| where EventID == 10
| extend Callers=split(CallTrace, "|")
| extend FirstCaller=toString(split(Callers[0], "+")[0])
| where not(FirstCaller has_any (ValidDlls))
```

A simple Sysmon configuration for Process Access can be used to capture the relevant events:

```
<ProcessAccess onmatch="include">
  <Rule groupRelation="and">
    <CallTrace condition="not begin
with">C:\Windows\SYSTEM32\ntdll.dll</CallTrace>
    <CallTrace condition="not begin
with">C:\Windows\SYSTEM32\win32u.dll</CallTrace>
    <CallTrace condition="not begin
with">C:\Windows\SYSTEM32\wow64win.dll</CallTrace>
  </Rule>
```

# Medium

Sign up to discover human stories that deepen your understanding of the world.

## Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

## ★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app



```
| extend Callers=split(CallTrace, "|")
| where not(secondcaller has_any (ValidCallers))
| where not(ThirdCaller has_any (ValidCallers))
```

Unfortunately, writing a Sysmon filter to capture all relevant events for this second detection rule is challenging since Sysmon only allows for simple string based filtering. This makes it hard to write a Sysmon filter that only captures the Process Access events related to directly calling the undocumented APIs in NTDLL. The rule can still be useful if you already have an existing Sysmon config that logs suspicious Process Access calls based on other heuristics, in which case the fact that an undocumented API call was used could be an additional indicator to identify malicious events.

Hopefully we have shown that direct system calls can be detected, especially when calling the *NtOpenProcess* system call which directly generates a Sysmon event which includes a call trace.

. . .

**Bonus Content — Getting Direct System Calls to Work on Windows 21H1 and above**

While testing a number of BOFs that use direct system calls we noticed that they ran fine on Windows up to 20H2 but failed on 21H1. It turns out that the direct system call stubs generated by InlineWhispers / SysWhispers as used by most BOFs publicly available, rely on a static table that maps Windows versions to syscall numbers that changes with each new Windows version.

Medium

Sign up to discover human stories that deepen your understanding of the world.

Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

Written by **Gijs Hollestelle**

129 Followers · Editor for FalconForce

Follow



More from Gijs Hollestelle and FalconForce

Gijs Hollestelle in FalconForce

## FalconFriday — Detecting Active Directory Data Collection — 0xFF21

Active Directory data collection

Nov 11, 2022



55



Olaf Hartong in FalconForce

## Sysmon vs Microsoft Defender for Endpoint, MDE Internals 0×01

It is not a big secret that we at FalconForce work a lot with, and are big fans of, both...

Oct 15, 2021



278



# Medium

Sign up to discover human stories that deepen your understanding of the world.

### Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

### ★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app

To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.

Recommended from Medium

Kostas

Telemetry on Linux vs. Windows: A Comparative Analysis

A look at how Windows and Linux manage telemetry to support incident response...

Sep 3 177



RED TEAM

Malware Development Part 8 : Reverse Shell Via Dll Hijacking

“From DLL to Shell: A Step-by-Step Guide to Reverse Shell via DLL Hijacking”

Jun 22 147



Lists



Staff Picks

755 stories · 1416 saves

Stories to Help You Level-Up at Work

19 stories · 852 saves

Self-Improvement 101

20 stories · 2961 saves

Productivity 101

20 stories · 2506 saves

Medium

Sign up to discover human stories that deepen your understanding of the world.

Free


- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app



To make Medium work, we log user data. By using Medium, you agree to our [Privacy Policy](#), including cookie policy.


 Alex Teixeira in Detect FYI

## Sysmon: a viable alternative to EDR?

I've been recently engaged in workshops with distinct clients from completely different...

★ Jul 4  144  9



 Sagar Pandita

## Understanding AMSI Bypass Techniques on Windows: An...

The Windows Antimalware Scan Interface (AMSI) is a pivotal component in Microsoft's...

Sep 30



See more recommendations

[Help](#) [Status](#) [About](#) [Careers](#) [Press](#) [Blog](#) [Privacy](#) [Terms](#) [Text to speech](#) [Teams](#)

# Medium

Sign up to discover human stories that deepen your understanding of the world.

### Free

- ✓ Distraction-free reading. No ads.
- ✓ Organize your knowledge with lists and highlights.
- ✓ Tell your story. Find your audience.

### ★ Membership

- ✓ Read member-only stories
- ✓ Support writers you read most
- ✓ Earn money for your writing
- ✓ Listen to audio narrations
- ✓ Read offline with the Medium app