Open in app ↗ Sign up Sign in

Medium Search Write

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

Gijs Hollestelle · Follow

Medium

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

Free

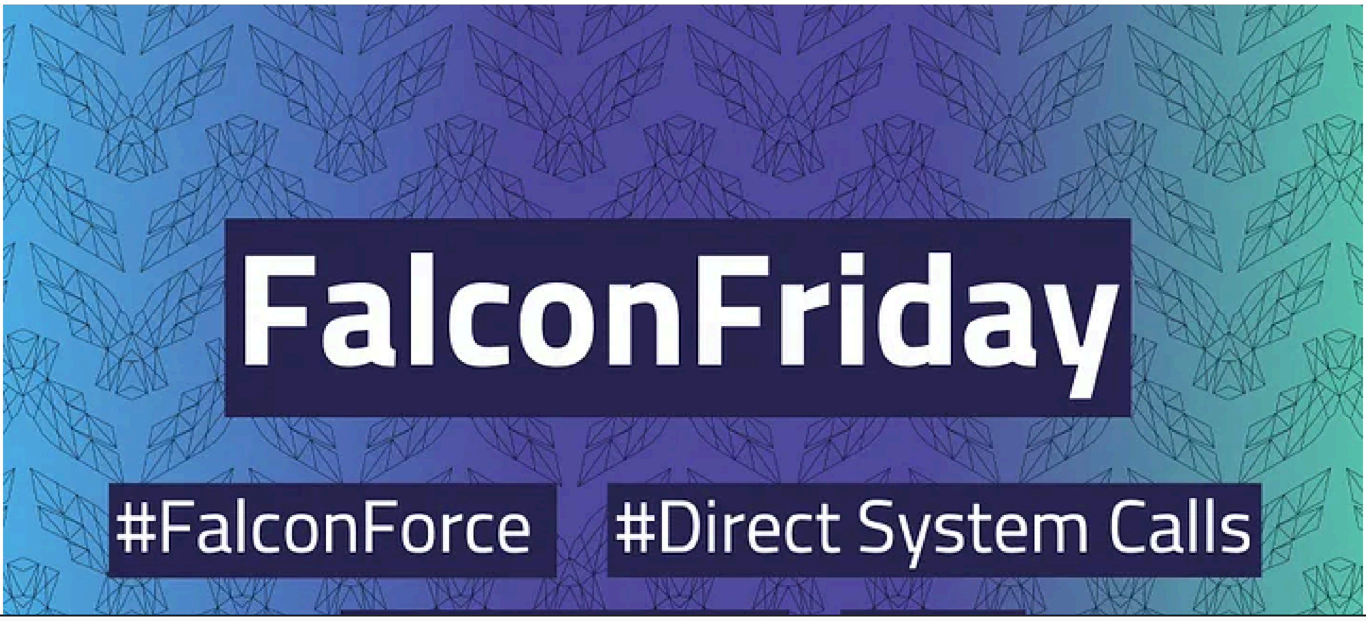
- Distraction-free reading. No ads.
- Organize your knowledge with lists and highlights.
- Tell your story. Find your audience.

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



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

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.

. . .

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

```
#include <windows.h>
#include <stdio.h>
#include "beacon.h"

WINBASEAPI HANDLE kernel32$OpenProcess(
  DWORD dwDesiredAccess,
  BOOL  bInheritHandle,
  DWORD dwProcessId
);

WINBASEAPI void kernel32$Sleep(
  DWORD dwMilliseconds
);

void go(char* args, int length) {
    int target_pid = 6244; // hardcoded pid of an explorer.exe
```

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

```
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;

BeaconPrintf(CALLBACK_OUTPUT, "Opening process: %d\n",
target_pid);
```

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

```
TargetProcessId: 6244
TargetImage: C:\Windows\Explorer.EXE
GrantedAccess: 0x1FFFFF
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:

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

These three traces provide some insight into how direct system call behaviour by BOFs can be detected:

- Cobalt Strike BOFs run from a memory page which is not mapped to an executable or DLL, showing as UNKNOWN in the Sysmon call traces above. This is even true in case the Cobalt Strike option module_x64 / module_x86 is used which makes the Beacon payload itself appear to originate from a memory mapped DLL.
- Using the regular *OpenProcess* API shows a call stack that goes from

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

```
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

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

```
let ValidCallers=dynamic([
  "kernelbase.dll",
  "wow64.dll",
  "kernel32.dll",
  "lsasrv.dll",
  "themeservice.dll",
  "wow64win.dll"
]);
Sysmon
| where EventID == 10
| extend Callers=split(CallTrace, "|")
| extend FirstCaller=tostring(split(Callers[0], "+")[0])
| extend SecondCaller=tostring(split(Callers[1], "+")[0])
| extend ThirdCaller=tostring(split(Callers[2], "+")[0])
| where not(SecondCaller has_any (ValidCallers))
```

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

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.

We were able to get these BOFs to run by regenerating the syscall headers

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

Written by Gijs Hollestelle

127 Followers · Editor for FalconForce

Follow

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