Detecting Cobalt Strike Default Modules via Named Pipe Analysis

labs.f-secure.com/blog/detecting-cobalt-strike-default-modules-via-named-pipe-analysis

Introduction

During recent years, the Cobalt Strike framework has gained significant popularity amongst red teamers and threat actors alike. Its functionality, flexibility and stability make it the state of the art when it comes to commercially available Command & Control frameworks.

Considerable efforts have been made to build robust signatures for Cobalt Strike and its implant, Beacon. The aim of this post is to examine some previously unknown Indicators of Compromise (IoCs). This post is not going to cover signatures for the default Cobalt Strike configuration - other papers offer an in-depth look at this. Instead, we will focus our attention on some of the built-in modules that provide Cobalt Strike's post exploitation capability, such as the keylogger, Mimikatz and the screenshot modules.

It must be noted that the IoC/behaviour was raised with the Cobalt Strike's author and subsequently exposed to operators as a customisable setting in the 4.2 malleable profile.

The hope is that this post will help both defenders in strengthening their detection capabilities, and force red teamers to use more sophisticated and customised techniques.

Analysis

Cobalt Strike is known to use a specific pattern, known as "Fork-n-Run", when executing some of its commands. The "Fork-n-Run" pattern comprises the spawning of a new process (also referred to as a sacrificial process) and injecting capabilities into it. This pattern offers a number of benefits, one being the ability to execute long running tasks, the "keylogger" being a prime example, without blocking the main Beacon thread. Often, these capabilities are implemented as Reflective DLLs.

Recent versions of the framework have given operators great flexibility in how to customise the capability injection process. However, some general aspects haven't changed much, and that's what we are going to focus on.

More specifically, a characteristic that remained unchanged was the ability to retrieve the output of an injected module. The "keylogger" module, for example, is able to send the pressed keys back to the main beacon process. But since the "keylogger" module is fully fileless, how does the communication with the main beacon process happen?

The answer: pipes!

Pipes are shared memory used for processes to communicate between each other. Fundamentally there are two types of pipe: named and unnamed.

- Named pipes, as the name implies, have a name and can be accessed by referencing this.
- Unnamed pipes, that need their handle to be passed to the other communicating process in order to exchange data. This can be done in a number of ways.

Cobalt Strike uses both named and unnamed pipes to exchange data between the beacon and its sacrificial processes.

Named Pipes

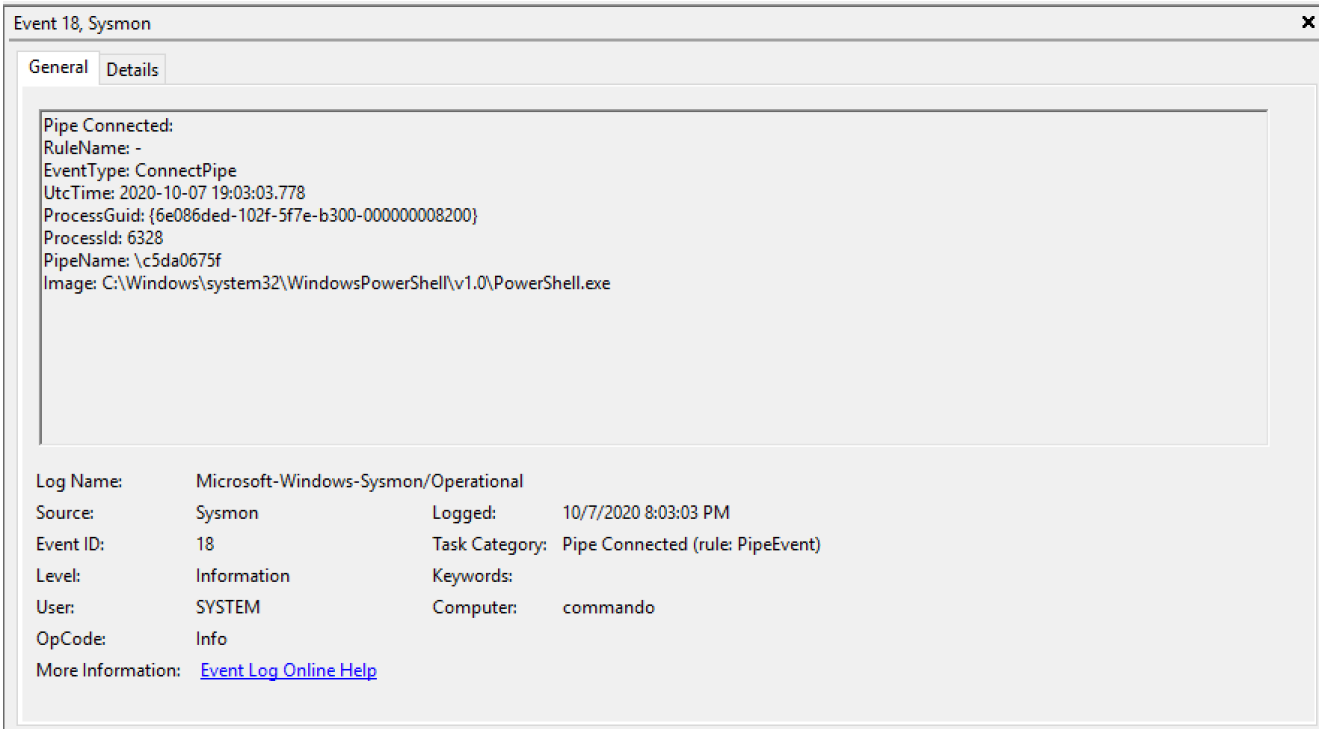
F-Secure observed that when using some of the Cobalt Strike's modules that injected a reflective DLL into a sacrificial process, a named pipe was created with a predictable pattern.

Note that these named pipes are not the SMB named pipes used for lateral movement that can be customised via the malleable profiles. Prior to version 4.2, this named pipe's name could not be modified by operators.

More specifically, it was observed that once a "job" was launched, the beacon created a named pipe; the name of the pipe comprised only hexadecimal characters, and its length was found to be equal to the length of the module name (e.g. 10 characters for the screenshot module) . Some of the modules that were found to have this behaviour:

- Keylogger
- Screenshot
- Mimikatz (dcsync, dpapi, logonpasswords)
- Powerpick
- Net (netview)

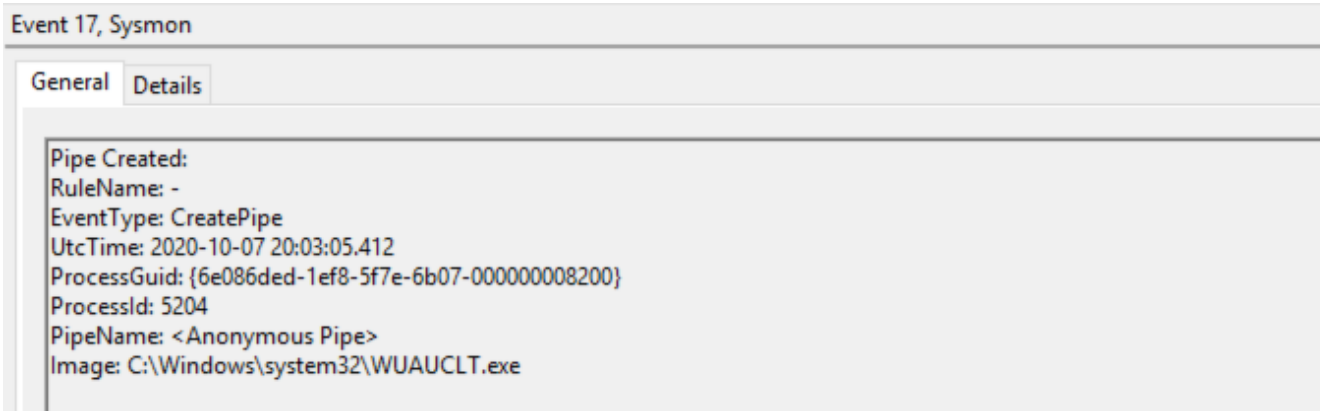
The screenshot below shows an example of Sysmon event ID 17 and 18 (pipe created and accessed, respectively) after the "keylogger" command was executed:



A limited number of experiments were made, but no other legitimate application was found to create named pipes with the same naming convention. We will use this information later to create Splunk searches that use Sysmon and a Yara rule to scan the processes memory.

Anonymous Pipes

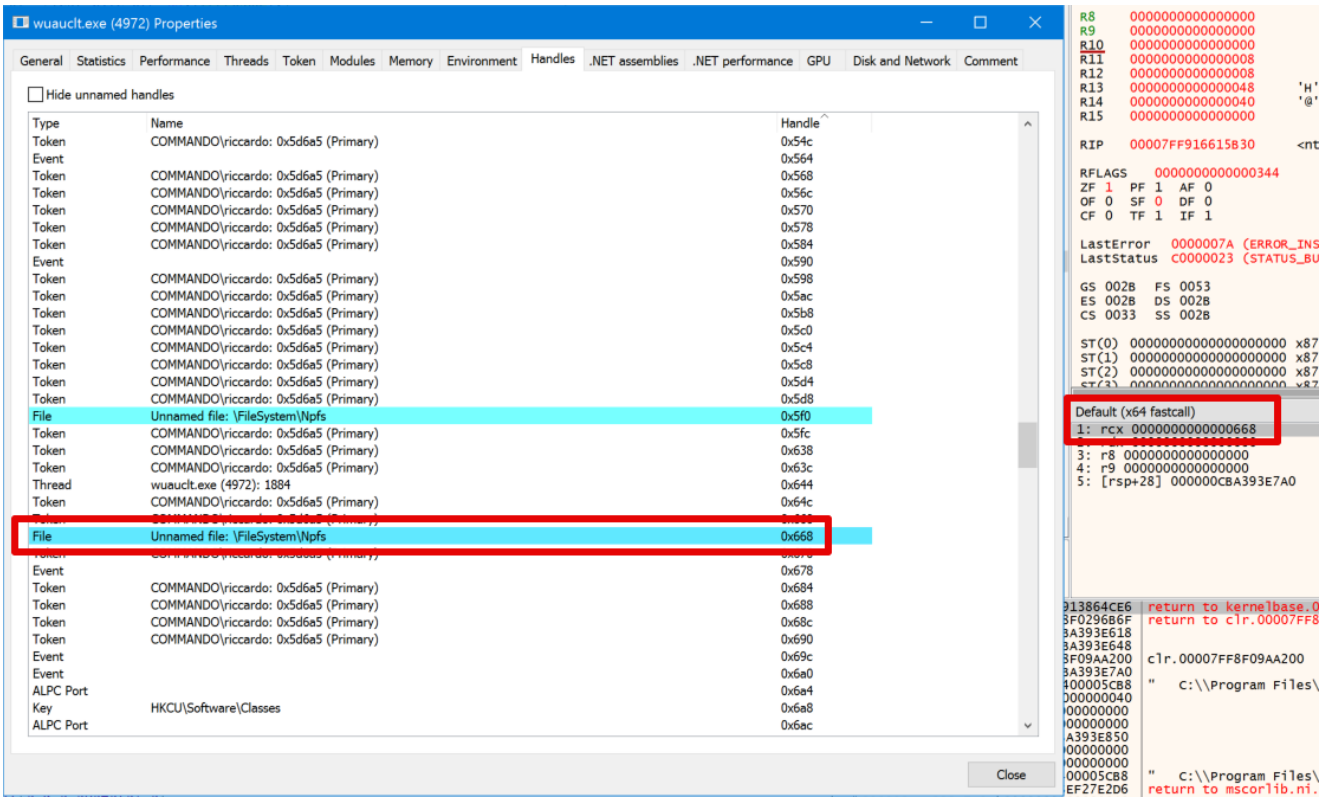
Not every Cobalt Strike command creates a named pipe, some of them will use anonymous (unnamed) pipes to achieve the same result. The image below shows an instance of a pipe created after issuing the "execute-assembly" command:



We can confirm that by debugging the sacrificial process spawned after launching a long-running assembly:

```
beacon> execute-assembly /mnt/hgfs/Downloads/Tools/Rubeus.exe monitor /interval:5
[*] Tasked beacon to run .NET program: Rubeus.exe monitor /interval:5
[+] host called home, sent: 318033 bytes
[+] received output:


  ______        _
  (_____ \      | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v1.5.0

[*] Action: TGT Monitoring
[*] Monitoring every 5 seconds for new TGTs
```

A breakpoint was set on the "ntdll!NtWriteFile" function, and as it is possible to see, the handle where the sacrificial process was trying to write to was associated with an unnamed file that belonged to the pipe filesystem (Npfs):



As we can see, spotting commands such as "execute-assembly" is not as trivial as the examples above. Is there anything that we can do using pipes?

In theory, we could baseline processes that use anonymous pipes. The interesting result is that native Windows processes do not use anonymous pipes that often. So we could look for Windows processes that connect to an anonymous pipe and investigate from there.

We mention "Windows processes" because, more often than not, attackers use native Windows binaries as sacrificial processes within their malleable profiles. Examples of such are the binaries listed in the C2Concealer repository, a project used to create randomised malleable profiles. We can see the executables from the C2Concealer default configuration below:

```
''
#################################################
Data set containing post_ex block data, including
spawn-to processes.
#################################################
'''
#CUSTOMIZE THIS LIST#
spawn_processes = ['runonce.exe','svchost.exe','regsvr32.exe','WUAUCLT.exe']
```

As it is possible to see, the above-mentioned processes are used for post exploitation jobs. None of them usually use anonymous pipes to communicate with different processes; it would therefore be possible to use this to perform hunting and eventually create detection rules.

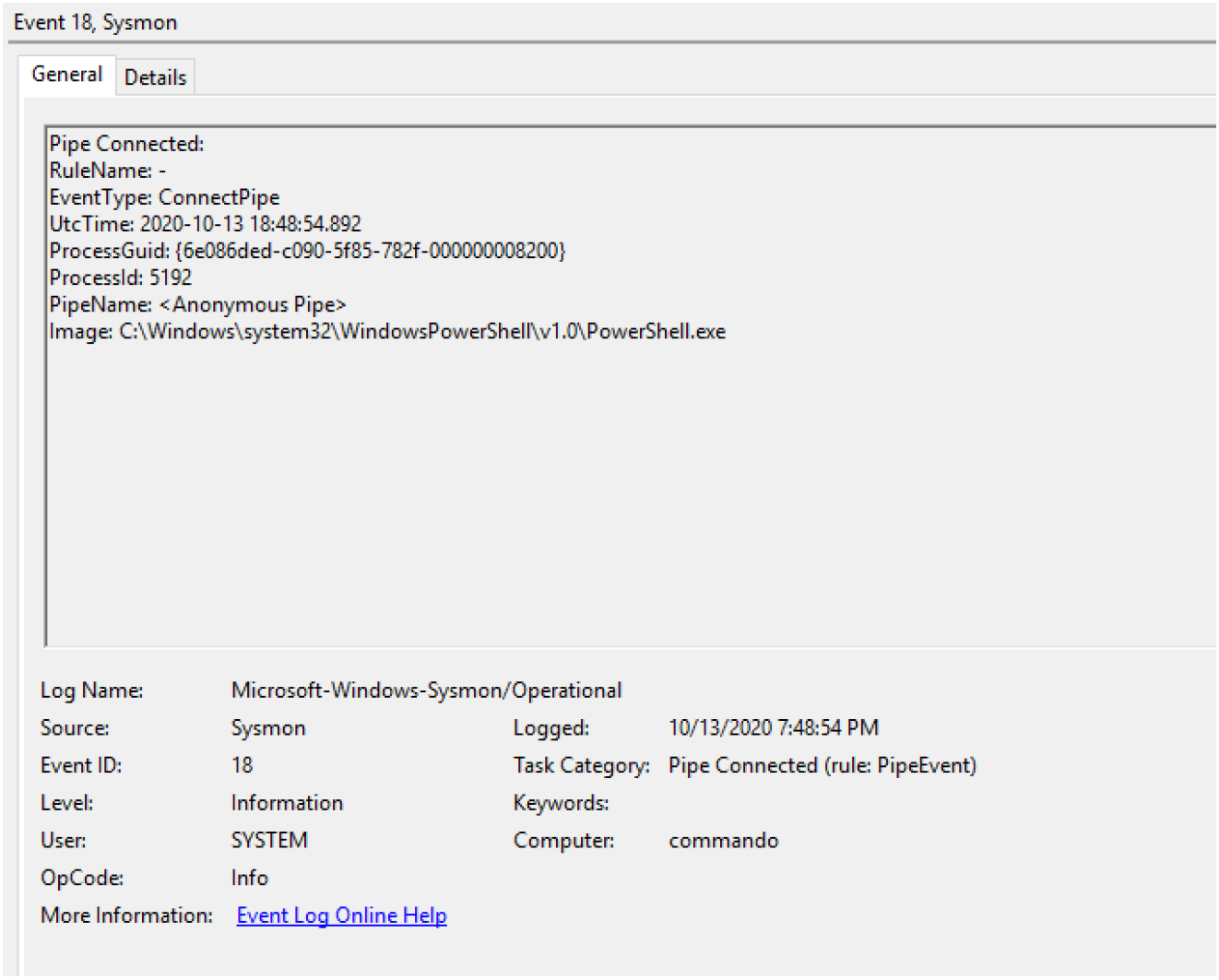
During experiments, the following Windows binaries were found to be using anonymous pipes for interprocess communication:

- wsmprovhost.exe
- ngen.exe
- splunk.exe
- splunkd.exe
- firefox.exe

The same applies to custom reflective DLLs that are executed via Cobalt Strike's dllspawn API, as the underlying mechanism for communication is the same. An example of such is the Outflank's Ps-Tools repository. Ps-Tools is a collection of rDLL fully compatible with Cobalt Strike that allow operators to monitor process activity. Let's execute the "psw" module, used to enumerate the active Windows, as shown below:

```
beacon> psw
[+] Enumerating processes with Active Windows.
[*] Tasked beacon to spawn PsW
[+] host called home, sent: 115228 bytes
[+] received output:

[+] ProcessName:   mmc.exe
    ProcessID:     436
    WindowTitle:   Event Viewer

[+] ProcessName:   explorer.exe
    ProcessID:     9000
    WindowTitle:   msbuild-templates

[+] ProcessName:   ApplicationFrameHost.exe
    ProcessID:     7084
    WindowTitle:   Settings

[+] ProcessName:   powershell.exe
    ProcessID:     10444
    WindowTitle:   Administrator: PowerShell

[+] ProcessName:   devenv.exe
    ProcessID:     10176
    WindowTitle:   Mimir - Microsoft Visual Studio
```

Executing this module, we can identify the same anonymous pipe behaviour we've seen before:



Detection Rules

Detection of the anomalous named pipes can be achieved in a number of ways. As a proof-of-concept, we developed a Yara signature that could be used to scan process memory and find live instances, and a Splunk search that could be used in conjunction with Sysmon.

The Yara rule is shown below:

```
rule cs_job_pipe
{
    meta:
        description = "Detects CobaltStrike Post Exploitation Named Pipes"
        author = "Riccardo Ancarani & Jon Cave"
        date = "2020-10-04"
    strings:
        $pipe = /\\\\\.\\pipe\\[0-9a-f]{7,10}/ ascii wide fullword
        $guidPipe = /\\\\\.\\pipe\\[0-9a-f]{8}\-/ ascii wide
    condition:
        $pipe and not ($guidPipe)
}
```

An example of execution against a sacrificial process:

```
.\yara64.exe .\cs-job-pipe.yar -s 9908
cs_job_pipe 9908
0x13372b7b698:$pipe: \\.\pipe\928316d80
0x13372bf3940:$pipe:
\\x00\\x00.\x00\\x00p\x00i\x00p\x00e\x00\\x009\x002\x008\x003\x001\x006\x00d\x008\x000
```

The Splunk search below can be used to alert on the creation of named pipes that match the aforementioned pattern:

```
index="YOUR_INDEX" source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
EventCode=17 PipeName!="&lt;Anonymous Pipe&gt;"   | regex PipeName="^\\\\[a-f0-9]
{7,10}$"
```

In regards to using anonymous pipes for automatic detection, this approach can be more prone to false positives. However, it can be used in conjunction with other IOCs to achieve better results.

An example of a Splunk search that can be used to obtain the processes that created an anonymous pipe, sorted by least frequency:

```
index="YOUR_INDEX" source="XmlWinEventLog:Microsoft-Windows-Sysmon/Operational"
EventCode=17 PipeName="&lt;Anonymous Pipe&gt;"| rare limit=20 Image
```

Opsec Considerations

From a red teaming perspective, Cobalt Strike version 4.2 gives operators the ability to modify the aforementioned named pipe naming convention. In fact, it would be possible to configure the "pipename" parameter within the "post-ex" block with a name that would, ideally, blend-in with the pipes used in the environement.

An example of a "post-ex" block is shown below:

```
post-ex {



    set spawnto_x86 "%windir%\\syswow64\\dllhost.exe";
    set spawnto_x64 "%windir%\\sysnative\\dllhost.exe";

    set obfuscate "true";
    set smartinject "true";
    set amsi_disable "true";

    set pipename "pipe\\CtxSharefilepipe###,";

    }
```

Additionally, choosing binaries that legitimately use anonymous pipes in the "spawnto_x86" and "spawnto_x64" parameters will decrease the chances of being detected.

The official malleable command reference and ThreatExpress' jQuery example profile are great resources for learning more about Cobal Strike's malleable profile options.

Closing Thoughts

This post showed two different strategies for identifying Cobalt Strike usage within an endpoint: we started by analysing anomalous named pipes associated with default modules and then we shifted our focus on a more statistical approach to identify even more sophisticated attacks.

For attackers, we have reinforced the importance of staying away from default settings and modules. While for defenders, we hope we gave some practical advice on how to spot this specific tool and more generally, monitor pipe anomalies using tools such as Sysmon.