Malware Analysis and Deobfuscation With Procmon - Smokeloader Example

By Matthew

Published: 2023-06-24 · Archived: 2026-04-05 15:34:22 UTC

This post will show you how to manually decode a SmokeLoader visual basic (.vbs) script using Procmon. From here you will see how to retrieve additional stages using Powershell and identify a malware sample using sandbox tooling.

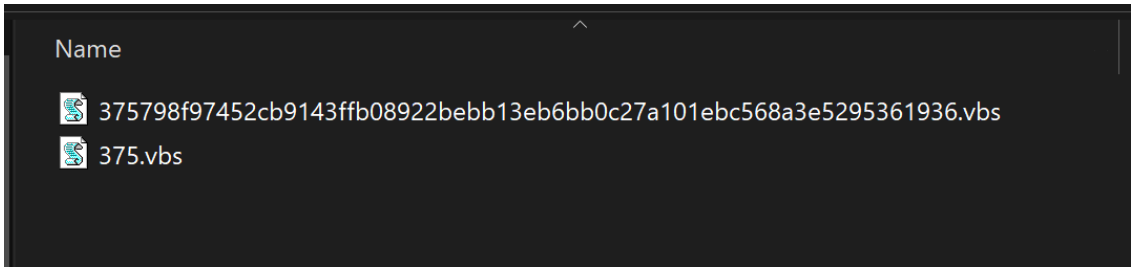
The initial file can be [downloaded from malware bazaar] and unzipped using the password `infected`.

`SHA256:375798f97452cb9143ffb08922bebb13eb6bb0c27a101ebc568a3e5295361936`

Initial Analysis

The initial file after unzipping is a visual basic `.vbs` script.

An additional copy `375.vbs` was made in order to preserve the original and work with a simpler filename.



Since visual basic is a text-based language, the file can be opened using a text editor.

This blog will utilise sublime text, but visual code, notepad++, or any other text editor will work equally well. (Any text editor with language highlighting and find/replace with regex support)

The script is "only" 10 lines long and primarily consists of a large blob of decimal values (line 1), a large blob of text (line 3).

The remainder of the surrounding code is used to decode the decimal and text blob.

Decoding Malware with Process Monitor (Procmon)

This is by far the simplest method for decoding script-based malware. This method involves executing the script inside of a safe virtual machine and simultaneously running the "Process Monitor" tool from Sysinternals.

This method will capture any new processes spawned by the obfuscated script, revealing any decoded command line arguments that were used to spawn the new process. This bypasses a lot of the obfuscation that may be present in an original encoded script.

There are downsides to this method as it assumes that a new process will be spawned, but it is the easiest method and is a great skill to have.

If you are using flare-vm, you will already have Procmon installed. If not, you can obtain it from the following link.

How to monitor the malware with ProcMon

To "decode" the malware using Procmon, you must first start the Procmon process and perform a few basic actions.

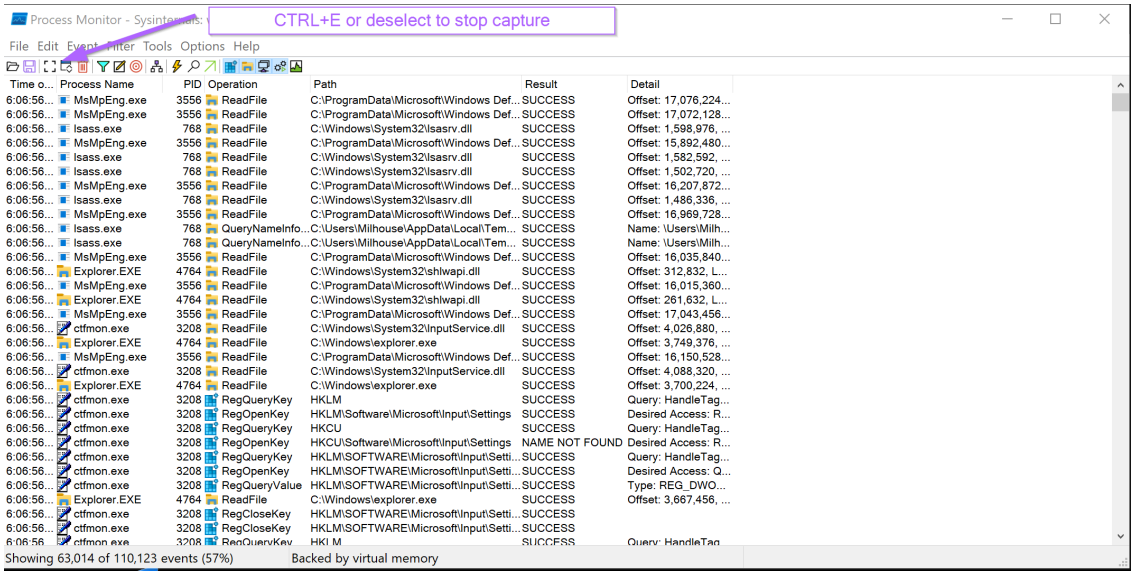
These basic actions are needed to focus on only the events related to the malware.

Since Procmon can capture hundreds of thousands of events per second, this can quickly eat up memory, so you want to make sure to capture the right events.

1. Locate and open the Procmon process
2. Stop Capture (CTRL+E), or manually deselect the capture button.
3. Clear the window (CTRL+X)
4. Set a filter on `WScript.exe` - (CTRL+L)
5. Turn on capture and run the malware.

Here we can see the initial screen when Procmon is first opened. Within seconds, 63,014 total events are captured. We want to stop this as soon as possible.

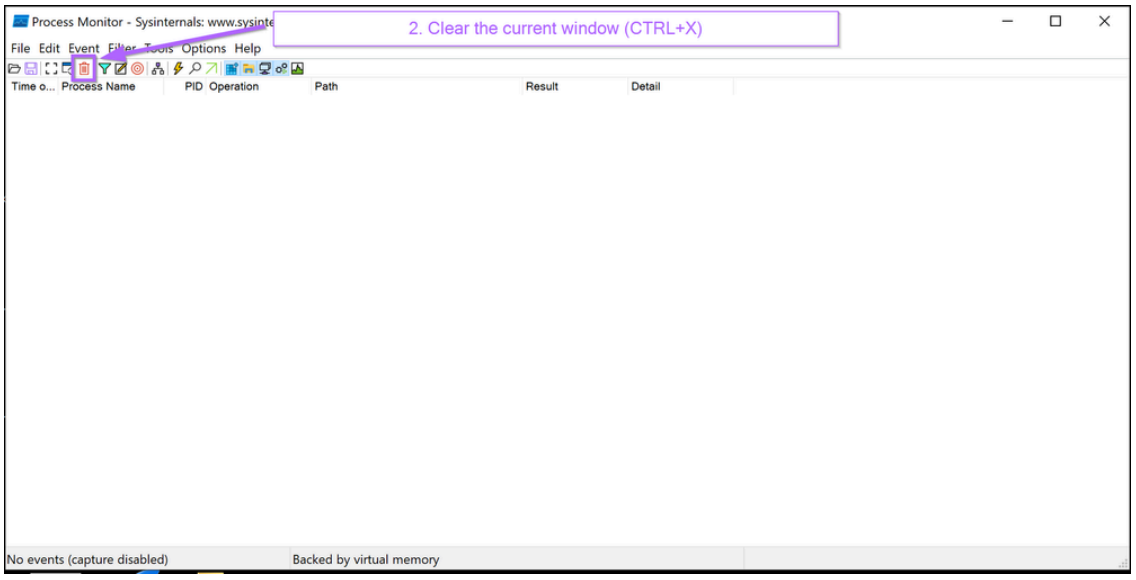
The stop capture can be done with CTRL+E or by manually de-selecting the capture button.



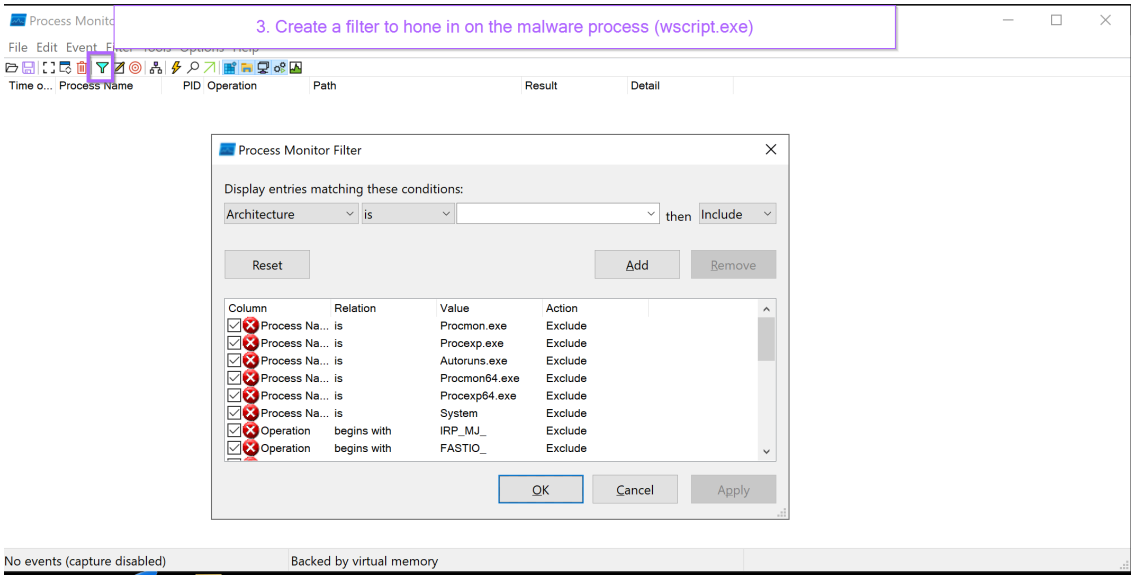
Opening procmon and stopping capture

Once the capture has been stopped, the already captured events will need to be cleared from the screen.

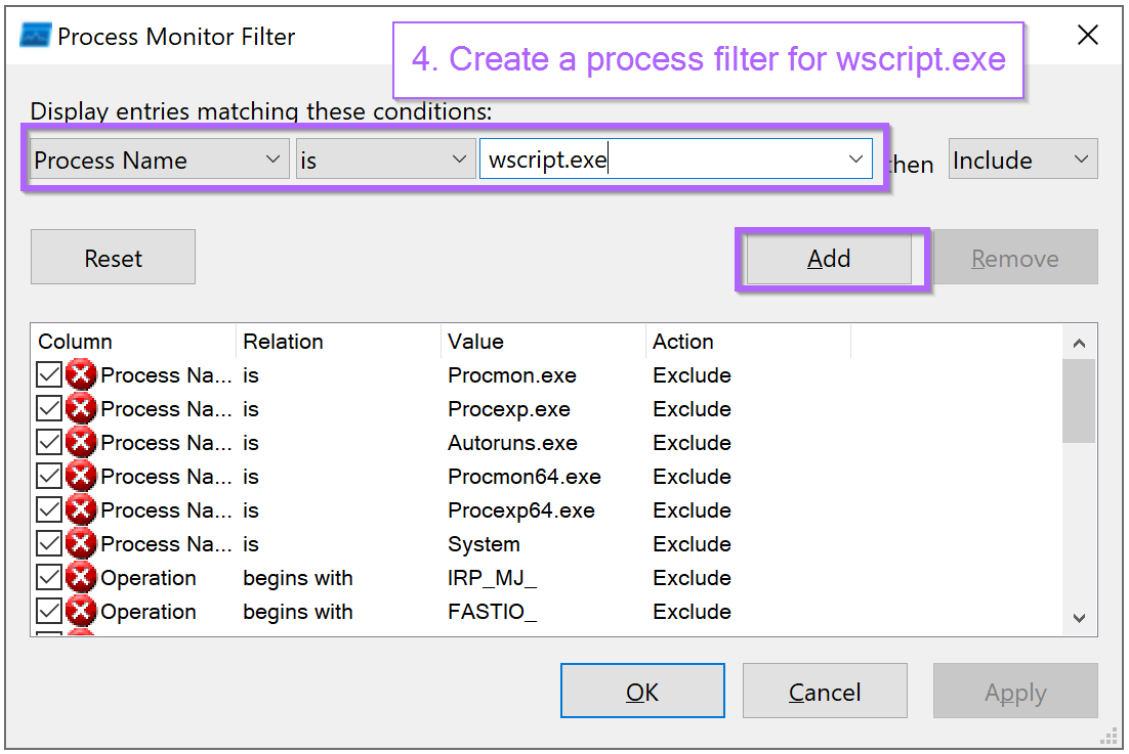
The captured events can be cleared with CTRL+X or by hitting the trash can button. This creates a clean screen for easy future analysis.



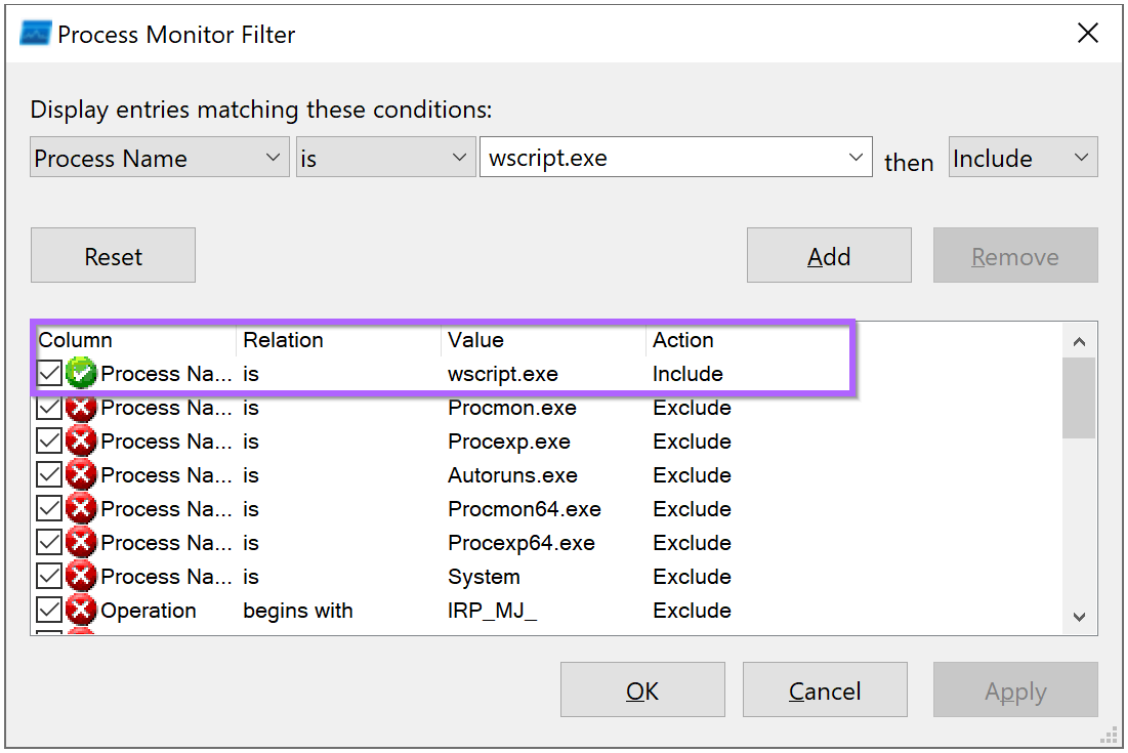
With the window now cleared, a new filter can be created with CTRL+L or by hitting the filter button. This will allow us to "hone in" on only wscript.exe, which is the process responsible for running .vbs scripts.



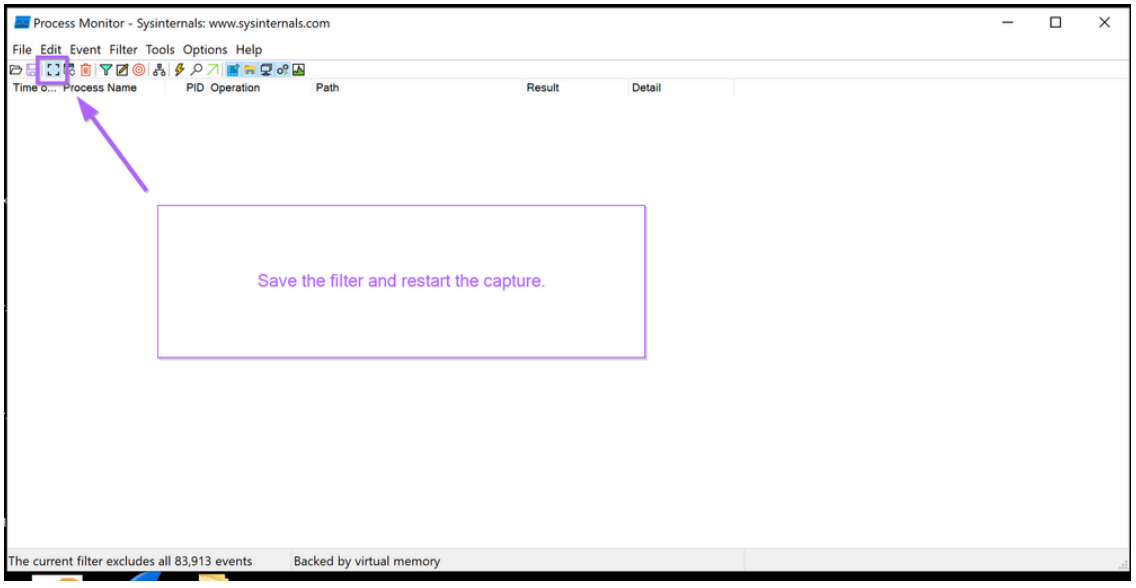
A new process filter for `wscript.exe` can now be created. Ensuring to press "add" to save the new filter.



This will result in a new filter entry for `wscript.exe`.

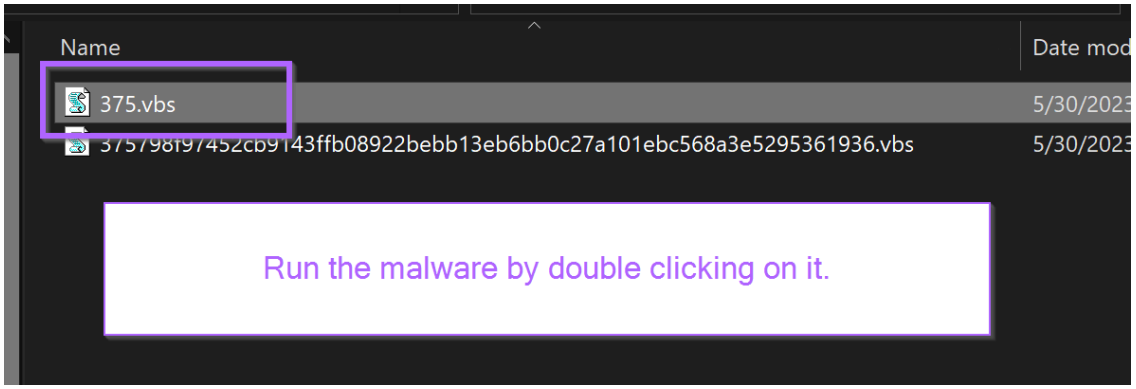


At this stage, the event capture can be re-started. This will begin capturing all events related to `wscript.exe`

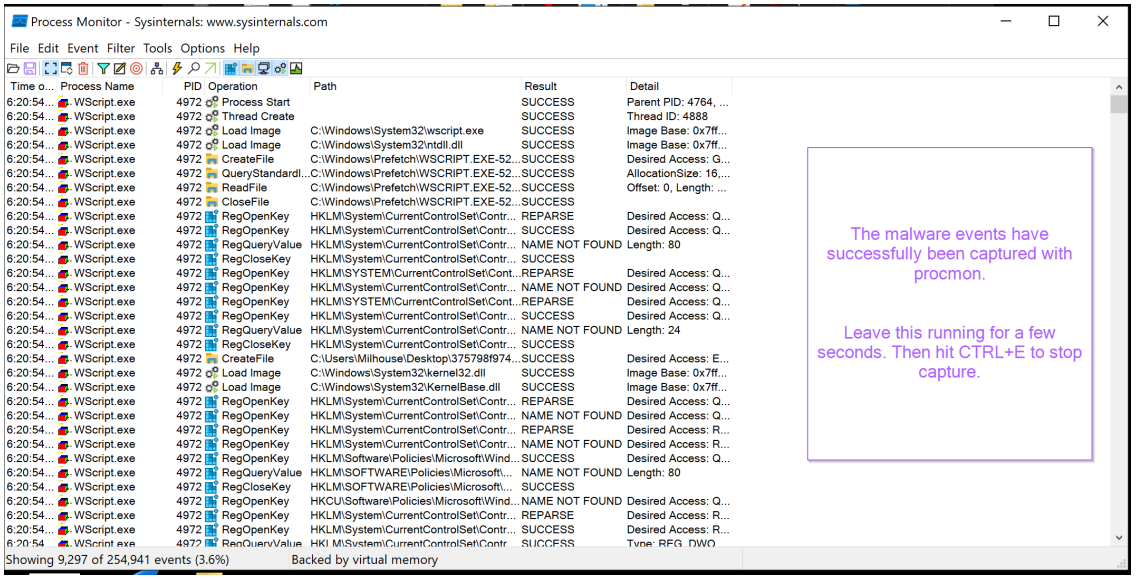


Now that the capture is ready, it's time to run the original malware script.

This is as simple as double clicking on the original .vbs file. Windows will run the script using `wscript.exe` by default.



With the filters correctly set, the events will now be captured using Procmon.



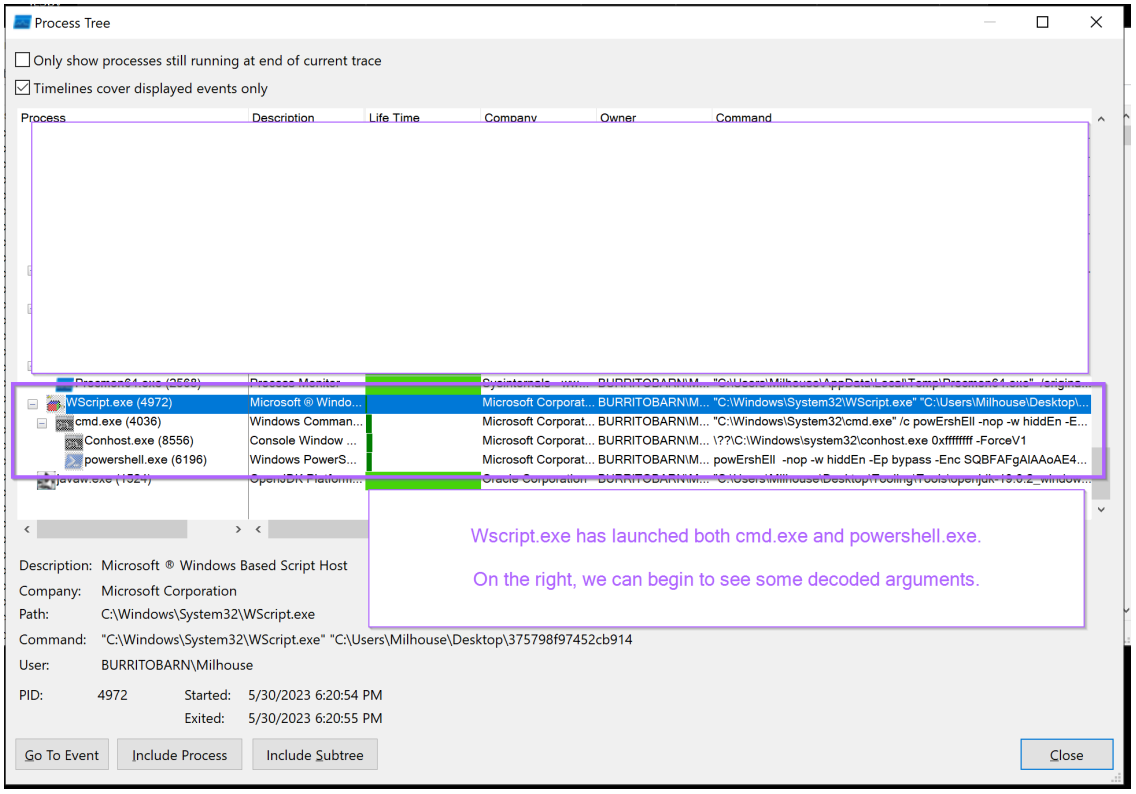
At first glance this is a lot (9297 events in just a few seconds) but we will soon filter down to a manageable number.

> The primary focus here is to identify if any new processes were spawned during the execution of the script. If a new process has been launched, we want to observe any arguments that have been passed and see if this reveals the functionality of the malware or at least brings us closer to something that allows us to determine what it does.

Identifying Spawned Processes Using Procmon

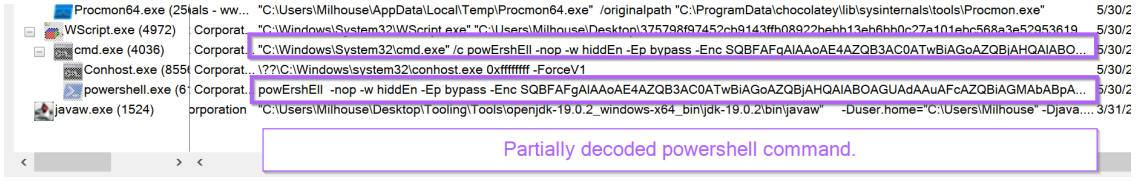
The process tree is the best way to identify newly spawned processes. This can be accessed by pressing `CTRL+T` or browsing the Procmon menu `Tools -> Process Tree`.

This will reveal a window similar to below. The top half has been covered to improve readability.



In the screenshot above - We can see that `WScript.exe` has ultimately spawned 3 new processes. `Cmd.exe, Conhost.exe and powershell.exe.`

By honing in on the right-most column titled `command`, you can observe the decoded commands that were used to spawn each process.



In the `cmd.exe` command - You can see that cmd.exe was used to spawn Powershell via the `/c` argument. The cmd.exe serves no malicious purpose, it serves only to spawn the Powershell.

> The `/c` argument will cause the powershell process to terminate after it has finished. This avoids a powershell terminal hanging around on the screen if powershell was launched directly.

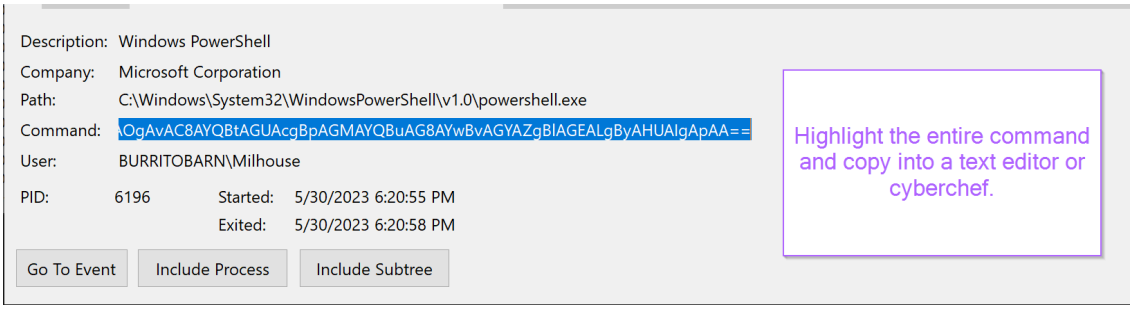
Detection related tangent

> The usage of `cmd.exe` also introduces the process relationship of `WScript.exe -> cmd.exe -> powershell.exe`. This may hinder detection in some SIEM tooling that do not capture grandparent processes.
> Eg Wscript.exe -> powershell is not common and would make a simple and reliable detection.
> Cmd.exe -> powershell.exe and wscript.exe -> cmd.exe are both very common and would require tuning and additional filtering for reliable detection.

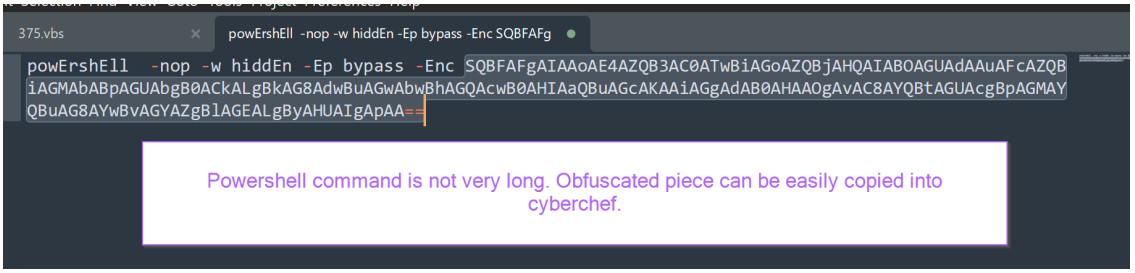
Returning back to the Procmon output. The final command can be easily observed by clicking on the line containing PowerShell.

The content has not been fully de-obfuscated yet. But we now have a powershell command with a seemingly simple base64. This is much better than the initial obfuscated .vbs script.

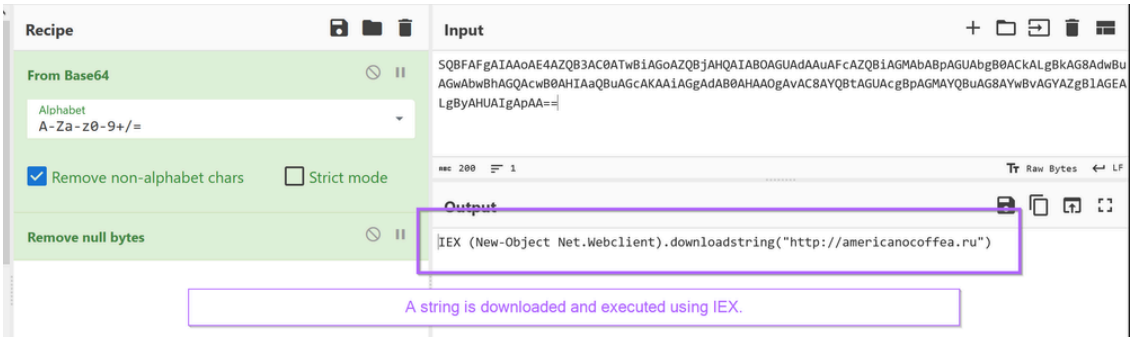
To obtain the full contents, you can highlight the command window and hit `CTRL+C`.



Pasting back into a text-editor, the semi-decoded powershell command can be observed.



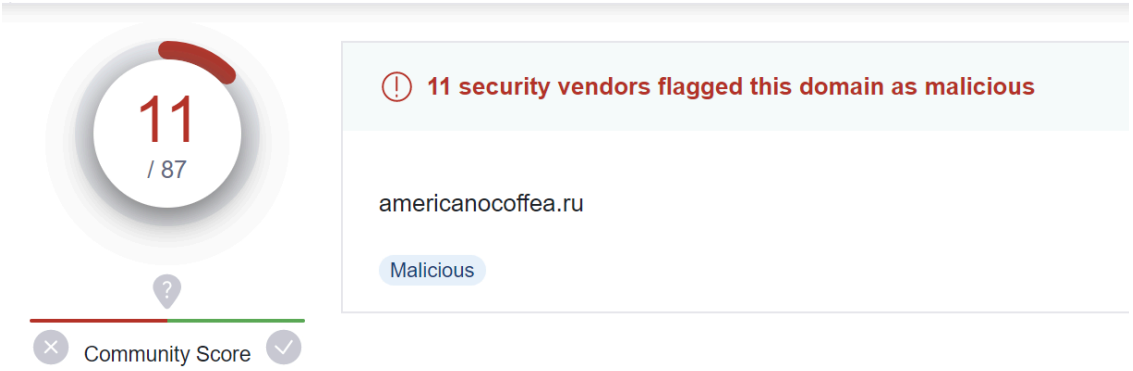
The final decoded component is easily obtained using CyberChef and `From Base64`. Remember to add "Remove Null Bytes" if you observe any dots or weird red lines. This is due to the utf-16 encoding common in windows.



We can now observe a decoded command that downloads a string from `americanocoffea[.]ru`. The resulting string is then executed using Invoke-Expression (IEX). Since the decoded string is executed within Powershell, it is likely another Powershell script.

Domain Analysis

The domain `americanocoffea[.]ru` had 11/87 detections at the time of writing `2023/05/23`. There was no information available on VirusTotal to determine which malware was being downloaded.



A malicious domain has now been identified and can be used as an IOC. However, there is no information on the malware that may be downloaded.

Retrieving a Malware Payload with Powershell

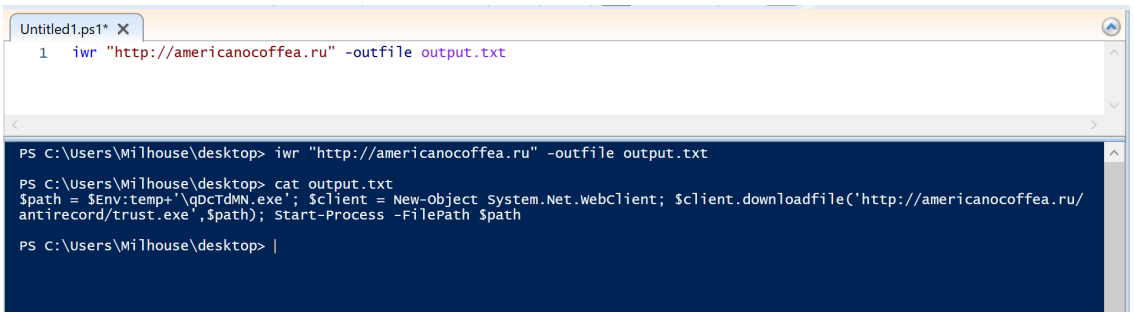
If the malware infection and script are recent enough, then the next stage can be obtained directly from the malicious server using Powershell.

The simplest way to do this is to use Powershell `invoke-webrequest` (iwr) from within a safe virtual machine (and ideally behind a VPN).

> If you require additional anonymity from attacker infrastructure, there are also tools such as grabbrapp for safely probing infrastructure and obtaining malware payloads.

In an ideal situation such as this one, PowerShell works just fine. The sample can be obtained by running the following command.

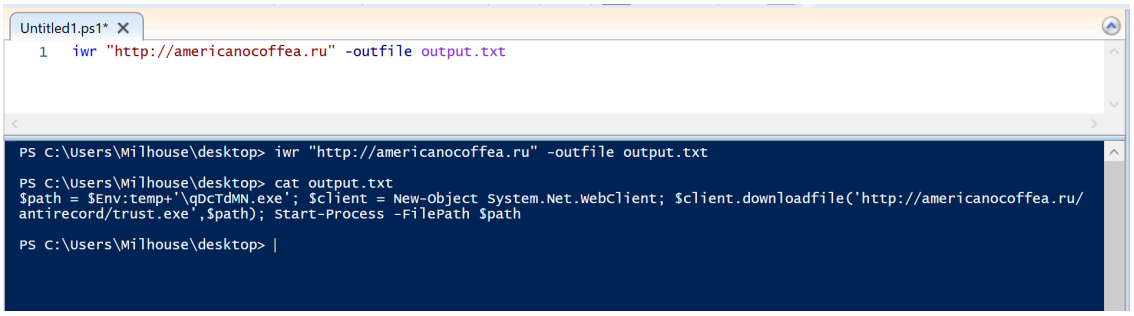
```
powershell.exe -iwr http://americanocoffea[.]ru -outfile output.txt
```



In the output (blue) of the screenshot above, we can see the downloaded content is *another* small Powershell script. This script retrieves and executes an executable file `qScTdMN.exe` and writes it to the user's temp directory `$env:temp` .

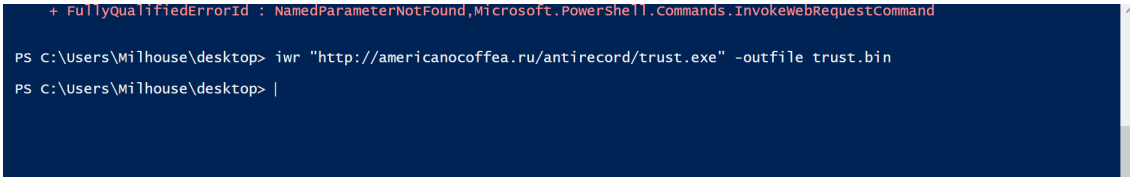
`http://americanocoffea[.]ru/antirecord/trust[.]exe`

This full URL and exe name would both make for good indicators in an IR situation. The use of randomly named .exe files in the user temp directory may also be a good indicator.



The next `.exe` can be retrieved using the same technique with `iwr` . Retrieving this additional file can be useful to obtain a full hash and binary information. In particular, you would now have a working sample that you can provide to a sandbox tool, malware analyst or any other tooling for the analysis of malware.

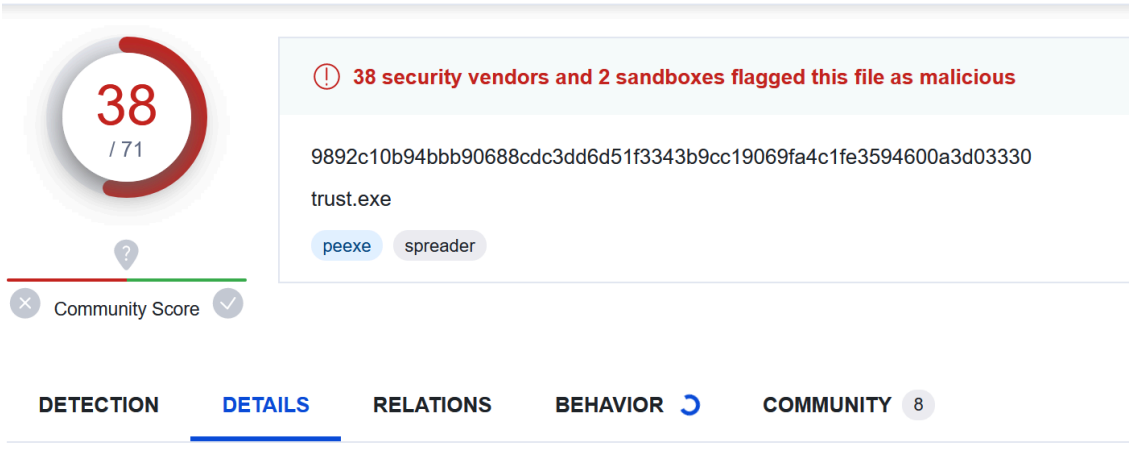
Below is a simple `iwr` command to download the next stage containing the `.exe` file.



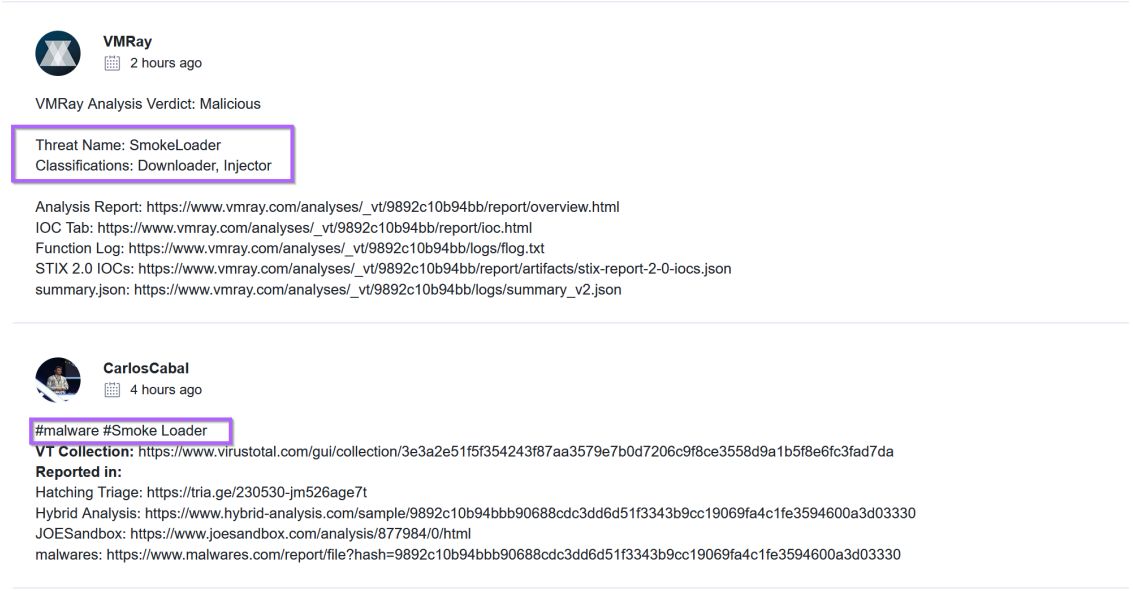
Checking the file hash on VirusTotal. There are 37/71 detections.

In many cases, you do not need (and may not be allowed) to submit the file to Virustotal. In this situation you can obtain a file hash (using detect-it-easy, pestudio etc) and submit the hash to see if it has previously been analysed.

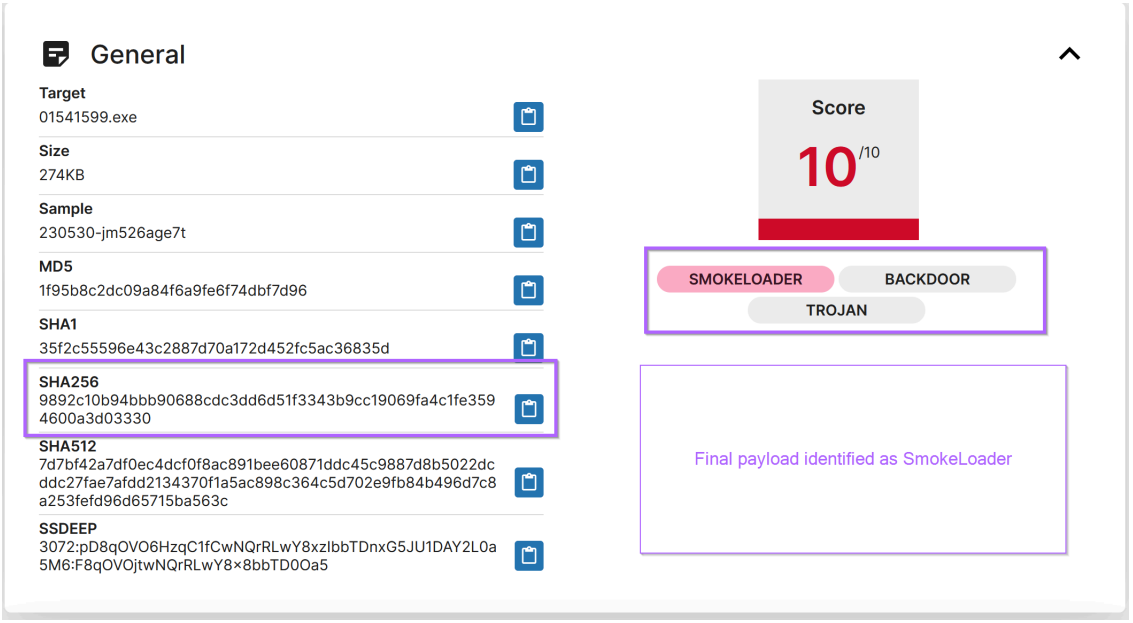
In this case, the file hash was already known to Virustotal.



Reviewing the comments reveals multiple references to SmokeLoader.



One of the links above is to a Triage analysis. With a confident 10/10 verdict of `SmokeLoader` for an identical file.



At this point, I would be confident to label the malware sample and incident as Smokeloader.

Conclusion

The malware has now been decoded and identified as Smokeloader. A domain and full URL have now been obtained, as well as an exe hash and confident verdict of the malware family.

Sign up for Embee Research

Malware Analysis and Threat Intelligence Research

No spam. Unsubscribe anytime.
