We're continuing to fight for universal access to quality information—and you can help as we continue to make improvements. Will you chip in?





14 captures 29 Sep 2020 - 4 Jan 2024

https://blog.menasec.net/2019/07/interesting-difr-traces-of-net-clr.html

Applied Security Research

Home About us

Tuesday, 16 July 2019

Interesting DFIR traces of .NET CLR Usage Logs

As most of you already know .NET has become an increasingly important component in the offensive world, with attackers making increasing direct use of it as well as indirect use of it via existing windows scripting utilities. One good example of the indirect approach is DotNetToJScript, which allow to deliver managed code via a simple JavaScript.

We decided to take a closer look to this category of malicious code delivery, which lead us to this great Offensive tool by MDSec "SharpShooter" (at it's heart make use of DotNetJScript).

SharpShooter allow to generate multiple payload formats (hta, js, jse, vba, vbe, vbs, wsf), if your are interested about how it works or how to use it please refer to this MDSec post.

For testing purposes we will be using the .hta payload as an example, below an example of the content of our test payload (will spawn notepad.exe):

```
OQZGUURG TO LV 30g Ld JSR LMC 31th -44 mBFYQL 40 mENT pictrophikp = 64 to AVML O'D invers J/Ogl MSYLYTY SUUNKEERBOCK 20X LWINSE; JME LST EXSKHUNGENGT HUNGER ON MANT TO LAKE 65 UNK RGYMEN PJ LS 14 YEAR 14 LWINGER ON MANT TO LAKE 65 UNK RGYMEN PJ LS 14 YEAR 14 LWINGER ON MANT TO LAKE 65 UNK RGYMEN PJ LS 14 YEAR 14 LWINGER ON MANT TO LAKE 65 UNK RGYMEN PJ LS 14 YEAR 14 LWINGER ON MANT TO LAKE 65 UNK RGYMEN PJ LS 14 YEAR 14 LWINGER ON MANT TO LAKE 65 UNK RGYMEN PJ LS 14 YEAR 14
```

As you can see above, it uses RC4 with the key "wxzomjyhto" to decrypt a base64 decoded blob, and then execute the resulting VBScript, below the decoded script:

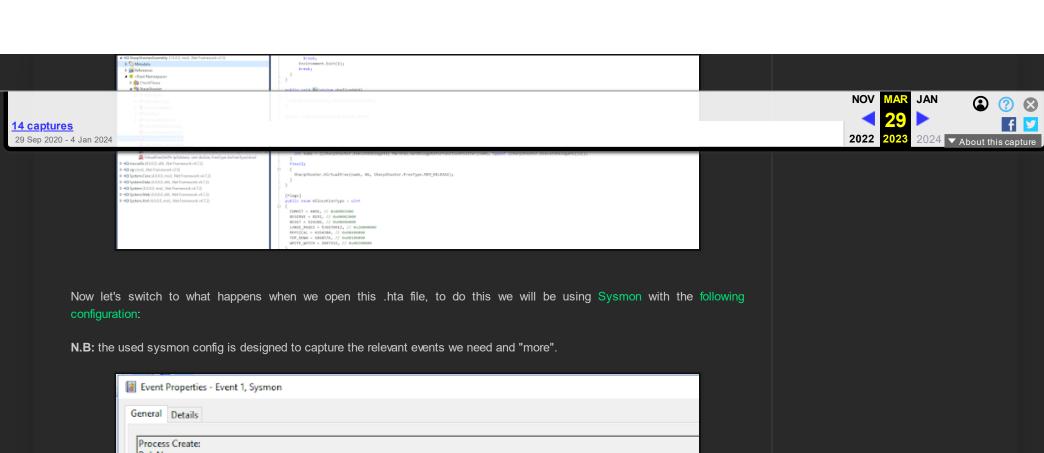
As you can see above, it uses the Deserialize_2 method of the

"System.Runtime.Serialization.Formatters.Binary.BinaryFormatter" COM Object which is "high level" how the "DotNetToJScript" technique works to load managed code via object Deserialization.

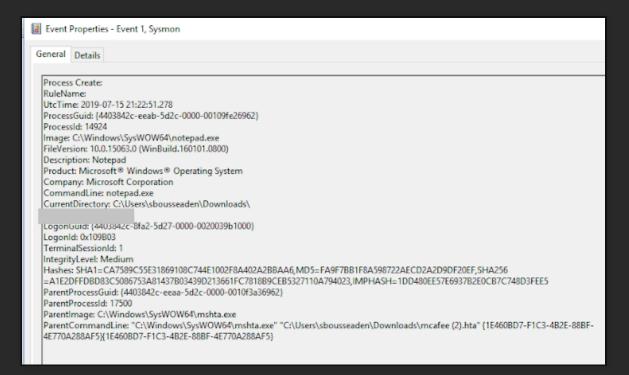
Decoding the base64 encoded blob will lead us to a .NET executable, which will be used to load and execute our msfvenom base64 encoded shellcode (see "o.Go" method call), the shellcode will simply launch notepad.exe.

Blog Archive

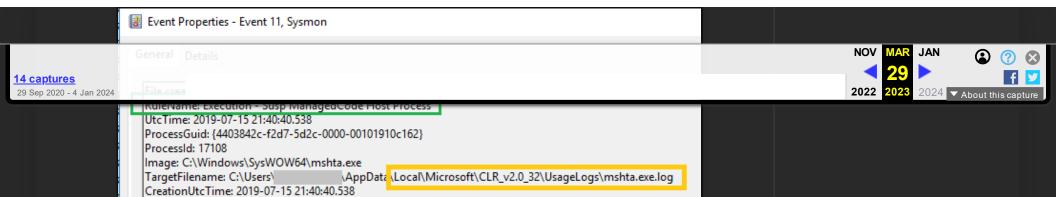
- **2022 (2)**
- **2021 (3)**
- **2020 (4)**
- **2019** (39)
 - ► November (2)
 - ▼ July (1)
 Interesting DFIR traces of .NET
 - April (3
 - ► March (7)
 - ► February (26)



RuleName: UtcTime: 2019-07-15 21:22:50.337 ProcessGuid: {4403842c-eeaa-5d2c-0000-0010f3a36962} Image: C:\Windows\SysWOW64\mshta.exe FileVersion: 11.00.15063.0 (WinBuild.160101.0800) Description: Microsoft (R) HTML Application host Product: Internet Explorer Company: Microsoft Corporation CurrentDirectory: C:\Users\sbousseaden\Downloads\ LogonGuid: {4403842c-8fa2-5d27-0000-0020039b1000} Logonld: 0x109B03 TerminalSessionId: 1 IntegrityLevel: Medium Hashes: SHA1=8AA61DC441A1A04C50F16B38EF82D495584BFE35,MD5=06DBEE04BF1523C1F4BAD78A94CFF822,SHA256 =B407BC80B3B0E7CC327986C8423E99DEAF536349521724EABAFD5A6152892484,IMPHASH=3977AA5A1E947ED0E8323A1974EAC30D ParentProcessGuid: {4403842c-8fd7-5d27-0000-001006273500} ParentProcessId: 12152 ParentImage: C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
ParentCommandLine: "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" Log Name: Microsoft-Windows-Sysmon/Operational 7/15/2019 11:22:50 PM Source: Sysmon Logged: Event ID: 1 Task Category: Process Create (rule: ProcessCreate) Information Keywords:



As you can see above, from process execution events, we don't see any clear traces of .NET code execution. enabling CLR common modules loading logging via sysmon is not an option since it's very noisy and lot of processes loads those DLLs. Lucklily for us while observing mshta.exe execution via ProcMon, we saw an interesting file being created under Microsoft .NET CLR usage logs [%localappdata%\Microsoft\CLR_v<version_number>\UsageLogs\ProcessName.exe.Log]:



7/15/2019 11:40:40 PM

Task Category: File created (rule: FileCreate)

The file creation date indicate the first time the process was executed, for any further executions of the same process, the same file is updated and no file creation event is recorded. Content of the file display the list of linked assembly modules and their versions:

Logged:

Keywords:

Microsoft-Windows-Sysmon/Operational

Sysmon

Information

11

Log Name:

Source:

Event ID:

Level:

```
Indicates the Normal No
```

First question that comes to our mind after observing this file system precious artifact, is what are the windows native system processes that normally loads .NET code, to find out we've used 3 months of EDR process and file creation telemetry covering more 700 Windows 10 endpoints and we filtred for any process starting from "c:\windows\s*" which covers wscript.exe, cscript.exe and other processes:

✓ C:\Windows\System32\AppV\AppVStreamingUX.exe ▼ C:\Windows\System32\DriverStore\FileRepository\prosetswcomponent.inf_amd64_e9a24c476cc5252e\INFAppRunner.exe ✓ C:\Windows\System32\eed_sl.exe ✓ C:\Windows\System32\gpresult.exe C:\Windows\System32\inetsrv\InetMgr.exe ✓ C:\Windows\System32\mmc.exe ✓ C:\Windows\System32\rundll32.exe C:\Windows\System32\sdiagnhost.exe ✓ C:\Windows\System32\taskhostw.exe ✓ C:\Windows\System32\tzsync.exe ✓ C:\Windows\System32\vmconnect.exe C:\Windows\System32\wbem\WmiPrvSE.exe ·☑ C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe C:\Windows\System32\WindowsPowerShell\v1.0\powershell_ise.exe ✓ C:\Windows\SysWOW64\msiexec.exe C:\Windows\SysWOW64\rundll32.exe C:\Windows\SysWOW64\sdiagnhost.exe C:\Windows\SysWOW64\WindowsPowerShell\v1.0\powershell.exe

As you can see above, the windows system processes that loads managed code are quite limited and can be baselined, for instance a straightforward detection is to alert for the following:

```
<TargetFilename condition="end with" name="Execution - Susp ManagedCode Bost Process">\UsageLogs\cscript.exe.log</TargetFilename> <!-- suspicious NET executions -->

<TargetFilename condition="end with" name="Execution - Susp ManagedCode Bost Process">\UsageLogs\wmic.exe.log</TargetFilename> <!-- suspicious NET executions -->

<TargetFilename condition="end with" name="Execution - Susp ManagedCode Bost Process">\UsageLogs\wmic.exe.log</TargetFilename> <!-- suspicious NET executions -->

<TargetFilename condition="end with" name="Execution - Susp ManagedCode Bost Process">\UsageLogs\wshta.exe.log</TargetFilename> <!-- suspicious NET executions -->

<TargetFilename condition="end with" name="Execution - Susp ManagedCode Bost Process">\UsageLogs\wswtost.exe.log</TargetFilename> <!-- suspicious NET executions -->
```

While googling for extra information about .NET UsageLogs, we come accross this interesting article explaining how to use CLR Load logging (different than UsageLogs) for debugging purposes and that can be enabled via a simple registry change and specifying where a path where to store those logs, doing so resulting in the following interesting details after the .hta execution:

> This PC > Windows (C:) > clrloadlogs					
	Name	Date modified	Туре	Size	Date created
yt.	mshta.exe.CLRLoad05.log	7/15/2019 11:40 PM	Text Document	8 KB	7/15/2019 11:40 PM
	mshta.exe.CLRLoad04.log	7/15/2019 11:22 PM	Text Document	8 KB	7/15/2019 11:22 PM
x	mshta.exe.CLRLoad03.log	7/15/2019 11:21 PM	Text Document	8 KB	7/15/2019 11:21 PM
7th	mshta.exe.CLRLoad02.log	7/15/2019 10:56 PM	Text Document	8 KB	7/15/2019 10:56 PM
A.	NGenTask.exe.CLRLoad01.log	7/15/2019 5:26 PM	Text Document	8 KB	7/15/2019 5:26 PM
	ngen.exe.CLRLoad94.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:23 PM
	ngen.exe.CLRLoad95.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:23 PM
	ngen.exe.CLRLoad96.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:23 PM
	ngen.exe.CLRLoad97.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:23 PM
	ngen.exe.CLRLoad98.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:23 PM
	ngen.exe.CLRLoad99.log	7/15/2019 5:23 PM	Text Document	4 KB	7/15/2019 5:23 PM
	ngen.exe.CLRLoad87.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:22 PM
	ngen.exe.CLRLoad88.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:22 PM
	gen.exe.CLRLoad89.log	7/15/2019 5:22 PM	Text Document	4 KB	7/15/2019 5:22 PM

For every execution a log file is created, below an example of SharpShooter .hta payload:



Although the CLR Load Logs provide more detailed information including invoked .NET COM objects, FunctionCall and Methods's names it's quite verbose and you can't exclude noisy processes.

TakeAway:

Using your EDR or Sysmon, hunt for File Creation with file path and name matching the following logic:

 $\verb|\w| wscript.exe.log| wscript.exe.log$

The advantage of this detection method is that you can hunt for it using just powershell or alike to scan filesystem for any matching file that related to a potential previous infections..

For RedTeamers, go for the vba payload as winword.exe and excel.exe are legit managed code host processes. and make sure you delete the corresponding .NET usage .log file if you plan to use hta, vbscript or jscript payloads.

Bonus:

You can download example of evtx logs for SharpShooter sysmon traces here.

Posted by MENASEC at 00:27

1 comment:

Anonymous 11 November 2022 at 08:56

Until at least of|no much less than} 5000 surveys were completed, lpsos drew extra samples. In the method, lpsos utilized Massachusetts on-line panel members from seven partner vendors to supplement their own on-line panel pattern. However, 18,580 were not eligible (i.e., residing out of state), 2946 did not complete the survey, 293 surveys were not used because of a full gender and age quota, and forty eight were eradicated due to 1xbet poor information high quality. This chapter in all probability not|will not be} construed, interpreted, or utilized to the possession of a reverse merchandising machine.

Reply

To leave a comment, click the button below to sign in with Google.

SIGN IN WITH GOOGLE

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)

Simple theme. Powered by Blogger.