

WINDOWS EVENT LOGS & FINDING EVIL:

Link to challenge: <https://academy.hackthebox.com/module/216/>

(log in required)

Class: Tier II | Medium | Defensive

Introduction

Windows Event Logs:

Question: Analyze the event with ID 4624, that took place on 8/3/2022 at 10:23:25. Conduct a similar investigation as outlined in this section and provide the name of the executable responsible for the modification of the auditing settings as your answer. Answer format: T_W____.exe

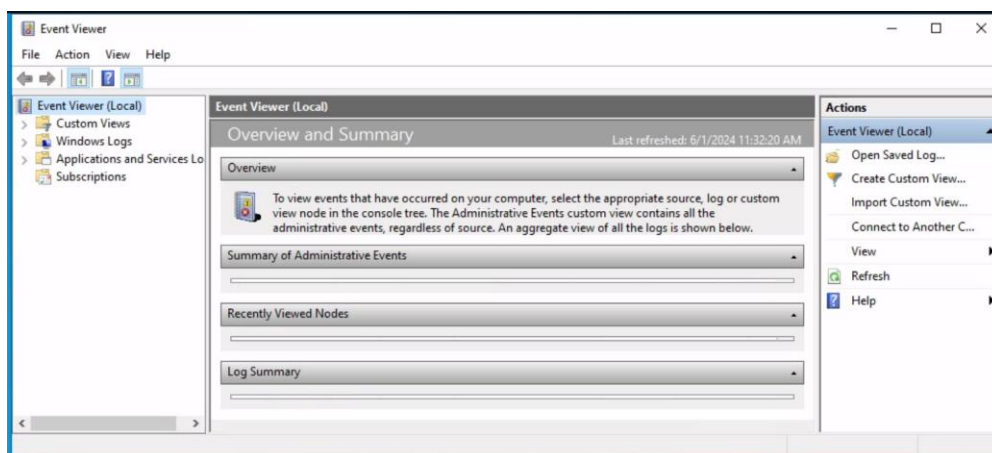
Answer: TiWorker.exe

Method:

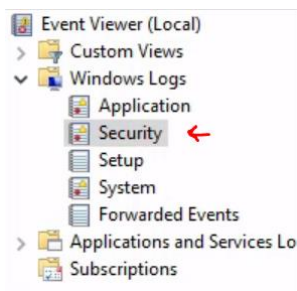
First (for all answers) – login to window machine with RDP with this linux command:

```
xfreerdp /u:Administrator /p:'HTB_@cad3my_lab_W1n10_r00t!@0' /v:[Target IP] /dynamic-resolution
```

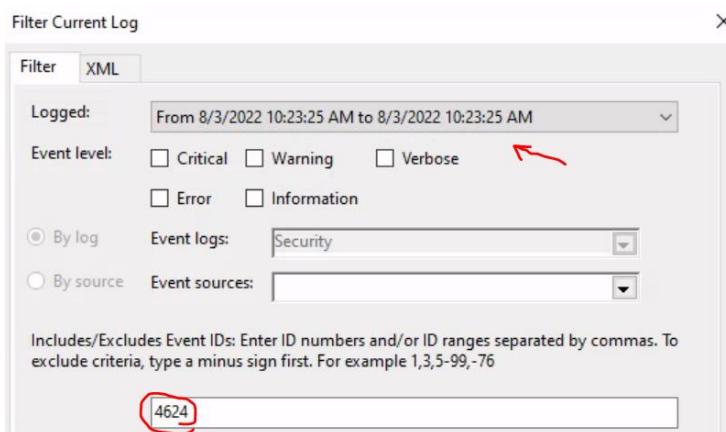
then, start event viewer on target Windows machine:



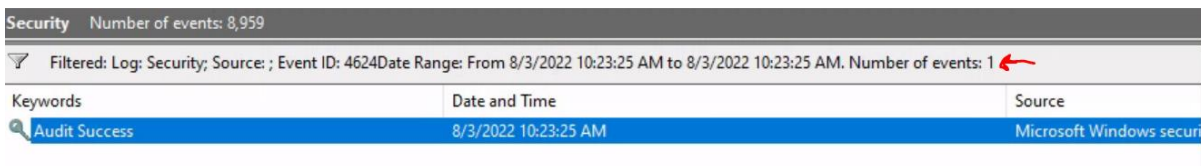
Enter on side bar Windows Logs->Security:



Then filter the logs by the specified instructions:



Observe there is only one log:



the 4624 eventID tell us that there was a successful logon on August 3rd 2022, 10:23:25 (we are looking at mm,dd,yyyy date format).

Now as we need to look for the process that is responsible for audit settings modifications that are subsequent to the Logon, that would be eventID 4907 (audit policy change).

We will search for eventID=4907, and the next 30 seconds after the logon success timeframe:

Filter Current Log

Filter XML

Logged: From 8/3/2022 10:23:25 AM to 8/3/2022 10:23:55 AM

Event level: ☐ Critical ☐ Warning ☐ Verbose
☐ Error ☐ Information

☒ By log Event logs: Security

☐ By source Event sources:

Includes/Excludes Event IDs: Enter ID numbers and/or ID ranges separated by commas. To exclude criteria, type a minus sign first. For example 1,3,5-99,-76

4907

We get several results:

Security Number of events: 9,313				
Filtered: Log: Security; Source: ; Keywords: win:AuditFailure, win:AuditSuccess; Event ID: 4907Date Range:				
Keywords	Date and Time	Source	Event ID	Task Category
Audit Success	8/3/2022 10:23:50 AM	Microsoft Windows security auditing.	4907	Audit Policy Change
Audit Success	8/3/2022 10:23:50 AM	Microsoft Windows security auditing.	4907	Audit Policy Change
Audit Success	8/3/2022 10:23:50 AM	Microsoft Windows security auditing.	4907	Audit Policy Change

Let's observe randomly the top result, and search for process name:

Event Properties - Event 4907, Microsoft Windows security auditing.

General Details

Process Information:

Process ID: 0x8

Process Name: C:\Windows\WinSxS\amd64_microsoft-windows-servicingstack_31bf3856ad364e35_10.0.19041.1790_none_7df2aec07ca10e81\TiWorker.exe

Auditing Settings:

Log Name: Security

Source: Microsoft Windows security

Event ID: 4907

Level: Information

User: N/A

OpCode: Info

More Information: [Event Log Online Help](#)

Logged: 8/3/2022 10:23:50 AM

Task Category: Audit Policy Change

Keywords: Audit Success

Computer: DESKTOP-NU10MTO

Question: Build an XML query to determine if the previously mentioned executable modified the auditing settings of C:\Windows\Microsoft.NET\Framework64\v4.0.30319\WPF\wpfgfx_v0400.dll. Enter the time of the identified event in the format HH:MM:SS as your answer.

Answer: 10:23:50

Method: we will construct xml script, the script itself can be downloaded from [here](#):

```
1 <QueryList>
2   <Query Id="0" Path="Security">
3     <Select Path="Security">
4       <!-- Audit Policy Change Events -->
5       * [System[ (EventID=4907) ]]
6       and
7       * [EventData[Data[@Name='SubjectLogonId'] and (Data='0x3E7')]]
8       and
9       * [EventData[Data[@Name='ProcessName'] and (Data='C:\Windows\WinSxS\amd64_microsoft-windows-servicingstack_31bf3856ad364e35_10.0.19041.1790
10      _none_7df2aec07ca10e81\TiWorker.exe')]]
11      and
12      * [EventData[Data[@Name='ObjectName'] and (Data='C:\Windows\Microsoft.NET\Framework64\v4.0.30319\WPF\wpfgfx_v0400.dll')]]
13    </Select>
14  </Query>
15 </QueryList>
```

In short – the script will search the parameters: EventID=4907 (audit policy change), LogonID=0x3e7, Process name of the 'TiWorker.exe' executable that committed the modification, and the target object in which the modification was committed upon.

Lets observe the results:

Security Number of events: 8,838	
Filtered: Advanced filter, click on "Filter" command to view filter configuration.. Number of events: <u>1</u>	
Keywords	Date and Time
 Audit Success	8/3/2022 <u>10:23:50 AM</u>

A single result, with the time of 10:23:50 AM.

Analyzing Evil With Sysmon & Event Logs:

Question: Replicate the DLL hijacking attack described in this section and provide the SHA256 hash of the malicious WININET.dll as your answer.

"C:\Tools\Sysmon" and "C:\Tools\Reflective DLLInjection" on the spawned target contain everything you need.

Answer:

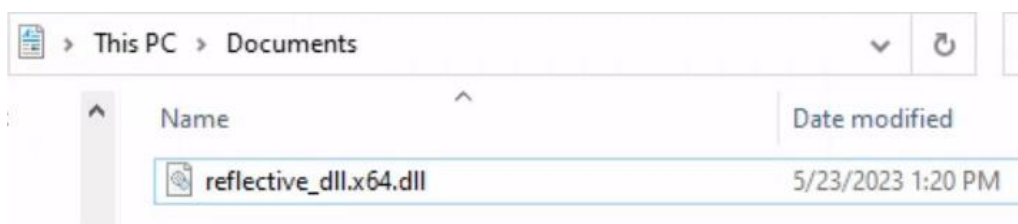
51F2305DCF385056C68F7CCF5B1B3B9304865CEF1257947D4AD6EF5FAD2E3B13

Method:

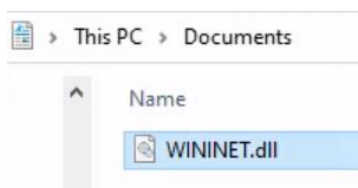
What we need to do is a. run the dll injection, and b. search for the activity in the event log.

First – according to the instructions, we can find the reflective_dll.x64.dll

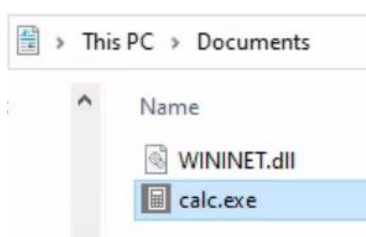
In the C:\Tools\Reflective DLL Injection path, we will move it from that to Documents folder:



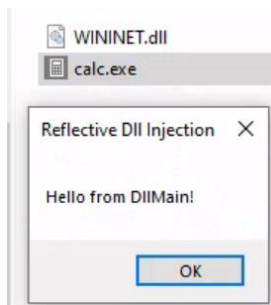
Then we will rename it 'WININET.dll', a dll used by calc.exe:



Then we will copy calc.exe from C:\Windows\System32\calc.exe to Documents folder:



Running the calc.exe in the Documents folder, will ensure the calc.exe will utilize the malicious WININET.DLL we just placed in the folder, instead the original one in C:\Windows\System32, lets test it:



Upon executing calc.exe – we got the message above instead the expected calculate app.

Now, we need to find the activity in the event viewer:

First – we need to modify the configuration file of sysmonconfig-export.xml, that is used by the tool Sysmon – so that relevant data about module Loads will be logged:

In our specific use case, we aim to detect a DLL hijack. The Sysmon event log IDs relevant to DLL hijacks can be found in the Sysmon documentation (<https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon>). To detect a DLL hijack, we need to focus on **Event Type 7**, which corresponds to module load events. To achieve this, we need to modify the **sysmonconfig-export.xml** Sysmon configuration file we downloaded from

(more detailed explanation in which the picture above is taken from - is the module section, at the link of the module above).

We will change the value from 'include' to 'exclude':

```
<RuleGroup name="" groupRelation="or">
  <ImageLoad onmatch="exclude">
    <!--NOTE: Using "include" with no rules means nothing in this section will be logged-->
  </ImageLoad>
</RuleGroup>
```

When done, we update the Sysmon configuration with the cmd command

```
sysmon.exe -c sysmonconfig-export.xml
```

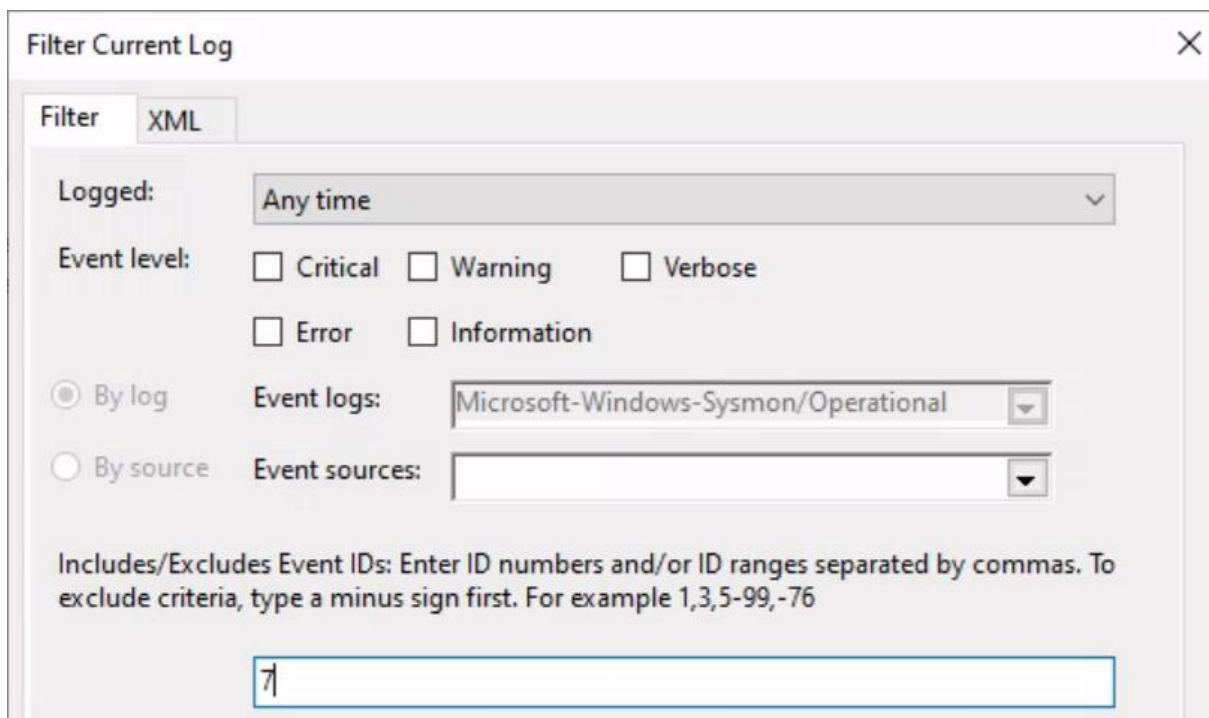
```
Administrator: C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Tools\Sysmon>sysmon.exe -c sysmonconfig-export.xml

System Monitor v14.16 - System activity monitor
By Mark Russinovich and Thomas Garnier
Copyright (C) 2014-2023 Microsoft Corporation
Using libxml2. libxml2 is Copyright (C) 1998-2012 Daniel Veillard. All Rights Reserved.
Sysinternals - www.sysinternals.com

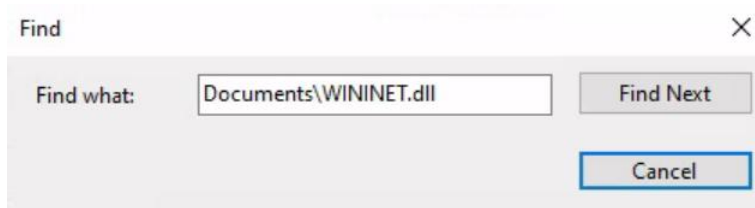
Loading configuration file with schema version 4.50
Sysmon schema version: 4.83
Configuration file validated.
Configuration updated.
```

After the Sysmon configuration is appropriately modified - we will open the event viewer on "Applications and Services" -> "Microsoft" -> "Windows" -> "Sysmon." logs, then we will filter only the EventID=7, which corresponds to module load events:

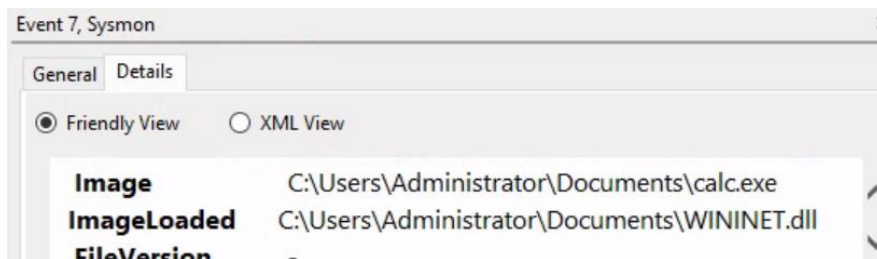


There will be still plenty of results, we need to find the result where the Image is C:\Users\Administrator\Documents\calc.exe and the Image Loaded is C:\Users\Administrator\Documents\WININET.dll.

The best search would be for the value "Documents\WININET.dll", on 'find' option:



When found – let's examine both the image and imageLoaded fields to confirm this is indeed the correct event:



Let's examine the hash field of this event, we look for SHA-256 value:

Hashes MD5=D4990A8D2FF6F2433ACDAD04521F85

->

6,SHA256=51F2305DCF385056C68F7CCF5B1B3B9304

Also, it should be worth noting that the Signed field of the module load is false, indicating invalid module loading.

Hashes: MD5=D4990A8D2FF6F2433
51F2305DCF385056C68F7CCF5B1B3
9FF3C0BBBF95713D517725CEE833
Signed: false
Signature: -
SignatureStatus: Unavailable

Question: Replicate the Unmanaged PowerShell attack described in this section and provide the SHA256 hash of clrjit.dll that spoolsv.exe will load as your answer. "C:\Tools\Sysmon" and "C:\Tools\PSInject" on the spawned target contain everything you need.

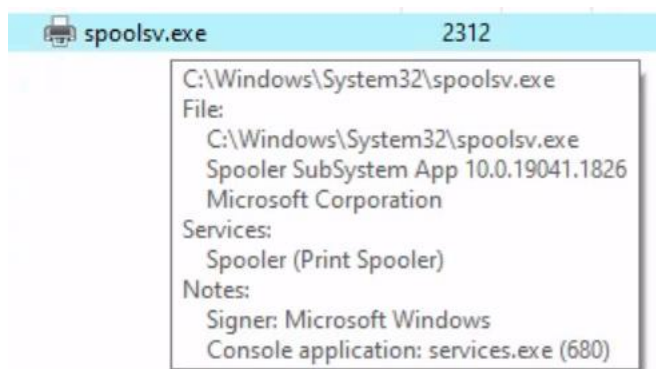
Answer:

8A3CD3CF2249E9971806B15C75A892E6A44CCA5FF5EA5CA89FDA951CD2C09AA9

Method: first – we will modify the sysmonconfig-export.xml just like we did on the previous question.

Then we will open event viewer on the same parameters as last questions (Sysmon logs, filter evenID=7).

Now, we will use tool which called 'process hacker' to observe processes details, We shall examine spoolsv.exe:



We can observe that the processID is 2312.

Then, when hovering the cursor over the process, we see it is unmanaged process.

In order to understand what is unmanaged process – so managed process in principle a process which were written, partially or whole – using C#.

So spoolsv.exe is unmanaged process – it is not built using C#.

Now we will inject powershell script into the process, as powershell is managed process, this procedure should modify the spoolsv.exe to managed process.

For that we shall use a provided script called 'Invoke-PSInject.ps1'.

so we shall open cmd on the script folder and run this script:

```
powershell -ep bypass
```

```
Import-Module .\Invoke-PSInject.ps1
Invoke-PSInject -ProcId [Process ID of spoolsv.exe] -
PoshCode "V3JpdGUtSG9zdCAiSGVsbG8sIEd1cnU5OSEi"
```

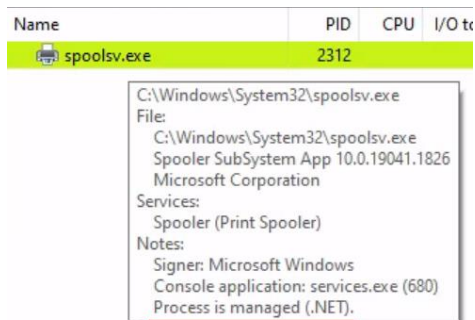
```
Administrator: C:\Windows\System32\cmd.exe - powershell -ep bypass
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Tools\PSInject>powershell -ep bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Tools\PSInject> Import-Module .\Invoke-PSInject.ps1
PS C:\Tools\PSInject> Invoke-PSInject -ProcId 2312 -PoshCode "V3JpdGUtSG9zdCAiSGVsbG8sIEd1cnU5OSEi"
PS C:\Tools\PSInject>
```

Now we gonna close the process hacker, and reopen it, and observe again spoolsv.exe process:



We can observe that now the process is managed.

Now what we need to do is to refresh the event viewer logs, and search for the event of image spoolsv.exe and imageLoaded slrjit.dll

Image	C:\Windows\System32\spoolsv.exe
ImageLoaded	C:\Windows\Microsoft.NET\Framework64 \v4.0.30319\clrjit.dll

Then we will run find on clrjit.dll:



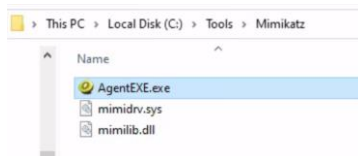
The search will find SHA-256 hash of

SHA256=8A3CD3CF2249E9971806B15C75A892E6A44CCA5FF5EA5CA89FD

Question: Replicate the Credential Dumping attack described in this section and provide the NTLM hash of the Administrator user as your answer.
"C:\Tools\Sysmon" and "C:\Tools\Mimikatz" on the spawned target contain everything you need.

Answer: 5e4ffd54b3849aa720ed39f50185e533

Method: we open the mimikatz program 'AgentExe.exe'



We run the commands according to instructions: a. 'privilege::debug' and then 'sekurlsa::logonpasswords'.

Then all we have to do is to locate the NTLM hash of the Administrator:

```
mimikatz # sekurlsa::logonpasswords
Authentication Id : 0 ; 439858 (00000000:0006b632)
Session          : RemoteInteractive from 2
User Name        : Administrator
Domain           : DESKTOP-NU10MTO
Logon Server      : DESKTOP-NU10MTO
Logon Time        : 6/2/2024 8:10:04 PM
SID               : S-1-5-21-2712802632-2324259492-1677155984-500
msv :
[00000003] Primary
* Username : Administrator
* Domain   : DESKTOP-NU10MTO
→ * NTLM    : 5e4ffd54b3849aa720ed39f50185e533
* SHA1     : e6cd3020bb3da2cd8f02dfeaf5c9f6d50812156b
```

In obtaining that information, mimikatz is doing what is known as 'LSASS dump'.

LSASS- Local Security Authority Subsystem Service is where the windows users credentials are managed.

So in LSASS dump, a process (in our case – mimikatz), is accessing the LSASS dump and retrieving the desired data.

Process accessing another process is of course coresponds to EventID=10 (ProcessAccess), and all suspicious LSASS dump activities would be monitored and detected with EventID=10 filter.

Lets see that on the event viewer:

First – we reconfigure the sysmonconfig-export.xml file, but this time we modify the ProcessAccess field, from 'include' to 'exclude', so its log appear on Sysmon logs on event viewer:

```
<RuleGroup name="" groupRelation="or">
  <ProcessAccess onmatch="exclude">
    <!--NOTE: Using "include" with no rules means nothing in this section will be logged-->
  </ProcessAccess>
</RuleGroup>
```

And as usual – we update the reconfiguration with the following cmd command:

```
sysmon.exe -c sysmonconfig-export.xml
```

now that is done – we filter the Sysmon event viewer with the following xml filter:

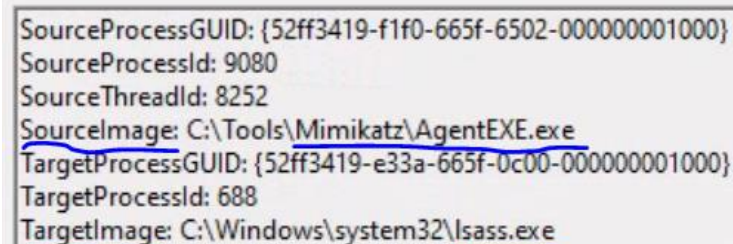
```
<QueryList>
  <Query Id="0" Path="Microsoft-Windows-Sysmon/Operational">
    <Select Path="Microsoft-Windows-Sysmon/Operational">
      *[System[EventID=10]]
      and
      *[EventData[Data[@Name='TargetImage'] and
(Data='C:\Windows\System32\lsass.exe')]]
      and
      *[EventData[Data[@Name='SourceImage'] and
(Data='C:\Tools\Mimikatz\AgentEXE.exe')]]
    </Select>
  </Query>
</QueryList>
```

Where TargetImage is the accessed process field (where we see its lsass.exe) And SourceImage is the accessing process field (Mimikatz\AgentEXE.exe).

When the filter is set – we run the following mimikatz cmd command:

```
AgentExE.exe "privilege::debug" "sekurlsa::logonpasswords"
exit
```

Then refresh the event viewer, and open the first log (each run should generate 3 logs), we get this:



```
SourceProcessGUID: {52ff3419-f1f0-665f-6502-000000001000}
SourceProcessId: 9080
SourceThreadId: 8252
SourceImage: C:\Tools\Mimikatz\AgentEXE.exe
TargetProcessGUID: {52ff3419-e33a-665f-0c00-000000001000}
TargetProcessId: 688
TargetImage: C:\Windows\system32\lsass.exe
```

Demonstrating the lsass dump to obtain credentials.

Additional Telemetry Sources

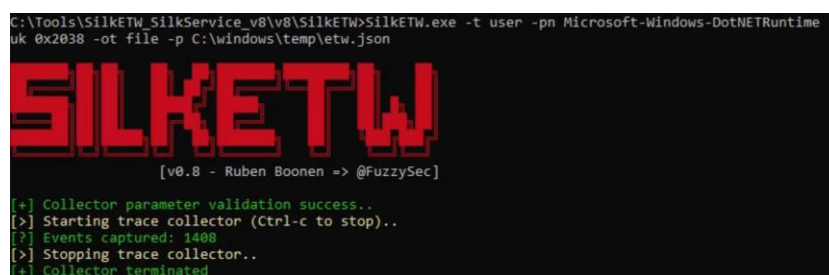
Tapping Into ETW:

Question: Replicate executing Seatbelt and SilkETW as described in this section and provide the ManagedInteropMethodName that starts with "G" and ends with "ion" as your answer. "c:\Tools\SilkETW_SilkService_v8\v8" and "C:\Tools\GhostPack Compiled Binaries" on the spawned target contain everything you need.

Answer: GetTokenInformation

Method: First, as instructed we start SilkETW and run the command:

```
SilkETW.exe -t user -pn Microsoft-Windows-DotNETRuntime -uk 0x2038 -ot file -p C:\windows\temp\etw.json
```



```
C:\Tools\SilkETW_SilkService_v8\v8\SilkETW>SilkETW.exe -t user -pn Microsoft-Windows-DotNETRuntime -uk 0x2038 -ot file -p C:\windows\temp\etw.json

SILKETW

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

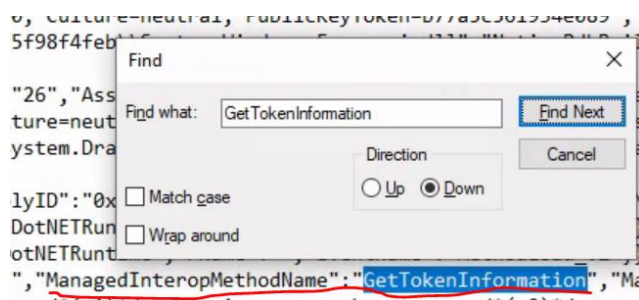
[+] Collector parameter validation success..
[>] Starting trace collector (Ctrl-c to stop)..
[?] Events captured: 1408
[>] Stopping trace collector..
[+] Collector terminated
```

then – **while** silkETW is running, we run seatbelt.exe with the command

```
Seatbelt.exe -group=all
```

(* the output of this command is long and the process takes a while so no picture here. ** approximate runtime: 100 seconds).

After the seatbelt execution has ended (or stopped), we stop the silkETW, then we open the output etw.json file, in there we search for 'ManagedInteropMethodName' keyword – after few tries the answer should appear somewhere within the json file:



* it may be possible more than one possible answer may appear, so some trial and error may be required until the current answer is found. Also it may be required to do the process more than once.

Analyzing Windows Event Logs En Masse

Get-WinEvent:

Question: Utilize the Get-WinEvent cmdlet to traverse all event logs located within the "C:\Tools\chainsaw\EVTX-ATTACK-SAMPLES\Lateral Movement" directory and determine when the *\PRINT share was added. Enter the time of the identified event in the format HH:MM:SS as your answer.

Answer: 12:30:30

Method: we will run the powershell script:

```
# Define the directory containing the .evtx files
$directory = "C:\Tools\chainsaw\EVTX-ATTACK-SAMPLES\Lateral Movement"

# Get all .evtx files in the directory
$evtxFiles = Get-ChildItem -Path $directory -Filter *.evtx

# Loop through each file and search for the event
foreach ($file in $evtxFiles) {
    $events = Get-WinEvent -Path $file.FullName

    foreach ($event in $events) {
        # Check if the event properties contain the share name
        foreach ($property in $event.Properties) {
            if ($property.Value -like "\\*\PRINT*") {
                # Output the time of the identified event in HH:MM:SS format
                $eventTime = $event.TimeCreated.ToString("HH:mm:ss")
                Write-Output "File: $($file.Name) - Time: $eventTime"
            }
        }
    }
}
```

on powershell:

```
PS C:\Users\Administrator> # Define the directory containing the .evtx files
$directory = "C:\Tools\chainsaw\EVTX-ATTACK-SAMPLES\Lateral Movement"

# Get all .evtx files in the directory
$evtxFiles = Get-ChildItem -Path $directory -Filter *.evtx

# Loop through each file and search for the event
foreach ($file in $evtxFiles) {
    $events = Get-WinEvent -Path $file.FullName

    foreach ($event in $events) {
        # Check if the event properties contain the share name
        foreach ($property in $event.Properties) {
            if ($property.Value -like "\\*\PRINT*") {
                # Output the time of the identified event in HH:MM:SS format
                $eventTime = $event.TimeCreated.ToString("HH:mm:ss")
                Write-Output "File: $($file.Name) - Time: $eventTime"
            }
        }
    }
}

File: net_share_drive_5142.evtx - Time: 12:30:30
```


Skills Assessment

Skills Assessment:

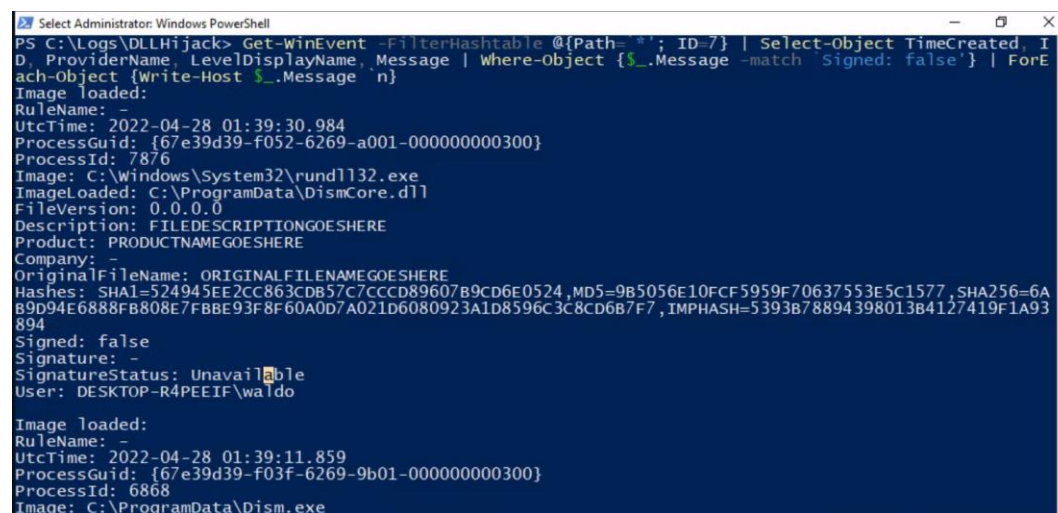
Question: By examining the logs located in the "C:\Logs\DLLHijack" directory, determine the process responsible for executing a DLL hijacking attack. Enter the process name as your answer. Answer format: _.exe

Answer: Dism.exe

Method: First, we modify the configuration file of sysmonconfig-export.xml (just like previous times). Now:

Method 1: run the PowerShell script on the C:\Logs\DLLHijack directory that containing the DLLHijack.evtx file:

```
Get-WinEvent -FilterHashtable @{Path='*'; ID=7} | Select-Object TimeCreated, ID, ProviderName, LevelDisplayName, Message | Where-Object {$_.Message -match 'Signed: false'} | ForEach-Object {Write-Host $_.Message `n}
```

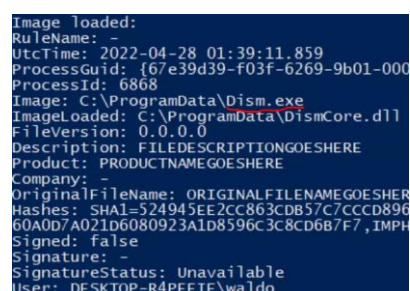


```
Select Administrator: Windows PowerShell
PS C:\Logs\DLLHijack> Get-WinEvent -FilterHashtable @{Path='*'; ID=7} | Select-Object TimeCreated, ID, ProviderName, LevelDisplayName, Message | Where-Object {$_.Message -match 'Signed: false'} | ForEach-Object {Write-Host $_.Message `n}
Image Loaded:
RuleName: -
UtcTime: 2022-04-28 01:39:30.984
ProcessGuid: {67e39d39-f052-6269-a001-000000000300}
ProcessId: 7876
Image: C:\windows\System32\rundll32.exe
ImageLoaded: C:\ProgramData\DismCore.dll
FileVersion: 0.0.0.0
Description: FILEDESCRIPTIONGoesHere
Product: PRODUCTNAMEGoesHere
Company: -
OriginalFileName: ORIGINALFILENAMEGoesHere
Hashes: SHA1=524945EE2CC863CDB57C7CCCD89607B9CD6E0524,MD5=9B5056E10FCF5959F70637553E5C1577,SHA256=6A89D94E6888FB808E7FB8E93F8F60A0D7A021D6080923A1D8596C3C8CD6B7F7,IMPHASH=5393B78894398013B4127419F1A93894
Signed: false
Signature: -
SignatureStatus: Unavailable
User: DESKTOP-R4PEEIF\waldo

Image Loaded:
RuleName: -
UtcTime: 2022-04-28 01:39:11.859
ProcessGuid: {67e39d39-f03f-6269-9b01-000000000300}
ProcessId: 6868
Image: C:\ProgramData\Dism.exe
ImageLoaded: C:\ProgramData\DismCore.dll
FileVersion: 0.0.0.0
Description: FILEDESCRIPTIONGoesHere
Product: PRODUCTNAMEGoesHere
Company: -
OriginalFileName: ORIGINALFILENAMEGoesHere
Hashes: SHA1=524945EE2CC863CDB57C7CCCD89607B9CD6E0524,MD5=9B5056E10FCF5959F70637553E5C1577,SHA256=6A89D94E6888FB808E7FB8E93F8F60A0D7A021D6080923A1D8596C3C8CD6B7F7,IMPHASH=5393B78894398013B4127419F1A93894
Signed: false
Signature: -
SignatureStatus: Unavailable
User: DESKTOP-R4PEEIF\waldo
```

- *a. make sure that the coloring of the command are as shown in the picture, else it might not work.
- b. there will be 2 possible options, trial and error might be required to see the correct process.

Now, look for the process name in the Image field:



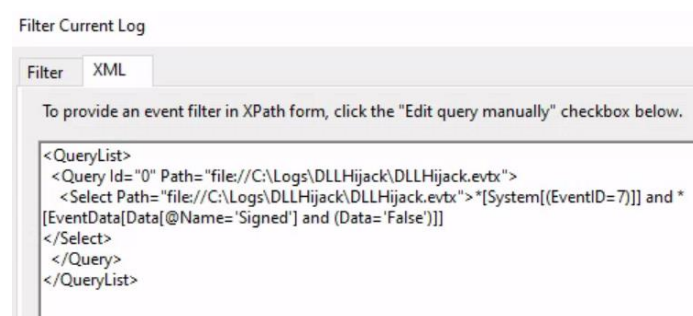
```
Image Loaded:
RuleName: -
UtcTime: 2022-04-28 01:39:11.859
ProcessGuid: {67e39d39-f03f-6269-9b01-000000000300}
ProcessId: 6868
Image: C:\ProgramData\Dism.exe
ImageLoaded: C:\ProgramData\DismCore.dll
FileVersion: 0.0.0.0
Description: FILEDESCRIPTIONGoesHere
Product: PRODUCTNAMEGoesHere
Company: -
OriginalFileName: ORIGINALFILENAMEGoesHere
Hashes: SHA1=524945EE2CC863CDB57C7CCCD89607B9CD6E0524,MD5=9B5056E10FCF5959F70637553E5C1577,SHA256=6A89D94E6888FB808E7FB8E93F8F60A0D7A021D6080923A1D8596C3C8CD6B7F7,IMPHASH=5393B78894398013B4127419F1A93894
Signed: false
Signature: -
SignatureStatus: Unavailable
User: DESKTOP-R4PEEIF\waldo
```


In our case it will be Dism.exe, also pay attention that the Signed=False, indicating the DLL was not injected 'appropriately'.

What the PowerShell script does is look for unsigned DLL injection from all module loads (EventID=7) logs.

Method 2: run xml filter on the event viewer:

```
<QueryList>
  <Query Id="0" Path="file://C:\Logs\DLLHijack\DLLHijack.evtx">
    <Select Path="file://C:\Logs\DLLHijack\DLLHijack.evtx">*[System[(EventID=7)]]
and *[EventData[Data[@Name='Signed'] and (Data='False')]]
    </Select>
  </Query>
</QueryList>
```



That leaves 2 with 2 logs:

Filtered: Advanced filter, click on "Filter" command to view filter configuration.. Number of events: 2				
Level	Date and Time	Source	Event ID	Task Category
Information	4/27/2022 6:39:30 PM	Sysmon	7	Image loaded...
Information	4/27/2022 6:39:11 PM	Sysmon	7	Image loaded...

Inspect the second log, for 'image' field value for the answer:

DLLHijack1 Number of events: 5,772

Filtered: Advanced filter, click on "Filter" command to view filter configuration.. Number of events: 2

Level	Date and Time	Source	Event ID	Task Category
Information	4/27/2022 6:39:30 PM	Sysmon	7	Image loaded...
Information	4/27/2022 6:39:11 PM	Sysmon	7	Image loaded...

Event 7, Sysmon

General Details

☒ Friendly View ☐ XML View

UtcTime	2022-04-28 01:39:11.859
ProcessGuid	{67e39d39-f03f-6269-9b01-000000000300}
ProcessId	6868
Image	C:\ProgramData\Dism.exe
ImageLoaded	C:\ProgramData\DismCore.dll

Question: By examining the logs located in the "C:\Logs\PowershellExec" directory, determine the process that executed unmanaged PowerShell code. Enter the process name as your answer. Answer format: _.exe

Answer: Calculator.exe

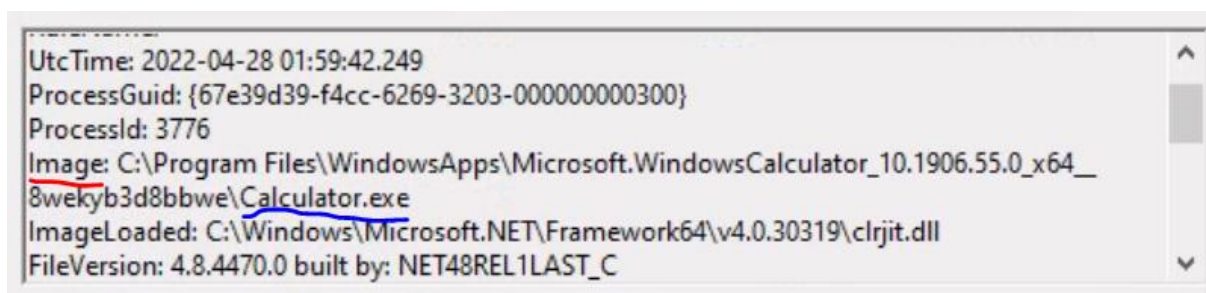
Method: first – we will modify the sysmonconfig-export.xml just like we did on the previous question.

Then we will open event viewer on the same parameters as last questions (Sysmon logs, filter evenID=7).

Then we will search by the keyword '.NET', as we are looking .NET made dll:



When a result found – we will inspect its image field:

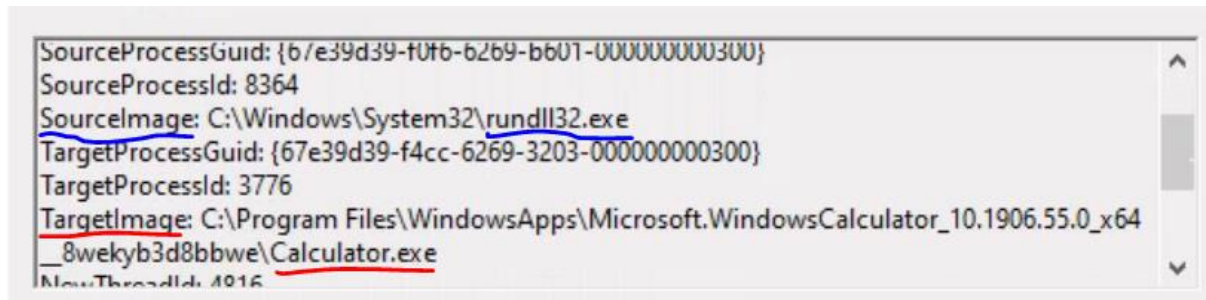


And get the result.

Question: By examining the logs located in the "C:\Logs\PowershellExec" directory, determine the process that injected into the process that executed unmanaged PowerShell code. Enter the process name as your answer. Answer format: _.exe

Answer: rundll32.exe

Method: change the filter for EventID=8 filter (CreateRemoteThread), to display, and search for the answer of the last question- 'Calculator.exe'



pay attention that the SourceImage is rundll32.exe, and the TargetImage is Calculator.exe, meaning rundll32.exe is the parent process of Calculator.exe, meaning that rundll32.exe is the injecting process.

Question: By examining the logs located in the "C:\Logs\Dump" directory, determine the process that performed an LSASS dump. Enter the process name as your answer. Answer format: _.exe

Answer: ProcessHacker.exe

Method: as this is a question of LSASS dump – we will need to deal with the corresponding EventID=10 filter for process creating.

first – we will modify the sysmonconfig-export.xml (for ProcessAccess field) just like last time.

Then we will open 'LsassDump.evtx' event viewer.

We will use the following xml filter:

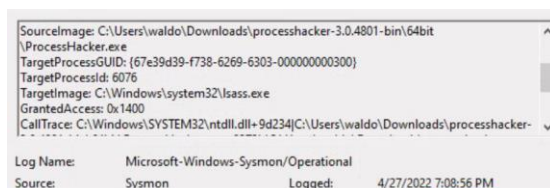
```
<QueryList>
  <Query Id="0" Path="file://C:\Logs\Dump\LsassDump.evtx">
    <Select Path="file://C:\Logs\Dump\LsassDump.evtx">
      *[System[EventID=10]]
      and
      *[EventData[Data[@Name='TargetImage'] and
(Data='C:\Windows\System32\lsass.exe')]]
    </Select>
  </Query>
</QueryList>
```

There would be several possible options, so we check for the field of 'SourceImage' values in the logs until we find the correct solution.

Question: By examining the logs located in the "C:\Logs\Dump" directory, determine if an ill-intended login took place after the LSASS dump. Answer format: Yes or No

Answer: No

Method: the LSASS dump took place in 19:08:56



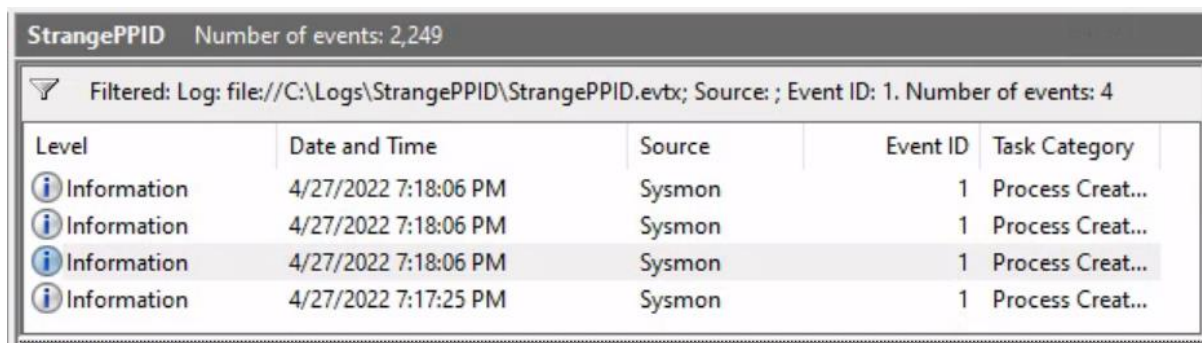
And no events indicating login were found after the mentioned time (according to ChatGPT (the parameters that were looked are for the field Image – the values cmd and powershell, and eventID=1 (process creation)).

Question: By examining the logs located in the "C:\Logs\StrangePPID" directory, determine a process that was used to temporarily execute code based on a strange parent-child relationship. Enter the process name as your answer. Answer format: `_.exe`

Answer: WerFault.exe

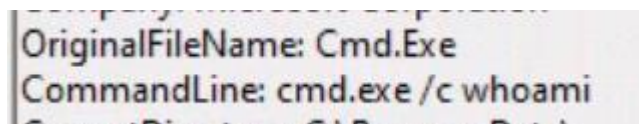
Method: First, we filter the log on EventID=1 (process creation).

That leaves us with 4 options:



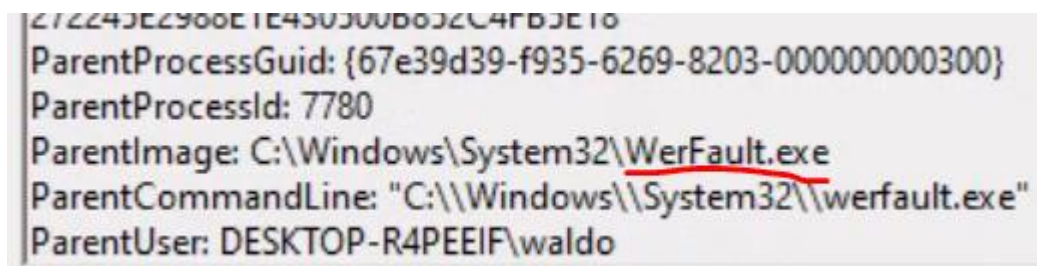
Level	Date and Time	Source	Event ID	Task Category
Information	4/27/2022 7:18:06 PM	Sysmon	1	Process Creat...
Information	4/27/2022 7:18:06 PM	Sysmon	1	Process Creat...
Information	4/27/2022 7:18:06 PM	Sysmon	1	Process Creat...
Information	4/27/2022 7:17:25 PM	Sysmon	1	Process Creat...

In one of these options we see a cmd were run, and the command 'whoami' were executed:



OriginalFileName: Cmd.Exe
CommandLine: cmd.exe /c whoami

All we have to do is to find the parent image that created the cmd process and his parent:



ParentProcessGuid: {67e39d39-f935-6269-8203-000000000300}
ParentProcessId: 7780
ParentImage: C:\Windows\System32\WerFault.exe
ParentCommandLine: "C:\\Windows\\System32\\werfault.exe"
ParentUser: DESKTOP-R4PEEIF\\waldo