Developing Sigma Rules

In this section, we'll cover manual Sigma rule development.

Let's now navigate to the bottom of this section and click on "Click here to spawn the target system!". Then, let's RDP into the Target IP using the provided credentials. The vast majority of the actions/commands covered from this point up to end of this section can be replicated inside the target, offering a more comprehensive grasp of the topics presented.

Manually Developing a Sigma Rule

Example 1: LSASS Credential Dumping

Let's dive into the world of Sigma rules using a sample named `shell.exe` (a renamed version of mimikatz) residing in the `C:\Samples\YARASigma` directory of this section's target as an illustration. We want to understand the process behind crafting a Sigma rule, so let's get our hands dirty.

After executing `shell.exe` as follows, we collected the most critical events and saved them as `lab_events.evtx` inside the `C:\Events\YARASigma` directory of this section's target.

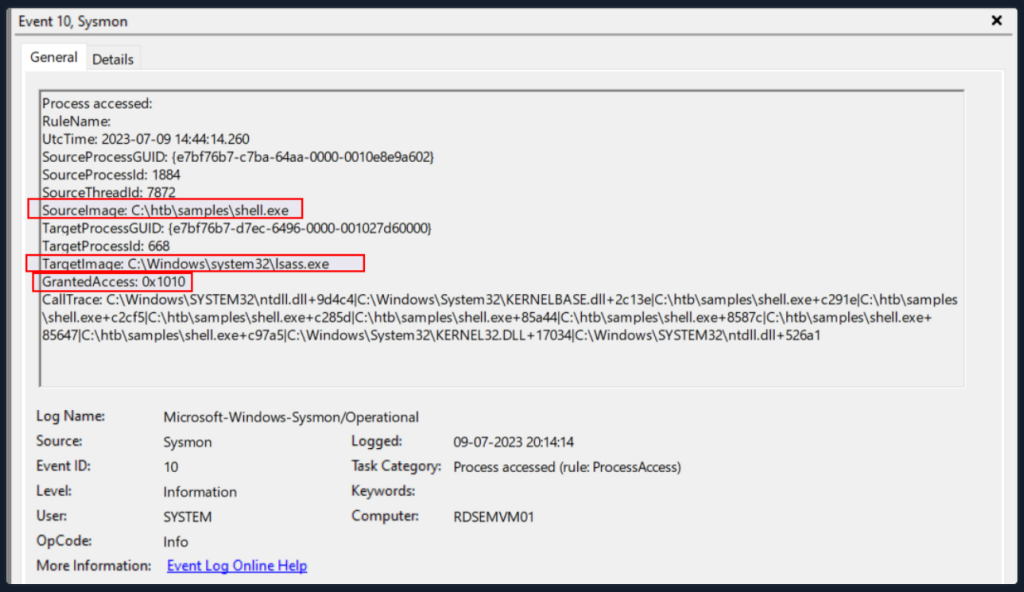
The process created by `shell.exe` (mimikatz) will try to access the process memory of `lsass.exe`. The system monitoring tool Sysmon was running in the background and captured this activity in the event logs (Event ID 10).

Developing Sigma Rules

```
C:\Samples\YARASigma>shell.exe

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX            ( vincent.letoux@gmail.com )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz # privilege::debug
Privilege '20' OK

mimikatz # sekurlsa::logonpasswords
---SNIP---
Authentication Id : 0 ; 100080 (00000000:000186f0)
Session           : Interactive from 1
User Name         : htb-student
Domain            : DESKTOP-VJF8GH8
Logon Server      : DESKTOP-VJF8GH8
Logon Time        : 8/25/2023 2:17:20 PM
SID               : S-1-5-21-1412399592-1502967738-1150298762-1001
        msv :
         [00000003] Primary
         * Username : htb-student
         * Domain   : .
         * NTLM     : 3c0e5d303ec84884ad5c3b7876a06ea6
         * SHA1     : b2978f9abc2f356e45cb66ec39510b1ccca08a0e
        tspkg :
        wdigest :
         * Username : htb-student
         * Domain   : DESKTOP-VJF8GH8
         * Password : (null)
        kerberos :
         * Username : htb-student
         * Domain   : DESKTOP-VJF8GH8
         * Password : (null)
        ssp :
        credman :
```

```
        cloudap :

Authentication Id : 0 ; 100004 (00000000:000186a4)
Session           : Interactive from 1
User Name         : htb-student
Domain            : DESKTOP-VJF8GH8
Logon Server      : DESKTOP-VJF8GH8
Logon Time        : 8/25/2023 2:17:20 PM
SID               : S-1-5-21-1412399592-1502967738-1150298762-1001
        msv :
         [00000003] Primary
         * Username : htb-student
         * Domain   : .
         * NTLM     : 3c0e5d303ec84884ad5c3b7876a06ea6
         * SHA1     : b2978f9abc2f356e45cb66ec39510b1ccca08a0e
        tspkg :
        wdigest :
         * Username : htb-student
         * Domain   : DESKTOP-VJF8GH8
         * Password : (null)
        kerberos :
         * Username : htb-student
         * Domain   : DESKTOP-VJF8GH8
         * Password : HTB_@cademy_stdnt!
        ssp :
        credman :
        cloudap :
---SNIP---
```

First off, Sysmon `Event ID 10` is triggered when a process accesses another process, and it logs the permission flags in the `GrantedAccess` field. This event log contains two important fields, `TargetImage` and `GrantedAccess`. In a typical LSASS memory dumping scenario, the malicious process needs specific permissions to access the memory space of the LSASS process. These permissions are often read/write access, among other things.



Now, why is `0x1010` crucial here? This hexadecimal flag essentially combines `PROCESS_VM_READ (0x0010)` and `PROCESS_QUERY_INFORMATION (0x0400)` permissions. To translate that: the process is asking for read access to the virtual memory of LSASS and the ability to query certain information from the process. While `0x0410` is the most common GrantedAccess flag used for reading LSASS memory, `0x1010` implies both reading and querying information from the process and is also frequently observed during credential dumping attacks.

So how can we weaponize this information for detection? Well, in our security monitoring stack, we would configure Sysmon to flag or alert on any `Event ID 10` where the `TargetImage` is `lsass.exe` and `GrantedAccess` is set to `0x1010`.

A Sigma rule that checks for the abovementioned conditions can be found below.

Code: yaml

```yaml
title: LSASS Access with rare GrantedAccess flag
status: experimental
description: This rule will detect when a process tries to access LSASS memory with suspicious acce
```

```yaml
description: This rule will detect when a process tries to access LSASS memory with suspicious acce
date: 2023/07/08
tags:
    - attack.credential_access
    - attack.t1003.001
logsource:
    category: process_access
    product: windows
detection:
    selection:
        TargetImage|endswith: '\lsass.exe'
        GrantedAccess|endswith: '0x1010'
    condition: selection
```

Sigma Rule Breakdown

- `title`: This title offers a concise overview of the rule's objective, specifically aimed at detecting interactions with LSASS memory involving a particular access flag.

Code: yaml

```yaml
title: LSASS Access with rare GrantedAccess flag
```

- `status`: This field signals that the rule is in the testing phase, suggesting that additional fine-tuning or validation may be necessary.

Code: yaml

```yaml
status: experimental
```

- `description`: Rule description.

Code: yaml

```yaml
description: This rule will detect when a process tries to access LSASS memory with suspicious acce
```

- `date`: This field marks the date when the rule was either updated or originally created.

Code: yaml

```yaml
date: 2023/07/08
```

- `tags`: The rule is tagged with `attack.credential_access` and `attack.t1003.001`. These tags help categorize the rule based on known attack techniques or tactics related to credential access.

Code: yaml

```yaml
tags:
    - attack.credential_access
    - attack.t1003.001`
```

- `logsource`: The logsource specifies the log source that the rule is intended to analyze. It contains `category` as `process_access` which indicates that the rule focuses on log events related to process access (`Sysmon Event ID 10`, if we use Sigma's default config files). Also, `product: windows` specifies that the rule is specifically designed for Windows operating systems.

Code: yaml

```yaml
logsource:
    category: process_access
    product: windows
```

- `detection`: The detection section defines the conditions that must be met for the rule to trigger an alert. The selection part specifies the criteria for selecting relevant log events where the `TargetImage` field ends with `\lsass.exe` and `GrantedAccess` field ends with the hexadecimal value `0x1010`. The `GrantedAccess` field represents the access rights or permissions associated with the process. In this case, it targets events with a specific access flag of `0x1010`. Finally, the condition part specifies that the selection criteria must be met for the rule to trigger an alert. In this case, both the `TargetImage` and `GrantedAccess` criteria must be met.

Code: yaml

```yaml
detection:
    selection:
        TargetImage|endswith: '\lsass.exe'
        GrantedAccess|endswith: '0x1010'
    condition: selection
```

Our first Sigma rule above can be found inside the `C:\Rules\sigma` directory of this section's target as `proc_access_win_lsass_access.yml`. Let's explore the `sigmac` tool that can help us transform this rule into queries or configurations compatible with a multitude of SIEMs, log management solutions, and other security analytics tools.

The `sigmac` tool can be found inside the `C:\Tools\sigma-0.21\tools` directory of this section's target.

Suppose that we wanted to convert our Sigma rule into a PowerShell (`Get-WinEvent`) query. This could have been accomplished with the help of `sigmac` as follows.

Developing Sigma Rules

```
PS C:\Tools\sigma-0.21\tools> python sigmac -t powershell 'C:\Rules\sigma\proc_access_win_lsass_acc
Get-WinEvent | where {($_.ID -eq "10" -and $_.message -match "TargetImage.*.*\\lsass.exe" -and $_.m
```

Let's adjust the Get-WinEvent query above by specifying the .evtx file that is related to LSASS access by another process (`lab_events.evtx` inside the `C:\Events\YARASigma` directory of this section's target) and see if it will identify the Sysmon event (`ID 10`) that we analyzed at the beginning of this section.

Note: Please open a PowerShell terminal as administrator to run the query.

Developing Sigma Rules

```
PS C:\Tools\sigma-0.21\tools> Get-WinEvent -Path C:\Events\YARASigma\lab_events.evtx | where {($_.I


TimeCreated : 7/9/2023 7:44:14 AM
Id          : 10
RecordId    : 7810
ProcessId   : 3324
MachineName : RDSEMVM01
Message     : Process accessed:
              RuleName:
              UtcTime: 2023-07-09 14:44:14.260
              SourceProcessGUID: {e7bf76b7-c7ba-64aa-0000-0010e8e9a602}
              SourceProcessId: 1884
              SourceThreadId: 7872
              SourceImage: C:\htb\samples\shell.exe
              TargetProcessGUID: {e7bf76b7-d7ec-6496-0000-001027d60000}
              TargetProcessId: 668
              TargetImage: C:\Windows\system32\lsass.exe
              GrantedAccess: 0x1010
              CallTrace: C:\Windows\SYSTEM32\ntdll.dll+9d4c4|C:\Windows\System32\KERNELBASE.dll+2c1
```

```
ell.exe+c291e|C:\htb\samples\shell.exe+c2cf5|C:\htb\samples\shell.exe+c285d|C:\htb\sa
4|C:\htb\samples\shell.exe+8587c|C:\htb\samples\shell.exe+85647|C:\htb\samples\shell.
\System32\KERNEL32.DLL+17034|C:\Windows\SYSTEM32\ntdll.dll+526a1
SourceUser: %12
TargetUser: %13
```

The related Sysmon event with ID 10 is successfully identified!

But let's not stop there - remember, false positives are the enemy of effective security monitoring.

- We should also cross-reference the `SourceImage` (the process initiating the access) against a list of known, safe processes that commonly interact with LSASS.
- If we see an unfamiliar or unusual process trying to read LSASS with a `GrantedAccess` that ends with `10`, `30`, `50`, `70`, `90`, `B0`, `D0`, `F0`, `18`, `38`, `58`, `78`, `98`, `B8`, `D8`, `F8`, `1A`, `3A`, `5A`, `7A`, `9A`, `BA`, `DA`, `FA`, `0x14C2`, and `FF` (these suffixes come from studying the `GrantedAccess` values that various LSASS credential dumping techniques require), that's a red flag, and our incident response protocol should kick in.
- Especially, if the `SourceImage` resides in suspicious paths containing, `\Temp\`, `\Users\Public\`, `\PerfLogs\`, `\AppData\`, `\htb\` etc. that's another red flag, and our incident response protocol should kick in.

A more robust version of the Sigma rule we created taking the above points into consideration can be found inside the `C:\Rules\sigma` directory of this section's target as `proc_access_win_lsass_access_robust.yml`

Code: yaml

```yaml
title: LSASS Access From Program in Potentially Suspicious Folder
id: fa34b441-961a-42fa-a100-ecc28c886725
status: experimental
description: Detects process access to LSASS memory with suspicious access flags and from a potenti
references:
    - https://docs.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights
    - https://onedrive.live.com/view.aspx?resid=D026B4699190F1E6!2843&ithint=file%2cpptx&app=PowerP
    - https://web.archive.org/web/20230208123920/https://cyberwardog.blogspot.com/2017/03/chronicle
    - https://www.slideshare.net/heirhabarov/hunting-for-credentials-dumping-in-windows-environment
    - http://security-research.dyndns.org/pub/slides/FIRST2017/FIRST-2017_Tom-Ueltschi_Sysmon_FINAL
author: Florian Roth (Nextron Systems)
date: 2021/11/27
modified: 2023/05/05
tags:
    - attack.credential_access
    - attack.t1003.001
    - attack.s0002
logsource:
    category: process_access
    product: windows
detection:
    selection:
        TargetImage|endswith: '\lsass.exe'
        GrantedAccess|endswith:
            - '10'
            - '30'
            - '50'
            - '70'
            - '90'
            - 'B0'
            - 'D0'
            - 'F0'
            - '18'
            - '38'
            - '58'
            - '78'
            - '98'
            - 'B8'
            - 'D8'
```

```
            - 'F8'
            - '1A'
            - '3A'
            - '5A'
            - '7A'
            - '9A'
            - 'BA'
            - 'DA'
            - 'FA'
            - '0x14C2'  # https://github.com/b4rtik/ATPMiniDump/blob/76304f93b390af3bb66e4f451ca165
            - 'FF'
        SourceImage|contains:
            - '\Temp\'
            - '\Users\Public\'
            - '\PerfLogs\'
            - '\AppData\'
            - '\htb\'
    filter_optional_generic_appdata:
        SourceImage|startswith: 'C:\Users\'
        SourceImage|contains: '\AppData\Local\'
        SourceImage|endswith:
            - '\Microsoft VS Code\Code.exe'
            - '\software_reporter_tool.exe'
            - '\DropboxUpdate.exe'
            - '\MBAMInstallerService.exe'
            - '\WebexMTA.exe'
            - '\WebEx\WebexHost.exe'
            - '\JetBrains\Toolbox\bin\jetbrains-toolbox.exe'
        GrantedAccess: '0x410'
    filter_optional_dropbox_1:
        SourceImage|startswith: 'C:\Windows\Temp\'
        SourceImage|endswith: '.tmp\DropboxUpdate.exe'
        GrantedAccess:
            - '0x410'
            - '0x1410'
    filter_optional_dropbox_2:
        SourceImage|startswith: 'C:\Users\'
        SourceImage|contains: '\AppData\Local\Temp\'
        SourceImage|endswith: '.tmp\DropboxUpdate.exe'
        GrantedAccess: '0x1410'
    filter_optional_dropbox_3:
        SourceImage|startswith:
            - 'C:\Program Files (x86)\Dropbox\'
            - 'C:\Program Files\Dropbox\'
        SourceImage|endswith: '\DropboxUpdate.exe'
        GrantedAccess: '0x1410'
    filter_optional_nextron:
        SourceImage|startswith:
            - 'C:\Windows\Temp\asgard2-agent\'
            - 'C:\Windows\Temp\asgard2-agent-sc\'
        SourceImage|endswith:
            - '\thor64.exe'
            - '\thor.exe'
            - '\aurora-agent-64.exe'
            - '\aurora-agent.exe'
        GrantedAccess:
            - '0x1fffff'
            - '0x1010'
            - '0x101010'
    filter_optional_ms_products:
        SourceImage|startswith: 'C:\Users\'
        SourceImage|contains|all:
            - '\AppData\Local\Temp\'
            - '\vs_bootstrapper_'
        GrantedAccess: '0x1410'
    filter_optional_chrome_update:
        SourceImage|startswith: 'C:\Program Files (x86)\Google\Temp\'
        SourceImage|endswith: '.tmp\GoogleUpdate.exe'
        GrantedAccess:
            - '0x410'
            - '0x1410'
```

```
        - 0x1410
    filter_optional_keybase:
        SourceImage|startswith: 'C:\Users\'
        SourceImage|endswith: \AppData\Local\Keybase\keybase.exe
        GrantedAccess: '0x1fffff'
    filter_optional_avira:
        SourceImage|contains: '\AppData\Local\Temp\is-'
        SourceImage|endswith: '.tmp\avira_system_speedup.tmp'
        GrantedAccess: '0x1410'
    filter_optional_viberpc_updater:
        SourceImage|startswith: 'C:\Users\'
        SourceImage|contains: '\AppData\Roaming\ViberPC\'
        SourceImage|endswith: '\updater.exe'
        TargetImage|endswith: '\winlogon.exe'
        GrantedAccess: '0x1fffff'
    filter_optional_adobe_arm_helper:
        SourceImage|startswith:  # Example path: 'C:\Program Files (x86)\Common Files\Adobe\ARM\1.0
            - 'C:\Program Files\Common Files\Adobe\ARM\'
            - 'C:\Program Files (x86)\Common Files\Adobe\ARM\'
        SourceImage|endswith: '\AdobeARMHelper.exe'
        GrantedAccess: '0x1410'
    condition: selection and not 1 of filter_optional_*
fields:
    - User
    - SourceImage
    - GrantedAccess
falsepositives:
    - Updaters and installers are typical false positives. Apply custom filters depending on your e
level: medium
```

Notice how the condition filters out false positives (selection `and not 1 of filter_optional_*`).

Example 2: Multiple Failed Logins From Single Source (Based on Event 4776)

According to Microsoft, Event 4776 generates every time that a credential validation occurs using NTLM authentication.

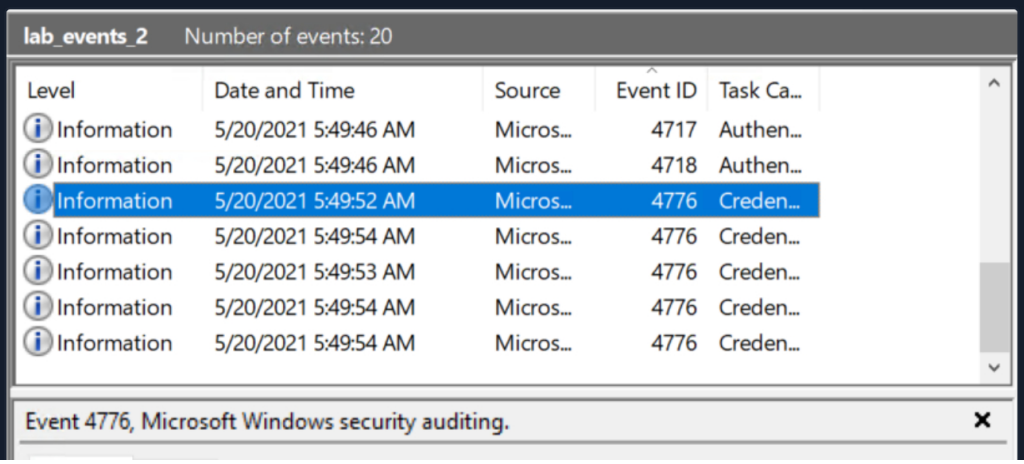
This event occurs only on the computer that is authoritative for the provided credentials. For domain accounts, the domain controller is authoritative. For local accounts, the local computer is authoritative.

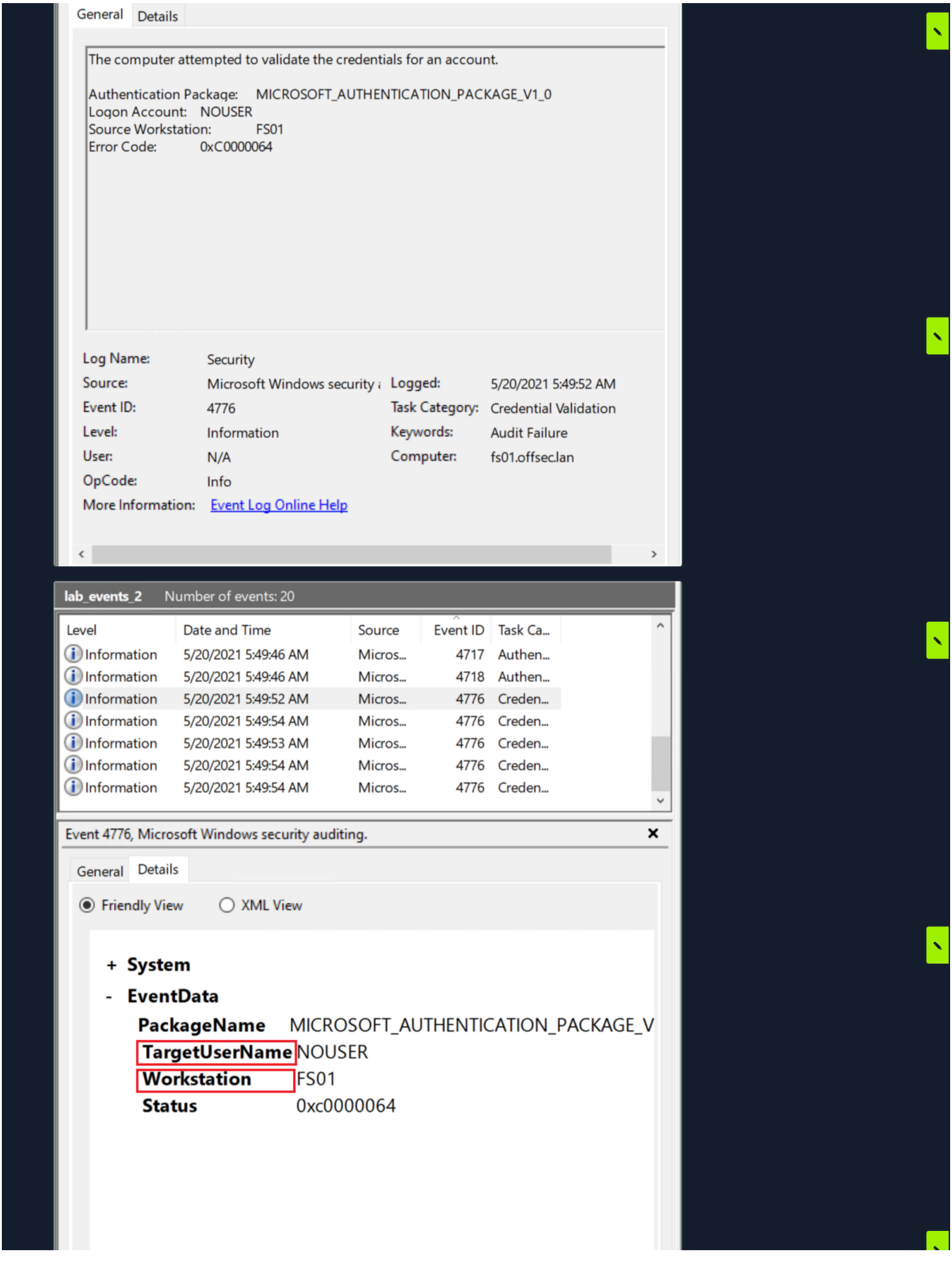
It shows successful and unsuccessful credential validation attempts.

It shows only the computer name (`Source Workstation`) from which the authentication attempt was performed (authentication source). For example, if you authenticate from CLIENT-1 to SERVER-1 using a domain account you'll see CLIENT-1 in the `Source Workstation` field. Information about the destination computer (SERVER-1) isn't presented in this event.

If a credential validation attempt fails, you'll see a Failure event with Error Code parameter value not equal to `0x0`.

`lab_events_2.evtx` inside the `C:\Events\YARASigma` directory of this section's target contains events related to multiple failed login attempts against `NOUSER` (thanks to mdecrevoisier).




General
Details
The computer attempted to validate the credentials for an account.
Authentication Package: MICROSOFT_AUTHENTICATION_PACKAGE_V1_0
Logon Account: NOUSER
Source Workstation: FS01
Error Code: 0xC0000064
Log Name: Security
Source: Microsoft Windows security
Logged: 5/20/2021 5:49:52 AM
Event ID: 4776
Task Category: Credential Validation
Level: Information
Keywords: Audit Failure
User: N/A
Computer: fs01.offsec.lan
OpCode: Info
More Information: Event Log Online Help
lab_events_2
Number of events: 20
Level
Date and Time
Source
Event ID
Task Ca...
Information 5/20/2021 5:49:46 AM Micros... 4717 Authen...
Information 5/20/2021 5:49:46 AM Micros... 4718 Authen...
Information 5/20/2021 5:49:52 AM Micros... 4776 Creden...
Information 5/20/2021 5:49:54 AM Micros... 4776 Creden...
Information 5/20/2021 5:49:53 AM Micros... 4776 Creden...
Information 5/20/2021 5:49:54 AM Micros... 4776 Creden...
Information 5/20/2021 5:49:54 AM Micros... 4776 Creden...
Event 4776, Microsoft Windows security auditing.
General
Details
Friendly View
XML View
+ System
- EventData
PackageName MICROSOFT_AUTHENTICATION_PACKAGE_V
TargetUserName NOUSER
Workstation FS01
Status 0xc0000064

A valid Sigma rule to detect multiple failed login attempts originating from the same source can be found inside the `C:\Rules\sigma` directory of this section's target, saved as `win_security_susp_failed_logons_single_source2.yml`

Code: yaml

```yaml
title: Failed NTLM Logins with Different Accounts from Single Source System
id: 6309ffc4-8fa2-47cf-96b8-a2f72e58e538
related:
    - id: e98374a6-e2d9-4076-9b5c-11bdb2569995
      type: derived
status: unsupported
description: Detects suspicious failed logins with different user accounts from a single source sys
author: Florian Roth (Nextron Systems)
date: 2017/01/10
modified: 2023/02/24
tags:
    - attack.persistence
    - attack.privilege_escalation
    - attack.t1078
logsource:
    product: windows
    service: security
detection:
    selection2:
        EventID: 4776
        TargetUserName: '*'
        Workstation: '*'
    condition: selection2 | count(TargetUserName) by Workstation > 3
falsepositives:
    - Terminal servers
    - Jump servers
    - Other multiuser systems like Citrix server farms
    - Workstations with frequently changing users
level: medium
```

Sigma Rule Breakdown:

- `logsource`: This section specifies that the rule is intended for Windows systems (`product: windows`) and focuses only on `Security` event logs (`service: security`).

Code: yaml

```yaml
logsource:
    product: windows
    service: security
```

- `detection: selection2` is essentially the filter. It's looking for logs with EventID `4776` (`EventID: 4776`) regardless of the `TargetUserName` or `Workstation` values (`TargetUserName: '*'`, `Workstation: '*'`). `condition` counts instances of `TargetUserName` grouped by `Workstation` and checks if a workstation has more than `three` failed login attempts.

Sigma Rule Development Resources

As you can imagine, the best Sigma rule development resource is the official documentation, which can be found at the

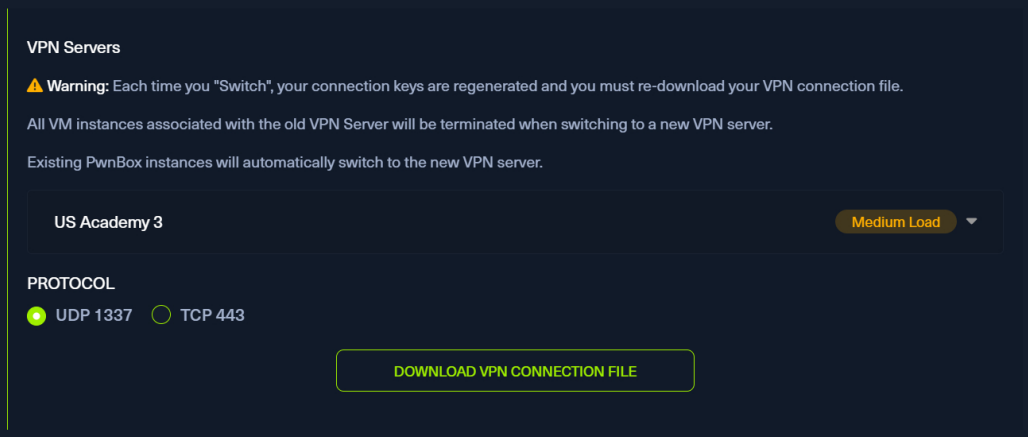
following links.

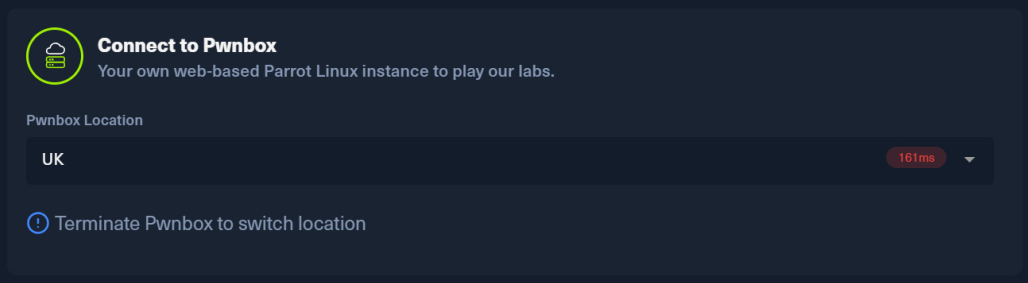
- https://github.com/SigmaHQ/sigma/wiki/Rule-Creation-Guide
- https://github.com/SigmaHQ/sigma-specification

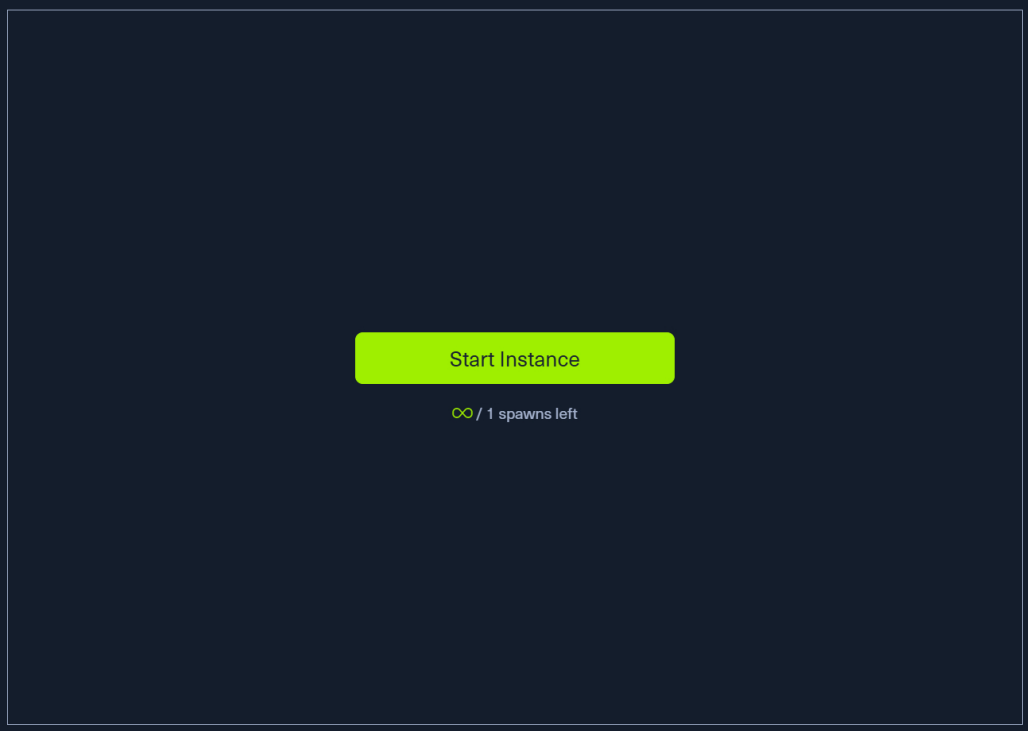
The following series of articles is the next best resource on Sigma rule developement.

- https://tech-en.netlify.app/articles/en510480/
- https://tech-en.netlify.app/articles/en513032/
- https://tech-en.netlify.app/articles/en515532/

In the upcoming section, we'll explore how to use Sigma rules for large-scale analysis of event logs with the aim of swiftly identifying threats.







Waiting to start...

Enable step-by-step solutions for all questions

Questions

Answer the question(s) below to complete this Section and earn cubes!

Download VPN Connection File

Target(s): Click here to spawn the target system!

RDP to with user "htb-student" and password "HTB_@cademy_stdnt!"

+ 2 Using sigmac translate the "C:\Tools\chainsaw\sigma\rules\windows\builtin\windefend\win_defender_threat.yml" Sigma rule into the equivalent PowerShell command. Then, execute the PowerShell command against "C:\Events\YARASigma\lab_events_4.evtx" and enter the malicious driver as your answer. Answer format: _.sys

mimidrv.sys

Submit

Previous Next

Mark Complete & Next

Powered by HACKTHEBOX