



# Sigma

SIEM Detection Format

## Operationalization of Sigma Rules with Processing Pipelines

2025-04-21

Thomas Patzke

TLP:CLEAR

# Agenda

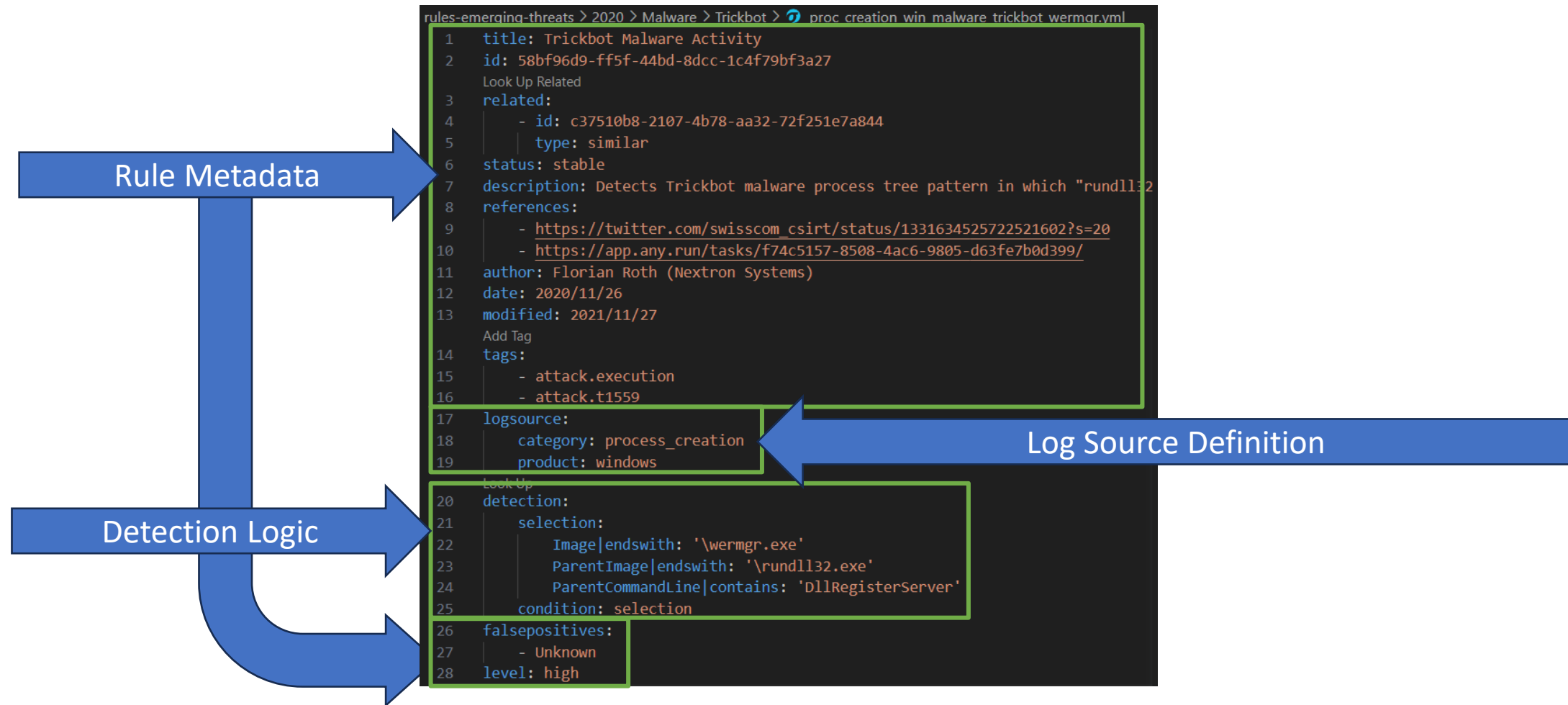


1. Introduction: What & Why?
2. Setting up Prerequisites
3. Mapping Field Names
4. Value Transformations
5. Add Conditions
6. Placeholders
7. Embedding Queries into Structures
8. Q&A

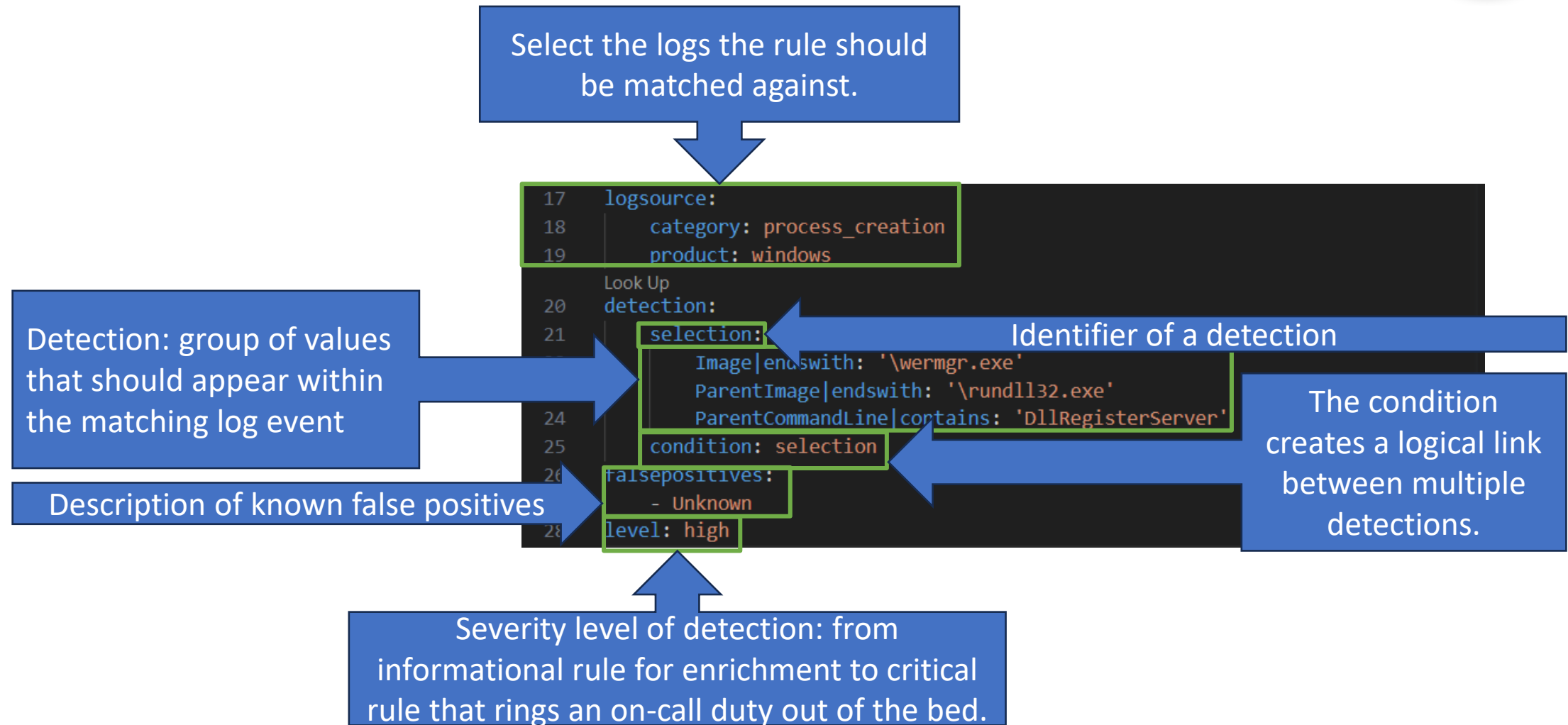
## Prerequisites:

- Python 3.9 to 3.13
- Sigma CLI: sigma-cli on PyPI
  - Install sigma-cli. Recommendation: separate Python virtual environment or pipx  
`pipx install sigma-cli`
- Clone of the Sigma workshop repository:  
`git clone`  
<https://github.com/SigmaHQ/sigma.git>
- Clone of the workshop repository:  
`git clone --recurse-submodules`  
`https://github.com/thomaspatzke/sigma-workshop-operationalization`

# Sigma Rule: Big Picture



# Sigma Rule: Log Source & Detection





# Introduction: Motivation

- Goal: utilize the SigmaHQ Sigma rule repository (or another one) to detect threats.
- Sigma rules must be converted into the query language of your SIEM.

## Challenges:

- Field names in Sigma rules are different than in your SIEM or data model:
  - CommandLine
  - process
  - process.command\_line
- Values differ too:
  - C:\Windows\System32\cmd.exe
  - \Device\Harddiskvolume3\Windows\System32\cmd.exe
  - cmd.exe



# Introduction: Advanced Use Cases

- Additional conditions, often search restrictions to increase performance
  - `index=windows-logs`
- Placeholders, e.g. domain controllers, vulnerability scanners etc.
  - Stored as list that should be integrated into query.
  - Dynamic via lookup in SIEM
- Data structures for bulk import of rules or other purposes.
  - Splunk `savedsearches.conf`
  - Elastic NDJSON exports
  - Documentation
- All these use cases don't require to touch the Sigma rules, everything can be done with processing pipelines!



# Prerequisites

You need:

- Python 3.9 to 3.13
- Sigma CLI: sigma-cli on PyPI
  - Install sigma-cli. Recommendation: separate Python virtual environment or pipx  
`pipx install sigma-cli`
- Clone of the Sigma workshop repository:  
`git clone https://github.com/SigmaHQ/sigma.git`
  - Use it as working directory, all paths in the slides are relative to this repos root directory.
- Clone of the workshop repository:  
`git clone --recurse-submodules  
https://github.com/thomaspatzke/sigma-workshop-operationalization`



# Target Query Language & Data Model

- I will show everything with Splunk and sometimes Elasticsearch, but the workshop is mostly query language-agnostic.
- The target data model is hypothetical. It's chosen to show as much features as possible. Nevertheless, there are parallels to real existing data models.
- If you want to “just follow”:  
`sigma plugin install splunk elasticsearch`
- Feel free to use your favorite query language or some real challenges!





# First Conversion

- Let's start to convert a Sigma rule into a query.
- Sigma rule:  
rules\windows\process\_creation\proc\_creation\_win\_rundll32\_obfuscate\_d\_ordinal\_call.yml
- Command:  

```
sigma convert -t splunk --without-pipeline  
sigma\rules\windows\process_creation\proc_creation_win_rundll32_obfuscate_d_ordinal_call.yml
```



# Conversion Result

`Image="*\\rundll32.exe"` OR `OriginalFileName="RUNDLL32.EXE"` OR  
`CommandLine="*rundll32*"` `CommandLine IN ("*#+*", "*#-")`

In our target data model:

- The field *Image* is named *ImageFileName*.
- As the field name says, *ImageFileName* doesn't contains a path but just the file name.
- There's no *OriginalFileName* field.



# Writing a Processing Pipeline

- Metadata
- Field Mapping Transformation
- Drop detection transformation
  - Field name condition restricts it to the defined field.
- Value replacement path to file name:
  - Transformation with pattern & replacement
  - Restrict transformation to defined field.

```
! pipeline-1.yml
1  name: Target data model
2  priority: 30
3  transformations:
4    - id: field_mapping
5      type: field_name_mapping
6      mapping:
7        Image: ImageFileName
8    - id: drop_fields
9      type: drop_detection_item
10     field_name_conditions:
11       - type: include_fields
12         fields:
13           - OriginalFileName
14    - id: path_to_file_name
15      type: replace_string
16      regex: "^\\*\\\\\\\\([\\\\\\\\]+)$"
17      replacement: "\\1"
18     field_name_conditions:
19       - type: include_fields
20         fields:
21           - ImageFileName
```



# Applying the First Pipeline

- **Command:**

```
sigma convert -t splunk -p pipeline-1.yml  
sigma\rules\windows\process_creation\proc_creation_win_rundll32_ob  
fuscated_ordinal_call.yml
```

- **Result:**

```
ImageFileName="rundll32.exe" OR CommandLine="*rundll32*"  
CommandLine IN ("*#+*", "*#-")
```

- **Just to Compare, the previous result:**

```
Image="*\\rundll32.exe" OR OriginalFileName="RUNDLL32.EXE" OR  
CommandLine="*rundll32*" CommandLine IN ("*#+*", "*#-")
```



# Fields & Values: Hints & Caveats

- Caveat: removing detection items changes the detection logic!
  - If it removes an essential part this can result in a useless detection with many hits. Don't use rule then (or try to improve it).
  - Here it was the original file name one of various attributes to identify the file.
- Fields can be mapped 1:n results in OR-ing all fields.
  - Useful in migration scenarios between data models.



# Controlled Failure

- What's if path also specifies directories and can't be mapped to the data model?
- Fail! It's important that no detection query is emitted that is not functional. This gives a false sense of security.
- Try it:

```
sigma convert -t splunk -p pipeline-2.yml  
sigma\rules\windows\process_creation\proc_creation_win_appvlp_unco  
mmon_child_process.yml
```

```
14 - id: fail_on_dir_in_path  
15   type: detection_item_failure  
16   message: "ImageFileName should not contain a directory path"  
17   field_name_conditions:  
18     - type: include_fields  
19       fields:  
20         - ImageFileName  
21   detection_item_conditions:  
22     - type: match_string  
23       cond: any  
24       pattern: "^\\.*\\\\\\?[^\\\\\\]+$"   
25       negate: true
```



# Rule Conditions

- There's another type of condition: rule conditions
- Use Cases:
  - Restricting to log sources
  - Match on rule tags and (custom) attributes
  - Check existence of field or detection item on rule level.
- Evaluation of conditions
  - Rule conditions: always, once for every rule
  - Detection item conditions: only for detection item transformations, once for every detection item
  - Field name conditions: additionally on field name list items of Sigma rule
- Example: `detection_item_failure` vs `rule_failure`



# Add Conditions

- Always searching everywhere isn't efficient.
- Example: searching Windows process creation events in Linux logs.
- Solution: add conditions based on log source.
- Example:
  - All logs are contained in the index *windows* (`index=windows`)
  - The sourcetype corresponds to the category:  
`sourcetype=Custom:<Sigma rule category>`

```
34 - id: windows_conditions
35   type: add_condition
36   conditions:
37     index: windows
38     sourcetype: Custom:$category
39   template: yes
40   rule_conditions:
41     - type: logsource
42       product: windows
```





# Placeholders

- Sigma supports %placeholders%
- There are different possibilities to handle placeholders:
  - Replace with a value or a list of values (value\_placeholders)
  - Replace with a query expression, e.g. lookup (query\_expression\_placeholders)
  - Ignore them by replacing with \* as last resort (wildcard\_placeholders)

```
detection:  
  selection:  
    CommandLine|contains: 'echo '  
    CommandLine|contains|expand: '%userdomain%'
```

- Try it with:

```
46 - id: placeholder_fallback  
47 | type: wildcard_placeholders
```

```
43 - id: placeholder_userdomain  
44 | type: value_placeholders  
45 | include: userdomain
```

rules-

placeholder\windows\process\_creation\proc\_creation\_win\_userdomain\_  
variable\_enumeration.yml

```
- id: placeholder_userdomain_lookup  
  type: query_expression_placeholders  
  expression: '[ inputlookup userdomain | fields domain | rename domain as {field} ]'
```

# Query Postprocessing & Finalization: Generating Importable Output



- Convert all process creation rules:

```
sigma convert -t splunk -p pipeline-4.yml -s  
rules\windows\process_access
```

- It's just a collection of queries, you can just copy&paste.
- SIEMs usually have a data format to import queries:
  - Splunk: savedsearches.conf
  - Elasticsearch: NDJSON import
- There's a *savedsearches* output format in the Splunk backend, but it lacks flexibility.

# Query Postprocessing & Finalization: Embedding Query in Use Case Framework



Usually, detections are embedded in some use case framework:

- Results are enriched with detection information, e.g. rule metadata.
- Results are written to an index or sent to other system (e.g. ticketing, SOAR) via API.
- Different scheduling parameters depending on log source:
  - Near real-time search for real-time logs.
  - Query delay for logs ingested in batches.

# Example: Splunk savedsearches.conf

Put query into structure

Extend query by enrichment & storage in notable events index

```
- type: template
  template: |
    [{{ rule.id }}]
    search = {{ query }} | eval rule="{{ rule.id }}", title="{{ rule.title }}" | collect index=notable_events
    description = {{ rule.description }}
finalizers:
- type: concat
  prefix: |
    [default]
    cron_schedule = */15 * * * *
    dispatch.earliest_time = -20m@m
    dispatch.latest_time = -5m@m
```

Prefix output with some defaults.



# Recap: What have learned

- Mapping field names
- Transforming values
- Fail if something is not possible
- Adding conditions
- Field name, detection item and rule conditions/transformations
- Placeholders: make detections flexible
- Post-processing and finalization: embedding generated queries into data structures



# But there's more...

## Transformations:

- Adding/removing/changing field lists ((add|change|remove)\_field)
- Mapping strings with a source/target value dictionary (map\_string)
- Stateful pipelines (set\_state)
- Nesting of multiple processing item in pipeline (nest)

## Conditions:

- Check processing state (processing\_state)
- Check if former transformation was applied (processing\_item\_applied)
- Check if rule attribute is set to a value (rule\_attribute)
- Check for tags (tag)

...and many more!

# Further Trainings



## **Writing and converting Sigma rules:**

Applied Network Defense: Detection Engineering with Sigma

- <https://www.networkdefense.co/courses/sigma/>



## **Building a detection pipeline with Sigma:**

Taggart Institute: Automated Detection with Sigma

- <https://taggartinstitute.org/p/detection-with-sigma>



# Contact & Resources



[thomas@patzke.org](mailto:thomas@patzke.org)



Sigma Discord



Documentation



Sponsoring



Blog: Pipelines



Blog: Post-Processing



Blog: Placeholders