Sigma

SIEM Detection Format

Operationalization of Sigma Rules with Processing Pipelines

2024-10-22

Thomas Patzke

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 3.12
- 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:
  `https://github.com/thomaspatzke/sigma-workshop-operationalization`

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 3.12
- 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:
  `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_obfuscated_ordinal_call.yml
- Command:

```
sigma convert -t splunk --without-pipeline
rules\windows\process_creation\proc_creation_win_rundll32_obfuscated_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

```yaml
name: Target data model
priority: 30
transformations:
- id: field_mapping
  type: field_name_mapping
  mapping:
    Image: ImageFileName
- id: drop_fields
  type: drop_detection_item
  field_name_conditions:
    - type: include_fields
      fields:
        - OriginalFileName
- id: path_to_file_name
  type: replace_string
  regex: "^\\*\\\\([^\\\\]+)$"
  replacement: "\\1"
  field_name_conditions:
    - type: include_fields
      fields:
        - ImageFileName
```

Applying the First Pipeline

- Command:

```
sigma convert -t splunk -p pipeline-1.yml
rules\windows\process_creation\proc_creation_win_rundll32_obfuscat
ed_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
```

```yaml
- id: fail_on_dir_in_path
  type: detection_item_failure
  message: "ImageFileName should not contain a directory path"
  field_name_conditions:
    - type: include_fields
      fields:
        - ImageFileName
  detection_item_conditions:
    - type: match_string
      cond: any
      pattern: "^\\*\\\\?[^\\\\]+$"
      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 is 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%

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

- 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)
- Try it with:

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

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

`rules-placeholder\windows\process_creation\proc_creation_win_userdomain_variable_enumeration.yml`

```yaml
- 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_creation
  ```
- 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!

Contact & Resources

thomas@patzke.org

Sigma Discord

Documentation

Sponsoring

Blog: Pipelines

Blog: Post-Processing

Blog: Placeholders