



Sigma
SIEM Detection Format

Log Hunting with Sigma

A hands-on workshop for Sigma and the pySigma/Sigma CLI toolchain

Created by Thomas Patzke

Agenda



1. Intro: What is Sigma?
2. Writing a Sigma Rule
3. Sigma Rule Validation
4. Conversion of Sigma Rules into Queries with Sigma CLI
5. Processing Pipelines: Adapting to the Target Environment
6. Creating Custom Output Formats with Query Post-Processing
7. Building Support for new Query Languages: Introduction to Backend Development.
8. The Open Source Sigma Rule Repository



Prerequisites

You need:

- Python 3.8 to 3.11
- Sigma CLI: sigma-cli on PyPI
 - Install with your favorite PIP package manager, e.g.:
`pipx install sigma-cli`
- **Clone of the Sigma workshop repository:**
`git clone https://github.com/SigmaHQ/sigma-workshop.git`
- Internet access 😊

Sigma Goals and Scope

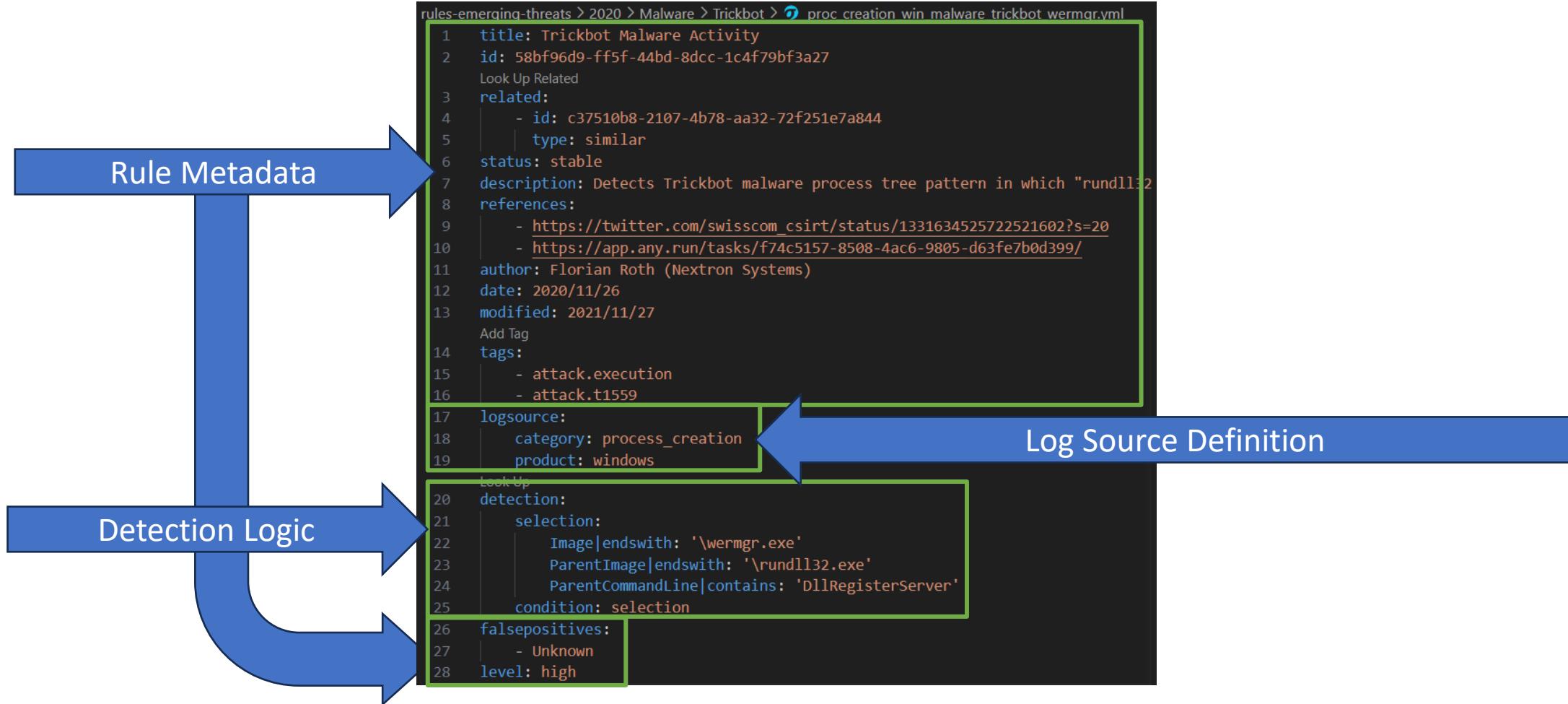


- Human readable/writable
- Machine readable/writable
- Keep it simple stupid
 - Describe log signatures for particular interesting events.
 - Instead of: expression of all imaginable detections and threat hunting techniques.
- Each Sigma rule should be convertible into as much target systems as possible.

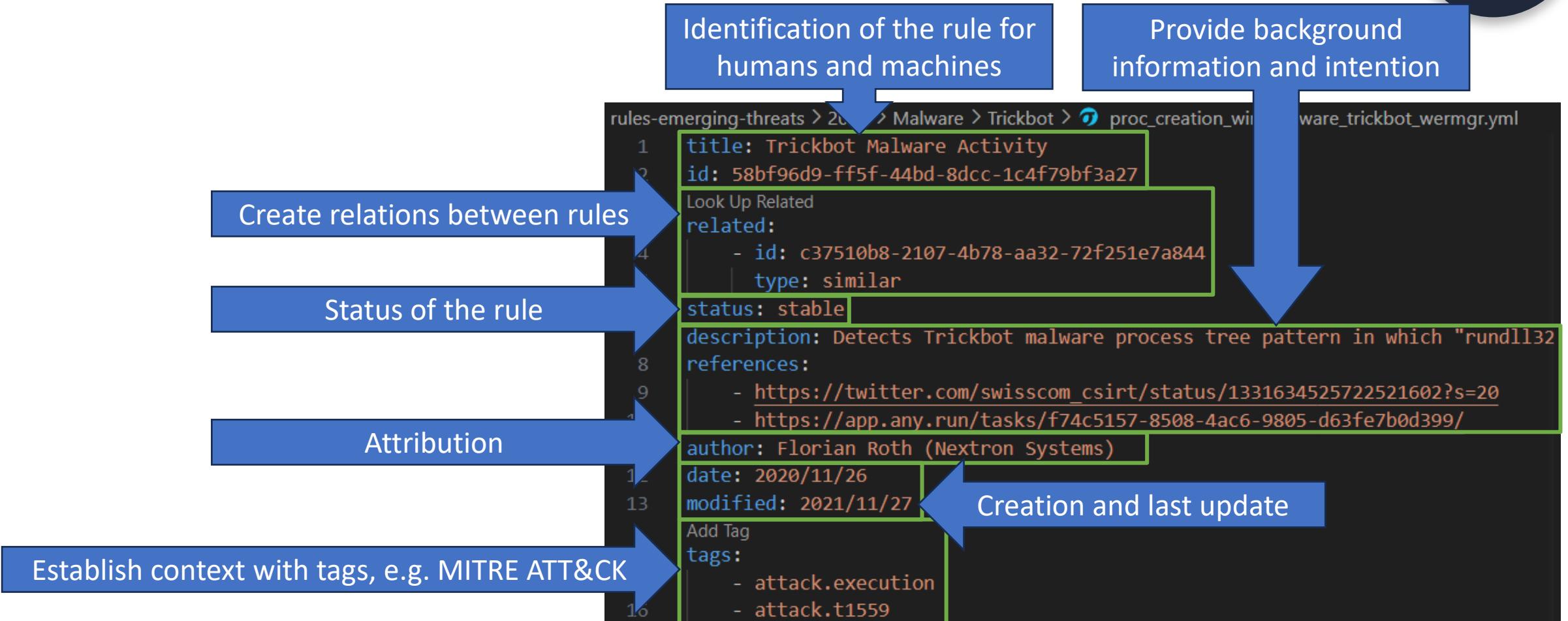


YAML as base format

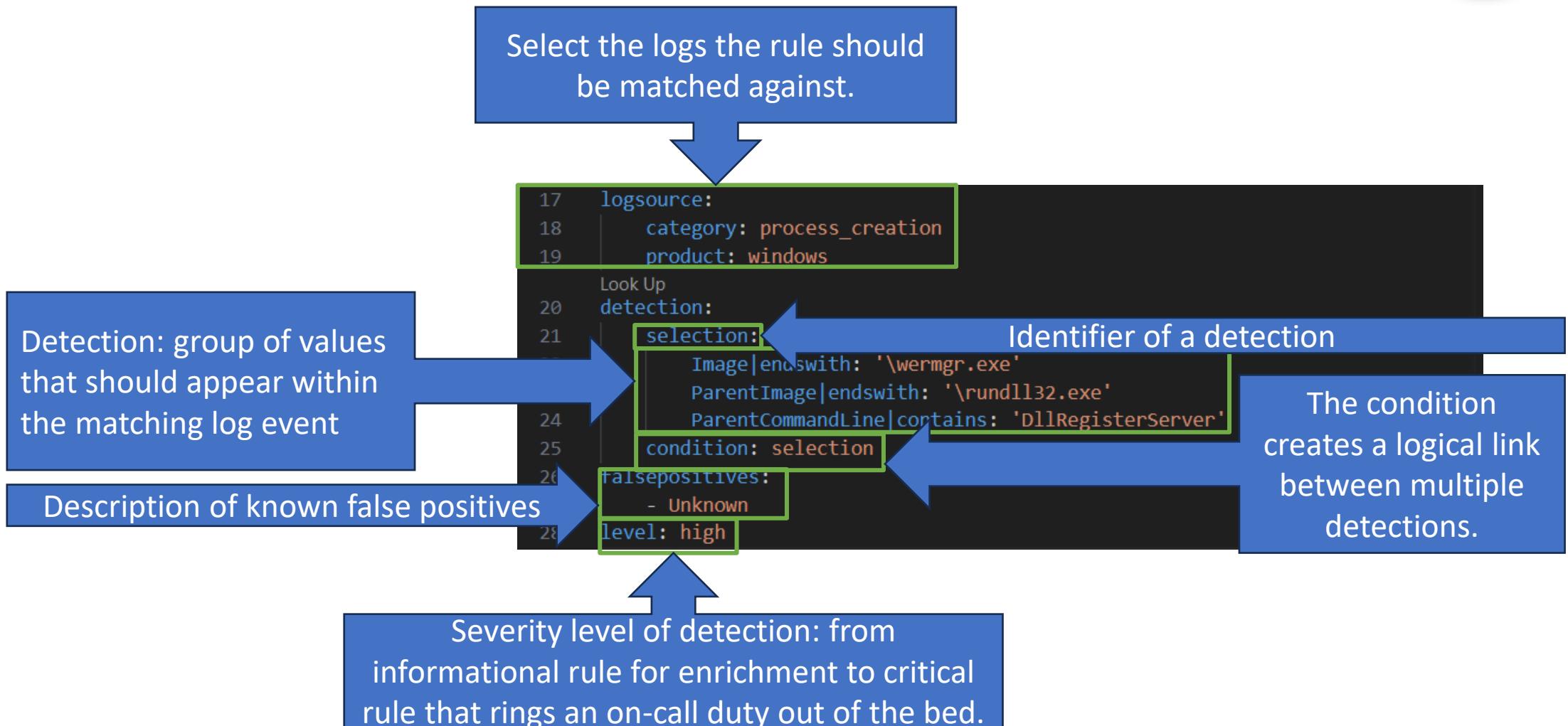
Sigma Rule: Big Picture



Sigma Rule: Metadata



Sigma Rule: Log Source & Detection





Sigma Rule: Condition

- Detections can be logically linked with the condition.
- Usual: exclude legitimate behavior or other known false positives.

```
detection:  
  selection:  
    cs-method: 'GET'  
    c-uri|endswith: '\?manifest=wac'  
    cs-host: 'onedrive.live.com'  
  filter:  
    c-uri|startswith: 'http'  
    c-uri|contains: '://onedrive.live.com/'  
  condition: selection and not filter
```

How to Write a Sigma Rule?



- Starting points: threat report, sandbox report, incident analysis, observation, ...
- Rule creation guide:
<https://github.com/SigmaHQ/sigma/wiki/Rule-Creation-Guide>
- Recommendation: VSCode Sigma extension
- Look after:
 - Unusual process relationships.
 - Unusual parameters, like encoded.
 - Unusual locations.
 - Events specific to actions like creation of scheduled tasks.

Exercise: Writing Sigma Rules



Pick one of the following sandbox analysis reports and write a Sigma rule that detects the threat:

1. QakBot:

<https://app.any.run/tasks/91840612-d4bb-4285-b86c-1e389f90156b/>

2. QakBot:

<https://www.joesecurity.org/reports/report-5cb20a0bfc5e3e2ae8398b1840adf7ae.html>



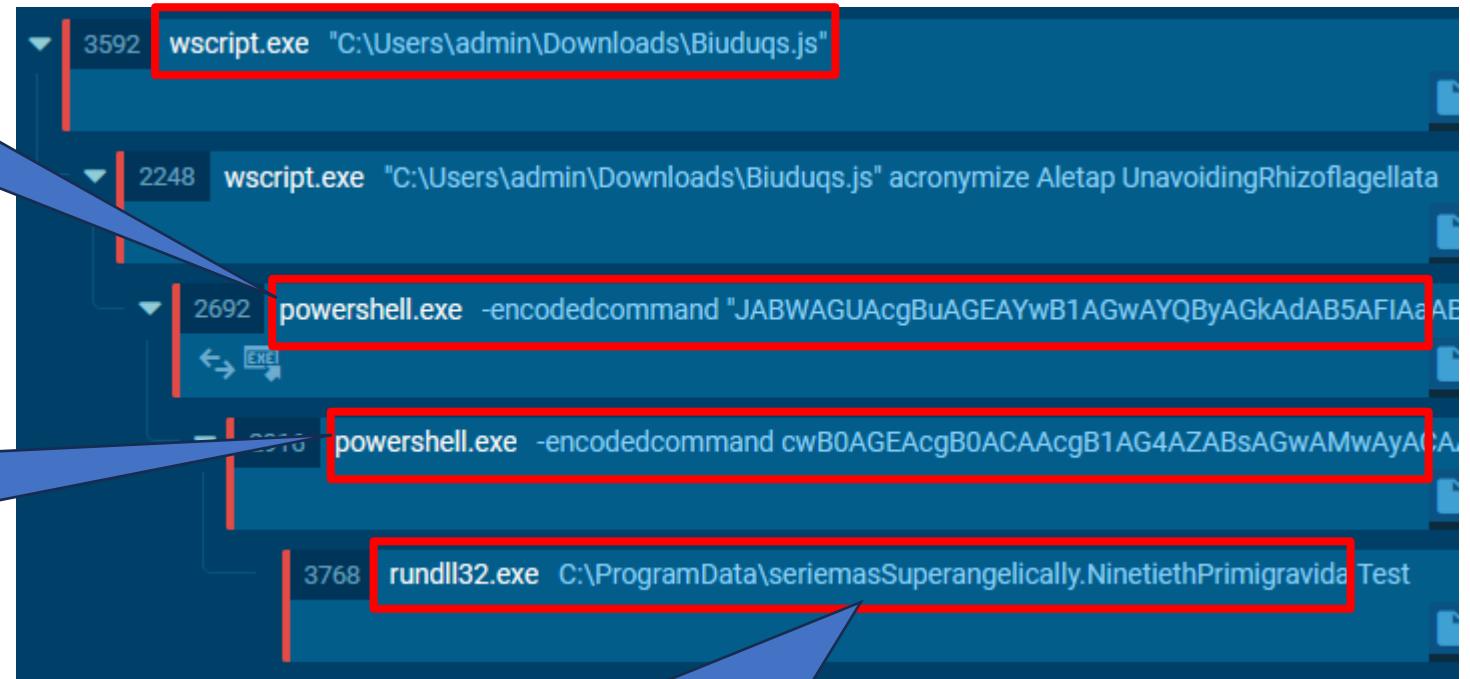
Sample 1



PowerShell with encoded command,
but beware of parameter shortening, -
enc is also valid! Payload itself
contains lots of randomized
identifiers.

Same as parent process, but encoded
part is “start rundll32
\$env:ProgramData\...” and more
applicable for a detection.

JS executed with wscript from users
download folder is quite unusual!



Rundll without a DLL? Beware: it's also called
sometimes with OCX and others.



Sample 2

The calculator shouldn't spawn anything!

- cmd.exe (PID: 1016 cmdline: "C:\Windows\System32\cmd.exe" /q /c calc.exe MD5: F3BDBE3BB6)
- conhost.exe (PID: 1300 cmdline: C:\Windows\system32\conhost.exe 0xffffffff -ForceV1 MD5: I...)
- calc.exe (PID: 5940 cmdline: calc.exe MD5: 60B7C0FEAD45F2066E5B805A91F4F0FC) 📁
- regsvr32.exe (PID: 7140 cmdline: C:\Windows\SysWOW64\regsvr32.exe 102755.dll MD5: ...)
- svchost.exe (PID: 6792 cmdline: C:\Windows\System32\svchost.exe -k netsvcs -p MI...)
- explorer.exe (PID: 6792 cmdline: C:\Windows\SysWOW64\explorer.exe MD5: 166AB...)

Service host usually only spawned by system processes and shouldn't ever appear further down the process tree.

Example Rule for Sample 1: Script Execution from Download Folder



```
proc_creation_win_wscript_download_folder.yml
1 title: Execution of Windows Scripting Host with File in Downloads Folder
2 id: ad41edbd-5a0f-47ae-8f60-a0222866c1c4
3 status: stable
4 description: Detects execution of script files from download folder. Often an indicator for malicious droppers.
5 author: Thomas Patzke
6 date: 2023/09/11
7 Add Tag
8 tags:
9 - attack.t1059
10 logsource:
11   category: process_creation
12   product: windows
13 Look Up
14 detection:
15   selection:
16     ImageFile|endswith: "*\\wscript.exe"
17     CommandLine|contains: "\\Downloads\\"
18   condition: selection
19 falsepositives:
20   - Unknown
21 level: high
```

Example Rule for Sample 2: Calculator Spawns a Process



```
release > rules > proc_creation_win_calc_spawns_process.yml
  1   title: Windows calculator spawns process
  2   id: 34381c6b-9e65-40f8-9270-9e733f8045fa
  3   status: stable
  4   description: Detects if the Windows calculator spawns a
                  process. Usually a strong indicator that something
                  injected into it.
  5   author: Thomas Patzke
  6   date: 2023/09/25
      Add Tag
  7   tags:
  8     - attack.t1055
  9   logsource:
10     category: process_creation
11     product: windows
      Look Up
12   detection:
13     selection:
14       ParentImage|endswith: "\Windows\System32\calc.exe"
15       condition: selection
16   falsepositives:
17     - Unknown
18   level: high
19
```

Example Rule for Sample 2: Unusual Service Host Parent



```
release > rules > proc_creation_unusual_svchost_parent.yml
  1 title: Service host spawned by unusual process
  2 id: 8f95b6b7-7f35-4f0d-9c6c-7c69d13a0cca
  3 status: experimental
  4 description: Detects Windows service host executions spawned by unusual processes.
  5 author: Thomas Patzke
  6 date: 2023/09/26
    Add Tag
  7 tags:
  8   - attack.t1055
  9 logsource:
10   category: process_creation
11   product: windows
  Look Up
12 detection:
13   selection:
14     Image|endswith: \\svchost.exe
15     legitimate:
16       ParentImage|endswith: \\services.exe
17     condition: selection and not legitimate
18 falsepositives:
19   - Crashes could possibly different produce process relationships in rare cases.
20 level: high
```

Rule Validation



- Sigma CLI has built-in Sigma rule validation that checks:
 - Rules for syntactical correctness.
 - Conditions for correctness.
 - Rules for common bad practices or possible improvements.
 - Run `sigma check .` To check current directory.

```
$ sigma check .
Parsing Sigma rules  [#####
Checking Sigma rules  [#####
=====
Summary =====
Found 0 errors, 0 condition errors and 0 issues.
No rule errors found.
No condition errors found.
No validation issues found.
```

```

sigma check bad
Parsing Sigma rules [########################################] 100%
Checking Sigma rules [########################################] 100%
  Issues ==

issue:InvalidATTACKtagIssue severity:medium description="Invalid MITRE ATT&CK tagging" rules=[c:\Users\thoma\OneDrive\Sigma\2023-Sigma Workshop\Rule\bad\proc_creation_unusual_svhost_parent.yml] tag=attack.t1080
issue:DoubleWildcardIssue severity:low description="String contains multiple consecutive * wildcards" rules=[c:\Users\thoma\OneDrive\Sigma\2023-Sigma\Workshop\Rules\bad\proc_creation_unusual_svhost_parent.yml] string="*services.exe"
issue:WildcardInsteadOfEndswithIssue severity:low description="String contains wildcard at beginning instead of being modified with endswith modifier" rules=[c:\Users\thoma\OneDrive\Sigma\2023-Sigma Workshop\Rules\bad\proc_creation_unusual_svhost_parent.yml] detection_item=SigmaDetectionItem[parent=SigmaDetectionItem[parent=ConditionIdentifier[parent=ConditionAND[parent=None, args=[ConditionField>equalsValueExpression(parent=..., field="Image", value=(SpecialChars.WILDCARD_MULTI: 1), 'svchost.exe')], ConditionNOT[parent=..., args=[ConditionField>equalsValueExpression(parent=SigmaDetectionItem[parent=ConditionIdentifier[parent=..., args=[legitimate], source=None, identifier="legitimate"], detection_items=[...], source=SigmaRuleLocation(path=WindowsPath\b\proc_creation_unusual_svhost_parent.yml)], line=None, char=None}, item_linking=<class 'sigma.conditions.ConditionAND>, applied_processing_items=set(), field='Image', modifiers=[], value=(SpecialChars.WILDCARD_MULTI: 1), 'services.exe']], value_linking=<class 'sigma.conditions.ConditionOR>, source=SigmaRuleLocation(path=WindowsPath\b\proc_creation_unusual_svhost_parent.yml)], line=None, char=None}, item_linking=<class 'sigma.conditions.ConditionAND>, applied_processing_items=set(), field='Image', modifiers=[], value=(SpecialChars.WILDCARD_MULTI: 1), '\\svchost.exe')), source=SigmaRuleLocation(path=WindowsPath\b\proc_creation_unusual_svhost_parent.yml)], line=None, char=None}], sourc e=SigmaRuleLocation(path=WindowsPath\b\proc_creation_unusual_svhost_parent.yml)], line=None, char=None)], args=[selection], source=None, identifier="selection", detection_items=[...], source=SigmaRuleLocation(path=WindowsPath\b\proc_creation_unusual_svhost_parent.yml)], line=None, char=None}, item_linking=<class 'sigma.conditions.ConditionAND>, applied_processing_items=set(), field='Image', modifiers=[], value=(SpecialChars.WILDCARD_MULTI: 1), '\\svchost.exe']), value_linking=<class 'sigma.conditions.ConditionOR>, source=SigmaRuleLocation(path=WindowsPath\b\proc_creation_unusual_svhost_parent.yml)], line=None, char=None)

  Summary ==
Found 0 errors, 0 condition errors and 3 issues.
No rule errors found.
No condition errors found.

Validation issue summary:
+-----+-----+-----+
| Count | Issue | Severity | Description |
+-----+-----+-----+
| 1     | InvalidATTACKtagIssue | MEDIUM | Invalid MITRE ATT&CK tagging |
| 1     | DoubleWildcardIssue | LOW    | String contains multiple consecutive * wildcards |
| 1     | WildcardInsteadOfEndswithIssue | LOW    | String contains wildcard at beginning instead of being modified with endswith modifier |
+-----+-----+-----+

```



Rule Conversion Concepts

- The Sigma CLI command *convert* converts a Sigma rule into a query.
- The conversion into a *target query language* is conducted with a *backend*.
- The conversion is configured with a *processing pipeline*.
- Backends and processing pipelines are available as *plugins*.
- Custom processing pipelines can be written in YAML format.

```
$ sigma convert -t splunk .
Usage: sigma convert [OPTIONS] INPUT...
Try 'sigma convert --help' for help.

Error:
Processing pipeline required by backend! Define a custom pipeline or choose a predefined one.

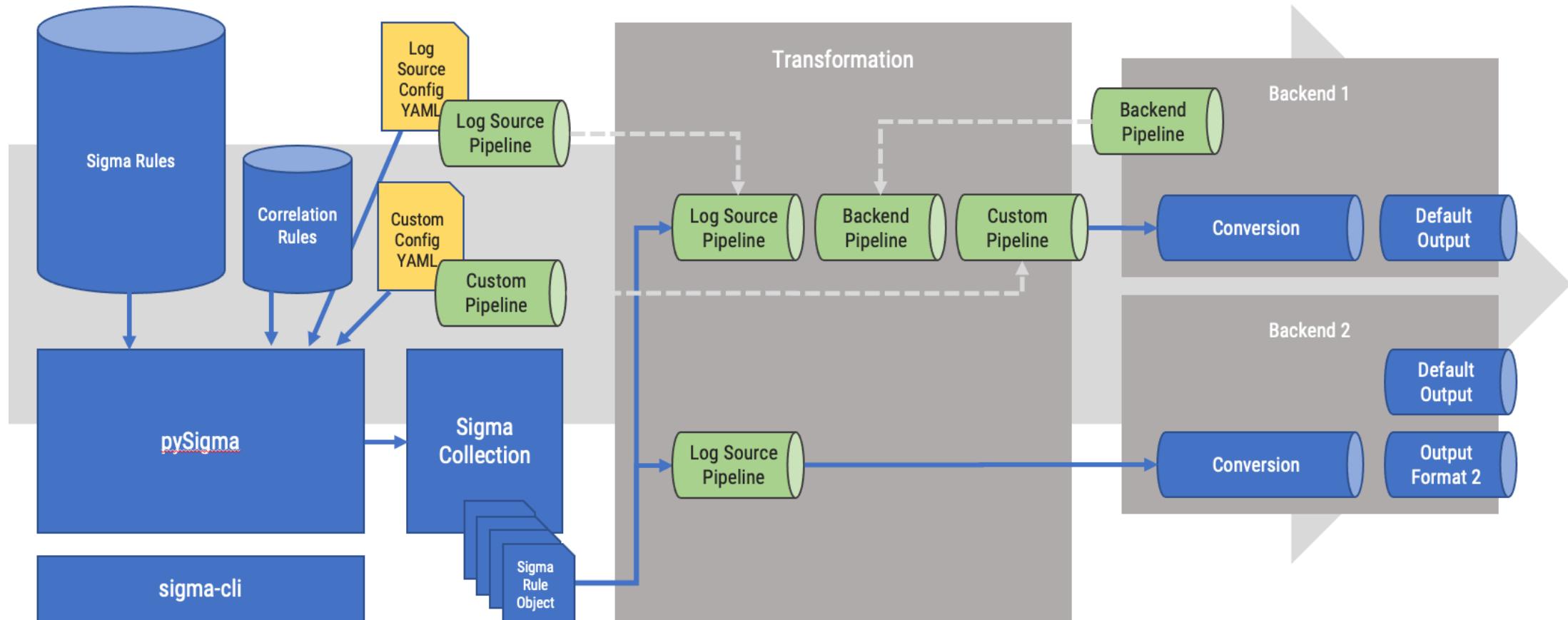
Get all available pipelines for splunk with:
sigma list pipelines splunk

If you never heard about processing pipelines you should get familiar with them
(https://sigmahq-pysigma.readthedocs.io/en/latest/Processing\_Pipelines.html).
If you know what you're doing add --without-pipeline to your command line to suppress this error.
```

```
$ sigma convert -t elasticsearch .
Usage: sigma convert [OPTIONS] INPUT...
Try 'sigma convert --help' for help.

Error: Invalid value for '--target' / '-t': 'elasticsearch' is not 'splunk'. -
run sigma plugin list --plugin-type backend for a list of available plugins.
```

Rule Conversion Flow





The Sigma CLI Plugin System

- Backends and pipelines are separated from pySigma and the CLI.
- Support for query languages and log sources must be installed.
- Subcommand `sigma plugin` for management of plugins.
- Example:

```
$ sigma plugin install splunk sysmon
Successfully installed plugin 'splunk'
Successfully installed plugin 'sysmon'
```

```
$ sigma plugin
Usage: sigma plugin [OPTIONS] COMMAND [ARGS]...

pySigma plugin management (backends, processing pipelines and validators).

Options:
  --help  Show this message and exit.

Commands:
  install  Install plugin by identifier or UUID.
  list     List installable plugins.
  uninstall Uninstall plugin by identifier or UUID.
$ sigma plugin list
```

| Identifier | Type | State | Description | Compatible? |
|----------------|----------|--------|---|-------------|
| ibm-qradar-aql | backend | stable | IBM QRadar backend for conversion into AQL queries. Contains mappings for fields and logsources | yes |
| cortexxdr | backend | stable | Cortex XDR backend that generates XQL queries. | yes |
| carbonblack | backend | stable | Carbon Black backend that supports queries for both Enterprise EDR (fka Threat Hunter) and EDR (fka Response) | yes |
| sentinelone | backend | stable | SentinelOne backend that generates Deep Visibility queries. | yes |
| sentinelone-pq | backend | stable | SentinelOne backend that generates PowerQuery queries. | yes |
| splunk | backend | stable | Splunk backend for conversion into SPL and tstats data model queries as plain queries and savedsearches.conf | no |
| insightidr | backend | stable | Rapid7 InsightIDR backend that generates LSQL queries. | no |
| qradar | backend | stable | IBM QRadar backend for conversion into AQL and extension packages. | yes |
| elasticsearch | backend | stable | Elasticsearch backend converting into Lucene queries, plain, embedded into DSL or as Kibana NDJSON. | no |
| opensearch | backend | stable | Opensearch backend converting into Lucene queries and Opensearch alerting rules. | no |
| ala-socprime | backend | devel | Azure Log Analytics backend with Windows log support maintained by SOC Prime. | no |
| ala-sifex | backend | devel | Azure Log Analytics backend with Windows log support maintained by @sifex. | yes |
| stix | backend | devel | STIX backend converting into plain STIX queries. Contains mappings for STIX 2.0 and STIX Shifter taxonomies. | yes |
| loki | backend | stable | Loki backend for conversion into Loki LogQL queries (plain and ruler YAML for alerts) and pipelines with mappings for Grafana and promtail Sysmon data. | no |
| windows | pipeline | stable | Windows logsource to Channel field and generic logsource to Windows audit events mapping. | no |
| sysmon | pipeline | stable | Mapping from generic log sources to Sysmon events. | no |
| crowdstrike | pipeline | stable | Mapping from generic log sources to CrowdStrike events from | no |



First Conversion

- sigma list targets shows available target query languages.
- sigma list pipelines shows available processing pipelines.
- sigma convert -t <target> -p <pipeline> <path> converts Sigma rules into queries.

```
$ sigma list targets
+-----+
| Identifier | Target Query Language | Processing Pipeline Required |
+-----+
| lucene     | Elasticsearch Lucene    | Yes
| splunk     | Splunk SPL & tstats data model queries | Yes
+-----+
$ sigma list pipelines
+-----+
| Identifier | Priority | Processing Pipeline | Backends |
+-----+
| crowdstrike_fdr | 10 | Generic Log Sources to CrowdStrike Falcon Data Replicator (FDR) Transformation | all
| ecs_windows | 20 | Elastic Common Schema (ECS) Windows log mappings from Winlogbeat from version 7 | lucene, opensearch
| ecs_windows_old | 20 | Elastic Common Schema (ECS) Windows log mappings from Winlogbeat up to version 6 | lucene, opensearch
| ecs_zeek_beats | 20 | Elastic Common Schema (ECS) for zeek using filebeat >= 7.6.1 | lucene, opensearch
| ecs_zeek_corelight | 20 | Elastic Common Schema (ECS) mapping from Corelight | lucene, opensearch
| splunk_windows | 20 | Splunk Windows log source conditions | splunk
| splunk_sysmon_acceleration | 25 | Splunk Windows Sysmon search acceleration keywords | splunk
| splunk_cim | 20 | Splunk CIM Data Model Mapping | splunk
| sysmon | 10 | Generic Log Sources to Sysmon Transformation | all
+-----+
$ sigma convert -t splunk -p sysmon .
Parsing Sigma rules [#####
EventID=1 Image="*\svhost.exe" NOT ParentImage="*\services.exe"
EventID=1 ParentImage="*\calc.exe"
EventID=1 ImageFile="*\wscript.exe" CommandLine="*\Downloads\*"
$ sigma convert -t lucene -p sysmon .
Parsing Sigma rules [#####
EventID:1 AND (Image:*\svhost.exe AND (NOT ParentImage:*\services.exe))
EventID:1 AND ParentImage:*\calc.exe
EventID:1 AND (ImageFile:*\wscript.exe AND CommandLine:*\Downloads\*)
$ sigma convert -t splunk -p crowdstrike_fdr .
Parsing Sigma rules [#####
event_simpleName="ProcessRollup2" ImageFileName="*\svhost.exe" NOT ParentBaseFileName="services.exe"
event_simpleName="ProcessRollup2" ParentBaseFileName="calc.exe"
event_simpleName="ProcessRollup2" ImageFile="*\wscript.exe" CommandLine="*\Downloads\*"
```



Is the query ready for usage?

- It is not specified which log is searched. Restriction to a specific index and Windows log source required.
- In Splunk the field *EventID* usually is named *EventCode*.
- In most Splunk setups this query will likely not match any results!
- EventID=1
Image="*\\svchost.exe" NOT ParentImage="*\\services.exe"
- EventID=1
ParentImage="*\\calc.exe"
- EventID=1
ImageFile="*\\wscript.exe"
CommandLine="*\\Downloads*"

Processing Pipelines Introduction



- Processing pipelines can be chained.
- Predefined processing pipeline `splunk_windows` defines log sources and field name mappings.
- Custom processing pipelines can be written in YAML format.
- Usual processing chain:
log source > target-specific > environment-specific

```
$ sigma convert -t splunk -p sysmon -p splunk_windows .
Parsing Sigma rules [#####
] 100%
source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1
Image="*\svchost.exe" NOT ParentImage="*\services.exe"

source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1
ParentImage="*\calc.exe"

source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1
ImageFile="*\wscript.exe" CommandLine="*\Downloads\*"
```

Processing Pipeline Basics



- Processing pipeline has a *name*.
- The *priority* determines the order of multiple concatenated processing pipelines.
- There are three stages of processing:
 1. *Rule pre-processing*: transforms the rule before it is converted by the backend into a query. **Examples**: mapping field names, adding conditions.
 2. *Query post-processing*: transforms the query generated from a Sigma rule. Has access to the rule and query. **Example**: embedding query and rule parts into template.
 3. *Output finalization*: works on all post-processed queries to generate a finalized output from this. **Example**: embed queries into file format with header.
- In each stage *transformations* are executed which can be restricted by *conditions*.



Processing Pipeline Example

Name of pipeline is just informational.

Each transformation should have an identifier

Defines what the transformation does.

Another transformation:
add prefix to all field names...

`pipeline.yml`

```
1 name: Fixing the field naming mess
2 priority: 30
3 transformations:
```

```
- id: field_mapping
  type: field_name_mapping
```

mapping:

```
  EventID:
    - event_id
    - evtid
```

```
- id: windows_field_prefix
```

```
  type: field_name_prefix
  prefix: "win."
```

```
  field_name_cond_not: true
```

```
  field_name_conditions:
```

```
    - type: processing_item_applied
```

```
    processing_item_id: field_mapping
```

Priority applies pipeline before possibly existing automatically applied backend pipelines and after log source pipeline.

Transformation type specific parameters. Here: field name mapping from EventID to multiple field names in target data scheme.

...that match the condition. Here, the field mapping from the former transformation was not applied to the field.

Exercise: Writing a Custom Processing Pipeline



- Build a processing pipeline that adds an index condition to each query.
- The *add_condition* transformation is a possible way.

```
processing_pipelines > ! splunk_customization.yml
1   name: Custom Splunk environment
2   priority: 30
3   transformations:
4     - id: index_condition
5       type: add_condition
6       conditions:
7         index: windows
```

```
$ sigma convert -t splunk -p sysmon -p splunk_windows -p .\processing_pipelines\splunk_customization.yml rules
Parsing Sigma rules [#####
index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1 Image="*\svchost.exe" NOT ParentImage="*\services.exe"

index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1 ParentImage="*\calc.exe"

index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1 ImageFile="*\wscript.exe" CommandLine="*\Downloads\*"
```



Custom Output Formats

- Plain query output is sufficient for searching for specific threats once, but not sufficient for continuous detection of threats.
- To operationalize detections the queries must be
 - *persisted* into a SIEM.
 - *scheduled* to run regularly or continuous.
 - *emit detection events* that can be picked up by case handling or SOAR systems.
- This means that queries must be embedded into data structures that configure the functions mentioned above.
- Processing pipelines allow this in the *query post-processing* and *output finalization* stages.



Post-Processing Stages

1. Starting point: the *pre-processed rule* was converted into a query by the backend.
2. In the *query post-processing* stage the query can be transformed, embedded into a template and enriched with information from the Sigma rule.
3. The *output finalization* stage merges and embeds all *post-processed queries* into a final output.

```
index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1
Image="*\svchost.exe" NOT ParentImage="*\services.exe"

index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1
ParentImage="*\calc.exe"

index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1
ImageFile="*\wscript.exe" CommandLine="*\Downloads\*"
```

```
[splunk-savedsearches-concat.yml]
1 postprocessing:
2   - type: template
3     template: |+
4       [{{ rule.id }}]
5       search = {{ query }} | eval rule="{{ rule.id }}", title="{{ rule.title }}" | collect index=notable_events
6       description = {{ rule.description }}

7
8   finalizers:
9     - type: concat
10       prefix: |
11         [default]
12         cron_schedule = */15 * * * *
13         dispatch.earliest_time = -20m@m
14         dispatch.latest_time = -5m@m
```

```
[default]
cron_schedule = */15 * * * *
dispatch.earliest_time = -20m@m
dispatch.latest_time = -5m@m
[8f95b6b7-7f35-4f8d-9c6c-7c6d13a0cca]
search = index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1 Image="*\svchost.exe" NOT ParentImage="*\services.exe" | eval rule="8f95b6b7-7f35-4f8d-9c6c-7c6d13a0cca", title="Service host spawned by unusual process" | collect index=notable_events
description = Detects Windows service host executions spawned by unusual processes.

[34381c6b-9e65-46f8-9270-9e733f88045fa]
search = index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1 ParentImage="*\calc.exe" | eval rule="34381c6b-9e65-46f8-9270-9e733f88045fa", title="Windows calculator spawns process" | collect index=notable_events
description = Detects if the Windows calculator spawns a process. Usually a strong indicator that something injected into it.

[44114edbd-5a0f-47ac-8f68-a0222866c1c4]
search = index="windows" source="WinEventLog:Microsoft-Windows-Sysmon/Operational" EventCode=1 ImageFile="*\wscript.exe" CommandLine="*\Downloads\*" | eval rule="44114edbd-5a0f-47ac-8f68-a0222866c1c4", title="Execution of Windows Scripting Host with File in Downloads Folder" | collect index=notable_events
description = Detects execution of script files from download folder. Often an indicator for malicious droppers.
```

Exercise: Create a Post-Processing Pipeline



Create a Splunk savedsearches.conf file

(<https://docs.splunk.com/Documentation/Splunk/9.1.0/Admin/Savedsearchesconf>) with the following properties:



- All searches are scheduled each 15 minutes and search logs in a window from 5 to 20 minutes before.
- Each hit is enriched with the rule ID (rule.id) and its title (rule.title).
- All enriched hits are written into an index *notable_events*.

Possible Solutions for the Post-Processing Pipeline



Embedding the rule identifier as query identifier.

This section defines the *query post-processing* stage transformations, here embedding the query and rule parts into a template

Enriching the matched event with rule metadata provides context to the analysts who are handling the event.

The query generated by the backend is embedded here.

Write enriched event into an index to be picked up by a SOAR or ticketing system.

```
processing_pipelines/splunk-savedsearches-concat.yml
1 postprocessing:
2   - type: template
3     template: |+
4       [{{ rule.id }}]
5         search = {{ query }} | eval rule="{{ rule.id }}", title="{{ rule.title }}"
6           collect index=notable_events
7           description = {{ rule.description }}
8
9 finalizers:
10  - type: concat
11    prefix: |
12      [default]
13      cron_schedule = */15 * * * *
14      dispatch.earliest_time = -20m@m
15      dispatch.latest_time = -5m@m
```

Use the rule description as description for the saved query.

Definition of the *output finalization* stage transformations. Here it's a simple concatenation of all post-processed queries with a prefix for the whole output.

The static prefix defines the scheduling and search window for all queries.

Advanced Post-Processing



- pySigma uses Jinja2 as templating language. This enables:
 - Filtering and transforming values in the template
 - Conditionals:
 - Additional actions like sending out mails or push notifications on critical severity matches.
 - Different scheduling and search windows for different log sources. Real-time ingested logs vs delayed ingestion in batches.
- Processing pipeline including variables is accessible. Use this to configure post-processing outside of the template.
- Future plan: HTTP API transformation to directly interface to SIEMs and EDRs.

Supporting new Query Languages with a Custom Backend



- Problem: Your query language is not yet supported by a backend.
- Solution: Create a new backend.
- This requires some Python development.
- In many cases you don't need to dive deep into Python code but only to declare tokens and properties of the query language.
- There's a backend template: <https://backend-template.sigmahq.io>

Using the Template



- The project's dependencies and packaging will be managed by Poetry.
Install it:

```
pipx install poetry
```

- It's a cookiecutter template, so you need to install cookiecutter:

```
pipx install cookiecutter
```

or

```
python3 -m pip install --user cookiecutter
```

- When cookiecutter is ready clone the template with:

```
cookiecutter
```

<https://github.com/SigmaHQ/cookiecutter-pySigma-backend.git>



Backend Creation from Template

Most of these values are derived from target_name. Usually the default values are sufficient and can simply be confirmed.

This should be a well-considered selection because the license impacts the usability as well as how others changes have to be contributed back.

You can also use a different license than offered here. Just replace the license references in your backend.

```
$ cookiecutter https://github.com/SigmaHQ/cookiecutter-pySigma-backend.git
target_name [Target query language name]: Foobar
backend_package_name [foobar]:
backend_name [Foobar]:
backend_class_name [FoobarBackend]:
Select package_type:
1 - backend
2 - pipeline
Choose from 1, 2 [1]:
package_name [pySigma-backend-foobar]:
package_description [pySigma Foobar backend]
author [Thomas Patzke]:
email [thomas@patzke.org]
Select license:
1 - LGPL-3.0-only
2 - MIT
3 - Apache-2.0
Choose from 1, 2, 3 [1]:
github_account [SigmaHQ]:
test_badge [True]:
coverage_badge [True]:
coverage_gist [GitHub Gist identifier containing coverage badge JSON expected by shields.io.]: xxx
status_badge [True]:
Select status:
1 - pre--release-orange
2 - release-green
Choose from 1, 2 [1]:
additional_output_formats [False]:
output_formats [format1 format2]:
Using version ^0.10.5 for pysigma
```

This is the most important value, choose it wisely!

Change these unless we share the same name or you're a clone of me 😊

Lots of details: where will your backend be hosted? Test status, coverage and project status badges. Output formats (use post-processing where possible!).

The template setups the backend source



Backend Structure

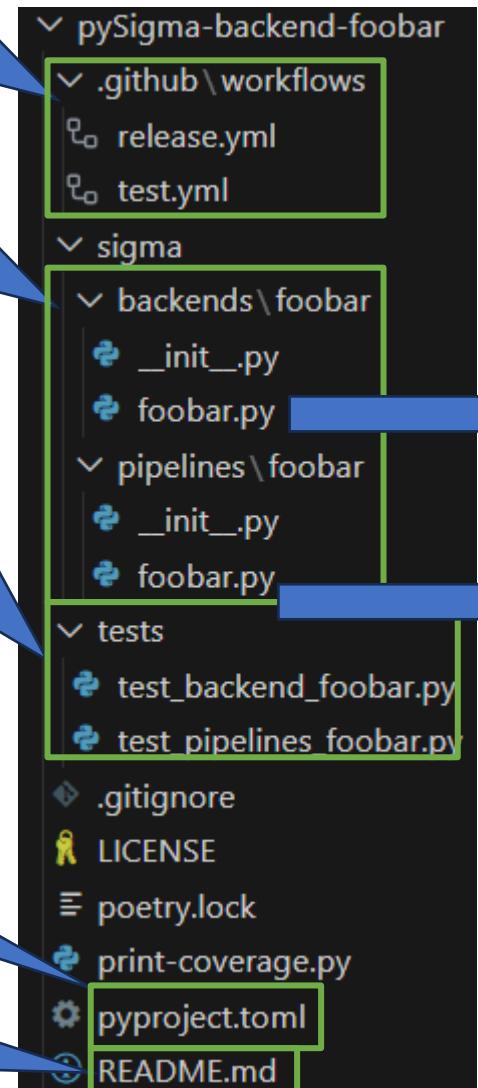
GitHub Actions configuration for CI testing and PyPI release.

The core of your backend! Open these and implement it here.

CI tests implemented with pyTest. Strong recommendation to implement them!

Project metadata: dependencies, license etc.

Likely the first thing people will see. Put some work into it.



```
class FoobarBackend(TextQueryBackend):
    """Foobar backend."""
    name : ClassVar[str] = "Foobar backend"
    formats : Dict[str, str] = { ... }
    requires_pipeline : bool = False # ...

    precedence : classVar[Tuple[ConditionItem, ConditionItem]] = ...
    group_expression : ClassVar[str] = "{expr}"

    # Generated query tokens
    token_separator : str = " " # separator between tokens
    or_token : ClassVar[str] = "OR"
    and_token : ClassVar[str] = " "
    not_token : ClassVar[str] = "NOT"
    eq_token : ClassVar[str] = "=" # Token inserted before value

@Pipeline
def foobar_pipeline() -> ProcessingPipeline: # Processing pipelines should
    return ProcessingPipeline(
        name="Foobar example pipeline",
        allowed_backends=frozenset(),
        priority=20, # The priority defines the order pipelines run
        items=[
            ProcessingItem( # This is an example for processing items generated by logsource_windows
                identifier=f"foobar_windows_{service}",
                transformation=AddConditionTransformation({ "source": source}),
                rule_conditions=[logsource_windows(service)],
            )
            for service, source in windows_logsource_mapping.items()
        ] +
        [
            ProcessingItem( # Field mappings
                identifier="foobar_field_mapping",
                transformation=FieldMappingTransformation({
                    "EventID": "event_id", # TODO: define your own field mapping
                })
            )
        ],
    ),
```

A code snippet showing the implementation of the 'FoobarBackend' class and its 'foobar_pipeline' processing pipeline. The 'FoobarBackend' class inherits from 'TextQueryBackend' and defines various configuration options like 'name', 'formats', and 'requires_pipeline'. It also defines query tokens for OR, AND, NOT, and EQUAL operations. The 'foobar_pipeline' is defined as a ProcessingPipeline with a specific name, priority, and items. The items consist of ProcessingItems that map log sources to fields, with some transformations and conditionals applied.

Implement your query language by providing the tokens and configuration. You can also override methods to implement advanced behavior.

Processing pipelines are implemented as Python code. Advantages: looping over log sources, efficiency etc. Results in the same data structure as if written in YAML.

Integrating the new backend into pySigma and Sigma CLI



- By default the backend is set up as Python package that can be installed into a Sigma CLI virtual Python environment.
- If you installed it with pipx use:
`pipx inject sigma-cli <path to your backend source directory>`
- If you are otherwise in an activated Python virtual environment with installed Sigma CLI use:
`pip install <path to your backend source directory>`
- Try if it works. After changes to the backend source code you have to reinstall it in your Sigma CLI virtual Python environment with the above commands.



Publishing your backend

- Your backend works? Great! Consider publishing it as open source project 
- Ideally: PyPI package because you can define clear releases.
- It's also possible to install backends from a public source location.
- Sigma CLI (and hopefully other tools) can find backend by the pySigma plugin directory: <https://pysigma-plugin-directory.sigmahq.io>
- Create a pull request with a description of your plugin in the directories JSON.

```
"4af37b53-f1ec-4567-8017-2fb9315397a1": {  
    "id": "splunk",  
    "type": "backend",  
    "description": "Splunk backend for conversion into SPL and tstats data model queries as plain queries and savedsearches.conf",  
    "package": "pysigma-backend-splunk",  
    "project-url": "https://github.com/SigmaHQ/pySigma-backend-splunk",  
    "report-issue-url": "https://github.com/SigmaHQ/pySigma-backend-splunk/issues/new",  
    "state": "stable",  
    "pysigma-version": "~=0.10.3"  
},
```

The Open Source Sigma Rule Repository



- URL: <https://rules.sigmahq.io>
- Rules are DRL-licensed (Detection Rule License): use as you want, even commercially. Just mention the rule author in a visible way (UI, not hidden in some text files on the hard disk).
- All contributions are automatically and manually checked against common bad practices, high detection count on clean system logs and much more.
- Rule packages for different purposes are offered as releases.
- Some statistics (averages): 15k clones/day, 150 unique cloners/day, 15k viewers/day and 1k visitors/day



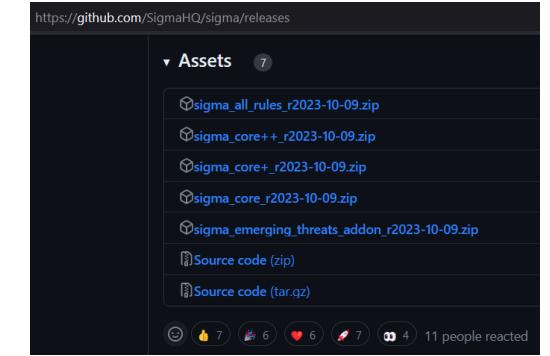
Rule Packages

Base:

- Core: high quality+confidence, low FP ratio. Severity \geq high, Status testing or stable.
- Core+: includes medium severity rules. Tuning needed.
- Core++: includes experimental rules. More FPs.

Add-on:

- Emerging Threats: specific current threats, low FP ratio.
- All Rules: All rules with severity \geq medium and status from experimental. High FP ratio.



Common Mistakes



- **Using detection rules without ingesting the corresponding logs** is just as pointless as it sounds. Sigma itself don't detects anything and don't provides any logs.
- **Ingesting all rules into your SIEM** will certainly destroy it. Choose carefully according to existing detection gaps, realistic threats and what you and your SIEM are able to handle.
- **Alerting on each Sigma rule match** will certainly destroy your analysts. Not all rules are intended for detection, some are more suitable for highlighting interesting events in log analysis. Look at the severity, use Sigma rule packages!
- **Applying the wrong or no processing pipelines** will result in no detections because generated queries don't match anything. Conversion must be configured properly and tested!



Further Resources

- Sigma website: <https://sigmahq.io>
- Rule repository: <https://rules.sigmahq.io>
- Sigma CLI: <https://cli.sigmahq.io>
- Web Converter: <https://sigconverter.io/>
- Sigma training by Applied Network Defense:
<https://www.networkdefense.co/courses/sigma/>

Want to Support Sigma?



- Contribute rules to the rules repository.
 - Most simple way: update rule status to stable if you use it in production.
 - Add known false positives: <https://fp.sigmahq.io>
 - Add rules for new or missing threats.
- Provide feedback on rules and code by opening GitHub issues or pull requests.
- Contribute backends and pipelines.
- Contribute to pySigma or Sigma CLI.
 - Please reach out before implementing new features. Some good ideas are intentionally not implemented for good reasons.
- Sponsor:
 - <https://github.com/sponsors/thomaspatzke>
 - Check out Sigma-related projects like backends etc. for further sponsoring opportunities!

The End!



Follow-up questions? Contact me:

- thomas@patzke.org
- [@thomaspatzke](https://infosec.exchange/@thomaspatzke)

Please give feedback: <https://forms.office.com/r/xJA3PjMfYh>



Sponsor my work on Sigma with GitHub Sponsors:
<https://sponsor.sigmahq.io>

