Sigma and Sigma Rules

Sigma is a generic signature format used for describing detection rules for log analysis and SIEM systems. It allows SOC analysts to create and share rules that help identify specific patterns or behaviors indicative of security threats or malicious activities. Sigma rules are typically written in YAML format and can be used with various security tools and platforms.

SOC analysts use Sigma rules to define and detect security events by analyzing log data generated by various systems, such as firewalls, intrusion detection systems, and endpoint protection solutions. These rules can be customized to match specific use cases and can include conditions, filters, and other parameters to determine when an event should trigger an alert.

The main advantage of Sigma rules is their portability and compatibility with multiple SIEM and log analysis systems, enabling analysts to write rules once and use them across different platforms.

Sigma can be considered as standardized format for analysts to create and share detection rules. It helps in converting the IOCs into queries and can be easily integrated with security tools, including SIEM and EDRs. Sigma rules can be used to detect suspicious activities in various log sources. This also helps in building efficient processes for Detection as Code by automating the creation and deployment of detection rules.

Usages of Sigma

- **Universal Log Analytics Tool**: We can write detection rules once and then convert them to various SIEM and log analytics tool formats, sparing us the repetitive task of rewriting logic across different platforms.
- **Community-driven Rule Sharing**: With Sigma, we have the ability to tap into a community that regularly contributes and shares their detection rules. This ensures that we constantly update and refine our detection mechanisms.
- **Incident Response**: Sigma aids in incident response by enabling analysts to quickly search and analyze logs for specific patterns or indicators.
- **Proactive Threat Hunting**: We can use Sigma rules for proactive threat hunting sessions. By leveraging specific patterns, we can comb through our datasets to pinpoint anomalies or signs of adversarial activity.
- **Seamless Integration with Automation Tools**: By converting Sigma rules into appropriate formats, we can seamlessly integrate them with our SOAR platforms and other automation tools, enabling automated responses based on specific detections.
- **Customization for Specific Environments**: The flexibility of Sigma rules means that we can tailor them according to the unique characteristics of our environment. Custom rules can address the specific threats or scenarios we're concerned about.
- **Gap Identification**: By aligning our rule set with the broader community, we can perform gap analysis, identifying areas where our detection capabilities might need enhancement.

How Does Sigma Work?

Table of Contents

Introduction to YARA & Sigma

Leveraging YARA

YARA and YARA Rules

Developing YARA Rules

Hunting Evil with YARA (Windows Edition)

Hunting Evil with YARA (Linux Edition)

Hunting Evil with YARA (Web Edition)

Leveraging Sigma

Sigma and Sigma Rules

Developing Sigma Rules

Hunting Evil with Sigma (Chainsaw Edition)

Hunting Evil with Sigma (Splunk Edition)

Skills Assessment

Skills Assessment

How Does Sigma Work?

At its heart, Sigma is about expressing patterns found in log events in a structured manner. So, instead of having a variety of rule descriptions scattered in various proprietary formats, with Sigma, we have a unified, open standard. This unified format becomes the lingua franca for log-based threat detection.

Sigma rules are written in YAML. Each Sigma rule describes a particular pattern of log events which might correlate with malicious activity. The rule encompasses a title, description, log source, and the pattern itself.

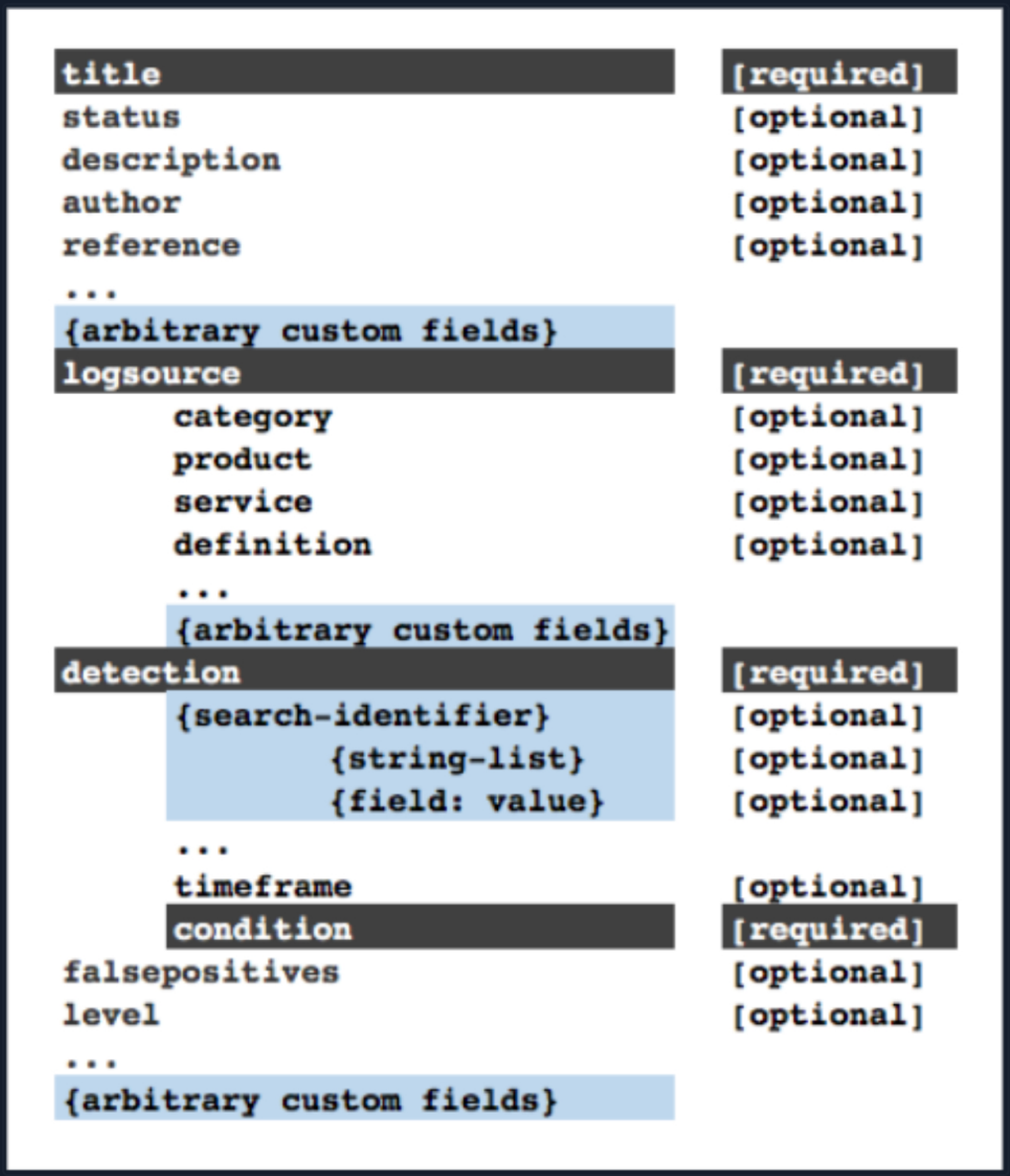
But here comes the magic. The true power of Sigma lies in its convertibility. We might ask, "If it's a standard format, how do we use it with our specific logging tools and platforms?" That's where the Sigma converter (`sigmac`) steps in. This converter is the linchpin of our workflow, transforming our Sigma rules into queries or configurations compatible with a multitude of SIEMs, log management solutions, and other security analytics tools.

With `sigmac`, we can take a rule written in the Sigma format and translate it for ElasticSearch, QRadar, Splunk, and many more, almost instantaneously.

Note: pySigma is increasingly becoming the go-to option for rule translation, as `sigmac` is now considered obsolete.

Sigma Rule Structure

As mentioned already, Sigma rule files are written in YAML format. Below is the structure of a Sigma rule.



Source: https://github.com/SigmaHQ/sigma/wiki/Specification

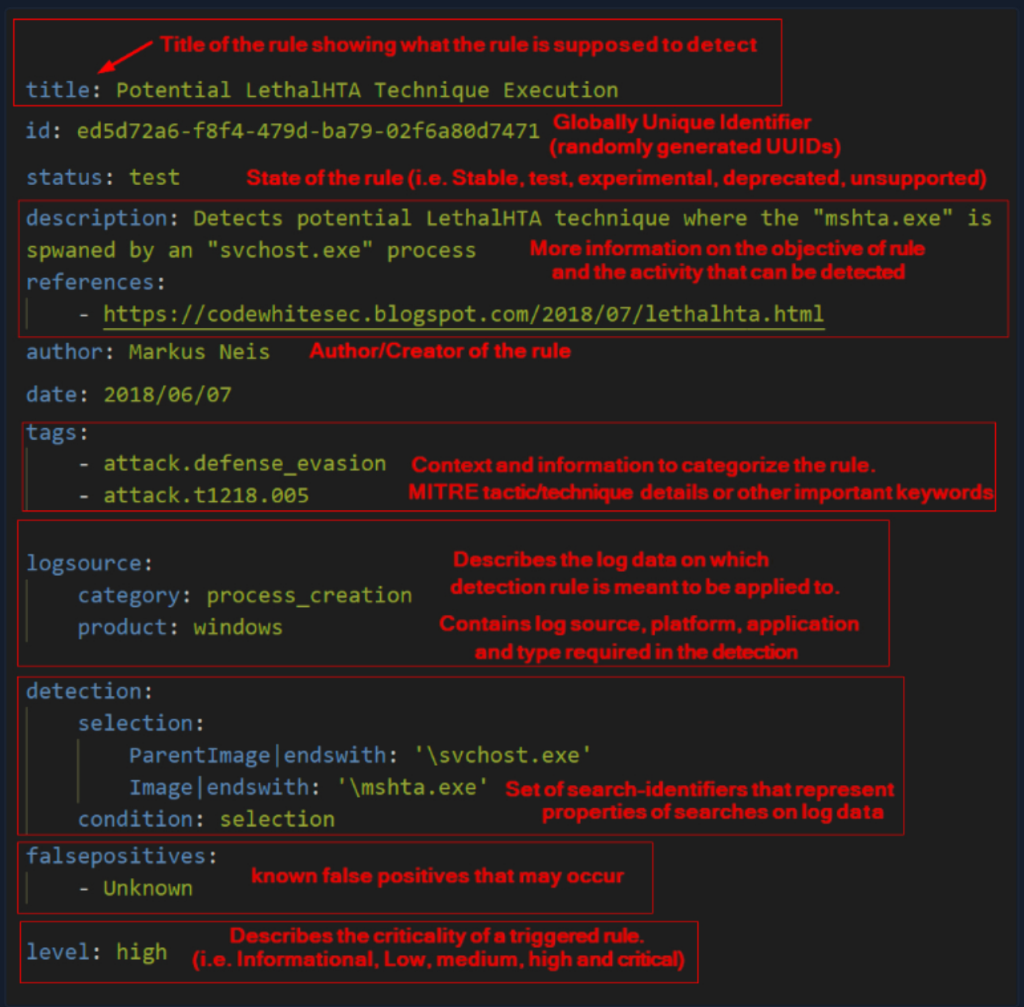
Let's understand the structure of a Sigma rule with an example

Let's understand the structure of a Sigma rule with an example:

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
description: Detects potential LethalHTA technique where "mshta.exe" is spawned by an "svchost.exe"
references:
    - https://codewhitesec.blogspot.com/2018/07/lethalhta.html
author: Markus Neis
date: 2018/06/07
tags:
    - attack.defense_evasion
    - attack.t1218.005
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        ParentImage|endswith: '\svchost.exe'
        Image|endswith: '\mshta.exe'
    condition: selection
falsepositives:
    - Unknown
level: high
```

The below screenshot shows the different components that form a Sigma Rule:



Sigma Rule Breakdown (based on Sigma's specification):

- `title`: A brief title for the rule that should contain what the rule is supposed to detect (max. 256 characters)

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
...
```

- id: Sigma rules should be identified by a globally unique identifier in the id attribute. For this purpose randomly generated UUIDs (version 4) are recommended but not mandatory.

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
...
```

- status (optional): Declares the status of the rule.
  - stable: The rule didn't produce any obvious false positives in multiple environments over a long period of time
  - test: The rule doesn't show any obvious false positives on a limited set of test systems
  - experimental: A new rule that hasn't been tested outside of lab environments and could lead to many false positives
  - deprecated: The rule is to replace or cover another one. The link between rules is made via the related field.
  - unsupported: The rule can not be used in its current state (special correlation log, home-made fields, etc.)

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
...
```

- description (optional): A short description of the rule and the malicious activity that can be detected (max. 65,535 characters)

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
description: Detects potential LethalHTA technique where "mshta.exe" is spawned by an "svchost.exe"
...
```

- references (optional): Citations to the original source from which the rule was inspired. These might include blog posts, academic articles, presentations, or even tweets.

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
description: Detects potential LethalHTA technique where "mshta.exe" is spawned by an "svchost.exe"
references:
    - https://codewhitesec.blogspot.com/2018/07/lethalhta.html
...
```

- `author` (optional): Creator of the rule (can be a name, nickname, twitter handle, etc).

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
description: Detects potential LethalHTA technique where "mshta.exe" is spawned by an "svchost.exe"
references:
    - https://codewhitesec.blogspot.com/2018/07/lethalhta.html
author: Markus Neis
...
```

- `date` (optional): Rule creation date. Use the format `YYYY/MM/DD`.

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
description: Detects potential LethalHTA technique where "mshta.exe" is spawned by an "svchost.exe"
references:
    - https://codewhitesec.blogspot.com/2018/07/lethalhta.html
author: Markus Neis
date: 2018/06/07
...
```

- `logsource`: This section describes the log data on which the detection is meant to be applied to. It describes the log source, the platform, the application and the type that is required in the detection. More information can be found in the following link: https://github.com/SigmaHQ/sigma/tree/master/documentation/logsource-guides

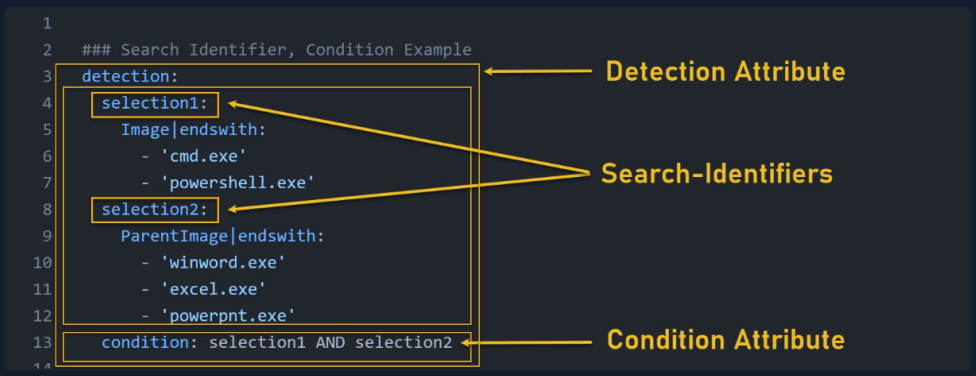
  It consists of three attributes that are evaluated automatically by the converters and an arbitrary number of optional elements. We recommend using a "definition" value when further explanation is necessary.

  - `category`: The `category` value is used to select all log files written by a certain group of products, like firewalls or web server logs. The automatic converter will use the keyword as a selector for multiple indices. Examples: `firewall`, `web`, `antivirus`, etc.
  - `product`: The `product` value is used to select all log outputs of a certain product, e.g. all Windows event log types including `Security`, `System`, `Application` and newer types like `AppLocker` and `Windows Defender`. Examples: `windows`, `apache`, `check point fw1`, etc.
  - `service`: The `service` value is used to select only a subset of a product's logs, like the `sshd` on Linux or the `Security` event log on Windows systems. Examples: `sshd`, `applocker`, etc.

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
description: Detects potential LethalHTA technique where "mshta.exe" is spawned by an "svchost.exe"
references:
    - https://codewhitesec.blogspot.com/2018/07/lethalhta.html
author: Markus Neis
date: 2018/06/07
logsource:
    category: process_creation
    product: windows
...
```

- `detection`: A set of search-identifiers that represent properties of searches on log data. Detection is made up of two components:
    - Search Identifiers
    - Condition



Source: blusapphire.io

Code: yaml

```yaml
title: Potential LethalHTA Technique Execution
id: ed5d72a6-f8f4-479d-ba79-02f6a80d7471
status: test
description: Detects potential LethalHTA technique where "mshta.exe" is spawned by an "svchost.exe"
references:
    - https://codewhitesec.blogspot.com/2018/07/lethalhta.html
author: Markus Neis
date: 2018/06/07
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        ParentImage|endswith: '\svchost.exe'
        Image|endswith: '\mshta.exe'
    condition: selection
...
```

The values contained in Sigma rules can be modified by value modifiers. Value modifiers are appended after the field name with a pipe character (`|`) as separator and can also be chained, e.g. `fieldname|mod1|mod2: value`. The value modifiers are applied in the given order to the value.

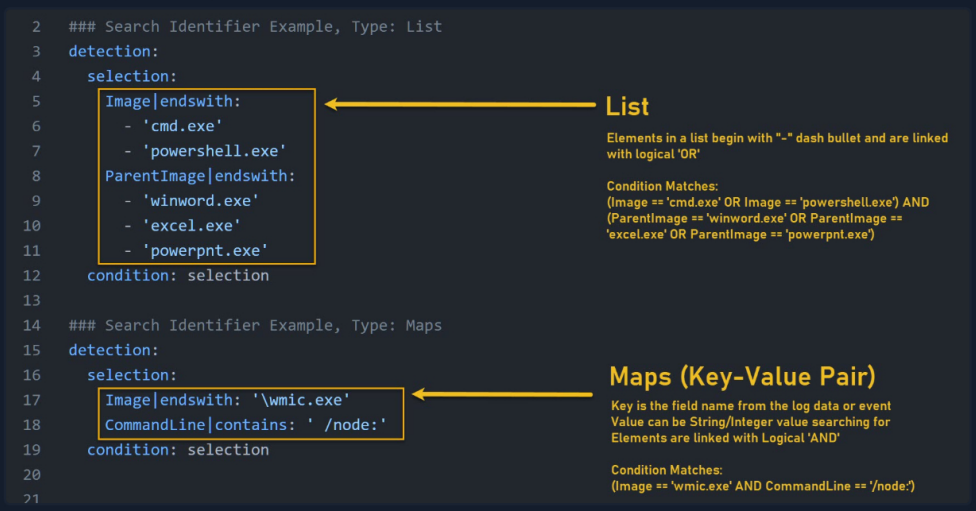
The behavior of search identifiers is changed by value modifiers as shown in the table below :

Value Modifier	Explanation	Example
contains	Adds wildcard (*) characters around the value(s)	CommandLine|contains
all	Links all elements of a list with a logical "AND" (instead of the default "OR")	CommandLine|contains|all
startswith	Adds a wildcard (*) character at the end of the field value	ParentImage|startswith
endswith	Adds a wildcard (*) character at the begining of the field value	Image|endswith
re:	This value is handled as regular expression by backends	CommandLine|re: '\[String\]\s*\$VerbosePreference

Source: blusapphire.io

Search identifiers include multiple values in two different data structures:

1. Lists, which can contain:
    - strings that are applied to the full log message and are linked with a logical OR.
    - maps (see below). All map items of a list are linked with a logical OR.



Source: blusapphire.io

Example list of strings that matches on `evilservice` or `svchost.exe -n evil`.

Code: yaml

```yaml
detection:
    keywords:
        - evilservice
        - svchost.exe -n evil
```

Example list of maps that matches on image file `example.exe` or on a executable whose description contains the string `Test executable`.

Code: yaml

```yaml
detection:
    selection:
        - Image|endswith: '\example.exe'
        - Description|contains: 'Test executable'
```

2. Maps: Maps (or dictionaries) consist of key/value pairs, in which the key is a field in the log data and the value a string or integer value. All elements of a map are joined with a logical AND.

Example that matches on event log `Security` and (`Event ID 517` or `Event ID 1102`)

Code: yaml

```yaml
detection:
    selection:
        EventLog: Security
        EventID:
          - 517
          - 1102
    condition: selection
```

Example that matches on event log `Security` and `Event ID 4679` and `TicketOptions 0x40810000` and `TicketEncryption 0x17`.

Code: yaml

```yaml
detection:
    selection:
        EventLog: Security
        EventID: 4769
        TicketOptions: '0x40810000'
        TicketEncryption: '0x17'
    condition: selection
```

- Condition: Condition defines how fields are related to each other. If there's anything to filter, it can be defined in the condition. It uses various operators to define relationships for multiple fields which are explained below.

Operator	Example
Logical AND/OR	`keywords1 or keywords2`
1/all of them	`all of them`
1/all of search-identifier-pattern	`all of selection*`
1/all of search-id-pattern	`all of filter_*`
Negation with 'not'	`keywords and not filters`
Brackets - Order of operation '()'	`selection1 and (keywords1 or keywords2)`

Source: blusapphire.io

Example of a condition:

Code: yaml

```yaml
condition: selection1 or selection2 or selection3
```

Sigma Rule Development Best Practices

Sigma's specification repository contains everything you may need around Sigma rules.

Sigma rule development best practices and common pitfalls can be found on Sigma's Rule Creation Guide.