Sigma

SIEM Detection Format

Lessons Learned

from (almost) 8 Years of Sigma Development

Thomas Patzke, 2024-10-22

Sigma Introduction

- Generic language for log detection rules.
- Rules are converted into target query language and data model.
- Content: detection rule repository with 2.000+ rules
- Code: toolchain for parsing, transformation and conversion of rules.
- https://sigmahq.io/

```
title: Potential Obfuscated Ordinal Call Via Rundll32
id: 43fa5350-db63-4b8f-9a01-789a427074e1
status: test
description: Detects execution of "rundll32" with potent
references:
    - Internal Research
author: Nasreddine Bencherchali (Nextron Systems)
date: 2023-05-17
Add Tag
tags:
    - attack.defense-evasion
logsource:
    category: process_creation
    product: windows
Look Up
detection:
    selection_img:
        - Image|endswith: '\rundll32.exe'
        - OriginalFileName: 'RUNDLL32.EXE'
        - CommandLine|contains: 'rundll32'
    selection_cli:
        CommandLine|contains:
            - '#+'
            - '#-'
    condition: all of selection_*
falsepositives:
    - Unknown
level: medium
```

Project History

2016/17: Start

- Everything in one Repository.
- Monolithic PoC-grade Toolchain code (Sigmac)

2020: Restructure & Rewrite

- Splitting rules, repositories, backends and pipelines into separate projects.
- Toolchain rewrite: pySigma and Sigma CLI

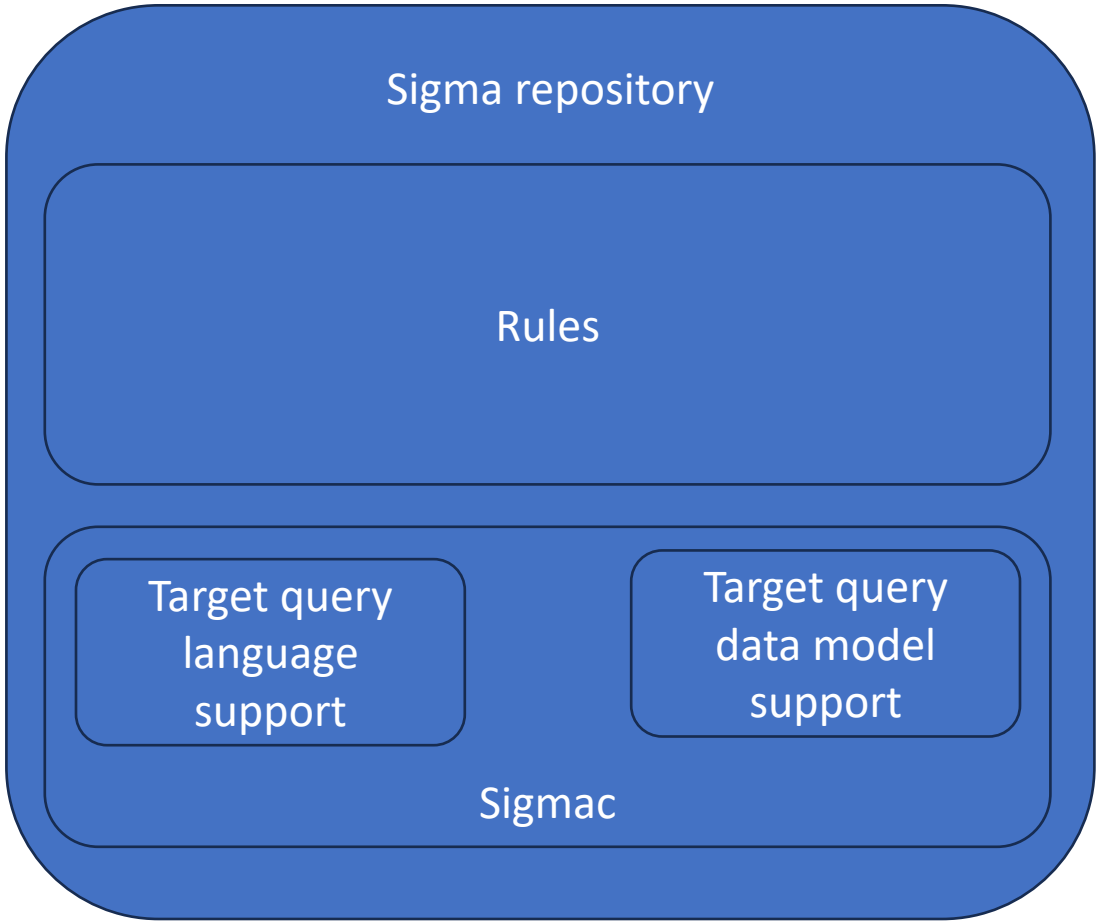
2023/24: Evolution

- Correlations
- Filters
- Query postprocessing
- …

Project Structure: How it has started

- Started with everything in one repository with code and content.
- Sigmac (Sigma Converter) as monolithic script.
- Worked very well in the beginning.

Ferdinand Reus / CC-BY-SA-2.0



The good and the bad…

Advantages

- User friendly: one project contains everything.
- Low overhead
- Dependent changes are done quickly
- No coordination between related projects.

Disadvantages

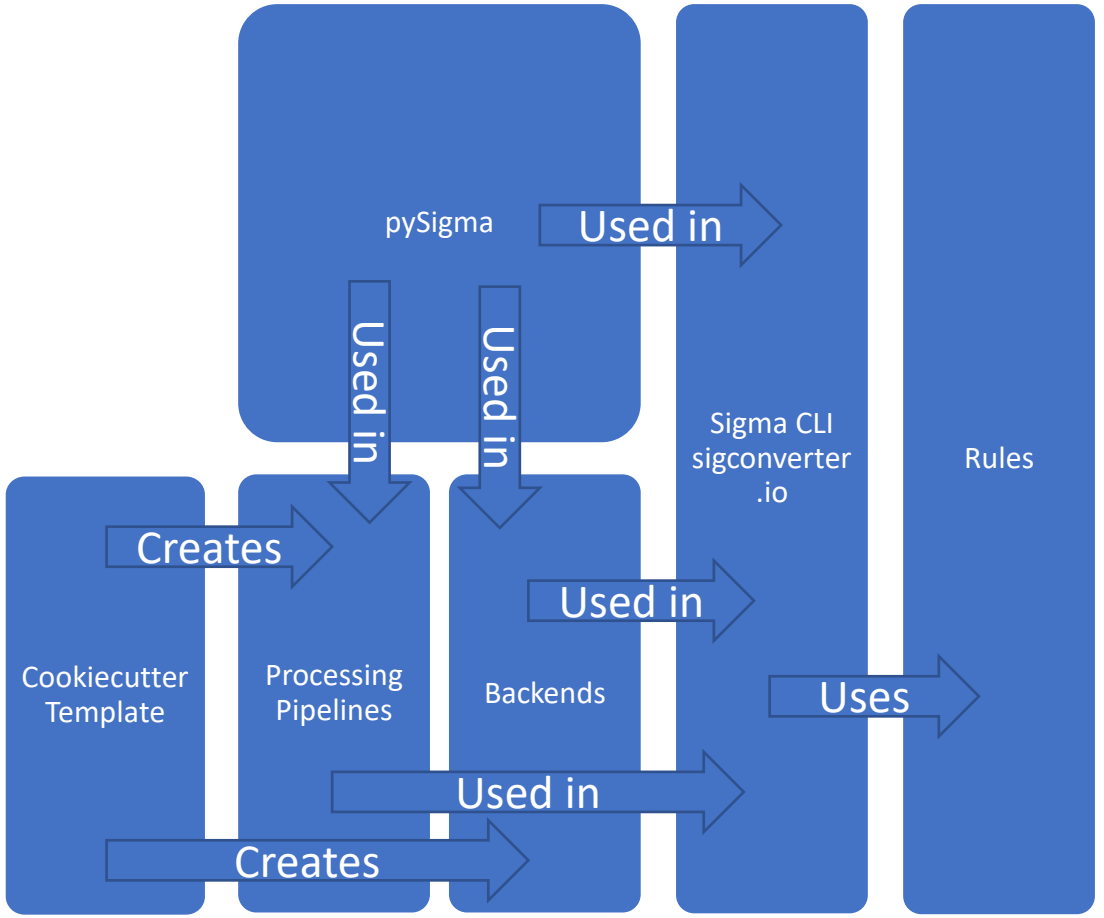
- Everything is mixed up, contributors and maintainers lose track.
- No ownership and lack of responsibility.
- No choice about contribution location.
- What does a release include?

Keep vs Rewrite

- Decision in 2020: rewrite the whole toolchain
- Rewrite from scratch: throw most that exists away.
- Opportunity to not making some mistakes again.
- Opportunity to adopt good practices from the beginning.
- Huge win: test-driven development
  - Before: code mostly without tests. Even smaller changes often broke something.
  - Now: >1.000 tests. Things broken by changes are directly visible (mostly).
  - Enables quick integration of contributions and new features.
  - Increases code quality.
- Drawback: few backends are still only available in legacy toolchain.

Project Structure: How it‘s gone

- Separation of code and content.
- Further separation of code into manageable sub-projects with pySigma at its core.
- Core components under SigmaHQ org, some backends are hosted/published independently.



Project Structure: Lessons Learned

- Starting in one big repository enabled us to quickly move forward with major changes.
- Increased complexity, decreased quality and development pace.
- Rewrite/split increased adoption of pySigma in other tools (e.g. sigconverter.io) and use cases (lots of private setups).
- Increased effort. Major/breaking changes now need release of several projects.
- Dependency complexity: backend depends on specific pySigma version, CLI on another.

License: How it has started

- GPL all the things!

But…

- How to integrate Sigma into a commercial product or context?
- How can the converted query be used?
- Must all changes (especially environmental baselining) to the detection be contributed back to the Sigma rule?

Licenses: how it‘s gone

- Specification is public domain
- Repository with detections is DRL-licensed (Detection Rule License)
  - Contributors agreed that it should be permissive.
  - Attribution of rule authors (and not the Sigma project, SigmaHQ etc.)
  - Attribution in the UI, not somewhere deep inside a program directory.

Toolchain for parsing, transformation and conversion of Sigma rules:

- Library pySigma and CLI (LGPL)
- Backends + Processing Pipelines: it depends

Lessons Learned: Licenses

- Choose wisely!
- Switching licenses is challenging.
- License must match the use case (code vs detection rules).
- Consider carefully if you really need a custom license!

Releases & Contributions

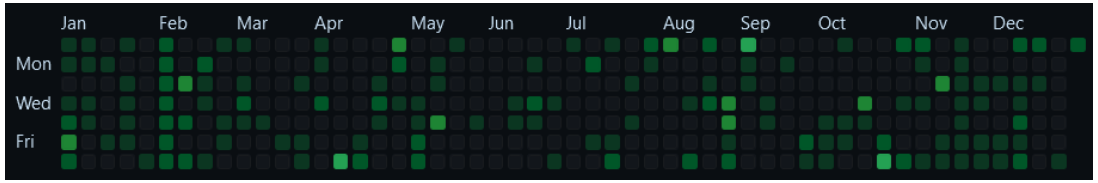
- Release early, release often!
- Contributions:
  - (Un)maintained?
  - Fix/small improvement or major change or new concept?
  - Balance between ensuring quality and tolerance.
  - Turning PRs that don’t meet requirements to such that do.
- Roadmap?
  - Yes, plans about features to be implemented
  - No dates or timeline

Staying Motivated

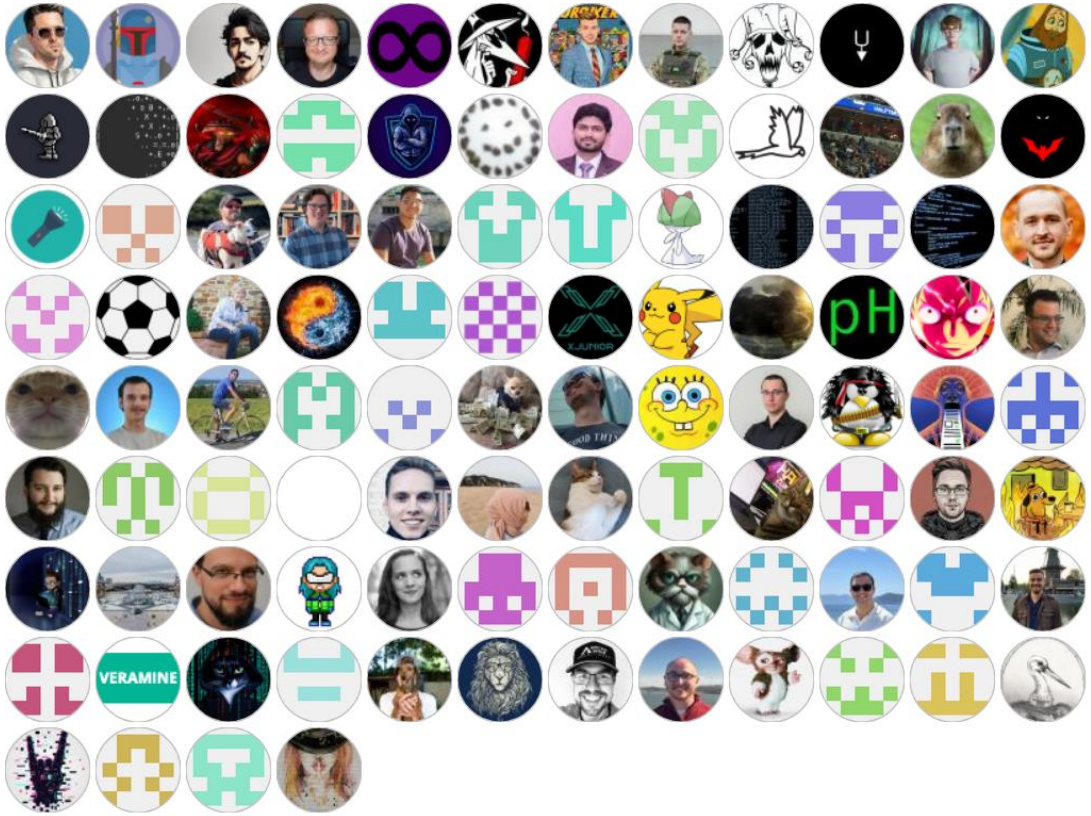
Developing & maintaining an open source project is…

- Fixing issues, sometimes very subtle ones.
- Support in solving problems
- Lots of communication
- Developing new features & Improving existing.

- Contribute
- Feedback (+ & -)
- Sponsoring
- Don’t stress yourself, take breaks!



Thanks & Contact



thomas@patzke.org
@thomaspatzke@infosec.exchange
https://github.com/thomaspatzke
https://sigmahq.io/