Log Hunting with Sigma

A hands-on workshop to Sigma and the toolchain

Thomas Patzke

Prerequisites

Requirements:

- Python 3.6
  https://www.python.org/downloads/release/python-369/
- Docker CE (current version)
- Clone of the Sigma workshop repository:
  https://github.com/thomaspatzke/sigma-workshop
  ```
  git clone --recursive \
  https://github.com/thomaspatzke/sigma-workshop.git
  ```
- Sigma dependencies:
  ```
  pip3 install -r sigma/tools/requirements.txt
  ```
  or `apt-get install python3-yaml`
- Elasticsearch and Kibana with log data:
  - `docker-compose -f es_kibana.docker-compose.yml up`
  - `./sigma_workshop_prepare_es.sh`
- MISP
  - `git clone https://github.com/DCSO/MISP-dockerized.git`
  - `make install`

Overview

- A short introduction to Sigma
- Writing a log Signature for:
  - Execution of a Mimikatz release binary (process execution by hash)
  - Common parameter usage of NirSoft’s NetPass tool (process execution by command line)
  - WCE LSASS injection behaviour
  - Encoded commands
- Building a Sigma Converter configuration
- Convert to Elasticsearch query and search in ELK instance
- Rule collections
- Generic log sources
- Value modifiers
- Importing Sigma rules in MISP
- Further tools

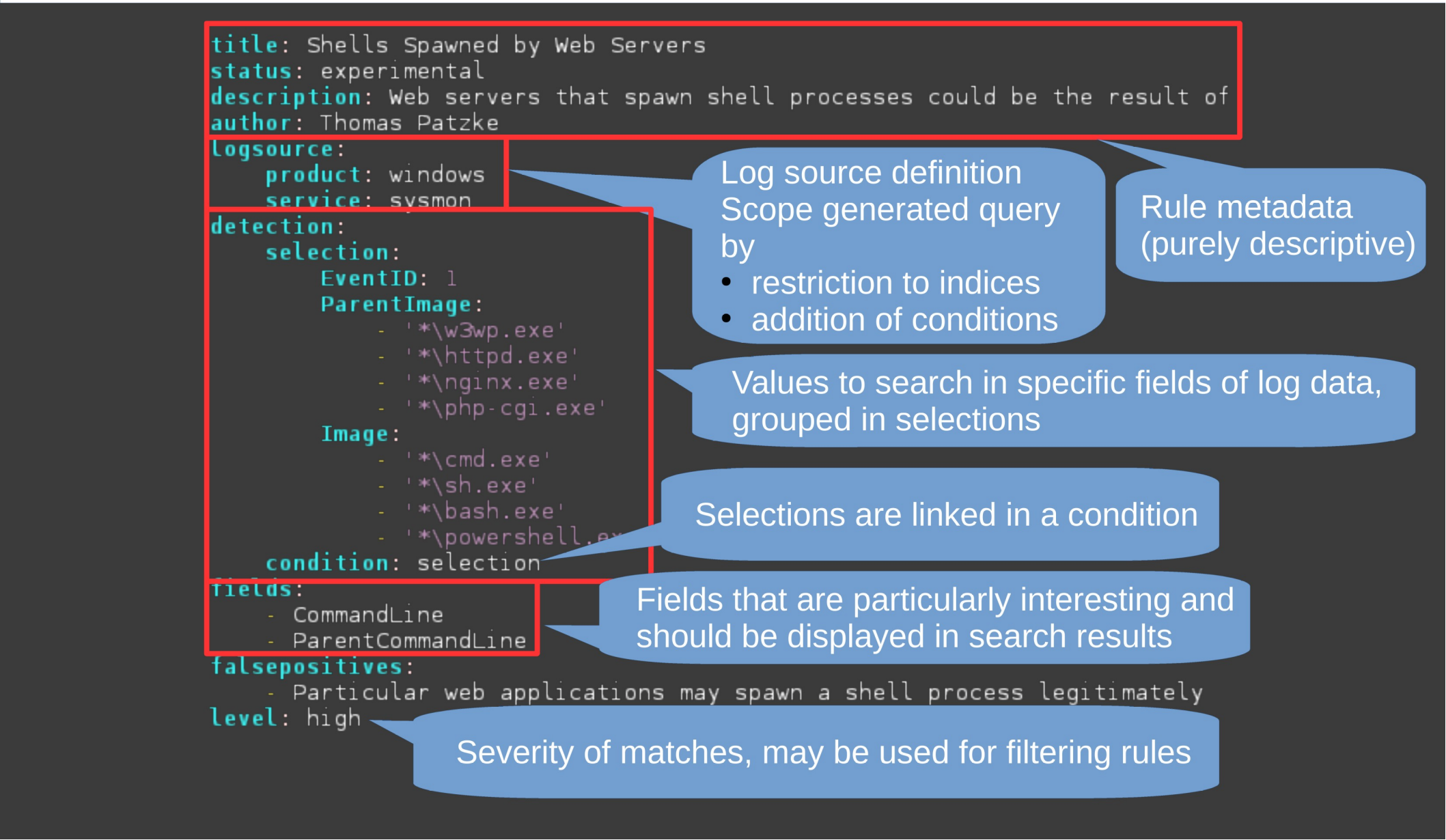
Sigma Goals and Scope

- Being human-writable and readable
  - No XML or JSON, no deeply nested structures
- Machine-readable and writable
  - YAML, no ambiguities
- Simpleness
  - Expressiveness for ~95% of log signatures
  - NOT: description of every imaginable SIEM use case or threat hunting technique
  - It should be relatively easy to build an own Sigma parser
- Tooling: it should be practicably usable, not just theory

Sigma Introduction

- Generic signature format for description of log events
  - YAML-based
  - Indicators: Key-Value, Key-List and Value-only
  - Conditions and aggregations
  - Meta-data: Title, description, authors, tags (ATT&CK), severity, ...
- Conversion tool sigmac
- Workflow:

Sigma Rule – Example 1



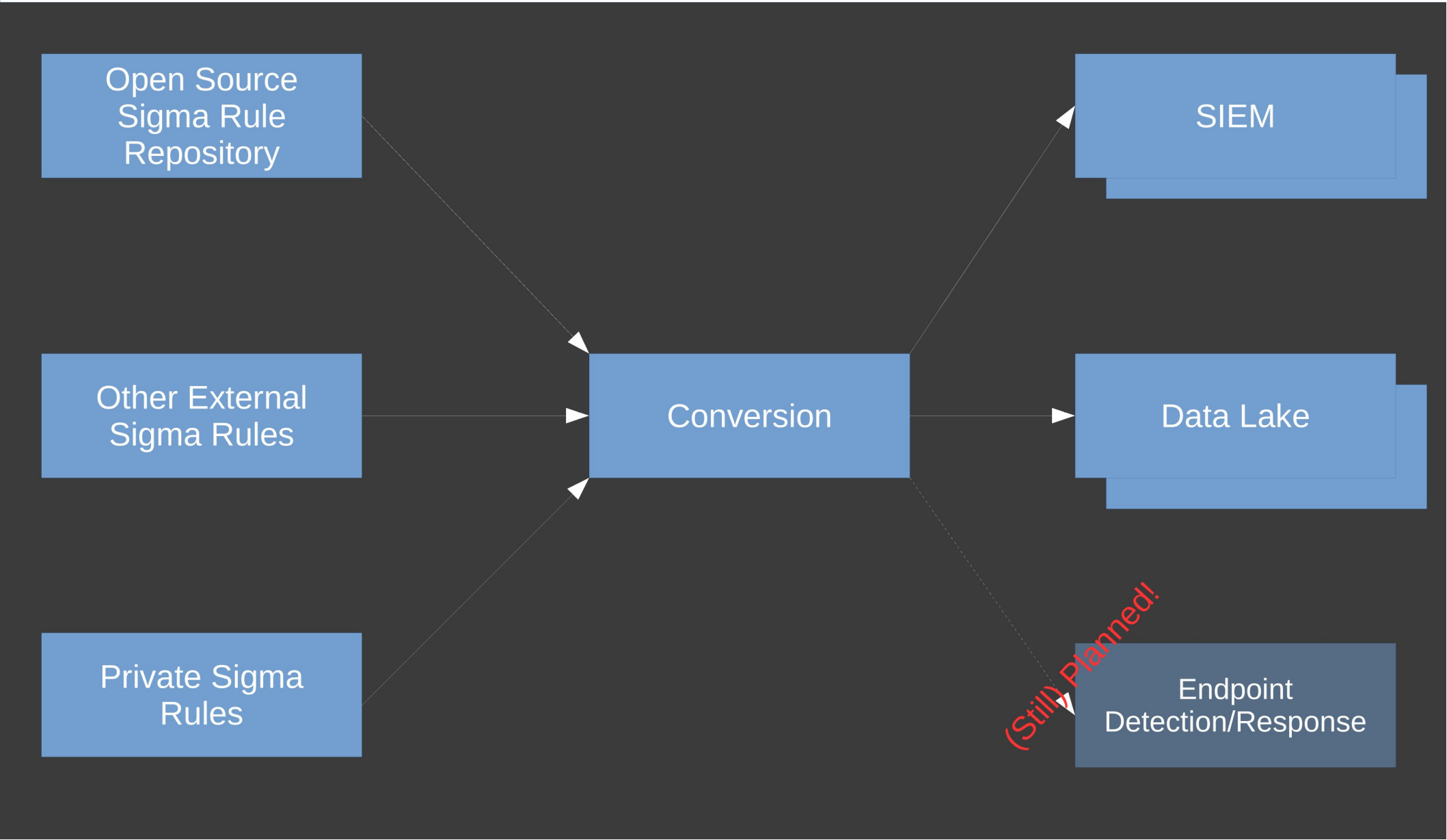
Sigma Rule – Example 2

```
title: Rundll32 Internet Connection
status: experimental
description: Detects a rundll32 that com
references:
    - https://www.hybrid-analysis.com/sa
author: Florian Roth
date: 2017/11/04
tags:
    - attack.t1085
    - attack.defense_evasion
    - attack.execution
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 3
        Image: '*\rundll32.exe'
    filter:
        DestinationIp:
            - '10.*'
            - '192.168.*'
            - '172.*'
    condition: selection and not filter
falsepositives:
    - Communication to other corporate s
level: medium
```

Tagging of rules with ATT&CK tactics, techniques and software tags. Can be used for filtering of rules.

Usual condition: search for selection and filter uninteresting events.

Possible Setup



Advantages

- Reduction of vendor lock-in
- Distribution of log signatures in heterogeneous environments or in blog posts/threat intel products
- Build one rule and use it in your SIEM, alerting, endpoint security solution or even for grepping in log files and querying from PowerShell
- 300+ open source Sigma rules in GitHub repository
- Evolving tool/services support: MISP conversion extension, online editor, ...
- Intermediate language for generation of queries from IOCs in other formats
- Increasing community contribution

Enough Theory!

Let’s get our hands dirty!

Exercise 1
Mimikatz Release Binary

- Let’s assume we’re targeted by an attacker who is known to use the Mimikatz 2.1.1 release
- SHA256 hashes (see challenges/1-Mimikatz_2.1.1_Hashes.txt):
  - 97f93fe103c5a3618b505e82a1ce2e0e90a481aae102e52814742baddd9fed41 ./Win32/mimilove.exe
  - 6bfc1ec16f3bd497613f57a278188ff7529e94eb48dcabf81587f7c275b3e86d ./Win32/mimikatz.exe
  - e46ba4bdd4168a399ee5bc2161a8c918095fa30eb20ac88cac6ab1d6dbea2b4a ./x64/mimikatz.exe
- Write a rule for Sysmon events that detects execution of the above binaries (EventID 1) by utilization of the *Hashes* field

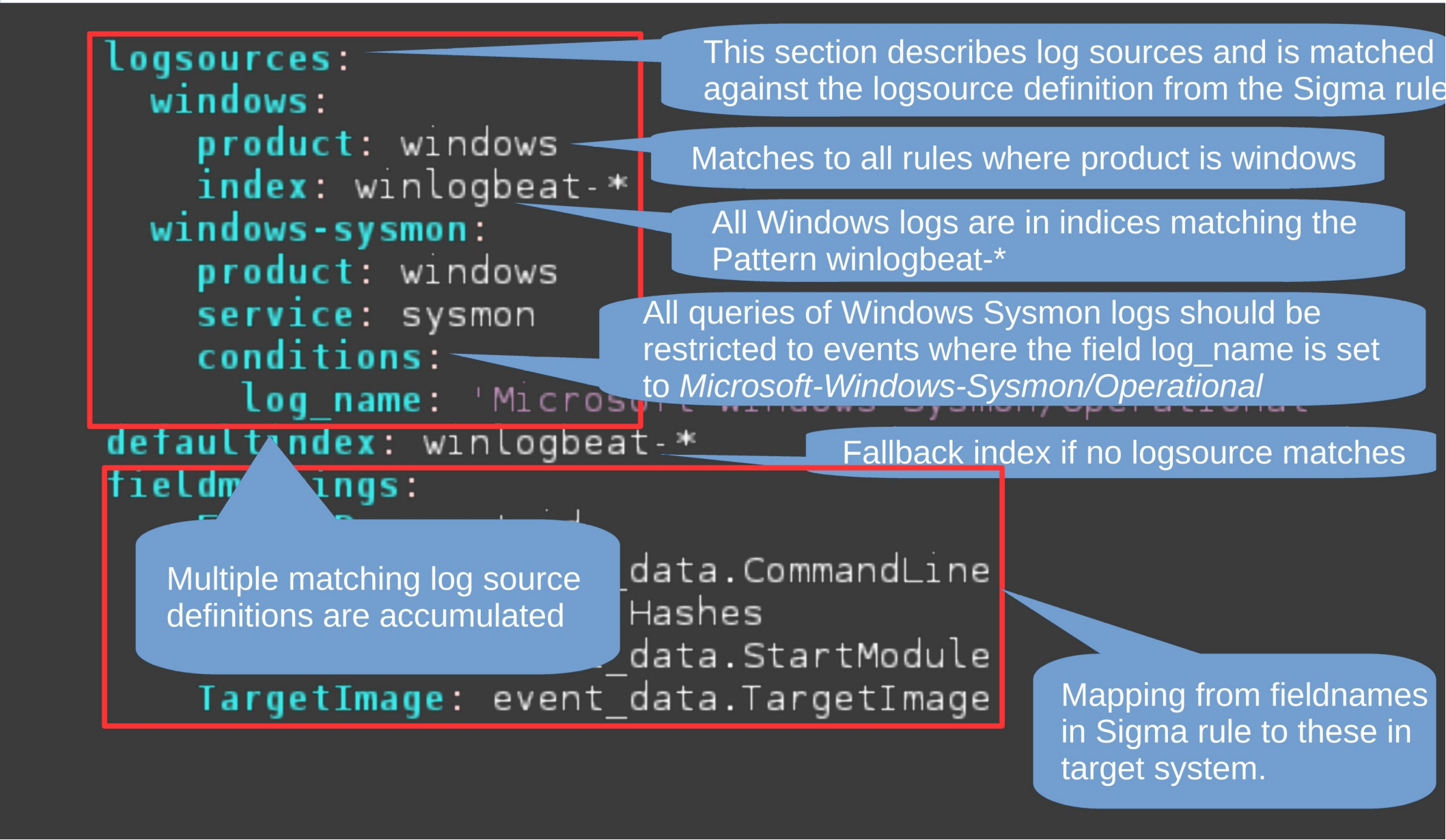
Exercise 1
Possible Solution

```yaml
title: Mimikatz detection
status: stable
description: Detects Mimikatz 2.1.1 release by recognition of executable hashes
tags:
    - attack.s0002
    - attack.t1003
    - attack.lateral_movement
    - attack.credential_access
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 1
        Hashes:
            - 97f93fe103c5a3618b505e82a1ce2e0e90a481aae102e52814742baddd9fed41
            - 6bfc1ec16f3bd497613f57a278188ff7529e94eb48dcabf81587f7c275b3e86d
            - e46ba4bdd4168a399ee5bc2161a8c918095fa30eb20ac88cac6ab1d6dbea2b4a
    condition: selection
level: high
```

Rule Conversion with Sigma Converter

- The Sigma Converter (sigmac) is located in tools/sigmac in the Sigma repository
- Run it with `--help` to get an overview
- Convert into target query language (-t) *es-qs* (Elasticsearch Query String)
- No matches! Why?
  - Sigma rules are (or should be) generic, so some further work is required
  - Mapping of field names:
    - EventID → event_id
    - Hashes → event_data.Hashes
  - EventID 1 may also appear from other sources, search needs to be restricted to log source by addition of further conditions
- Sigma conversion configuration defines the transformation

Sigma Converter Configuration



Try Again – with Configuration!

- Try to write your own configuration
- Configurations can be passed to Sigma converter with parameter -c

Exercise 2: NirSoft NetPass

- NetPass has some very characteristic parameter names: /stext, /stab, /scomma, /stabular, /shtml, /sverhtml, /sxml
- Write a rule for Sysmon process creation events and utilize the *CommandLine* field for identification of parameter usage, don’t:
  - Try to match hashes of any releases
  - Match the file name

Exercise 2
Possible Solution

```yaml
title: Detection of Nirsoft NetPass parameter usage
status: stable
description: NetPass supported some characteristic parameters
le
tags:
    - attack.credential_access
    - attack.t1003
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 1
        CommandLine:
            - "* /stext *"
            - "* /stab *"
            - "* /scomma *"
            - "* /stabular *"
            - "* /shtml *"
            - "* /sverhtml *"
            - "* /sxml *"
    condition: selection
level: high
```

Rule Collections & Generic Log Sources

- Both rules were Sysmon-specific. The last event could also be identified by a Security/4688 event (Windows Auditing).
- ...or by other sources that don’t know about these event identifiers (e.g. EDR, Windows Defender ATP, ...)
- There are two ways to express such similar events:
  - As rule collection (the old way)
    - Multiple rules in one file
    - *action* keys allow to define common parts for all rules
  - Generic log sources (the preferred way)
    - One rule, generic log source identifier
    - Converter generates the specific rule

Rule Collections

- Useful for grouping of rules. Examples:
  - APT group
  - Malware class
- Not the preferred solution for rule variants for different events/products
  - Redundancy: multiple rules for the same event
  - Inconsistency: only one rule for a event id that also may be recognized by another
  - Complex conversion (matching all EventIDs to target query language objects)

Generic Log Sources: Example

```yaml
title: Bitsadmin Download
status: experimental
description: Detects usage of bits
references:
    - https://blog.netspi.com/15-w
    - https://isc.sans.edu/diary/2
tags:
    - attack.defense_evasion
    - attack.persistence
    - attack.t1197
    - attack.s0190
author: Michael Haag
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 1
        Image:
            - '*\bitsadmin.exe'
        CommandLine:
            - '/transfer'
    condition: selection
fields:
    - CommandLine
    - ParentCommandLine
falsepositives:
    - Some legitimate apps use thi
level: medium
```

```yaml
title: Bitsadmin Download
status: experimental
description: Detects usage of bitsa
references:
    - https://blog.netspi.com/15-wa
    - https://isc.sans.edu/diary/22
tags:
    - attack.defense_evasion
    - attack.persistence
    - attack.t1197
    - attack.s0190
author: Michael Haag
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image:
            - '*\bitsadmin.exe'
        CommandLine:
            - '/transfer'
    condition: selection
fields:
    - CommandLine
    - ParentCommandLine
falsepositives:
    - Some legitimate apps use this
level: medium
```

Conversion to Generic Sigma Rule

- Tool sigma2genericsigma converts specific rules or rules collections into generic Sigma rules
  - Also supports bulk conversion of whole repositories
  - Output (-o) may be
    - single file/standard output: results in rule collection
    - Directory: one output per input file
  - Parameter -c creates list of converted files
- Try to convert rule from exercise 2 into generic Sigma rule

Generic Log Sources: Usage

- Usage: chained configurations
  1. Generic rule ➔ specific
     Process creation ➔ Sysmon/1
  2. Specific rule ➔ Environment-specific rule
     Sysmon/1 ➔ Sysmon/1 with field mappings and additional conditions
- Configurations for process creation to Sysmon and Windows Security Audit events already exist
- Let’s try it!
  - Sigma Converter with generic log source support in directory sigma_with_generic_logsources/

Exercises 3: WCE LSASS Injection

- WCE causes a burst of Sysmon CreateRemoteThread (EventID 8) events into lsass.exe (TargetImage)
- Further, some security products also inject into LSASS, but only WCE does without a *StartModule*. Filter these out.

Exercise 3
Possible Solution

```yaml
title: WCE Remote Thread Injection
status: stable
description: Detection of remote thread creation in LSASS by Windows Credential Editor
tags:
    - attack.credential_access
    - attack.t1003
      attack.s0005
logsource:
    product: windows
    service: sysmon
detection:
    selection:
        EventID: 8
        TargetImage: 'C:\Windows\System32\lsass.exe'
    filter:
        StartModule: '*'
    condition: selection and not filter
level: high
```

Introduction to Value Modifiers

Motivation:

- Writing encoded values in Sigma rules doesn’t increases readability
- Problematic encodings like UTF16
- Changing default behaviors (e.g. AND of list elements) required workarounds

Solution: Value Modifiers

- Post-processing of values
  - Encoding
  - Different logical linking of values
  - Treat as regular expression
- Syntax:
  `field name|modifier1|modifier2|...`

Existing Value Modifiers

- List provided by `sigmac -l`

```
Modifiers:
  contains : Add *-wildcard before and after all string(s)
       all : Override default OR-linking behavior for list with AND-linking of all list values
    base64 : Encode strings with Base64
base64offset : Encode string(s) with Base64 in all three possible shifted offsets
     utf16 : Encode string to UTF-16 byte sequence
   utf16le : Encode string to UTF-16 little endian byte sequence
      wide : Modifier 'wide' is an alias for the utf16le modifier.
   utf16be : Encode string to UTF-16 big endian byte sequence
        re : Treat value as regular expression
```

Excercise 4: Encoded Ping

- Ping command called from Powershell as Base64 encoded payload:
  `powershell -encoded ...`
  - It’s UTF16 encoded
  - The plain command might be prefixed or padded with something
  - Don’t make any assumption on the location of the base64 encoded part.
- Event from Endgames Red Team Automation project

Exercise 4: Possible Solution

```yaml
title: Base-64 encoded ping
status: stable
description: Detect all possibilities of a base64 encoded ping command
tags:
    - attack.t1140
    - attack.t1027
    - attack.defense_evasion
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        CommandLine|wide|base64offset|contains: 'ping'
    condition: selection
level: high
```

Handling many Sigma Rules

- Copy and pasting rules between terminal and browser is not very convenient.
- Build a Kibana import file from all previous solutions with the *kibana* backend
- Import the generated file into Kibana

Sharing Sigma Rules with MISP

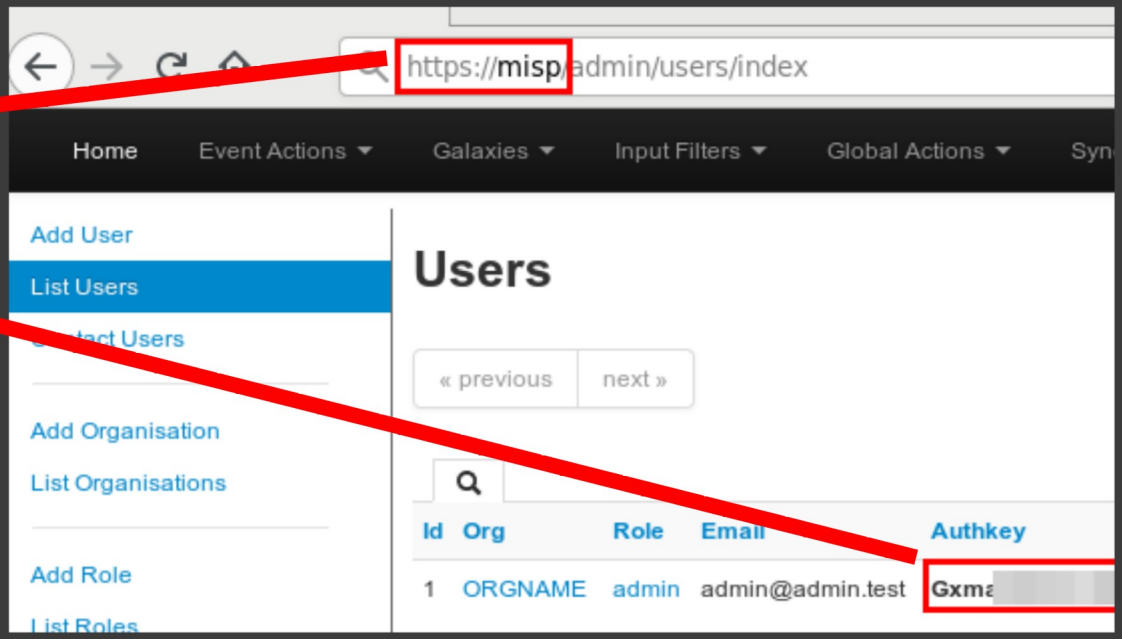
- MISP has a *sigma* attribute type
- The Sigma repository contains a tool sigma2misp for importing Sigma rules into MISP events

Configuring & Using sigma2misp

misp.conf

```
url https://misp
key Gxma…
insecure
```

Don‘t do this in production!



Example for import:

```
tools/sigma2misp @misp.conf –same-event \
  –event 42 rules/apt/apt_turla_*
```

Exercise: Import Simga Rules to MISP

- Build a config
- Import one or multiple Sigma rules into a new MISP event
- Importing Sigma rules into existing event

Further Tools

- evt2sigma: https://github.com/Neo23x0/evt2sigma
  - Convert Windows XML log entry into Sigma rule
- SPARK Core: https://www.nextron-systems.com/spark-core/
  - Host-based scanner with Sigma support
- SigmaUI: https://github.com/socprime/SigmaUI
  - Sigma editor in Kibana
- Uncoder.io
  - Convert between different languages (web based)
- A Splunk app with Sigma searches: https://github.com/dstaulcu/TA-Sigma-Searches
  - *splunkxml* backend generates something similar

Questions?

- E-Mail: thomas@patzke.org
- Open an issue or contribute on GitHub: https://github.com/Neo23x0/sigma/
- Twitter: @blubbfiction

Further Resources

- Windows Events:
  - https://github.com/MicrosoftDocs/windows-itpro-docs/tree/master/windows/security/threat-protection/auditing
  - https://www.ultimatewindowssecurity.com/securitylog/encyclopedia/
- Sigma Wiki (Specification, Taxonomy, Tags, …): https://github.com/Neo23x0/sigma/wiki
- MITRE ATT&CK: https://attack.mitre.org/
- Atomic Threat Coverage: https://github.com/krakow2600/atomic-threat-coverage