Sharing is Caring: Improving Detection with Sigma

John Hubbard (@SecHubb)
Blue Team Wins
So Everything Is Great Right?

STARTED FROM THE BOTTOM

NOW WE HERE
Network Traffic

- Full PCAP, Metadata, IDS, NetFlow
- Detection:
  - Threat data matching
  - Protocol / metadata anomaly detection
  - IDS signatures
File Analysis

• File analysis, classification and simulation
• Detection:
  • AV Signatures / ML-based detection
  • Sandboxed dynamic analysis
  • YARA signatures
Log Analysis

• Enterprise log collection + SIEM
• Detection:
  • Manual log analysis and hunting
  • ...proprietary format SIEM Analytics 😞
Log Signatures?

Why is it so difficult?
1. Even in deployments of *same SIEM*...
   * Field names differ
   * Data sources differ
2. We collect in different log *formats*:
   * Windows logs – Syslog, JSON, XML
3. We have *no common language* to specify analytics
What Would It Take?

Source_IP  
SrcIP  
src.ip  

SIEM Field Normalization

Identical Field Normalization

SIEM Alert Engine

source_ip

Generic:  
"Match if Source IP = 1.2.3.4"

SIEM Pseudo-Analytic:

if (source_ip) == 1.2.3.4 {
    Alert();
}

(Product and Environment Specific)
Sigma to the Rescue!

• Written by Florian Roth & Thomas Patzke
  • "To logs, what Snort is to network traffic and YARA is to files"
• High level generic language for analytics
• Best method so far of solving logging signature problem!
• Enables analytics re-use and sharing across orgs
  • MISP compatible - share and store aligned with threat intel
• Decouples rule logic from SIEM vendor and field names
  • Eliminates SIEM tribal knowledge
• Blue teams needs this!!!
How Sigma Works

title: Office Macro Starts Cmd
status: experimental
description: Detects a Windows
references:
  - https://www.hybrid-analysis
author: Florian Roth
logsource:
  - product: windows
  - service: sysmon
detection:
  selection:
    EventID: 1
    ParentImage:
      - '*\WINWORD.EXE'
      - '*\EXCEL.EXE'
    Image: '/*\cmd.exe'
  condition: selection
fields:
  - CommandLine
  - ParentCommandLine
Conversion of Signatures to Alert Queries

Analytics → Sigma Rule

- Splunk
- Qradar
- Elasticsearch
- Grep

Written by community

Mapping to your field names, written by you

Company 1 field names → Search Query → Alert Engine
Company 2 field names → Search Query → Alert Engine
Company 3 field names → Search Query → Alert Engine
Grep command → Alert Engine

Search Query

Alert Engine
Rule Format

- Plain text YAML files
- Easy schema

1. Metadata
   - Title, status, description, references, tags, etc.

2. Log Source
   - What type, brand, and service is the log from?

3. Detection – List of Selectors

4. Condition – Logic for selector matching
Title, Metadata, and Log Source

title: PowerShell PSAttack
status: experimental
description: Detects the use of PSAttack PowerShell hack tool
references:
- https://adsecurity.org/?p=2921
tags:
- attack.execution
- attack.t1086
author: Sean Metcalf (source), Florian Roth (rule)
logsource:
product: windows
service: powershell
description: 'It is recommended to use the new "Script Block Logging" of PowerShell v5
Optional Classifiers:

- **category**: proxy, firewall, AV, IDS
  - For all logs of a *group of products*
- **product**: Squid, pfSense, Symantec, Snort, Windows
  - For all log outputs of *one product*
- **service**: SSH, DNS, DHCP
  - For a *subset of a products logs* – sshd, named, ...
- **description**: Additional detail on log source, configs
Detection & Conditions

• **Condition**: Logic for rule matching
• **Detection**: Object containing items of interest
  - `[field name]` - referenced in the condition

Examples:

detection:
  selection:
    EventID: 5140
    ShareName: Admin$
    filter:
      SubjectUserName: '*$'

condition: selection and not filter

signature:
  - "*MeteTool*"
  - "*Meterpreter*"
  - "*Metasploit*"
  - "*PowerSploit*"
  - "*CobaltStrike*"

condition: selection

detection:
  service_installation:
    EventID: 7045
    ServiceName: 'PSEXESVC'
    ServiceFileName: '*\PSEXESVC.exe'

service_execution:
    EventID: 7036
    ServiceName: 'PSEXESVC'
    sysmon_processcreation:
      EventID: 1
      Image: '*\PSEXESVC.exe'
      User: 'NT AUTHORITY\SYSTEM'

condition: 1 of them
Example Rule

```yaml
$ cat inbound_ssh.yml

title: Inbound SSH login attempt
description: Alert when someone is trying
to connect via SSH
author: John Hubbard
status: experimental
logsource:
  category: firewall
  product: ufw
detection:
  selection_1:
    destination_port: 22
    destination_ip: 192.168.42.139
condition: selection_1

Log Sample:

Nov 20 13:23:51 ubuntu kernel:
[219663.525528] [UFW AUDIT] IN=
OUT=ens33 SRC=192.168.42.140
DST=192.168.42.139 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=41584 DF PROTO=TCP
SPT=52900 DPT=22 WINDOW=29200 RES=0x00
SYN URGP=0

Nov 20 13:23:52 ubuntu kernel:
[219664.531357] [UFW AUDIT] IN=
OUT=ens33 SRC=192.168.42.140
DST=192.168.42.139 LEN=60 TOS=0x00
PREC=0x00 TTL=64 ID=41585 DF PROTO=TCP
SPT=52900 DPT=22 WINDOW=29200 RES=0x00
SYN URGP=0
```
**Splunk Rule Conversion**

```
$ cat inbound_ssh.yml

title: Inbound SSH login attempt
logsource:
  category: firewall
  product: ufw
detection:
  selection_1:
    destination_port: 22
    destination_ip: 192.168.42.139
condition: selection_1
```

**Field Mapping:**

```
$ cat splunk-sigmaui.yml

logsources:
  ufw:
    category: firewall
    product: ufw
    index: main
fieldmappings:
  destination_port: destination_port
  destination_ip: destination_ip
defaultindex: main
```

**Rule Conversion:**

```
$ ./sigmac --target splunk --config splunk-sigmaui.yml inbound_ssh.yml
(index="main" destination_port="22" destination_ip="192.168.42.139")
```
<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/20/18 1:24:34 PM</td>
<td><strong>destination_ip</strong> = 192.168.42.139, <strong>destination_port</strong> = 22, host = ubuntu, length = 60, out_interface = ens33, protocol = TCP, source = ufw.log, source_ip = 192.168.42.140, source_port = 52906</td>
</tr>
</tbody>
</table>

**Splunk Results**

```
(index="main" destination_port="22" destination_ip="192.168.42.139")
```
**Elasticsearch Rule Conversion**

```
$ cat inbound_ssh.yml

```{.highlight}
title: Inbound SSH login attempt
logsource:
  category: firewall
  product: ufw
detection:
  selection_1:
    destination_port: 22
    destination_ip: 192.168.42.139
  condition: selection_1
```

**Field Mapping:**

```
$ cat elk-sigmaci.yml

```{.highlight}
logsources:
  ufw:
    category: firewall
    product: ufw
    index: firewall-ufw
fieldmappings:
  destination_port: DPT
  destination_ip: DST
defaultindex: firewall-ufw
```

**Rule Conversion:**

```
$ ./sigmac --target es-qs --config elk-sigmaci.yml inbound_ssh.yml
(DPT:"22" AND DST:"192.168.42.139")
```

//This would be run on the "firewall-ufw" index
**What Was Just Done**

- One generic rule goes in...
- Two formatted & customized search queries come out!

**Sigma Rule**

- Written by community

**Splunk Format Converter**

- splunk_sigmaui.yml
- Customized for each deployment

**Elasticsearch Format Converter**

- elk_sigmaui.yml
- Search Query

Written by community

Customized for each deployment
Currently Supported Outputs

- Splunk
- QRadar
- ArcSight
- Elasticsearch (Elastalert, Query strings, DSL, Watcher, & Kibana)
- Logpoint
- Qualys
- Windows Defender ATP
- PowerShell
- grep
Coming Soon: Generic Log Sources

- Many separate process creation event sources:
  - Security Event 4688
  - Sysmon Event 1
  - Defender ATP
  - OSQuery
  - EDR suites...etc.
- Mapping is difficult for each unique event ID / field name
- **Solution: Generic Log sources**
Inbound SSH login Attempt

**Author:** John Hubbard

**Logsourc**e: product: ufw; category: firewall

```
1 title: 'Inbound SSH login Attempt'
2 description: 'Alert when someone is trying to connect via SSH'
3 author: 'John Hubbard'
4 status: 'experimental'
5 logsourc:
6  | product: ufw
7  | category: firewall
8 detection:
9  | condition: selection_1
10  | selection_1:
11  |  | destination_port: 22
12  |  | destination_ip: 192.168.42.139
13 fields: []
14 falsepositives:
15   |
```

**logsourc**e:

- **product:** ufw
- **category:** firewall

**detection**:

- **condition:** selection_1
- **selection_1**:
  - **destination_port:** 22
  - **destination_ip:** 192.168.42.139

**fields:**

- ""
Rule Organization

Don't know who changed a rule, or why?
• Sigma can help with this too!
• Most SIEMs don't have alert change tracking
• Text based analytics = easy Git version control!
Continuous Updates Required

Assuming we already collect the correct:

- **Data sources**
- **Events** inside those data sources
  - Filter, parse, and enrich
- Have **analytics for known attacks**

We still need **continuous updates** for new attacks!

- Exploits move fast: ShellShock, Heartbleed, ETERNALBLUE, etc.
- Sigma can help here too....
Sigma + MISP

- MISP is one of the best, free Threat Intel Platforms
- Wide usage in enterprise
  - Integrates well with other tools via open API
  - “Event” driven data organization
    - All hashes, IPs, URLs, for incident go into an "event"
- Meant for sharing
  - Supports Sigma rules as object type
  - Tool sigma2misp pushes rules to events
Imagine a world…

- Where intelligence **reports come with Sigma rules**
- Don't have to write the analytics
- Don't even have to transcribe them
  - They came to you through MISP!
- **Analytics automatically appear in Threat Intel Platform**
  - Already associated with threat actors
  - Supporting IOCs included
  - Simply convert the rules you want!
Putting Sigma Rules Into MISP

```
student@ubuntu:/opt/sigma/tools$ ./sigma2misp --insecure --url https://localhost --key D8Z2pRacFKFkfeua7riu3haEtV6DaxMEEwzKpA ~/Docker/ SigmaUI/inbound_ssh.yaml
Importing Sigma rule /home/student/Docker/ SigmaUI/inbound_ssh.yaml into MISP event 1...
```
<table>
<thead>
<tr>
<th>Date</th>
<th>Org</th>
<th>Category</th>
<th>Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2018-11-21</td>
<td></td>
<td></td>
<td>sigma</td>
<td>title: Outbound SSH attempt</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>description: Alert when someone trying to</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>connect via SSH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>author: john</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>status: experimental</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>logsource:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>category: firewall</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>product: ufw</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>detection:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>selection_1:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>destination_port: 22</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>destination_ip: 192.168.42.139</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>condition: selection_1</td>
</tr>
</tbody>
</table>
Pulling Rules Out of MISP

Easy to use REST API:

```bash
student@ubuntu:/opt/sigma/tools$ curl -k --header "Authorization: D8Z2pRacFKFkfeua7riu3haEtkV6DaxMEEwtKpa " --header "Accept: application/json" --header "Content-Type: application/json" https://localhost/attributes/1 | python -m json.tool
```

```

"Attribute": {
    "category": "Payload delivery",
    "comment": ",",
    "deleted": false,
    "disable_correlation": false,
    "distribution": "3",
    "event_id": "1",
    "event_uuid": "5bf55b69-7fbc-4693-958e-0136ac110003",
    "id": "1",
    "object_id": "0",
    "object_relation": null,
    "sharing_group_id": "0",
    "timestamp": "1542806648",
    "to_ids": false,
    "type": "sigma",
    "uuid": "bd06b01f-e46d-46f7-ad63-2a17b61fd876",
    "value": "title: Outbound SSH attempt\n\ndescription: Alert when someone trying to connect via SSH\n\nstatus: experimental\n\nlogsource: r\n\ncategory: firewall\n\nproduct: ufw\n\ndetection: r\n\nselection_1:\r\n
destination_port: 22\n\n\ndestination_ip: 192.168.42.139\n\ncondition: selection_1"
}
```
Action Steps

- Clone the Sigma project from GitHub
- Create a custom mapping for your SIEM
- Write some analytics to try out
- Push rules into MISP (Docker image available)
- Start sharing - link MISP to a friend!

Optional:
- Try out my SigmaUI docker-compose script!
- Contribute - Write a parser to support your SIEM!
Using My SigmaUI Docker Container

To download and run container (requires newest docker-compose):

- $ git clone https://github.com/SecHubb/SigmaUI.git
- $ cd SigmaUI && docker-compose up
- You're ready to go!
  - Point browser to localhost:5601 and click Sigma side tab

QuickStart for Sigma rule converter tool (python 3 required):

- git clone https://github.com/Neo23x0/sigma.git
- $ sigma/tools/sigmac -format [format] -config [config.yml] [rule_file.yml]
Thank you to:
Florian Roth, Thomas Patzke, and all Sigma contributors!

Blue team – let's make it happen!