[III. DETECTION ENGINEERING]

Detection Engineering
* a set of processes that enable detection of potential threats in an environment
* encompasses:
  - ↳ collecting detection requirements
  - ↳ aggregating sys. telemetry
  - ↳ implementing & maintaining detection logic
  - ↳ validating program effectiveness
* motivation: earlier threat detection ⇒ ↓ remediation cost

Goals
- Mitigating security events
- Collection of events
- Receive feedback for further improvement
- Identify events that may indicate security incident
- Communication of high-value details of event to stakeholders
- Detecting threats
- Understand impact for a potential incident
- Qualifying threats

Process
- Discovery
  - → collect detection requirements
  - → determine criticality of detection
- Design, Development & Testing
  - → interpreting detection requirement
  - → plan & implement detection in a lab/test environment
- Implementation & post-implementation monitoring
  - → implementing in production environment
  - → monitor performance of detection
- Validation → routing testing to determine effectiveness of detection engineering program

Tools
SIGMA (log analysis) ≪

SNORT (network traffic)
an open src IDS\IPS which helps detect cyber threats at network level

Purpose
- → detect & alert on potential network security threats
- → analyzes network traffic in real-time
  - ↳ generates alerts that can be acted upon

YARA (files & binaries)
tool for creating descriptions of malware fams. based on textual or binary patterns to
- → detect sophisticated malware
- → identify & classify malicious samples

Anatomy (similar to C language)

```
rule ruleName {
    meta:       // descriptive info abt rule - name, author, brief of purpose
        /* ... */
    strings:    // patterns\sequences of bytes to search for in the files
        /* ... */
    conditions: // logical statements that specify when the rule should trigger
        /* ... */
}
```

favor hex values over alphanumeric strings
- diff. encoding possible for obscuring chars
- can specify checking for specific string signatures based on specific offsets

yara.readthedocs.io/stable/writingrules.html

Architecture
Network Traffic
↓
Sniffer
↓
Packet Decoder
↓
Preprocessor
↓
Detection Engine ⇆ Rules
↓
Logging & Alerting

Sniffer
- ↳ captures packets
- ↳ operates in promiscuous mode

Preprocessor
- ↳ prepares data for analysis
- ↳ packet reassembly, normalization, protocol decoding
- ↳ helps make sense of complex network protocols & ensure data is suitable for analysis

Detection Engine (core of snort's architecture)
- ↳ analyzes network traffic to identify potential security threats
- ↳ process preprocessed data & compares it against detection rules
- ↳ employs various detection forms (ex., anomaly-based, signature-based) to identify malicious or sus. patterns in traffic

Capabilities
* identify & classify malware
* find new malware samples based on feats.
* help w threat hunting
* find new exploits & zero-days
* help w incident response
* build effective cyber defense
* build own antivirus (AV)

github.com/Yara-Rules/rules virustotal.com/blog/yara-rules

Yara Rules

yara <yara_rule> <target_file>

String Examples

```
$pe = { 4D 5A } //PE file signature "MZ" at beginning

$office = { 50 4B 03 04 } // Office XML formats (DOCX, PPTX, XLSX, etc.)
          signature "PK" at beginning
                space chars
                use ZIP container

// Older Office formats (DOC, PPT, XLS)
// Compound File Binary Format signature
$officeOld = { D0 CF 11 E0 A1 B1 1A E1 }

/*
   PDF file signature "%PDF-" at the beginning
   signature is printed first when using "strings" cmd
   ex. value: %PDF-1.3
*/
$pdfStart = { 25 50 44 46 2D } // "%PDF-"  } $pdfStart at 0 and $end
$pdfEnd = { 25 25 45 4F 46 }   // "%%EOF"  }

//example loader strings
$s0 = "URLRequest" wide ascii   } 1 of ($s*)
$s1 = "URLLoader" wide ascii    }
$s2 = "loadAuf" wide ascii      }
$s3 = "myURLRequest" wide ascii }

$s4 = "<!-- The ID below indicates application support for Windows 10 -->"

$t1 = "cmd.exe" wide fullword

$a1 = "<description>WinRAR SFX module</description>" fullword ascii

$dotNet = ".NETFramework,Version" ascii fullword
```

Rule Header

```
rule Detect_PEOfficePDF
     identifier (alphanumeric, first char x digit)
↓
must start w "rule" keyword
```

String Modifiers

```
$a = "malwarestring" [fullword]   match against exact word
                                  "www.malwarestring.com"      ✓
                                  "www.abcmalwarestring.com"   ✗

$a = "malwarestring" [wide]     → match unicode strings separated by null bytes
                                  ex. "w.w.w.-.m.a.l.w.a.r.e.s.t.r.i.n.g...c.o.m."
$a = "malwarestring" [wide ascii] → match on unicode & ascii chars
$a = "malwarestring" [nocase]   → match string regardless of case
$a = { 5C ?? ?F }  question marks used as wildcards for detecting slight
                   variations of hex patterns with multiple samples
$a = { 5C [2-10] 6F }  start w "5C", then 2-10 random bytes
$a = { 5C (01 02 | 03 04) 6F }  start w "5C" then either "01 02" or "03 04"
```

Condition

```
condition:
    header_check
    file_size_limitation
    other_limitations
    string_combinations
    false_positive_filters
```

Example:

① condition:

```
uint16(0) == 0x5a4d
and filesize < 300KB
and pe.number_of_signatures == 0
and all of ($s*)
and not 1 of ($fp*)
```

② condition:

```
(
    uint16(0) == 0x5a4d  // MZ marker
    or uint16(0) == 0x457f // ELF marker
)
and filesize < 300KB
and pe.number_of_signatures == 0
and all of ($s*)
and not 1 of ($fp*)
```

SIGMA

Generic & open src rule format; allows describing relevant log events in straightforward manner

* main purpose: provide structured form for describing & sharing detection methods for malicious or dangerous behavior
* components work together to create comprehensive ecosystem for SIEM detection
  - ↳ SIEM detection supports creation, mgmt & sharing of detection rules across diff. platforms & use cases

Components

Sigma Format

* built as open-src std so Sigma detection rule can be used across entire security landscape of:
  - ↳ SIEMs (ex., Splunk, Azure Sentinel)
  - ↳ logging platforms (ex., graylog)
  - ↳ dbs (ex., MySQL)
* flexible, easy to write, & applicable to any type of log file

Sigma Tools (pySigma \ sigma-cli)

* 1° focus for Sigma ecosystem: converter
  - ↳ turns Sigma detection files into usable SIEM queries

Sigma Rules

rule collections in repos (ex., GitHub SigmaHQ/sigma)

* can cover wide range of threat types & atk scenarios
* useful for expanding detection capabilities or staying up to date w emerging threats

Components

Detection

Specifies what malicious behavior the rule is searching for in logs

* detection methods:
  - ↳ by keyword
    - ex., "rm *bash_history"
  - ↳ by field value
    - ex., Username: 'Administrator'
  - ↳ by multiple field vals. (lists)

Metadata

other info abt detection

Logsource

specifies what log data should be searched by rule

* to target specific variety of logs, rather than search over all types of logs w/in SIEM
* logsource def'n can provide more info abt how to onboard log data src correctly
* def'n split into:
  - ↳ category (of products) ex., webserver, firewall, edr
  - ↳ product ex., windows, linux, cisco
  - ↳ service (running w/ diff. product)
    - ex., kerberos, defender

ex.,
```
logsource:
    product: windows
    category: ps_script
    definition: Script Block Logging must be enabled
```

Sigma Backends

* "drivers" of Sigma conversion process for Sigma rule into a SIEM compatible query
* pySigma provides API for each Sigma backend to perform conversion, transformation & formatting of every Sigma rule

Example: YAML Format

```yaml
title: Cobalt Strike DNS Beaconing
id: 2975af79-28c4-4d2f-a951-9095f229df29
status: test
description: Detects suspicious DNS queries known from Cobalt Strike beacons
references:
   - https://www.icebrg.io/blog/footprints-of-fin7-tracking-actor-patterns
   - https://www.sekoia.io/en/hunting-and-detecting-cobalt-strike/
author: Florian Roth (Nextron Systems)
date: 2018/05/10
modified: 2022/10/09
tags:
   - attack.command_and_control
   - attack.t1071.004
logsource:
   category: dns
detection:
   selection1:
      query|startswith:
         - 'aaa.stage.'
         - 'post.1'
   selection2:
      query|contains: '.stage.123456.'
   condition: 1 of selection*
falsepositives:
   - Unknown
level: critical
```

Rule Template

github.com/SigmaHQ/sigma/wiki/Rule-creation-guide

Rule Template

The best way is to use an existing rule that gets close to what you plan like to write.

Make sure that the following fields are set in a rule that you would like to push to our public repository:

```yaml
title: a short capitalised title with less than 50 characters
id: generate one here https://www.uuidgenerator.net/version4
status: experimental
description: A description of what your rule is meant to detect
references:
    - A list of all references that can help a reader or analyst understand the meaning of a triggered rule
tags:
    - attack.execution  # example MITRE ATT&CK category
    - attack.t1059      # example MITRE ATT&CK technique id
    - car.2014-04-003   # example CAR id
author: Michael Haag, Florian Roth, Markus Neis  # example, a list of authors
date: 2018/04/06  # Rule date
logsource:                      # important for the field mapping in predefined or your additional config f:
    category: process_creation  # In this example we choose the category 'process_creation'
    product: windows            # the respective product
detection:
    selection:
        FieldName: 'StringValue'
        FieldName: IntegerValue
        FieldName|modifier: 'Value'
    condition: selection
fields:
    - fields in the log source that are important to investigate further
falsepositives:
    - describe possible false positive conditions to help the analysts in their investigation
level: one of five levels (informational, low, medium, high, critical)
```