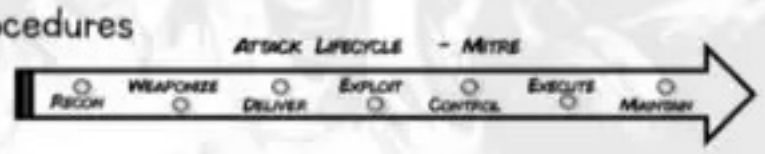
Practical Threat Intelligence

Define your requirements. Understand international relations and the geopolitical context.

Collect & classify intelligence reports:

- Advanced Persistent Threat, Threat Actors
- Tactics, Techniques and Procedures
- Vulnerability reports



Collect & classify Indicators of Compromises (IOC):

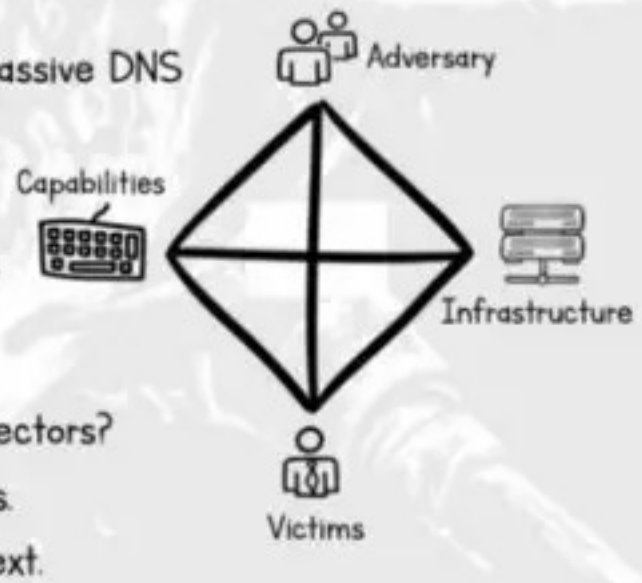
- Incident Response
- Open-Source Intelligence (OSINT)
- Threat Hunting

Analyze & triage IOCs:

- Malware and/or vulnerability analysis
- Infrastructures mapping. New domains.

Hunt & pivot for new attacks:

- Create Yara, Sigma, Snort Rules
- Identify code similarities
- Search for infrastructure overlap & passive DNS
- MassScanning to uncover new C2s
- Set up honeypots
- Get information from private sources



Understand victimology:

- Who/where are the targets? Which sectors?
- Make the connections to past attacks.
- Find a link with the geopolitical context.

Share intelligence, dispatch IOCs, improve the knowledge base.

Iterate & improve the process.

@FrØgger_
Thomas Roccia

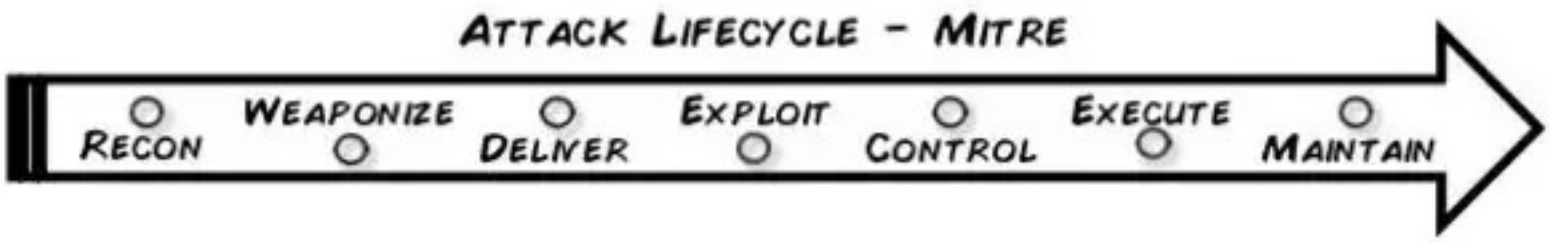
Tactics Techniques and Procedure (TTP)

TTP is a military term describing the operations of enemy forces.

In InfoSec TTP is an approach for profiling and contextualizing cyberattack operations.

Being able to break down complex TTP attacks will make detection much easier to understand.

Tactics **Techniques** **Procedure**



Tactics describes how an attacker operates during his operation. (Infrastructure reused, amount of entry point, compromised targets...)

Techniques describes the approach used to facilitate the tactical phase. (Tools used, malware, phishing attacks...)

Procedures describes a special sequence of actions used by attackers to execute each step of their attack cycle.

@FrØgger_
Thomas Roccia

MITRE ATT&CK MATRIX

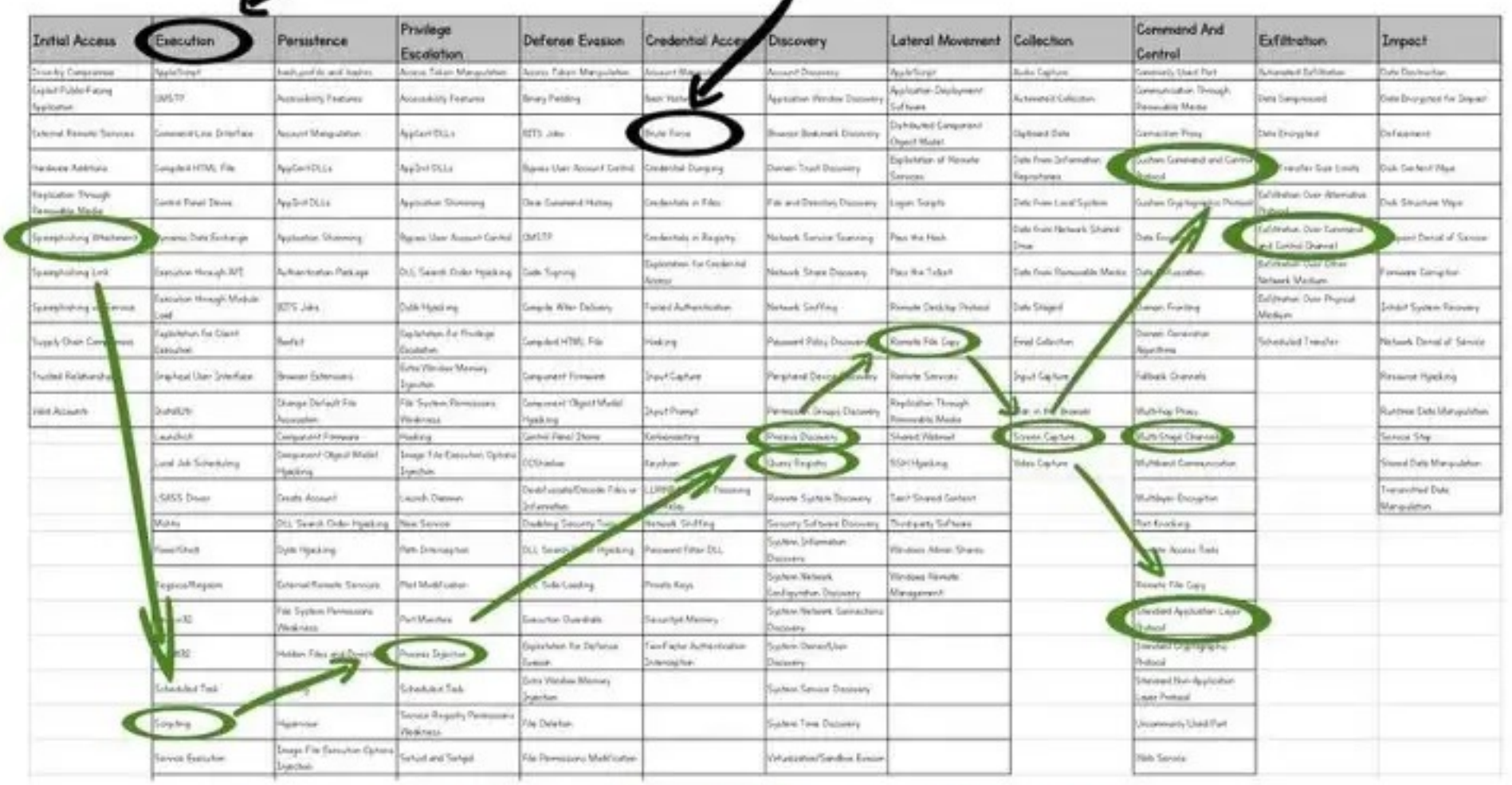
The matrix ATT&CK is a knowledge base of adversary tactics and techniques based on real-world observations.

ATT&CK stands for Adversarial Tactics, Techniques, and Common Knowledge. It documents tactics, techniques, and procedures (TTPs) that advanced persistent threats use.

ATT&CK organizes techniques into a set of tactics to provide context. It can be used to profile each step of a cyberattack operation.

TACTIC

TECHNIQUE



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Command And Control	Exfiltration	Impact
Drive-by Compromise	AppleScript	.bash_profile and .bashrc	Access Token Manipulation	Access Token Manipulation	Account Manipulation	Account Discovery	AppleScript	Audio Capture	Commonly Used Port	Automated Exfiltration	Data Destruction
Exploit Public-Facing Application	CMSTP	Accessibility Features	Accessibility Features	Binary Padding	Bash History	Application Window Discovery	Application Deployment Software	Automated Collection	Communication Through Removable Media	Data Compressed	Data Encrypted for Impact
External Remote Services	Command-Line Interface	Account Manipulation	AppCert DLLs	BITS Jobs	Brute Force	Browser Bookmark Discovery	Distributed Component Object Model	Clipboard Data	Connection Proxy	Data Encrypted	Defacement
Hardware Additions	Compiled HTML File	AppCert DLLs	AppInit DLLs	Bypass User Account Control	Credential Dumping	Domain Trust Discovery	Exploitation of Remote Services	Data from Information Repositories	Custom Command and Control Protocol	Exfiltration Size Limits	Disk Content Wipe
Replication Through Removable Media	Control Panel Items	AppInit DLLs	Application Shimming	Clear Command History	Credentials in Files	File and Directory Discovery	Logon Scripts	Data from Local System	Custom Cryptographic Protocol	Exfiltration Over Alternative Protocol	Disk Structure Wipe
Spearphishing Attachment	Dynamic Data Exchange	Application Shimming	Bypass User Account Control	CMSTP	Credentials in Registry	Network Service Scanning	Pass the Hash	Data from Network Shared Drive	Data Encoding	Exfiltration Over Command and Control Channel	Endpoint Denial of Service
Spearphishing Link	Execution through API	Authentication Package	DLL Search Order Hijacking	Code Signing	Exploitation for Credential Access	Network Share Discovery	Pass the Ticket	Data from Removable Media	Data Obfuscation	Exfiltration Over Other Network Medium	Firmware Corruption
Spearphishing via Service	Execution through Module Load	BITS Jobs	Dylib Hijacking	Compile After Delivery	Forced Authentication	Network Sniffing	Remote Desktop Protocol	Data Staged	Domain Fronting	Exfiltration Over Physical Medium	Inhibit System Recovery
Supply Chain Compromise	Exploitation for Client Execution	Bootkit	Exploitation for Privilege Escalation	Compiled HTML File	Hooking	Password Policy Discovery	Remote File Copy	Email Collection	Domain Generation Algorithms	Scheduled Transfer	Network Denial of Service
Trusted Relationship	Graphical User Interface	Browser Extensions	Extra Window Memory Injection	Component Firmware	Input Capture	Peripheral Device Discovery	Remote Services	Input Capture	Fallback Channels		Resource Hijacking
Valid Accounts	InstallUtil	Change Default File Association	File System Permissions Weakness	Component Object Model Hijacking	Input Prompt	Permission Groups Discovery	Replication Through Removable Media	Man in the Browser	Multi-hop Proxy		Runtime Data Manipulation
	Launchctl	Component Firmware	Hooking	Control Panel Items	Kerberoasting	Process Discovery	Shared Webroot	Screen Capture	Multi-Stage Channels		Service Stop
	Local Job Scheduling	Component Object Model Hijacking	Image File Execution Options Injection	CCleaner	Keychain	Query Registry	SSH Hijacking	Video Capture	Multiband Communication		Stored Data Manipulation
	LSASS Driver	Create Account	Launch Daemon	Deobfuscate/Decode Files or Information	LLMNR/NBT-NS Poisoning and Relay	Remote System Discovery	Taint Shared Content		Multilayer Encryption		Transmitted Data Manipulation
	Mshta	DLL Search Order Hijacking	New Service	Disabling Security Tools	Network Sniffing	Security Software Discovery	Third-party Software		Port Knocking		
	PowerShell	Dylib Hijacking	Path Interception	DLL Search Order Hijacking	Password Filter DLL	System Information Discovery	Windows Admin Shares		Remote Access Tools		
	Regsvcs/Regasm	External Remote Services	Plist Modification	DLL Side-Loading	Private Keys	System Network Configuration Discovery	Windows Remote Management		Remote File Copy		
	Regsvr32	File System Permissions Weakness	Port Monitors	Execution Guardrails	Securityd Memory	System Network Connections Discovery			Standard Application Layer Protocol		
	Rundll32	Hidden Files and Directories	Process Injection	Exploitation for Defense Evasion	Two-Factor Authentication Interception	System Owner/User Discovery			Standard Cryptographic Protocol		
	Scheduled Task	Hypervisor	Scheduled Task	Extra Window Memory Injection		System Service Discovery			Uncommonly Used Port		
	Scripting	Image File Execution Options Injection	Service Registry Permissions Weakness	File Deletion		System Time Discovery			Web Service		
	Service Execution	Logon Scripts	Setuid and Setgid	File Permissions Modification		Virtualization/Sandbox Evasion					

FANCY BEAR GROUP EXAMPLE

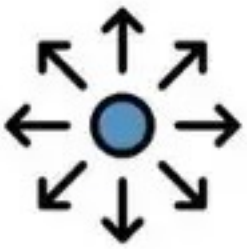
- Understand the operating method of an attacker.
- Identify the techniques and tactics used.
- Assess defensive coverage and identify high priority gaps.

Source : https://attack.mitre.org

@FRØGGER_
THOMAS ROCCIA

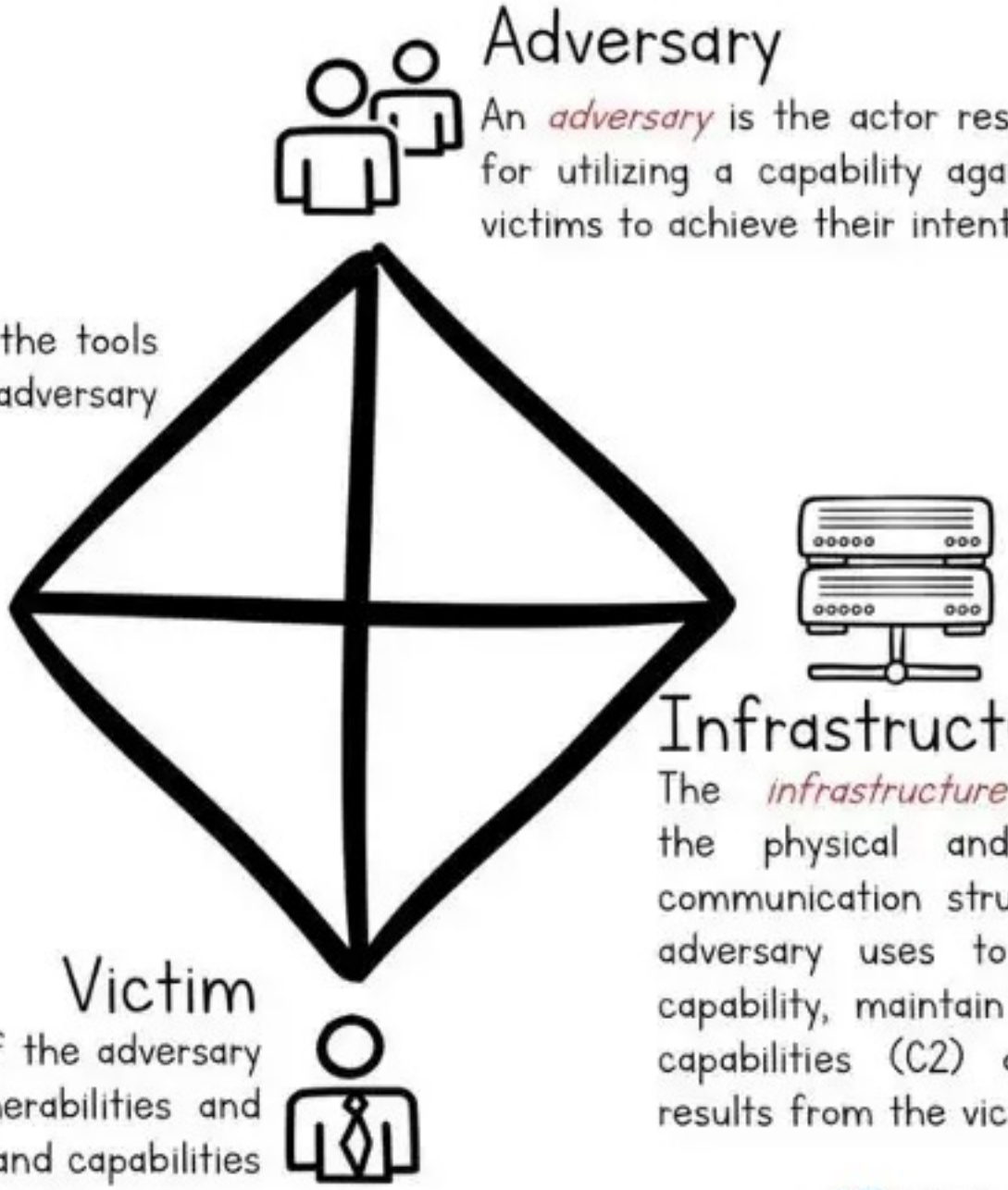
Diamond Model of Intrusion Analysis

The Diamond Model is an approach to conducting intelligence on network intrusion events.



This model relates four basic elements of an intrusion: adversary, capabilities, infrastructure and victim.

An intrusion event is defined as how the attacker demonstrates and uses certain capabilities and techniques over infrastructure against a target.



Adversary

An *adversary* is the actor responsible for utilizing a capability against the victims to achieve their intent.

Capabilities

The *capability* describes the tools and techniques of the adversary used in the event.

Infrastructure

The *infrastructure* describes the physical and/or logical communication structures, the adversary uses to deliver a capability, maintain control of capabilities (C2) and effect results from the victim.

Victim

A *victim* is the target of the adversary and against whom vulnerabilities and exposures are exploited and capabilities used.

Source : https://apps.dtic.mil/dtic/tr/fulltext/u2/a586960.pdf

@FrØgger_
Thomas Roccia

Anatomy of a yara Rule

Yara is a tool used to identify file, based on textual or binary pattern.

A rule consists of a set of strings and conditions that determine its logic.

Rules can be compiled with "yarac" to increase the speed of multiple Yara scans.

1 Import Module

Yara modules allow you to extend its functionality. The PE module can be used to match specific data from a PE:

- pe.number_of_exports
- pe.sections[0].name
- pe.imphash()
- pe.imports("kernel32.dll")
- pe.is_dll()

List of modules: pe, elf, hash, math, cuckoo, dotnet, time

2 Rule Name

The rule name identifies your Yara rule. It is recommended to add a meaningful name. There are different types of rules:

- Global rules: applies for all your rules in the file.
- Private rules: can be called in a condition of a rule but not reported.
- Rule tags: used to filter yara's output.

3 Metadata

Rules can also have a metadata section where you can put additional information about your rule.

- Author
- Date
- Description
- Etc.

4 Strings

The field strings is used to define the strings that should match your rule. It exists 3 type of strings:

- Text strings
- Hexadecimal strings
- Regex

5 Condition

Conditions are Boolean expressions used to match the defined pattern.

- Boolean operators:
  - and, or, not
  - <=, >=, ==, <, >, !=
- Arithmetic operators:
  - *, -, +, \, %
- Bitwise operators:
  - &, |, <<, >>, ^, ~
- Counting strings:
  - #string0 == 5
- Strings offset:
  - $string1 at 100

```
import "pe"

rule demo_rule : Tag1 Demo
{
    meta:
        author = "Thomas Roccia"
        description = "demo"
        hash = ""

    strings:
        $string0 = "hello" nocase wide
        $string1 = "world" fullword ascii
        $hex1 = { 01 23 45 ?? 89 ab cd ef }
        $re1 = /md5: [0-9a-zA-Z]{32}/ }

    condition:
        uint16(0) == 0x5A4D and filesize < 2000KB
        or pe.number_of_sections == 1 and
        any of ($string*) and (not $hex1 or $re1)
}
```

Advanced Condition

- Accessing data at a given position: uint16(0) == 0x5A4D
- Check the size of the file: filesize < 2000KB
- Set of strings: any of ($string0, $hex1)
- Same condition to many strings: for all of them : (# > 3)
- Scan entry point: $value at pe.entry_point
- Match length: !re1[1] == 32
- Search within a range of offsets: $value in (0,100)

Text Strings

Text strings can be used with modifiers:

- nocase: case insensitive
- wide: encoded strings with 2 bytes per character
- fullword: non alphanumeric
- xor(0x01-0xff): look for xor encryption
- base64: base64 encoding

Hexadecimal

Hex strings can be used to match piece of code:

- Wild-cards: { 00 ?2 A? }
- Jump: { 3B [2-4] B4 }
- Alternatives: { F4 (B4 | 56) }

Regex

Regular expression can also be used and defined as text strings but enclosed in forward slash.

@FrØgger_
Thomas Roccia

Anatomy of a SIGMA Rule

Sigma is a tool used to identify patterns in log events using rules.

A rule consists of a set of detection fields that describes malicious events to identify.

Sigma is for log files what Snort is for network traffic and YARA is for files.

Title
Title of your rule, that allows to quickly identify the goal. This is the alert name.

Rule ID
Universally Unique Identifier (UUID)
https://www.uuidgenerator.net

Related rule types :
- derived: Rule derived from the referred rule
- obsoletes: Obsoletes rule
- merged: Rule was merged from the referred rules
- renamed: The rule had previously the referred identifier or identifiers but was renamed

Status
- stable: the rule may be used in production systems or dashboards
- test: rule that could require some fine tuning
- experimental: rule that could lead to false results

Fields
Use for the evaluation of certain events

False Positives
Describe possible false positives

```
title: Demo Rule
id: dc827aee-f664-4c53-901c-2a55094960a2
related:
  - id: 08fbc97d-8a2f-491c-ae21-8ffcfd3174e9
    type: derived
status: experimental
description: Sigma rule for demo.
references:
    - https://den800.com
tags:
    - attack.execution
    - attack.t1003.001
author: Thomas Roccia, Harry Potter, Jack Sparrow
date: 2021/12/06
logsource:
    category: process_access
    product: windows
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.
level: critical
```

Description
Description of the current rule

References
External link or document for the rules. This field must be a list.

Tags
Tags from Mitre ATT&CK.
- Use lower-case tags only
- Replace space or hyphens with an underscore

Author
Specify the author(s) of the rules.

Date
Used to specify date of rule creation.

Log Source
Identify the log source that trigger the rule. If there is not a single rule use the following:
- product (e.g. linux, windows, cisco)
- service (e.g. sysmon, ldapd, dhcp)
- category (e.g. process_creation)

Level
Indicates the level of the rules.
- informational, critical, high, medium, low

Detection
Used to trigger your detection using selection and condition.

General
- All values are case-inconsitive strings
- You can use wildcard characters '*' and '?'
- Wildcards can be escaped with \, e.g. *
- Regular expressions are case-sensitive

Special Field Values
- An empty value is defined with ''
- A null value is defined with null

```
detection:
  selection:
    EventID: 4738
  filter:
    PasswordLastSet: null
condition:
  selection and not filter
```

FieldName
FieldName defines the value in your logs. It can be a list linked with a logical 'OR'.

```
detection:
  keywords:
    - EVILSERVICE
    - svchostexe -n evil
```

Or it can be a Dictionary consisting of key/value pairs. Lists of maps are joined with a logical 'OR'. All elements of a map are joined with a logical 'AND'.

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

Value Modifiers
Value modifiers are appended with a pipe character | as separator.
- contains: the value is matched anywhere in the field
- all: This modifier links all value with AND
- base64: The value is encoded with Base64
- base64offset: If a value appears in a base64-encoded value the representation might change depending on the position in the overall value
- endswith: The value is expected at the end of the field's content
- startswith: The value is expected at the beginning of the field's content
- utf16le: transforms value to UTF16-LE
- utf16be: transforms value to UTF16-BE
- wide: alias for utf16le modifier
- re: value is handled as regular expression by backends

Condition
- Logical AND/OR (keywords1 or keywords2)
- **1/all of search-identifier**
  - 1 (logical or across alternatives)
  - all (logical and across alternatives)
- **1/all of them**: Logical OR (1 of them) or AND (all of them)
- **1/all of search-identifier-pattern**: Same as 1/all of them but restricted to matching search identifiers
- **Negation with 'not'** (keywords and not filters)
- **Brackets**: "selection1 and" (keywords1 or keywords2)
- **Near aggregation expression**
  - near search-id-1 [[and search-id-2 | and not search-id-3] .]
- **Operator Precedence**
  - [or, and, not, x of search-identifier, (expression)

@FR0GGER_
Thomas Roccia

Log Parsing Cheat Sheet

GREP	GREP allows you to search patterns in files. ZGREP for GZIP files. $grep <pattern> file.log	-n: Number of lines that matches -i: Case insensitive -v: Invert matches -E: Extended regex -c: Count number of matches -l: Find filenames that matches the pattern
NGREP	NGREP is used for analyzing network packets. $ngrep -I file.pcap	-d: Specify network interface -i: Case insensitive. -x: Print in alternate hexdump -t: Print timestamp -I: Read pcap file
CUT	The CUT command is used to parse fields from delimited logs. $cut -d ":" -f 2 file.log	-d: Use the field delimiter -f: The field numbers -c: Specifies characters position
SED	SED (Stream Editor) is used to replace strings in a file. $sed s/regex/replace/g	s: Search g: Replace d: Delete w: Append to file -e: Execute command -n: Suppress output
SORT	SORT is used to sort a file. $sort foo.txt	-o: Output to file -r: Reverse order -n: Numerical sort -k: Sort by column. -c: Check if ordered -u: Sort and remove. -f: Ignore case -h: Human sort
UNIQ	UNIQ is used to extract uniq occurrences. $uniq foo.txt	-c: Count the number of duplicates -d: Print duplicates -i: Case insensitive
DIFF	DIFF is used to display differences in files by comparing line by line. $diff foo.log bar.log	How to read output? a: Add c: Change d: Delete #: Line numbers <: File 1 >: File 2
AWK	AWK is a programming language use to manipulate data. $awk {print $2} foo.log	Print first column with separator ":" $awk -F: '{print $1}' /etc/passwd Extract uniq value from two files: awk 'FNR==NR {a[$0]++; next} !($0 in a)' f1.txt f2.txt

@FrØgger_
Thomas Roccia

Log Parsing Cheat Sheet 2

HEAD	HEAD is used to display the first 10 lines of a file by default. **$head file.log**	-n: Number of lines to display -c: Number of bytes to display
TAIL	TAIL is used to display the last 10 lines of a file by default. **$tail file.log**	-n: Number of lines to display -f: Wait for additional data -F: Same as -f even if file is rotated
LESS	LESS is used to visualize the content of a file, faster than MORE. ZLESS for compressed files. **$less file.log**	space: Display next page /: Search n: Next g: Beginning of the file G: End of the file +F: Like tail -f
COMM	COMM is used to select or reject lines common to two files. **$comm foo.log bar.log**	Three columns as output: Column 1: lines only in file 1 Column 2: lines only in file 2 Column 3: lines in both files -1, -2, -3: Suppress columns output
CSVCUT	CSVCUT is used to parse CSV files. **$ csvcut -c 3 data.csv**	-n: Print columns name -c: Extract the specified column -C: Extract all columns except specified one -x: Delete empty rows
JQ	JQ is used to parse JSON files. **$jq . foo.json**	jq . f.json: Pretty print jq '.[]' f.json: Output elements from arrays jq '.[0].<keyname>' f.json
TR	TR is used to replace a character in a file. **$ tr ";" "," < foo.txt**	-d: Delete character -s: Compress characters to a single one Lower to upper every character: tr "[:lower:]" "[:upper:]" < foo.txt
CCZE	CCZE is used to color logs. **$ccze < foo.log**	-h: Output in html -C: Convert Unix timestamp -l: List available plugins -p: Load specified plugin

@FrØgger_
Thomas Roccia