Blue-Team-as-Code: Lessons From Real-world Red Team Detection Automation Using Logs

Oleg Kolesnikov, VP of Threat Research/Cybersecurity

Den Iuzvyk, Senior Security Researcher

INTRO/DISCLAIMER: Blue Team Singularity

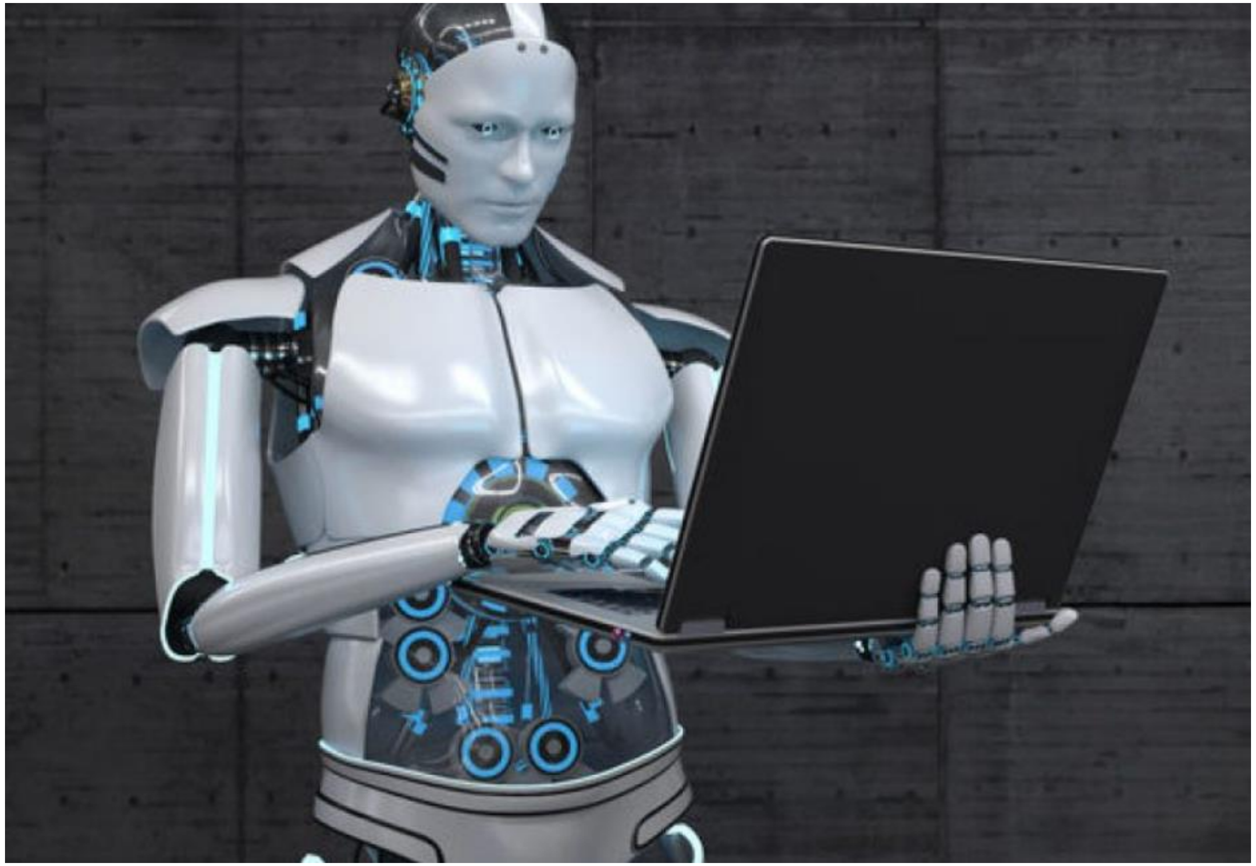


IMAGE CREDIT: FRANK ALFIERI

Agenda

0x01 **Blue-team-as-Code Fundamentals**

Intro, Key Concepts, and Tooling

0x02 **Tradecraft Highlights, Blindspots**

Common Blindspots, How to Address w/Code, etc

0x03 **Demos & Practical Examples**

Real-world Red Team detection using code

0x04 **Conclusions**

Takeaways and applying what you've learned

0x01 - Blue-team-as-code (BAC) Fundamentals

I

ATT&CK "Bathymetry"

Intro, bathymetry, specific & sensitive + code, detection spectrum

II

Tools/code

Sigma, jupyter, python

III

OMEGA

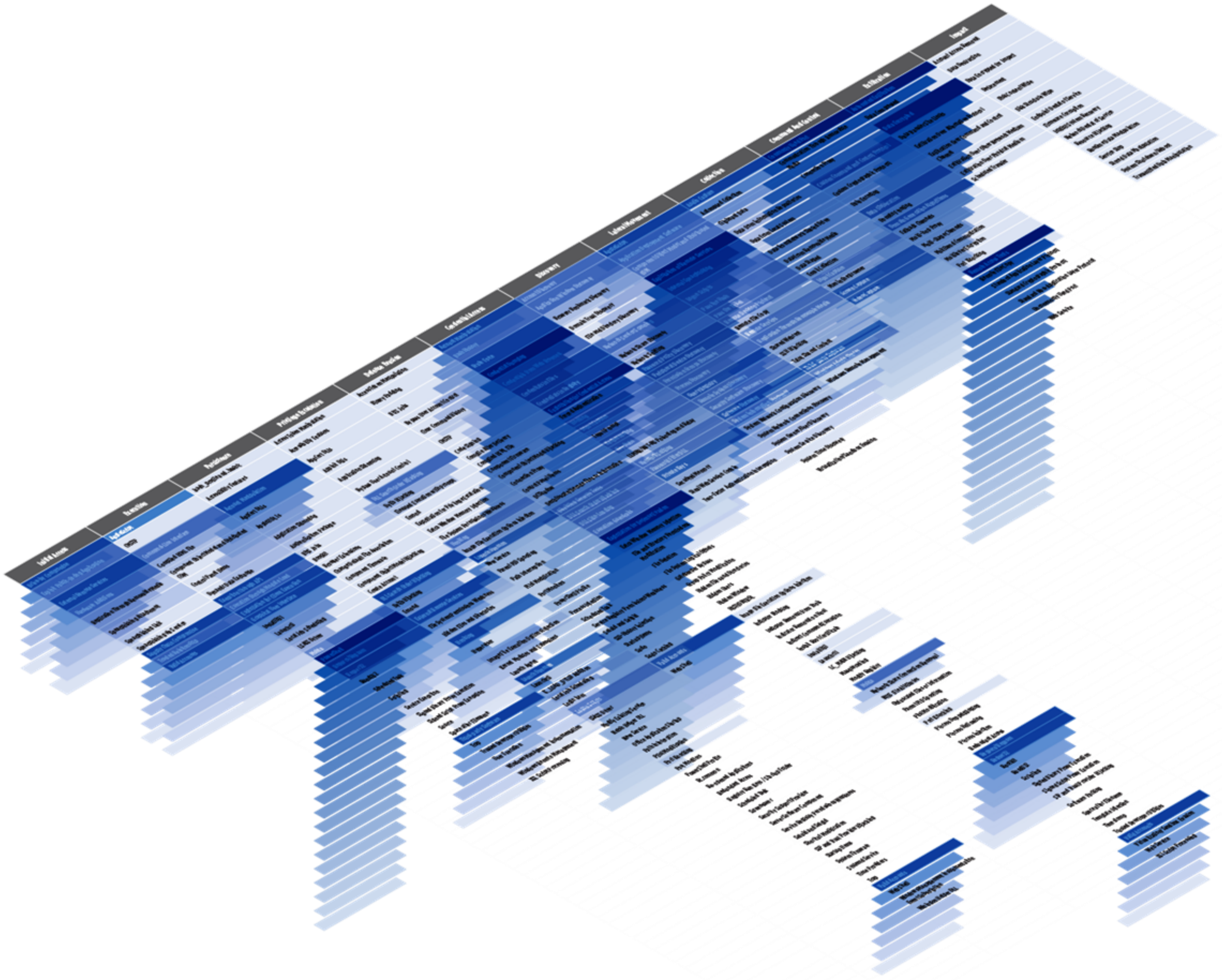
Detection, response, hunting as code

IV

Blue-Team-as-Code Spiral

5-steps, code-based evolution of detection

Blue-Team-as-Code ”Bathymetry”



Detection spectrum - What detection use cases work better for automation?

“Shallow water fishing”
More specific use cases

”Deep water fishing”
More open-ended (De-rabbit-holing)

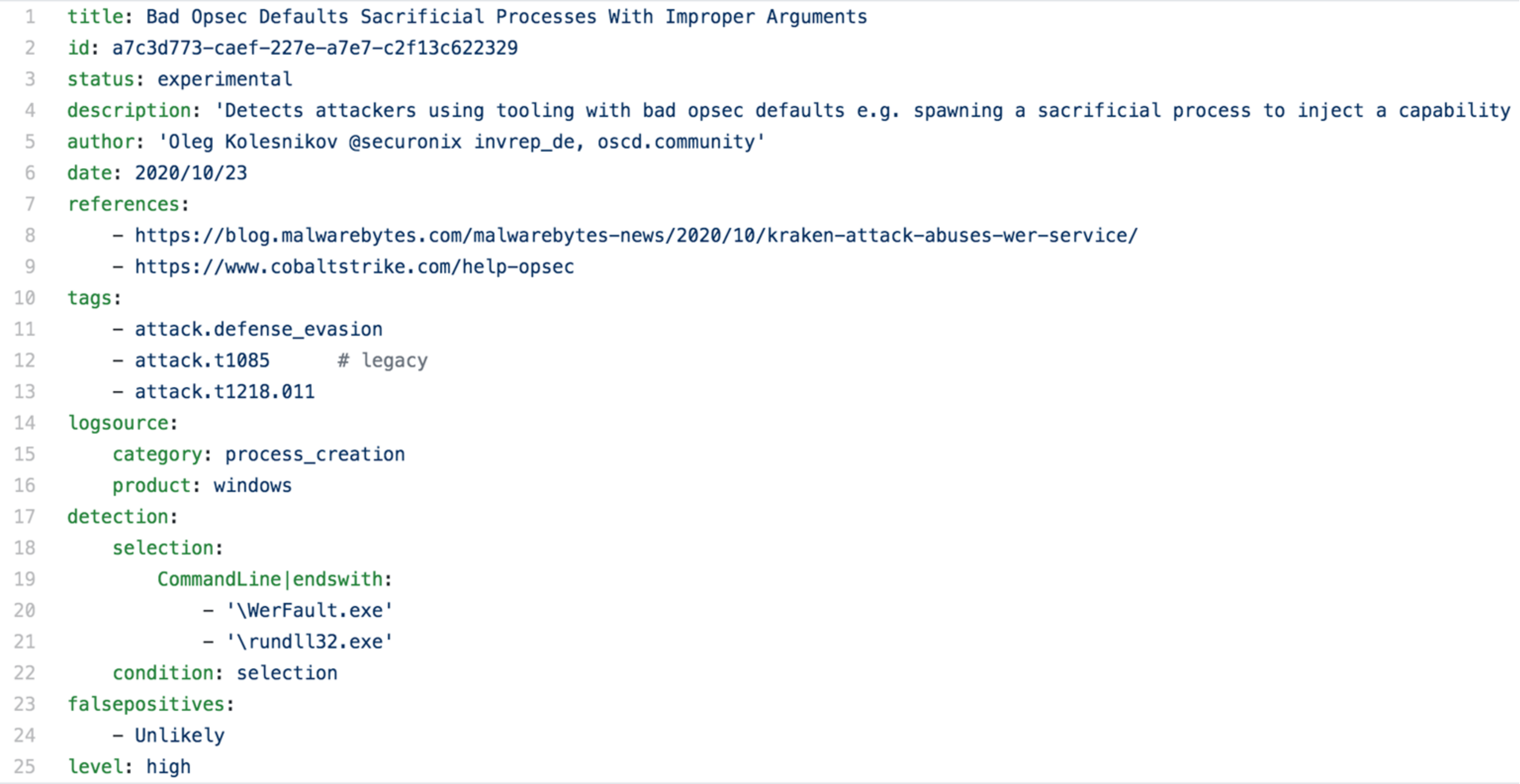
Higher Specificity
Lower fp
High fn

Zero/very few results
For Blue team
To review/pivot

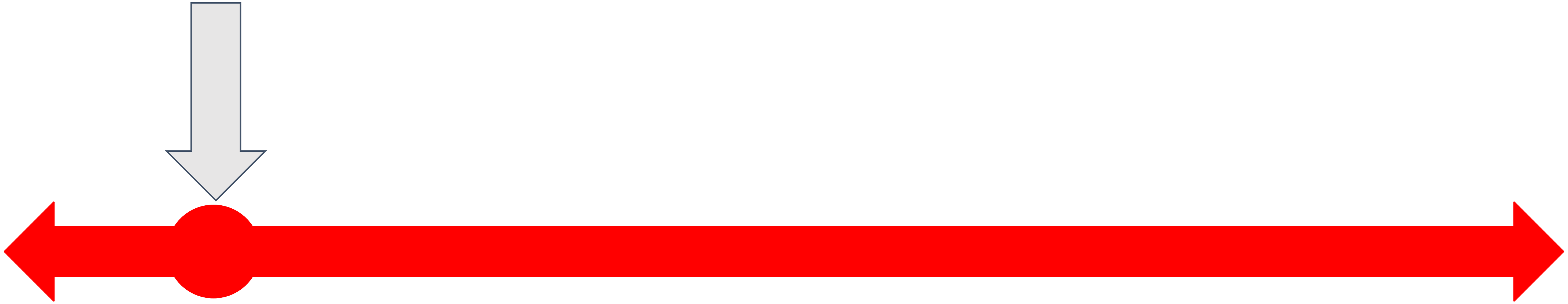
Higher Sensitivity
Higher fp
Low fn

A lot of results
For Blue team
To review/pivot

Example – Bad Opsec - Less deep/more specific



```
title: Bad Opsec Defaults Sacrificial Processes With Improper Arguments
id: a7c3d773-caef-227e-a7e7-c2f13c622329
status: experimental
description: 'Detects attackers using tooling with bad opsec defaults e.g. spawning a sacrificial process to inject a capability
author: 'Oleg Kolesnikov @securonix invrep_de, oscd.community'
date: 2020/10/23
references:
    - https://blog.malwarebytes.com/malwarebytes-news/2020/10/kraken-attack-abuses-wer-service/
    - https://www.cobaltstrike.com/help-opsec
tags:
    - attack.defense_evasion
    - attack.t1085      # legacy
    - attack.t1218.011
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        CommandLine|endswith:
            - '\WerFault.exe'
            - '\rundll32.exe'
    condition: selection
falsepositives:
    - Unlikely
level: high
```



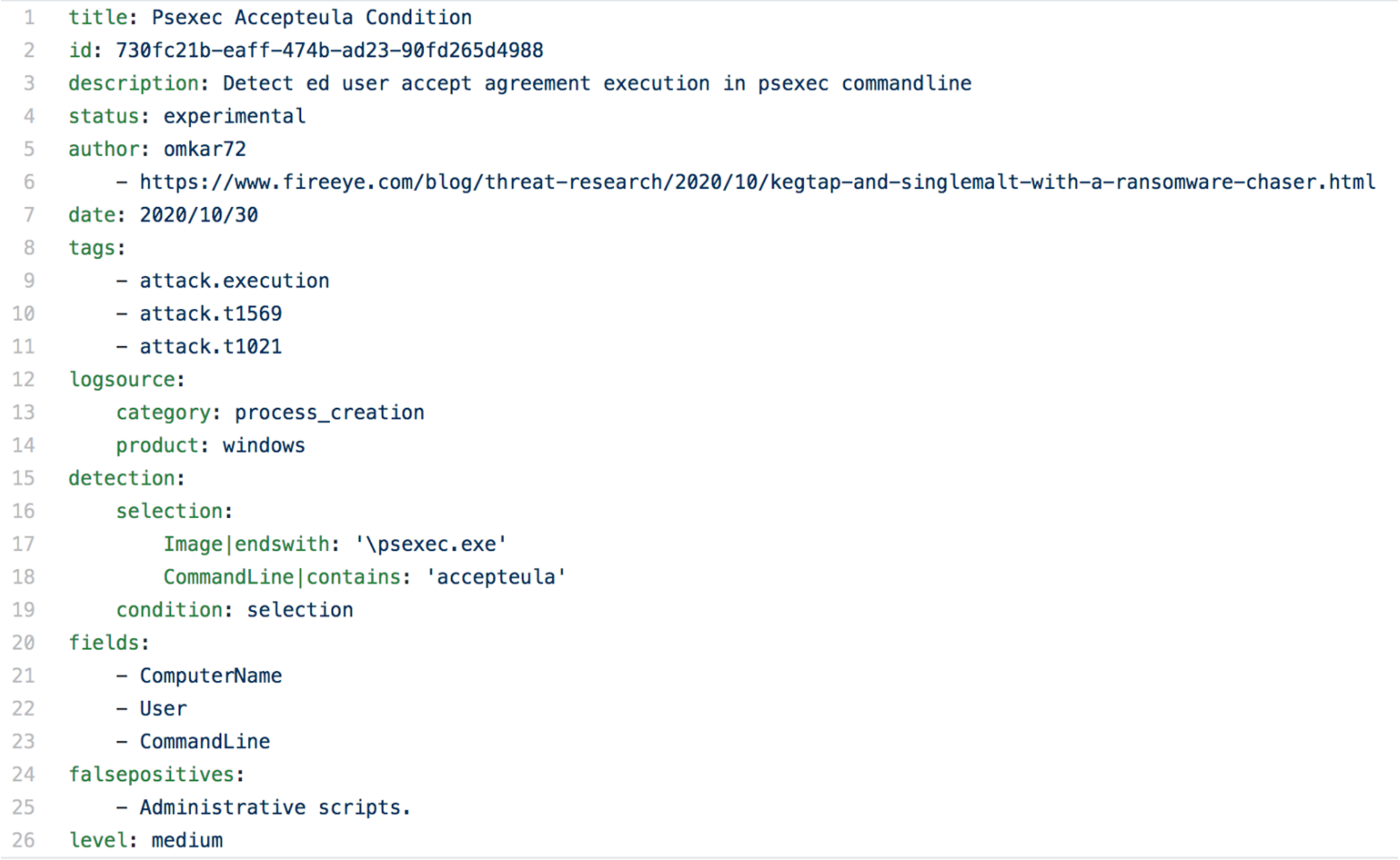
Higher Specificity
Lower fp

Higher Sensitivity
High fp
Low fn

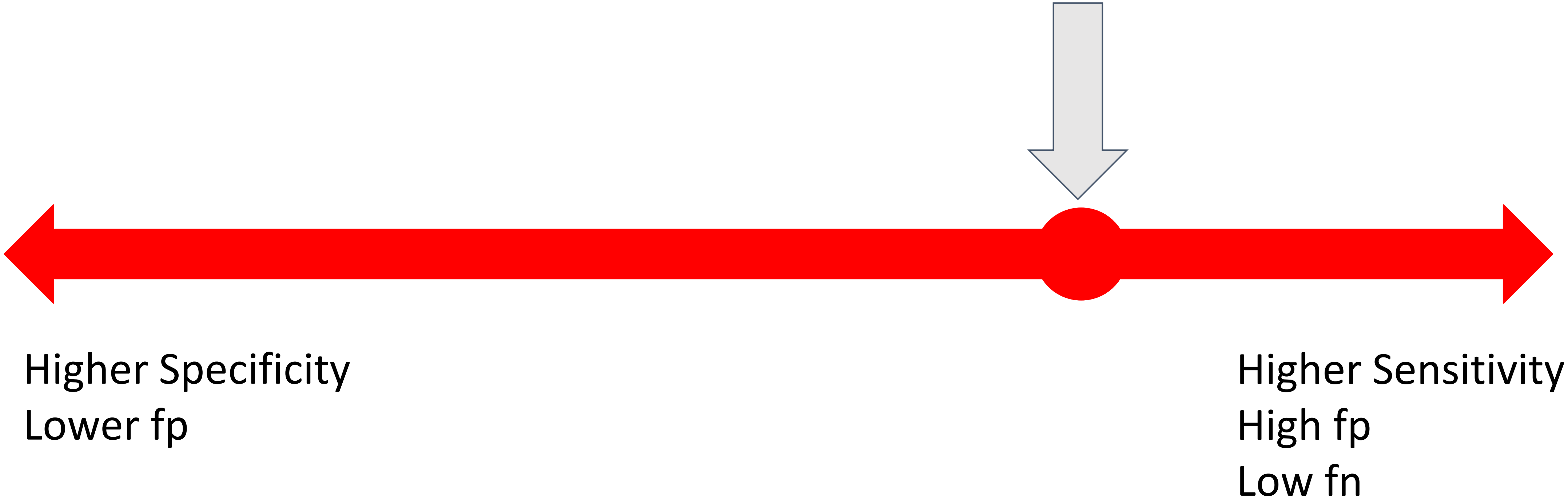
“Bad Opsec” Detection Example – “To the left”

Minimal need for Red Team Automated Detection, good results if caught, but very easy to evade – Red teams can just use, e.g. proc arg spoofing “argue” in CS beacon

Example – Deeper/more sensitive – PSEXEC ACCEPTEULA



```yaml
title: Psexec Accepteula Condition
id: 730fc21b-eaff-474b-ad23-90fd265d4988
description: Detect ed user accept agreement execution in psexec commandline
status: experimental
author: omkar72
    - https://www.fireeye.com/blog/threat-research/2020/10/kegtap-and-singlemalt-with-a-ransomware-chaser.html
date: 2020/10/30
tags:
    - attack.execution
    - attack.t1569
    - attack.t1021
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|endswith: '\psexec.exe'
        CommandLine|contains: 'accepteula'
    condition: selection
fields:
    - ComputerName
    - User
    - CommandLine
falsepositives:
    - Administrative scripts.
level: medium
```



Higher Specificity
Lower fp

Higher Sensitivity
High fp
Low fn

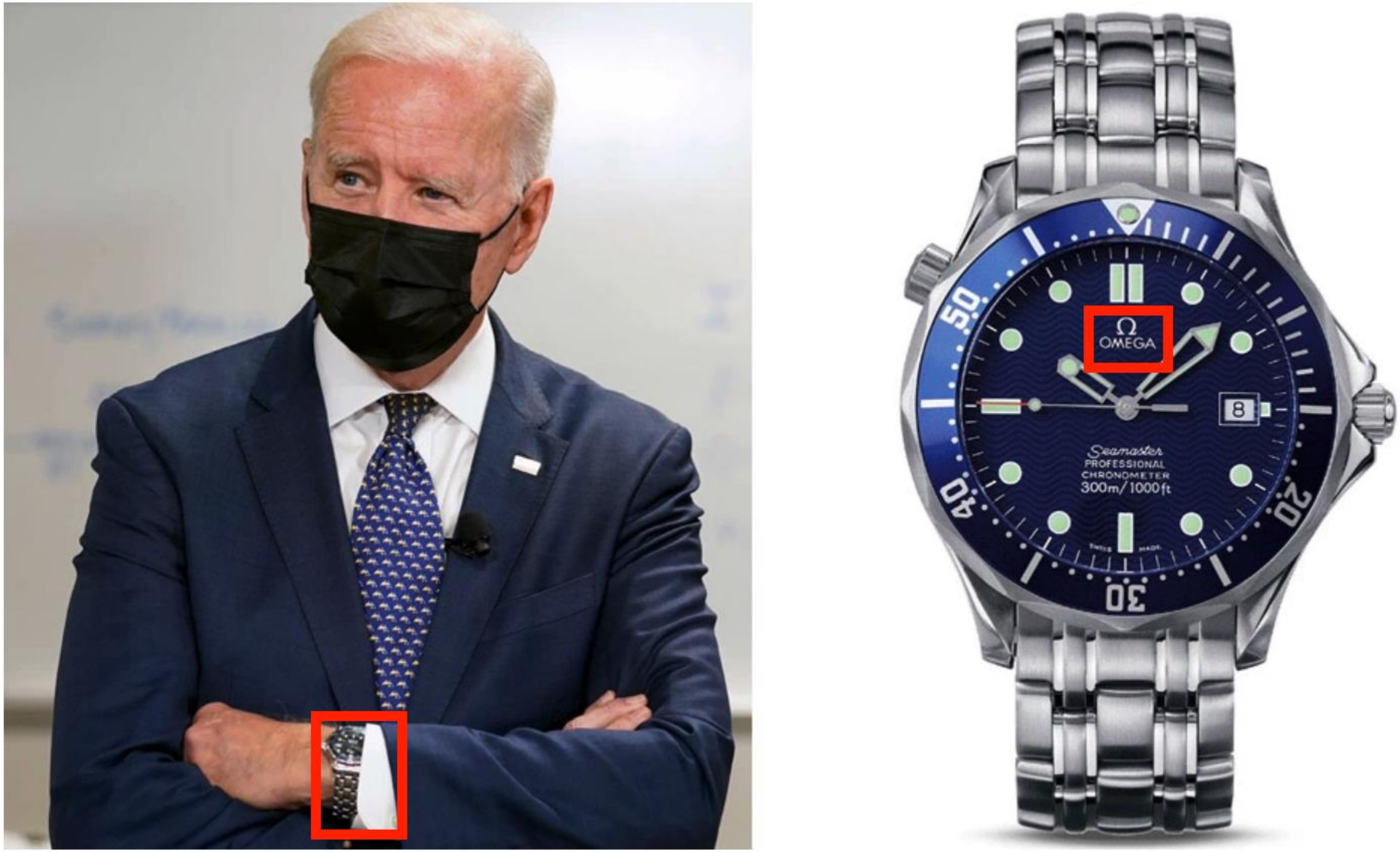
Trivial PsExec accepteula – relatively Sensitive, more to the right

In many envs, need to automate/de-rabbit-holing, especially when PsExec is used as part of admin activity

Use both specific & sensitive detection use cases to detect Red Teams, but how can we filter the results produced by more sensitive use cases?

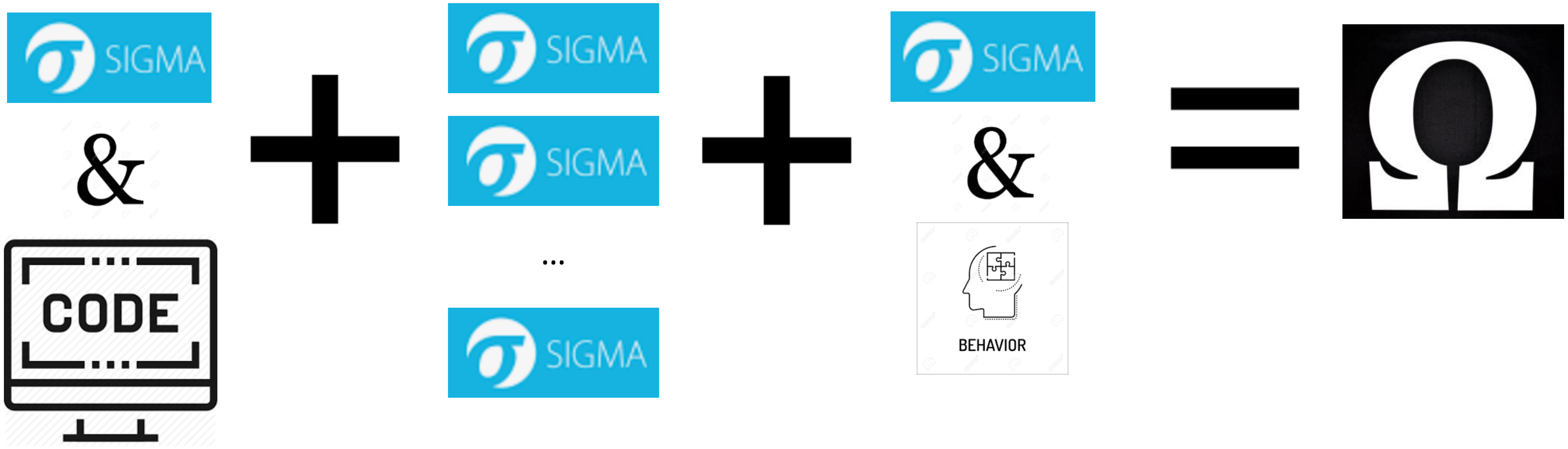
Use both specific & sensitive detection use cases to detect Red Teams, but how can we filter the results/fp produced by sensitive use cases?

Add code to your detections! You can use Ω

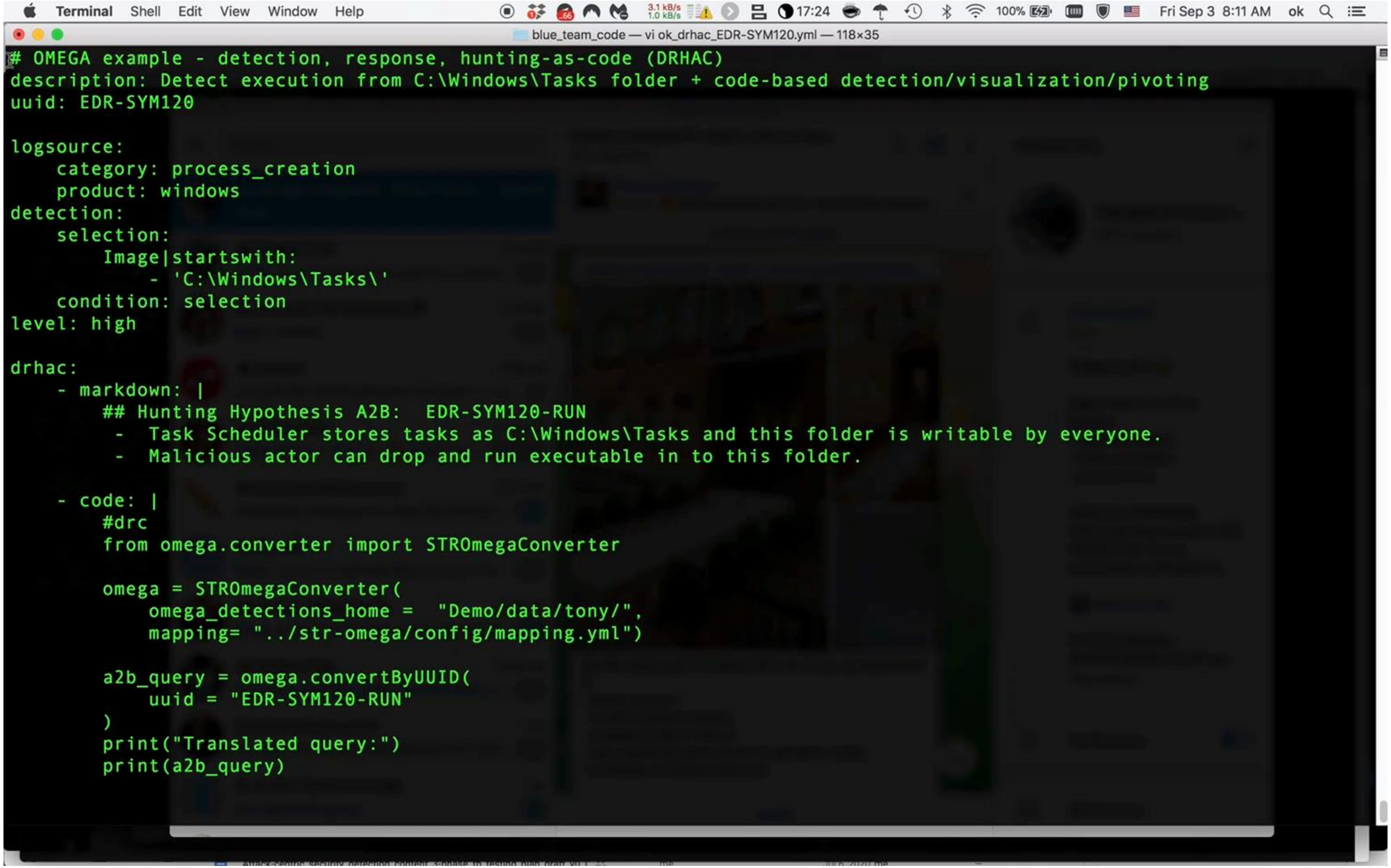


AP Photo/Evan Vucci/Omega

OMEGA - Open-source Framework, Backward-compatible with Sigma, part of Detection-Response-Hunting-as-Code (DRHAC) concept

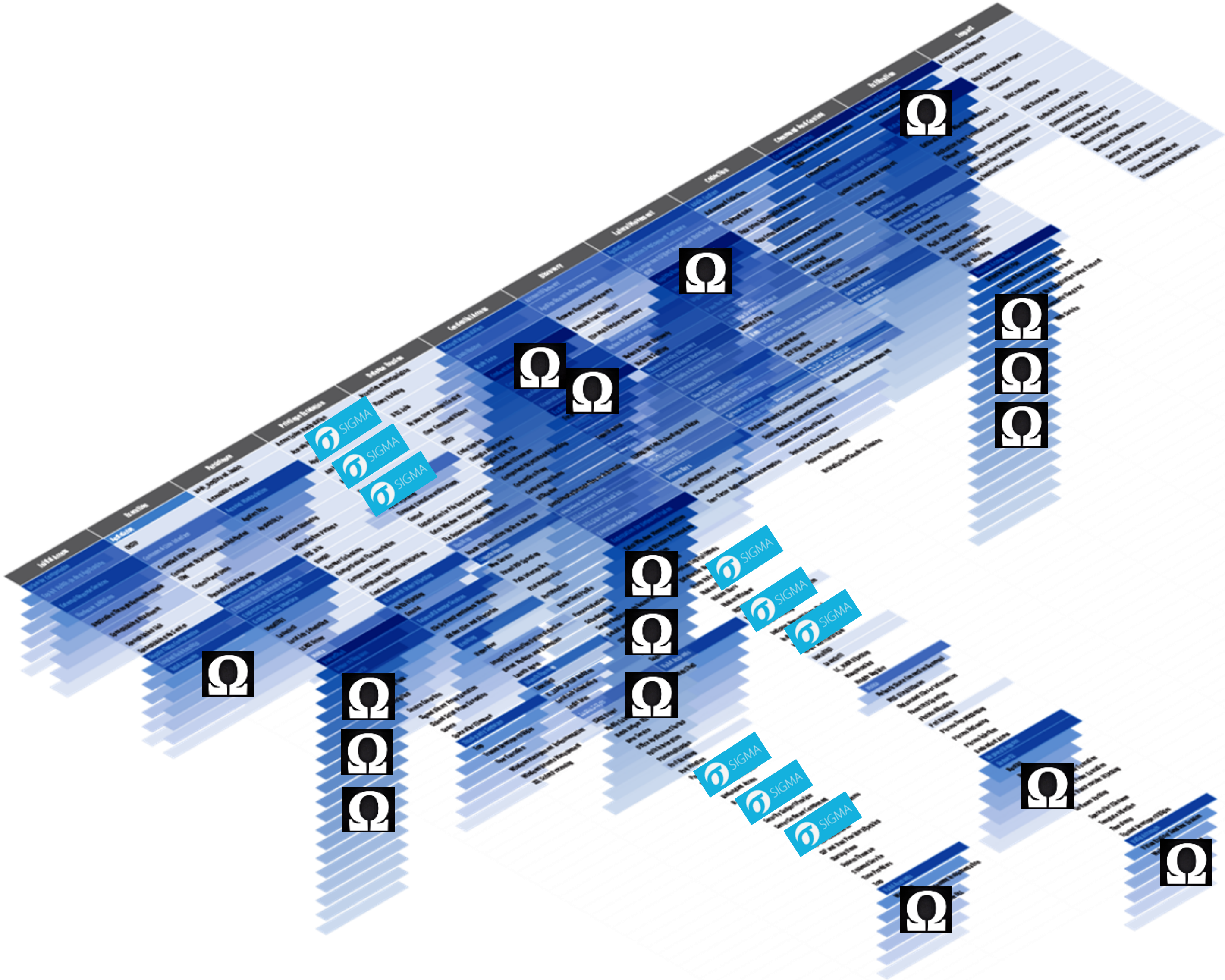


Example – OMEGA [see github for more]



```
# OMEGA example - detection, response, hunting-as-code (DRHAC)
description: Detect execution from C:\Windows\Tasks folder + code-based detection/visualization/pivoting
uuid: EDR-SYM120

logsource:
    category: process_creation
    product: windows
detection:
    selection:
        Image|startswith:
            - 'C:\Windows\Tasks\'
    condition: selection
level: high

drhac:
    - markdown: |
        ## Hunting Hypothesis A2B:  EDR-SYM120-RUN
         -  Task Scheduler stores tasks as C:\Windows\Tasks and this folder is writable by everyone.
         -  Malicious actor can drop and run executable in to this folder.

    - code: |
        #drc
        from omega.converter import STROmegaConverter

        omega = STROmegaConverter(
            omega_detections_home =  "Demo/data/tony/",
            mapping= "../str-omega/config/mapping.yml")

        a2b_query = omega.convertByUUID(
            uuid = "EDR-SYM120-RUN"
        )
        print("Translated query:")
        print(a2b_query)
```

Specific & Sensitive+Code Detections to improve Red Team Detection



0x02 – Tradecraft, Blindspots, and Ways to Address Using Code

I

Tradecraft Highlights from Red Teams

Examples of latest Red team tradecraft relevant to Blue-team-as-code

II

Common Blue Team Blindspots

Some common blue team blind spots

III

Ways to address using Code/Lessons Learned

Some Red Team Tradecraft Highlights/Trends related to Blue-team-as-code Automation

More Opsec-aware & automated Red Teams

HIDING IN BENIGN/KNOWN GOOD - More covert CS Beacon BOFs
.NET/C# payloads,
Evade telemetry/detection via Syswhispers, Opsec-safe Spawntos
Custom Open source/Under-detection-threshold payloads,
Malleable customizations,
Env-keyed Payloads,
AV/EDR Evasion/enum_filter_driver BOFs, Parent/args spoofing, etc.

Exploit Blue Team Visibility

Short-lived off-disk payloads, in-vitro probing, observing Blue Teams-Red ELK/SIEM, c2 reflector/backend covertness (minimal stagers, separate persistence/stagers, etc)

More Eradication-resistance

Resist/Hide from Get-InjectedThread.ps1,
More Process Hollowing/Module Stomping,
Artifact Kit telemetry obfuscation,
Sleep_masks, etc

Red-Team-as Code

Moving toward more code-driven red teams, aggressor .cna community automations [5]

Common Blue Team Blindspots relevant to Blue-team-as-Code – Required Telemetry

Blindspot #1: Basic Required Telemetry Missing

Having required telemetry is key for Blue-team-as-code automation, e.g.

- Raw EDR/process activity/network activity logs (basic AV/EDR often not enough!)
- Powershell SBL logs
- NTA logs
- More detailed logs for crown jewels e.g. ETW + custom/lower level logs, e.g. SMB/CIFS, if feasible etc.

Q: Our CISO scheduled a red team exercise next week, we have some basic AV, Proxy, Firewall logs, how can we do code automation for our detection use cases so our blue team can detect the red team activity more effectively?

A: You need to take care of the basics first, get the required telemetry, without it, your Return-on-Automation-Investment (ROAI) is going to be very limited, sorry, there is no way around it.”

Common Blindspots relevant to Blue-team-as-Code – Lack of automation tooling to leverage sensitive detections effectively

Blindspot #2: Inability to monitor known good/leverage sensitive

- ⇒ **Overreliance on specific known-bad detections/IOCs since sensitive are often unfeasible to use w/o automation;**
- ⇒ **Red teams often hide in benign/known good**
- ⇒ **Need tools to be able to help automate processing results of more sensitive detections effectively (visualize, baseline, pivot, chain etc);**

DON'T

- Expect red teams to use payloads that'll trigger your AV/EDR/whitelisting and/or write payloads to disk
- Expect red teams use unobfuscated payloads and often spawn suspicious parent-child processes
- Expect red teams to spawnto rundll32.* or svchost/notepad etc
- Expect red teams to use community attack frameworks as-is w/no changes
- Expect grab_beacon_config to work on many c2s – increased red team opsec awareness is trending

DO:

- Expect red team to run payloads *via* your whitelisted apps, dll sideload, less defense-sensitive lolbins [6] etc.
- Expect your AV/*DR to be evaded/disabled
- Expect customized/significantly obfuscated community attack tools and frameworks
- Expect in-memory payloads w/minimal footprint, no rwx, unhooking (partial telemetry), no unassociated thread start addrs (Get-InjectedThread.ps1 resistance), c2 to common cloud infra (Malleable) etc.

Ways to address using Code / Where Blue-team-as-Code Automation Could help

1 **Required Logs/Telemetry**
Critical, no BAC without it

2 **Visualize effectively**
Logon sessions, decoding etc

3 **Chain attack techniques**
Connect sensitive detections

4 **Cluster from code**
Dynamic clustering using code

5 **Auto baseline use cases**
Automated (canned) models

6 **Code-driven baseline**
Dynamic baselining from code + early chatter

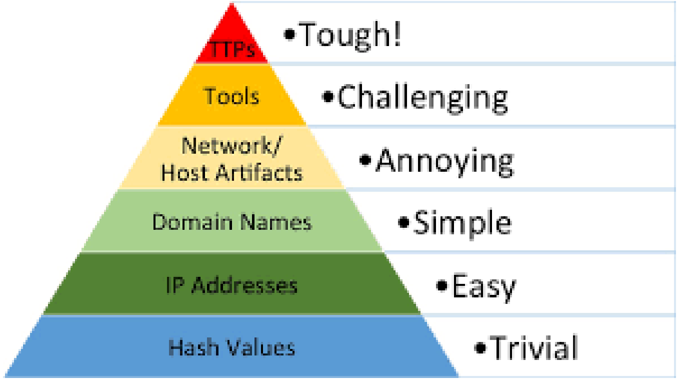
7 **Evolve using code**
Code-generated/evolved detections

8 **Specific+sensitive+code**
Combine both specific & sensitive

9 **Code-based eradication**
Automate eradication (SOAR etc)

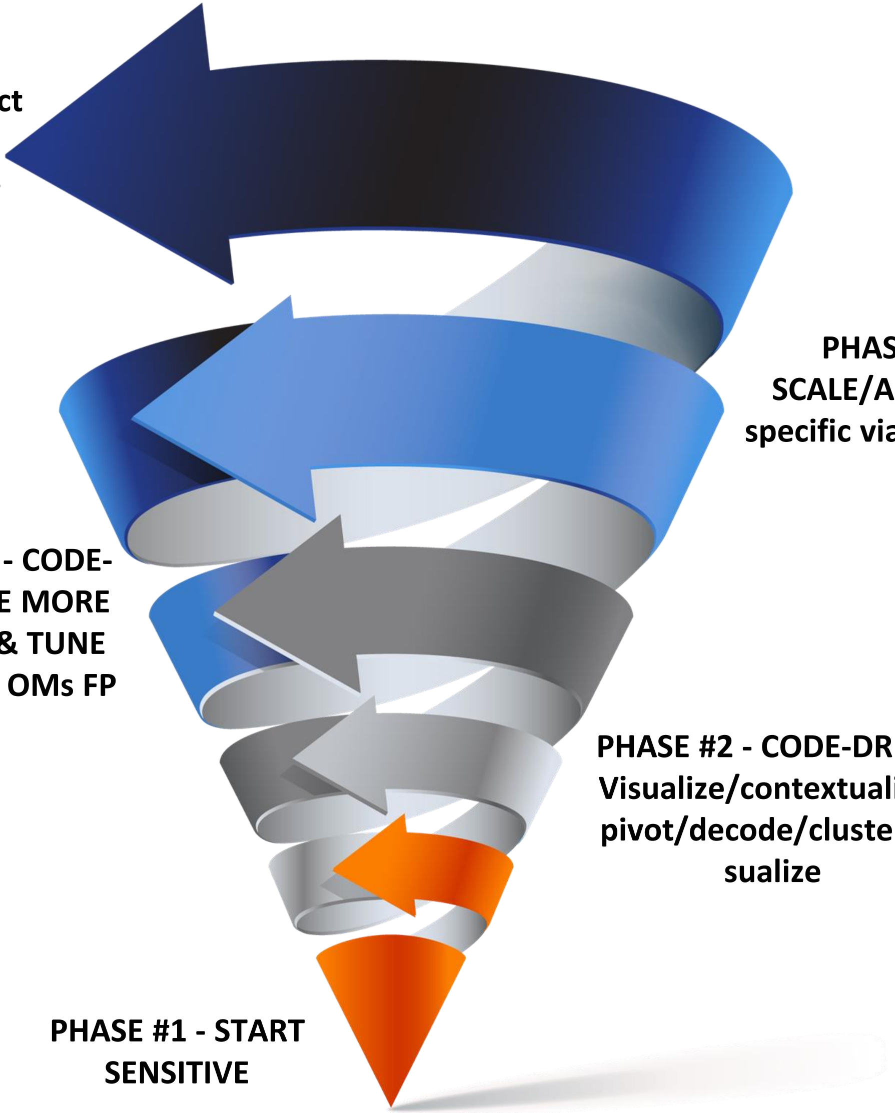
MAKE SURE TO ALSO SCRUTINIZE YOUR **KNOWN-GOOD**-THAT’S WHERE RED TEAMS INCREASINGLY HIDE

From “Pyramid of Pain” to Blue-team-as-code “Spiral of Joy”



TTPs	•Tough!
Tools	•Challenging
Network/ Host Artifacts	•Annoying
Domain Names	•Simple
IP Addresses	•Easy
Hash Values	•Trivial

→



PHASE #5 – JOY: Detect red team/eradicate w/SOAR using code-enhanced specific & sensitive detections

PHASE #4 - APPLY AT SCALE/ACTION: Apply more specific via Autonomous Threat Hunting.

PHASE #3 - CODE-GENERATE MORE SPECIFIC & TUNE SENSITIVE OMs FP

PHASE #2 - CODE-DRIVE: Visualize/contextualize/ pivot/decode/cluster/visualize

PHASE #1 - START SENSITIVE

0x03 – DEMO – Putting Blue-team-as-Code Fundamentals to Use

0x03

Demos & Practical Examples

Real-world Red Team detection using code

0x03 – DEMO - Real-world Red team detection using Code - Example

I

Setting the stage

Hypothesis, context, pivoting

II

Demos & Practical Examples

Putting Blue-team-as-Code Fundamentals to Use

III

Bonus - Demo

Trivial code-generated detection for Blue teams Spiral-of-Joy

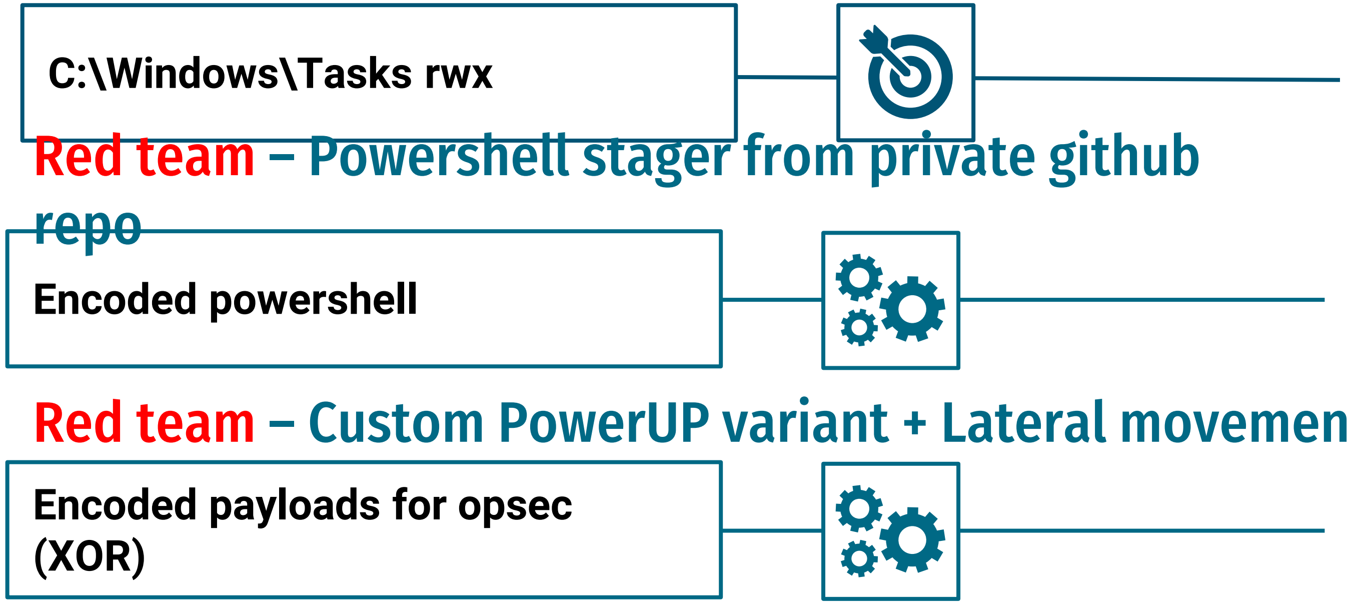
0x03 – DEMO - Real-world Red team detection using Code - Example

I

Setting the stage

Hypothesis, context, pivoting

Blue-team-as-code: Real-world Red Team Detection Example – Part I (Red team)



Red team – Multiple techniques, including Persistence payload drop

C:\Windows\Tasks rwx

Red team – Powershell stager from private github repo

Encoded powershell

Red team – Custom PowerUP variant + Lateral movement

Encoded payloads for opsec (XOR)

Blue-team-as-code: Real-world Red Team Detection Example – Part II (Blue team)

Blue-team-as-code – Use code-driven sensitive detection (scheduled tasks folder executable)

Use code to process detection results

Blue-team-as-code – Visualize logon session

Visualize logon sessions + code reuse

Blue-team-as-code – Extract artifact + perform auto code-driven scan/extrapolate

Extract custom encoded artifact (”JAB”) &
Perform code-driven scan on other systems

Blue-team-as-code– Detect customized PowerUP + pvt github repo OSINT = Identify Sr. Red Team member

RED TEAM DETECTED! =>Code-evolve Blue-team detections as next step

Hunting Hypothesis A2B: EDR-SYM120-RUN

- Task Scheduler stores tasks as C:\Windows\Tasks and this folder is writable by everyone.
- Malicious actor can drop and run executable in to this folder.

Load this query from A2B: EDR-SYM120-RUN

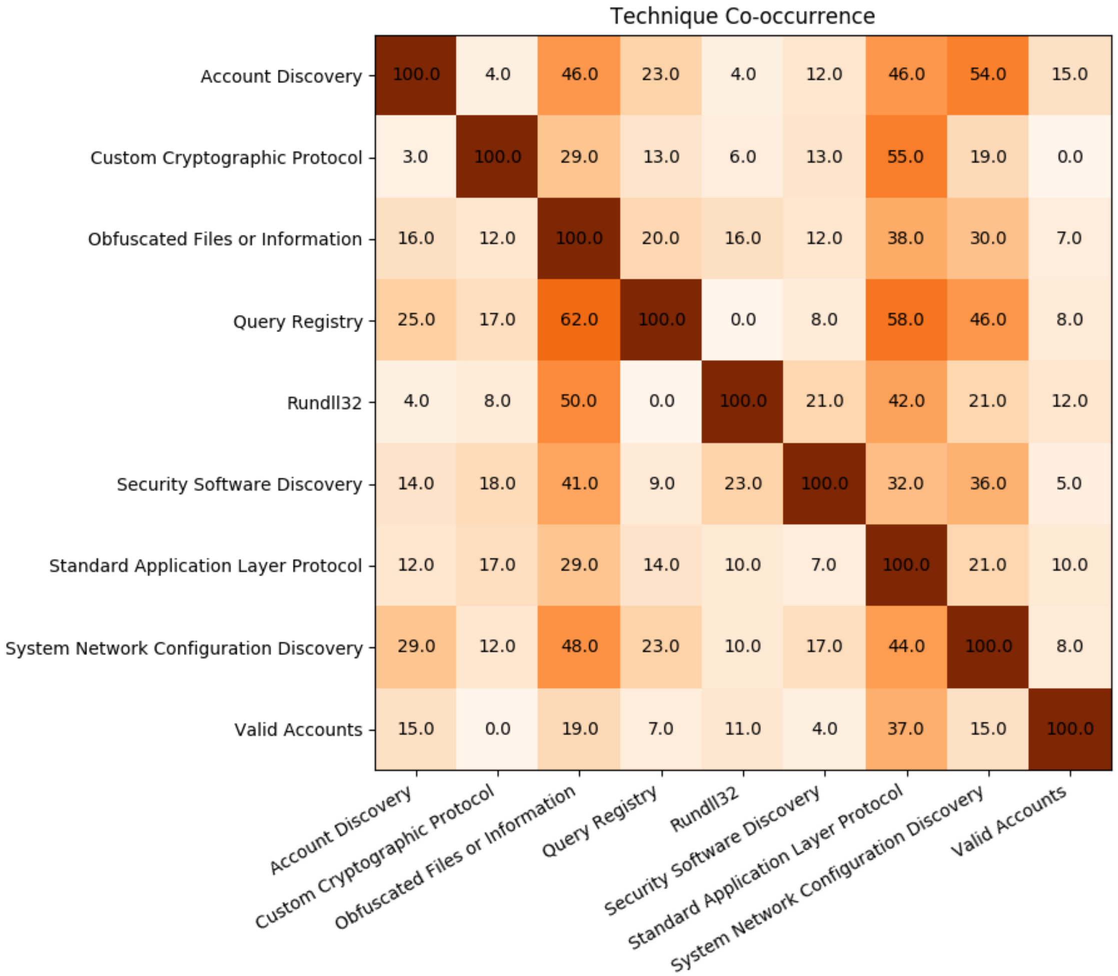
```
In [ ]:
```

```
In [ ]: from omega.converter import STROmegaConverter

        omega = STROmegaConverter(
            omega_rules_home =  "../oms/a2b/",
            mapping= "../str-omega/config/mapping.yml"
        )

        a2b_query = omega.convertByUUID(
            uuid = "EDR-SYM120-RUN"
        )
        print("Translated query:")
```

Demo

Putting Blue-team-as-Code Fundamentals to Use

Blue-team-as-code: Attack Chaining & De-rabbit-holing - How can we use code to automatically combine ATT&CK techniques that co-occur to improve S/N ratio

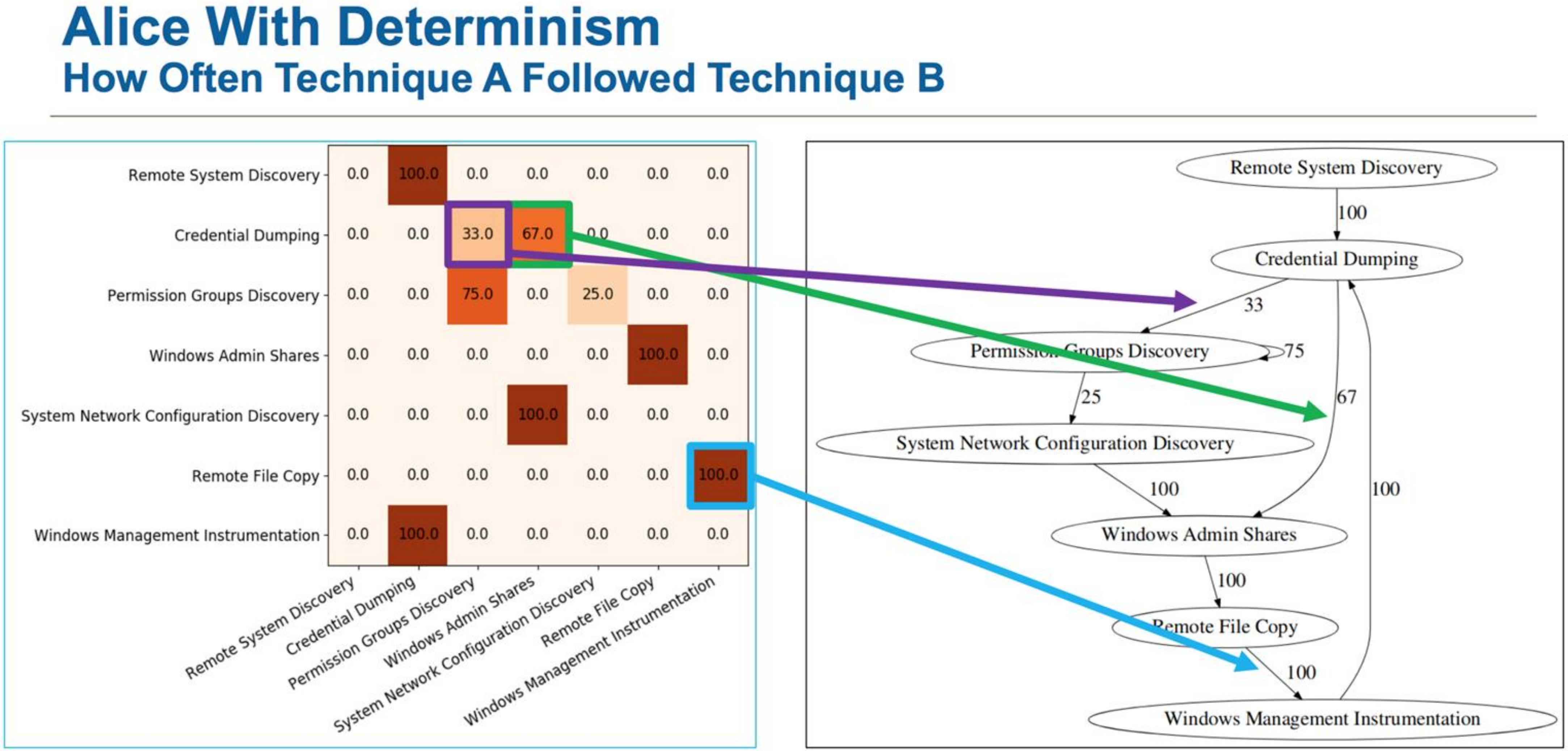


Technique Co-occurrence

	Account Discovery	Custom Cryptographic Protocol	Obfuscated Files or Information	Query Registry	Rundll32	Security Software Discovery	Standard Application Layer Protocol	System Network Configuration Discovery	Valid Accounts
Account Discovery	100.0	4.0	46.0	23.0	4.0	12.0	46.0	54.0	15.0
Custom Cryptographic Protocol	3.0	100.0	29.0	13.0	6.0	13.0	55.0	19.0	0.0
Obfuscated Files or Information	16.0	12.0	100.0	20.0	16.0	12.0	38.0	30.0	7.0
Query Registry	25.0	17.0	62.0	100.0	0.0	8.0	58.0	46.0	8.0
Rundll32	4.0	8.0	50.0	0.0	100.0	21.0	42.0	21.0	12.0
Security Software Discovery	14.0	18.0	41.0	9.0	23.0	100.0	32.0	36.0	5.0
Standard Application Layer Protocol	12.0	17.0	29.0	14.0	10.0	7.0	100.0	21.0	10.0
System Network Configuration Discovery	29.0	12.0	48.0	23.0	10.0	17.0	44.0	100.0	8.0
Valid Accounts	15.0	0.0	19.0	7.0	11.0	4.0	37.0	15.0	100.0

SOURCE: MITRE [2]

Starting point – Great work by MITRE/Caldera team on Chaining/pre/postconditions



Alice With Determinism

How Often Technique A Followed Technique B

	Remote System Discovery	Credential Dumping	Permission Groups Discovery	Windows Admin Shares	System Network Configuration Discovery	Remote File Copy	Windows Management Instrumentation
Remote System Discovery	0.0	100.0	0.0	0.0	0.0	0.0	0.0
Credential Dumping	0.0	0.0	33.0	67.0	0.0	0.0	0.0
Permission Groups Discovery	0.0	0.0	75.0	0.0	25.0	0.0	0.0
Windows Admin Shares	0.0	0.0	0.0	0.0	0.0	100.0	0.0
System Network Configuration Discovery	0.0	0.0	0.0	100.0	0.0	0.0	0.0
Remote File Copy	0.0	0.0	0.0	0.0	0.0	0.0	100.0
Windows Management Instrumentation	0.0	100.0	0.0	0.0	0.0	0.0	0.0

SOURCE: MITRE [2]

Blue-team-as-code: Attack Chaining Example – Applocker evasion/Lsass Minidump

Red team - Msbuild *.xml

Covert red team payload compilation

Red team - Lsass dump from C#

Covert lsass cred dump

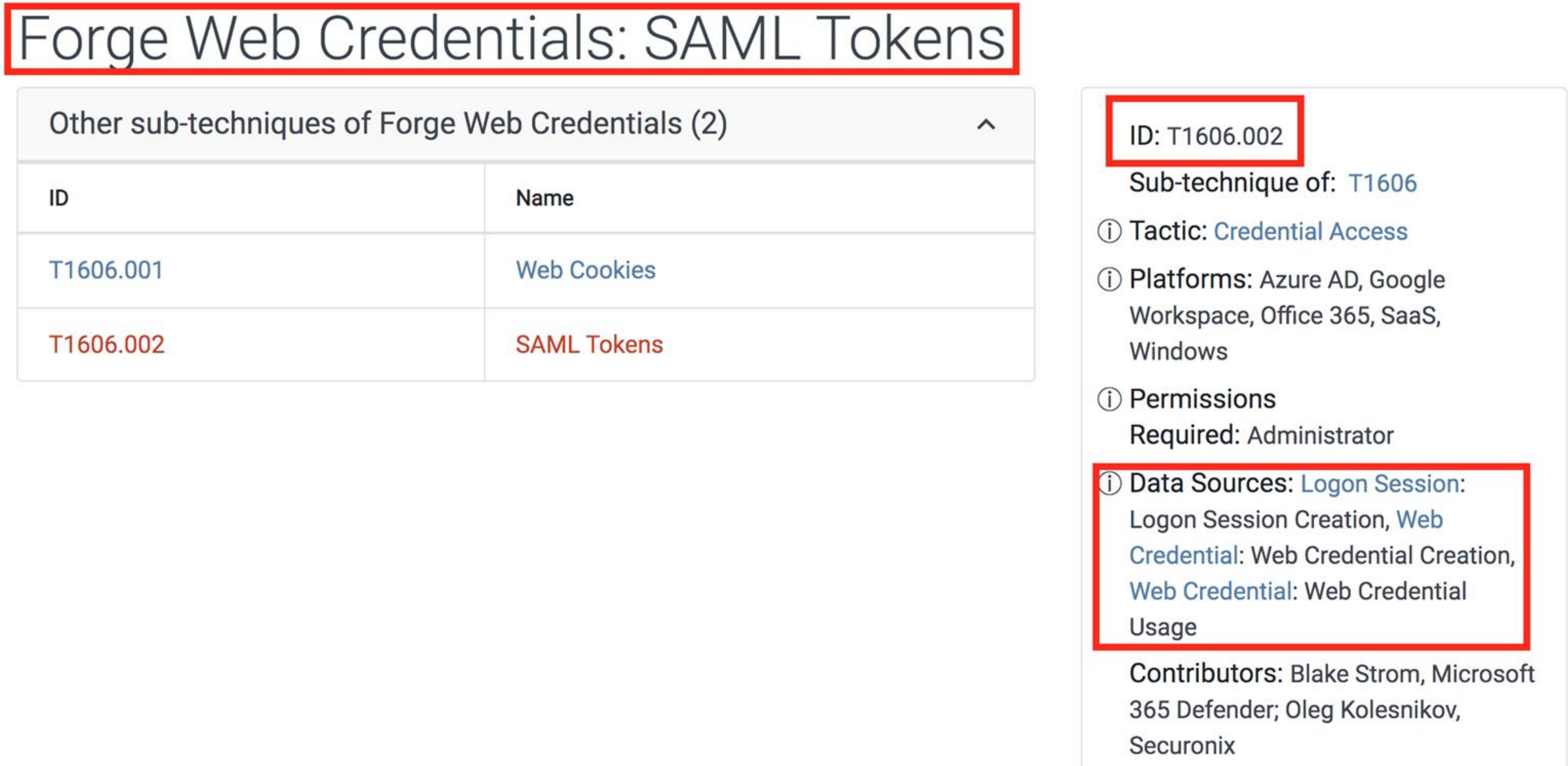
Blue-team-as-code – Chain two sensitive msbuild + lsass proc access flags

Automate by chaining two techniques

Nano - Demo

BAC Attack chaining

Blue-team-as-code – Code Filling in the Log Source gaps – Siloed data sources – on prem vs. cloud



Forge Web Credentials: SAML Tokens

Other sub-techniques of Forge Web Credentials (2)

ID	Name
T1606.001	Web Cookies
T1606.002	SAML Tokens

ID: T1606.002

Sub-technique of: T1606

ⓘ Tactic: Credential Access

ⓘ Platforms: Azure AD, Google Workspace, Office 365, SaaS, Windows

ⓘ Permissions Required: Administrator

ⓘ Data Sources: Logon Session: Logon Session Creation, Web Credential: Web Credential Creation, Web Credential: Web Credential Usage

Contributors: Blake Strom, Microsoft 365 Defender; Oleg Kolesnikov, Securonix

Detect logins to service providers using SAML SSO w/no corresponding 4769, 1200, and 1202 events in the Domain.

Example 2 - Malicious Oath2 App Phishing in Cloud by Red Team

Red team – Phish your users by sending a malicious app access request

Malicious app perm phishing to access your users O365 mailboxes

Blue-team-as-code - Fill gaps in disjoint logs & find malicious app quickly

Code pulls missing artifacts from different disjoint logs and enriches to review/detect

Nano - Demo

BAC Code Filling-in Gaps in Cross-realm/Disjoint logs

Blue-team-as-code: Detecting Chrome Cookie Dumping/Passing + Code-based ERADICATION - SYNOPSIS

Red team – Spawn Chrome w/debug port/incoming debug cookie dump

--remote-debugging-port + IEX

Blue-team-as-code – Identify cookie dump and eradicate

Identify cookie dump stager, check hash, and stop the malicious payload from code

Nano - Demo

Red Team Cookie Dumping/Passing + Code-based ERADICATION

Blue-team-as-code: Code-generated/evolved Detection Example: Rundll32 Sandbox bypass - SYNOPSIS

Red team - Sandboxing bypass through custom dll invocation

Rundll32.exe <dll_payload>.dll, <func>

Blue team-as-code – Use sensitive rundll32 detection + code to generate specific

Check for rundll3.exe w/DllRegisterServer

III

Bonus - Demo

BAC Code-generated Detection

0x04 – Conclusions/Key Takeaways

Required telemetry

Logs/telemetry critical for BAC implementation!

Use both specific & sensitive + code

Don't ignore sensitive, find a way to use / reduce noise w/code

Implement Blue team code automation

Make it part of your SOP

Blue Team Spiral-of-Joy

Code-generated artifacts, use code to evolve detections, tune sensitive detections

Apply What You Have Learned Today

- **Next week you should**: Identify missing feature-rich log/data sources (B1) & “more sensitive” use cases => candidates for your red team detection automation & scope initial automation/bathymetry (techniques, code primitives, combining techniques/clustering, visualization, logon sessions, etc).

- **In the first three months following the presentation you should:** Add missing telemetry and implement some initial basic Blue-team-as-code automation based on the “bathymetry” of your environment techniques from red team perspective, what is used and not used by your DevOps e.g. wmi, psexec, msbuild, etc.

- **Within 6-12 months you should:** Augment your blue team capabilities w/code by implementing key code primitive from Blue-team-as-Code “Spiral of Joy” into your SOP, automating-automation, “evolving” your detections to ensure you are not only operating on the basis of high-specificity but can also leverage high-sensitivity effectively.

Credits/References

[1] Orlando, Mark. Cobot Uprising: Smart Automation for Blue Teams with Mark Orlando. SANS Blue Team Summit 2020. https://www.youtube.com/watch?v=jp-PuLnd9EQ

[2] Applebaum, Andy. Finding Dependencies Between Adversary Techniques. MITRE/First Conference 2019. https://www.first.org/resources/papers/conf2019/1100-Applebaum.pdf

[3] Chuvakin, A. Can we have detection as code. https://medium.com/anton-on-security/can-we-have-detection-as-code-96f869cfdc79

[4] Security Boulevard. Detection Development Lifecycle. https://securityboulevard.com/2020/09/ddlc-detection-development-life-cycle/

[5] CobaltStrike Community Kit. https://cobalt-strike.github.io/community_kit/

[6] Lolbas Project. lolbas-project.github.io. Living Off The Land Binaries and Scripts (and also Libraries). https://lolbas-project.github.io

[7] Kolesnikov, O. et al. Detection, Response, and Hunting-as-code (DRHAC). Medium.

Special thanks to Florian Roth/Sigma community, Ismael Valenzuela/SANS, and all others security experts/Blue/Red teamers who provided their inputs/feedback.

Den Iuzvyk

Oleg Kolesnikov

Questions?

SECURONIX

Security Analytics. Delivered.