

Maturing SecOps with Detection-as-Code

David French & Wade Wells

Blue Team Con 2024

David French

- 18+ years in IT and cybersecurity
 - Blue Team: Detection Engineer, Threat Hunter, SOC Analyst
 - Vendor Stuff: Security Researcher, Detection Engineering, building SIEMs & EDRs
- Currently at Google Cloud (Google Security Operations)
- Formerly Twilio, Elastic, Endgame, Capital Group
- Enjoys sharing knowledge & research:
 - Speaker at Black Hat, BSides, FIRST
 - [Blog](#), [community contributions](#), [Detection-as-Code](#), [Dorothy](#)
- When I'm not working, I'm hiking, fishing, cycling, etc



Wade Wells

8 Years of Sec Ops(Threat Hunting, Threat Intel, Detection Engineering).



MS Cybersecurity Georgia Tech, GCFA, GMON, OMGBBQ,



BHIS Talkin' Bout News Regular, BSides San Diego Board , Antisyphon Instructor



Create By Dan Rearden(Haircutfish)

Agenda

1. A Symphony of SecOps Struggles
2. An Introduction to Detection-as-Code (DaC)
3. Implementing DaC
4. Maturing SecOps with DaC
5. Final Thoughts
 - a. Real-World Challenges
 - b. Key Takeaways
 - c. Useful Resources
6. Q&A

A Symphony of SecOps Struggles

Difficult to Understand Detection Coverage

- A lot of Detections
 - what's covered?
 - has it been tested?
- A lot of tools
 - Not everyone has access?
- Passing Knowledge on Detections



Passing Knowledge on Detections

- Character limit in detections and alerts
 - Hard to explain the goal in 255 characters.
- Bad or no formatting
- Limited amount of data
 - Tags
 - Dates
 - What a tune was made

Edit Custom Rule

Rule Details Condition Response Summary

Rule Details

Rule Name: PowerShell using FTP protocol

Description: The use of the legacy FTP protocol from PowerShell scripts (T1059) are quite likely an indicator of evasion or data exfiltration (TA0010) when using legacy FTP (T1071.002)

Rule Severity: Low Rule Type: Temporary Expiration Date: Sep 9, 2021

Condition

Scope Hierarchy: Account Query Type: Events

Query: SrcProcName = "powershell.exe" and NetProtocolName In AnyCase ("ftp")

Response

✓ Treat as a threat: Suspicious Threat Policy

☒ Activate rule immediately after saving

Back Activate

Possible Persistence Regmod - Urlsearchhooks Last updated: 11:39:53am, Mar 15, 2019 4

This query will identify processes that modify registry keys that may be used by attackers for malware persistence. Threat: In order to gain persistence on a target, attackers will often modify registry keys that will allow their malware to start up again across reboots. False Positives: Some applications may legitimately modify this registry key during installation. Score: 40

registry persistence attackframework attack windows evasion t1112

1 IOC

Take Action

IOC	ACTIONS
<input type="checkbox"/> ((regmod_name:"software\microsoft\internet explorer\urlsearchhooks")) -legacy:true	



No Change Control for Detection Content

- Traditional approach: Manually configure rules & signatures in security tools
- No input/review from anyone else on the team
- People make mistakes
- Many security tools don't provide version control for rules
- Context around changes is often lacking



An Introduction to Detection-as-Code

What is Detection-as-Code?

- A set of principles that use code & automation to manage detection content
- Leverages software development practices & tools to create, manage, & test detection content
- Treats detection content as code artifacts
 - Doesn't necessarily mean you're writing detections *in* code (e.g. Python)
- Modern SecOps teams want everything “as-code”

[Can We Have “Detection as Code”? — Anton Chuvakin](#)

[Automating Detection-as-Code — John Tuckner](#)

[Detection-as-code: Why it works and where to start — Kyle Bailey](#)

[Detection as Code: Detection Development Using CI/CD — Patrick Bareiß, Jose Hernandez](#)

Can We Have “Detection as Code”?



Anton Chuvakin · Follow

Published in Anton on Security · 5 min read · Sep 21, 2020



268



4



Automating Detection-as-Code

Last updated on September 28, 2022



Written by John Tuckner

Head of Research at Tines Labs, Tines

Detection-as-Code

Why it works and where to start.

Kyle Bailey (@kylebailey22)

Security Engineer @ Panther Labs

Core Technologies to Manage Detection Content “as code”

Version Control System (VCS)

Software that tracks changes to code over time

Facilitates structured development processes & rollbacks

Examples: Git, Subversion, Mercurial

Software Development Platform

Provides a centralized workspace for managing Git repositories

Provides issue tracking, pull requests, code reviews, etc

Examples: GitHub, GitLab, Bitbucket

Continuous Integration / Continuous Deployment Tool

CI/CD tools automate the building, testing, and deployment of code changes

Jobs are executed when the codebase for our detection content changes

Examples: Jenkins, CircleCI, GitLab CI/CD, GitHub Actions

Implementing Detection-as-Code

Preparation

- Explore security tool API
- Export detections from tools (crawl, walk, run)
- Detection Config files(format & schema)
- Secrets management
- Configure repository & CI/CD pipeline jobs



Simple Pipeline Design

▼ DETECTION-ENGINEERING

▼ apis

> edr

> siem

🔗 __init__.py

> rule_cli

▼ rules

≡ github_enterprise_audit_log_stream_destro...

≡ okta_new_api_token_created.yaral

≡ okta_suspicious_use_of_a_session_cookie...

Detection content and “orchestration” code stored in software development platform (e.g. GitHub)

CI/CD Pipeline Jobs

Run Tests

Get rules

Update rules

SIEM

EDR

Email
Gateway

DLP

Read, create, update, and
verify rules via APIs

Example Detection-as-Code Workflow

Propose Changes

Detection Engineer creates a new pull request in GitHub with their proposed rule changes

Example changes include creating a new rule or updating an existing rule

Run Tests

CI/CD pipeline job runs tests

Check for invalid rule configuration, duplicate rule names, verify rule syntax, etc

Execute tests to trigger rules and validate alert generation

Review & Approve

Security team discusses and collaborates on proposed changes in pull request

Changes are approved by one or more members of the security team

Deploy Changes

Changes are merged into the main branch of the GitHub repo

CI/CD pipeline detects changes to the main branch and pushes any pending updates to security tools

Optional: Pull the latest version of all rules from the security tool and commit them to the main branch



Implementation Considerations 🤔

- What is your authoritative source for your detection content?
- Prevent rules being modified in the UI of your security tools
 - Ensures collaboration, review, & approval happens for proposed changes
 - Maintains version history & audit trail
- Should your CI/CD pipeline overwrite any changes made in the UI?
- Monitor for rule modifications happening in the UI?



Testing Your Rules: Don't Skip This Step!

- Validates that your detection & alerting capabilities are working
- Common issues impacting detections:
 - Logging interruptions or parsing errors
 - Misconfigured data sources
 - Vendors changing their logging schema
- Testing helps you identify & fix issues before malicious activity goes unnoticed
 - Run tests in your pipeline when rule changes are being worked on (and on a regular basis)

18%

of all SIEM rules are broken and will never fire due to common issues such as misconfigured data sources and missing fields.

[State of SIEM Detection Risk \(2024\)](#) – CardinalOps

Options for Testing Rules

- Unit tests in CI/CD pipeline – Prevent/reduce risk of deploying breaking changes
 - Validate rules against schema
 - Verify rule syntax
 - Check for invalid config, duplicate rule names, etc
- End-to-end testing – Simulate attacker behavior
 - Developing complex tests is expensive
 - Sometimes impossible or impractical to implement
- Replay test events to your SIEM and validate alert generation



Maturing SecOps with Detection-as-Code

Maturing with a Detection Library

- One location for all detections
 - Mass edits
 - Queryable
- Detection/Logging gaps?
- Easier to Share with other teams
- Easier to collaborate
- Seeing the life cycle of a detection



Better Formatting

- More Characters
- Resources
- Mitre tagging
- Extra Fields

```
title: Compiled HTML
id: 8509ac58-51fd-4696-b598-ddc6643eb401
status: Active
description: This detection looks for the execution of compiled HTML files (CHM)
using hh.exe. A custom CHM file containing embedded payloads could be delivered
to a victim and then triggered by User Execution. CHM execution is a common
technique used by threat actors to bypass application whitelisting solutions.
references:
- https://attack.mitre.org/techniques/T1218/001/
- https://detection.fyi/sigma/sigma/windows/process_creation/
  proc_creation_win_hh_chm_execution/
- https://atomicredteam.io/defense-evasion/T1218.001/
author: Wade Wells
date: 2023/09/02
modified: 2023/09/08
Tested: 2024/08/23
tags:
- attack.t1218.001
- attack.t1218
- attack.execution
- attack.defense_evasion
logsource: EDR
detection: process_name = "hh.exe" AND command_line contains '.chm'
Falsepositives: none
level: Low
```


Possible Persistence Regmod - Urlsearchhooks Last updated: 11:39:53am, Mar 15, 2019 4

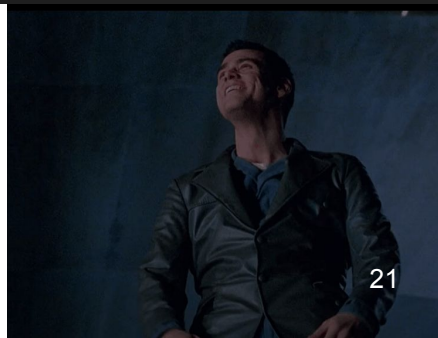
This query will identify processes that modify registry keys that may be used by attackers for malware persistence. Threat: In order to gain persistence on a target, attackers will often modify registry keys that will allow their malware to start up again across reboots. False Positives: Some applications may legitimately modify this registry key during installation. Score: 40

registry persistence attackframework attack windows evasion t1112

1 IOC

Take Action ▼

IOC	ACTIONS
<input type="checkbox"/> ((regmod_name:"software\\microsoft\\internet explorer\\urlsearchhooks")) -legacy:true	



Palantir Alerting & Detection Strategies Framework

Pros:

- Even easier to read
- Knowledge transfer
- Helps in Development
- Written in Markdown

Cons:

- Writing... a lot

Outline.md > # Additional Resources	Outline
1 # [Outline]()	
2	
3 # Goal	Goal
4	
5 # Categorization	Categorization
6	
7	
8 # Strategy Abstract	Strategy Abstract
9	
10 # Technical Context	Technical Context
11	
12	
13 # Blind Spots and Assumptions	Blind Spots and Assumptions
14	
15	
16 # False Positives	False Positives
17	
18 # Validation	Validation
19	
20 # Priority	Priority
21	
22	
23 # Response	Response
24	
25 # Additional Resources	Additional Resources
26	
27	
28 ✨	
29	



ADS Compare

title: Compiled HTML
id: 8509ac58-51fd-4696-b598-ddc6643eb401
status: Active
description: This detection looks for the execution of compiled HTML files (CHM) using hh.exe. A custom CHM file containing embedded payloads could be delivered to a victim and then triggered by User Execution. CHM execution is a common technique used by threat actors to bypass application whitelisting solutions.
references:

- <https://attack.mitre.org/techniques/T1218/001/>
- https://detection.fyi/sigma/q/sigma/windows/process_creation/proc_creation_win_hh_chm_execution/
- <https://atomicredteam.io/defense-evasion/T1218.001/>

author: Wade Wells

date: 2023/09/02

modified: 2023/09/08

Tested: 2024/08/23

tags:

- attack.t1218.001
- attack.t1218
- attack.execution
- attack.defense_evasion

logsource: EDR

detection: process_name = "hh.exe" AND command_line contains '.chm'

Falsepositives: none

level: Low

Compiled HTML

Goal

To detect the malicious execution of hh.exe launching an html file(.chm).

Categorization

These attempts are categorized as [Defense Evasion/System Binary Proxy Execution](#)

Strategy Abstract

Adversaries may abuse Compiled HTML files (.chm) to conceal malicious code. CHM files are commonly distributed as part of the Microsoft HTML Help system. CHM files are compressed compilations of various content such as HTML documents, images, and scripting/web related programming languages such as VBA, JScript, Java, and ActiveX. CHM content is displayed using underlying components of the Internet Explorer browser loaded by the HTML Help executable program (hh.exe).

Technical Context

The HTML Help executable program (Hh.exe) runs whenever a user clicks a compiled help (.chm) file or menu item that opens the help file inside the Help Viewer. The HTML Help executable program calls the HTML Help ActiveX control, which opens the help file and provides navigation and other features to the user.

Adversaries may abuse Compiled HTML files (.chm) to conceal malicious code. CHM files are commonly distributed as part of the Microsoft HTML Help system. CHM files are compressed compilations of various content such as HTML documents, images, and scripting/web related programming languages such as VBA, JScript, Java, and ActiveX. (Citation: Microsoft HTML Help May 2018) CHM content is displayed using underlying components of the Internet Explorer browser (Citation: Microsoft HTML Help ActiveX) loaded by the HTML Help executable program (hh.exe). (Citation: Microsoft HTML Help Executable Program)

A custom CHM file containing embedded payloads could be delivered to a victim then triggered by User Execution. CHM execution may also bypass application control on older and/or unpatched systems that do not account for execution of binaries through hh.exe. (Citation: MsitPros CHM Aug 2017) (Citation: Microsoft CVE-2017-8625 Aug 2017)

Locations

- C:\Windows\System32\hh.exe
- C:\Windows\SysWOW64\hh.exe

Examples of how hh.exe can be used include:


hh.exe C:\path\to\helpfile.chm
hh.exe https://example.com/helpfile.chm

hh.exe T1218.001.chm




Synergy with LLMs

CoPilot in VsCode

 wadingthrulogs

can you get several references for this detection


 GitHub Copilot

> Used 1 reference

Certainly! Here are several references that provide more information about the use of `hh.exe` and `.chm` files in both legitimate and malicious contexts:

- Microsoft Documentation on HTML Help:**
 - [HTML Help Overview](#)
 - This documentation provides an overview of HTML Help, including the use of `hh.exe` to display `.chm` files.
- MITRE ATT&CK Framework:**
 - [T1218.001: Signed Binary Proxy Execution: Compiled HTML File](#)
 - This technique describes how adversaries may use `hh.exe` to proxy execution of malicious code through `.chm` files.
- SANS Internet Storm Center:**
 - [Malicious Use of CHM Files](#)
 - This article discusses how `.chm` files can be used maliciously to deliver payloads and execute scripts.
- FireEye Blog:**
 - [CHM Abuse: A New Attack Vector](#)
 - This blog post details how attackers have been using `.chm` files to deliver malware and provides examples of such attacks.
- Trend Micro Research:**
 - [CHM Files Used in Malware Campaigns](#)

ChatGPT





ADS Framework bot

By Wade Wells 🤖

This bot helps you write Alerting and Detection Strategies Framework documents for detections. Give it detection logic or description of the detection and it will create a fully filled out ADS document

<https://github.com/palantir/alerting-detection-strategy-framework>

Write me and ADS for this detection logic "process ==..."

 Message ADS Framework bot 



ADS Framework bot

By Wade Wells 🔗

This bot helps you write Alerting and Detection Strategies Framework documents for detections. Give it detection logic or description of the detection and it will create a fully filled out ADS document
<https://github.com/palantir/alerting-detection-strategy-framework>

Write me and ADS
for this detection
logic "process == ...

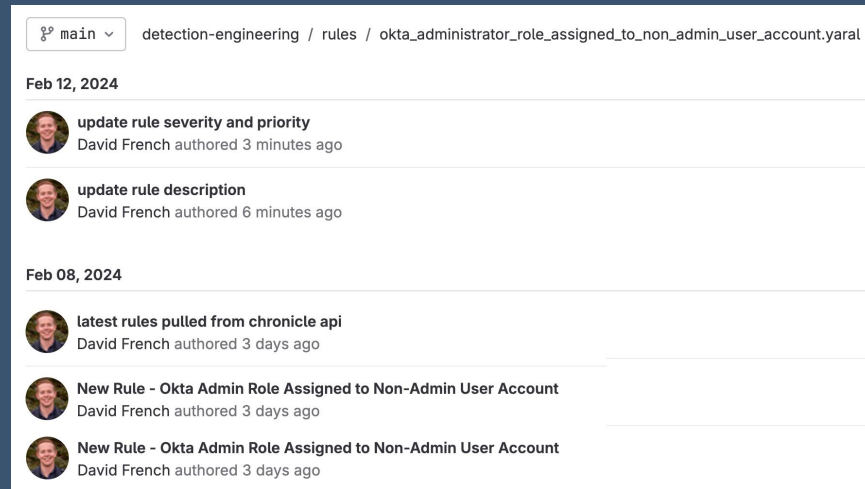


Message ADS Framework bot



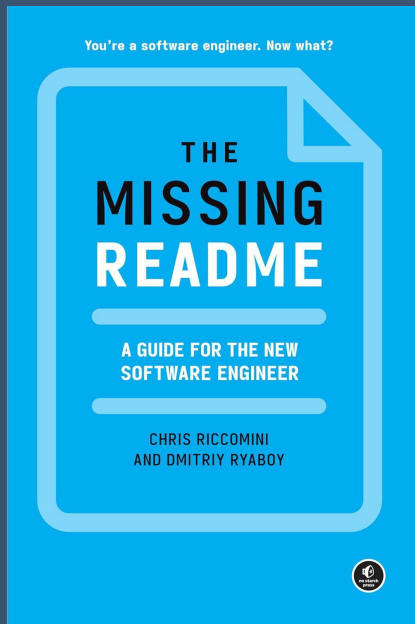
Version Control

- Commit history in VCS makes it easy to review prior versions of a rule
- Context around changes is preserved in pull requests
- Can revert to a previous version if needed
- Show auditors changes over time



Lessons Learned: Code Reviews

- Your rule may be criticized (its logic or the basis for the rule)
- Common for conflict to occur at this stage
- Authors: Assume positive intent - try to avoid getting defensive
- Reviewers:
 - Provide constructive feedback and make suggestions
 - Review in a timely manner
- Build a culture of trust and knowledge sharing
- Develop a rule style guide



“Don’t be the reason improvements
wither on the vine”

Final Thoughts

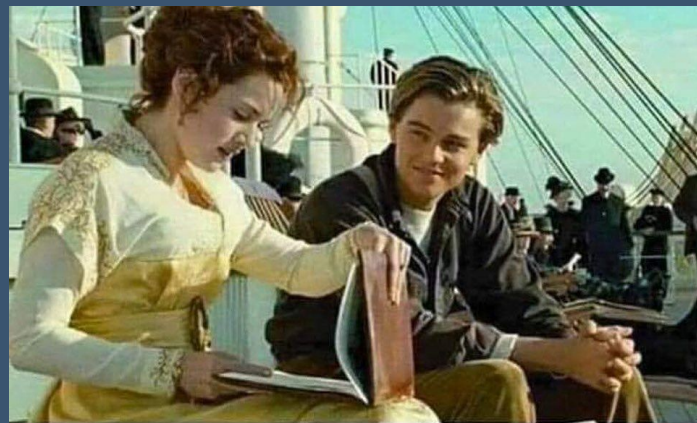
Hard Things About DaC

- The Naming
- Explaining what DaC is
- Why do it?
- Not going overboard with data
- Access to build
- Building in auto testing of rule



What Does the Future Hold for DaC?

- Workflows built directly into security tools
 - Detection content stored in software development platform like GitHub (e.g. [Sublime](#))
 - Peer review & approval
 - Version control (e.g. [Google SecOps](#))
- Pipelines augmented with AI for development, testing, and tuning of detections (e.g. [DIANA](#))
- Detection engineering continuing to adopt DevSecOps practices & tools



MY DETECTION RULES

SESSION COOKIE THEFT: ← ← → → ← + (A)

DATA COLLECTED FROM
CODE REPOSITORIES: → → ↓ ↑ + (B)

TWO-FACTOR AUTHENTICATION
DISABLED: ↑ ↑ ← → + (A)

Key Takeaways – Advantages of Adopting DaC

- Fosters collaboration to build the best detections possible
- Centralized knowledge repo provides clear insights into detection content
- Ensures changes are tested, reviewed, & approved before they're deployed
- Provides consistency regardless of rule author
- Version control



Useful Resources

- DaC deep dives, walkthroughs, and example code for implementation: [1](#), [2](#), [3](#) – David French
- [Can We Have “Detection as Code”?](#) – Anton Chuvakin
- [Automating Detection-as-Code](#) – John Tuckner
- [Detection-as-Code: Why it works and where to start](#) – Kyle Bailey
- [Detection as Code](#) – Patrick Bareiß, Jose Hernandez
- Free rules: [Google](#), [Elastic](#), [Sigma](#), [Splunk](#)

AND MORE AT THE BIT.LY 🙌

<https://bit.ly/DaC>



Thank You

Q&A