From soup to nuts: Building a Detection-as-Code pipeline

David French
Staff Adoption Engineer, Google Cloud
@threatpunter

Amsterdam 2024 FIRST Technical Colloquium
David French / About Me

- 18+ years in IT and cybersecurity
  - Blue team life: Detection Engineer, Threat Hunter, SOC Analyst
  - Vendor life: Threat Research, Detection Engineering, building SIEMs & EDRs
- Currently at Google Cloud (Chronicle Security Operations)
- Formerly Twilio, Elastic, Endgame, Capital Group
- Speaker at Black Hat and BSides
- Creator of [Dorothy](#) - Adversary simulation tool for Okta
- Likes to share knowledge & research: [Blog](#), [community contributions](#), MITRE ATT&CK
- Enjoys hiking, fishing, cycling, etc
Intended audience

- Anyone curious about how to manage detection content “as code” and how to get started
- Defensive security practitioners: Detection Engineers, SOC Analysts, etc
- Maybe you manage rules/signatures manually in your security tools and want to automate that
- If you’re already an expert in Detection-as-Code, you might not learn a ton 😊
Agenda

1. What is Detection-as-Code?
2. Example Detection Engineering workflow with Detection-as-Code
3. Benefits of managing detection rules “as code”
4. Designing the pipeline
5. Building a pipeline to manage detection content
6. Wrap up
   a. Key takeaways
   b. Links to useful resources
   c. Q&A
What is Detection-as-Code (DaC)?

- A set of principles that use code and automation to implement and manage threat detection content
- Traditional approach: Security team manually configures rules & signatures in security tools
- Detection-as-Code: Leverages software development practices & tools and treats detection content as code artifacts
- Gaining in popularity; growing acceptance
Core technologies to automate detection content management

<table>
<thead>
<tr>
<th>Version Control System (VCS)</th>
<th>Software Development Platform</th>
<th>Continuous Integration / Continuous Delivery Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software that tracks changes to code over time</td>
<td>Provides a centralized workspace for managing Git repositories</td>
<td>CI/CD tools automate the building, testing, and deployment of code changes</td>
</tr>
<tr>
<td>Facilitates structured development processes rollbacks</td>
<td>Provides issue tracking, pull requests, code reviews, etc</td>
<td>Examples: Jenkins, CircleCI, GitLab CI/CD, GitHub Actions</td>
</tr>
<tr>
<td>Examples: Git, Subversion, Mercurial</td>
<td>Examples: GitHub, GitLab, Bitbucket</td>
<td></td>
</tr>
</tbody>
</table>

6
Example Detection-as-Code workflow

**Propose Changes**
Detection Engineer creates a new pull request in GitLab with their proposed rule changes.
Example changes include creating a new rule or updating an existing rule.

**Run Tests**
GitLab 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 GitLab project.
A CI/CD pipeline detects changes to the main branch and pushes any pending updates to the SIEM.
The latest version of all rules is pulled from the SIEM and committed to the repo to include updated metadata.
Benefits of managing detection rules as code
Benefits of DaC: Collaboration (1)

- Challenge with traditional method of managing detection rules: People make mistakes
- DaC makes it easy for the team to discuss and contribute to changes to detection content
- A group of practitioners with unique insights working together will result in more accurate and effective rules
- Peer review reduces risk
  - False negatives
  - False positive explosions
Benefits of DaC: Collaboration (2)

- Easier to share detection content with the security community; stronger defense against attacks
  - Google: [https://github.com/chronicle/detection-rules](https://github.com/chronicle/detection-rules)
  - Elastic: [https://github.com/elastic/detection-rules](https://github.com/elastic/detection-rules)
  - Splunk: [https://github.com/splunk/security_content](https://github.com/splunk/security_content)
  - Microsoft: [https://github.com/Azure/Azure-Sentinel](https://github.com/Azure/Azure-Sentinel)
  - Sigma: [https://github.com/SigmaHQ/sigma](https://github.com/SigmaHQ/sigma)
Benefits of DaC: Change management

- DaC provides more control over changes made to detection content
- Detection content stored in a software development platform e.g. GitHub, GitLab
- Changes are tested, reviewed, and approved before getting deployed to prod
- Some organizations require robust change control for both preventive and detective security controls
Benefits of DaC: Automation

- CI/CD tools used to ensure continuous process for building, testing, and deploying changes to detection content
- Tests reduce risk of introducing false positives/negatives
  - Reduce problem of alert fatigue
- Test in dev before deploying to prod
Designing & building the pipeline
Pipeline design

Software development platform and version control system (VCS)

GitLab CI/CD Pipeline Jobs
- Run Tests
- Get rules
- Update rules

Write code to read, create, update, and verify rules via the SIEM’s API
At this point, we’re assuming:

- We have some rules configured in our SIEM
- Our SIEM has an API endpoint for managing rules

SIEM vendors may provide example code or engineers may have to write it themselves.

Users expect parity between what they can do in the UI of a security tool versus the API.

Managing detection rules via an API (2)

- Python modules are wrapped in a simple CLI to use in CI/CD pipeline jobs in GitHub, GitLab, etc
- Additional modules & logic written to handle logic for updating rules
Some teams use Infrastructure-as-Code tools to manage SIEM rules & configuration
  ○ e.g. Terraform, Pulumi

Code is stored in central repository and CI/CD jobs “apply” changes to “infrastructure” (security tools)

These tools can overwrite changes made in the UI if that’s your desired behavior

Example code for managing rules in Sumo Logic using Terraform: https://github.com/threat-punter/detection-as-code-example
GitLab project layout

Python modules for managing rules via SIEM’s API

SIEM rules stored as code artifacts

GitLab CI/CD pipeline configuration file

CLI with commands to retrieve, update, and verify rules via SIEM’s API

Rule configuration file

Used to configure rule state (e.g. enabled/disabled/archived)

Used to store rule metadata (e.g. rule ID, create time, etc)

Example code: https://github.com/chronicle/detection-rules/tree/main/tools/rule_manager
Defining a rule schema: Benefits

- Provides a way to structure and standardize rules
- Ensures rule structure is consistent across authors
- Define which parts are required/optional
- Automation - Easier to validate, test, and deploy detection content if it’s in a consistent format
- Easier to share rules within the community
- Example of a schema using Pydantic

```python
from pydantic import BaseModel

class Rule(BaseModel):
    """Data class for a YARA-L rule."""
    name: str
    id: str | None
    resource_name: str | None
    create_time: str | None
    revision_id: str | None
    revision_create_time: str | None
    enabled: bool
    alerting: bool
    archived: bool | None
    archive_time: str | None
    run_frequency: str | None
    type: str | None
text: str
```
Defining a rule schema: Popular formats

- YAML - Used by [Splunk](https://www.splunk.com) and [Sigma](https://github.com/splunk/sigmacore)
- TOML - Used by [Elastic](https://elasticsearch.org)
- Example of a YARA-L rule in TOML format ➡
- I decided to decouple the rule config & metadata from the rule logic
  - Granular control over deploying to multiple SIEM instances (e.g. if you’re deploying to dev, prod, etc or an MSSP deploying to multiple customers)
Validating rules against a schema

- Catch issues as early as possible; minimize risk of deploying broken rules
  - Missing/invalid values
  - Misconfigurations e.g. a rule that’s enabled cannot be archived until it’s disabled
  - Invalid rule/file names
- **Pydantic** and **Marshmallow** are great for this

```python
ValidationError occurred for rule
```

```json
{
  "type": "bool_type",
  "loc": [
    "enabled"
  ],
  "msg": "Input should be a valid boolean",
  "input": null,
  "url": "https://errors.pydantic.dev/2.6/v/bool_type"
}
```
Verifying rule syntax

- Options to verify the syntax of a rule:
  - Via your SIEM's API if supported
  - Develop your own linter for rule parsing & validation (++ effort to create and maintain)
- Some SIEMs prevent a rule from being created/modified if syntax errors are found
Pulling the latest rules from the SIEM

- We need to keep the GitLab project up-to-date with the latest version of all rules in the SIEM
- CLI argument pulls latest rules from SIEM and writes rule files and rule config file

```
$ python -m rule_cli --pull-latest-rules
```

```
19-Jan-24 23:03:35 UTC | INFO | <module> | Rule CLI started
19-Jan-24 23:03:35 UTC | INFO | <module> | Attempting to pull latest version of all Chronicle rules and update local files
19-Jan-24 23:03:35 UTC | INFO | get_remote_rules | Attempting to retrieve all rules from Chronicle
19-Jan-24 23:03:36 UTC | INFO | get_remote_rules | Retrieved 3 rules
19-Jan-24 23:03:36 UTC | INFO | get_remote_rules | Retrieved a total of 3 rules
19-Jan-24 23:03:36 UTC | INFO | get_remote_rules | Attempting to retrieve rule deployment state for 3 rules
19-Jan-24 23:03:38 UTC | INFO | dump_rules | Writing 3 rule files to /builds/
dump_rule_config | Writing rule config to /builds/
g.yaml
```
Dumping the rule logic and rule configuration

- Rule logic is written to the `rules` directory
- Rule configuration and metadata is written to a `rule_config.yaml` file
Syncing rules between the SIEM and GitLab

- CI/CD pipeline job runs on a schedule
- Pulls latest rules from SIEM
- Writes files containing rules and rule config
- Commits any changes to the main branch of the GitLab project

Example commit made by CI/CD job

Reviewing rule modifications that were made in the SIEM’s UI
Creating a new rule

Detection Engineer creates a GitLab pull request to create a new SIEM rule
Protecting the main branch

- Protect the main branch of your GitLab/GitHub project
- Prevent code from being merged until tests pass and approval is obtained

![Merge blocked: 2 checks failed]

```
08-Feb-24 22:35:14 UTC | INFO | verify_rules | Rule verification succeeded for 9 rules
08-Feb-24 22:35:14 UTC | ERROR | verify_rules | Rule verification failed for 1 rules
    "compilationDiagnostics": [
        {
```

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, explain your thought process, 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”
Testing rules: Don’t skip this step!

● If you’re not testing your detection rules on a regular basis, you’re on shaky ground
● Can you say with confidence that your logging, detection, and alerting is working properly?
● Broken detections result in false negatives 😞
● Challenges & considerations
  ○ Time: It can take longer to develop a test than the rule itself!
  ○ Build vs. buy: Do we have the expertise to develop & automate tests?
  ○ Tech debt: What if you have hundreds of rules without tests? 😞
The problem with untested rules

- Environments drift
- Infrastructure and technologies come and go, software is updated
- Logging interruptions occur
- Vendors change their logging schemas
- Attack techniques no longer work (relevancy)
- Active detection rules that will never fire waste detection engine resources
Options for testing rules

- Run the rule against sample data
  - Better than having no tests at all
- Trigger the rule and validate alerts were generated
  - More comprehensive
  - Validates logging, detection, and alerting pipeline is working
  - Get started with free projects like Atomic Red Team and Red Team Automation
  - You can’t test everything (and that’s okay) e.g. anomaly detections
Triggering the rule

Run code in CI/CD pipeline job to carry out actions via Okta API and trigger detection rule

Example code: https://github.com/threat-punter/detection-as-code-example/blob/main/detections_cli/triggers/assign_admin_role_to_okta_user.py
Validating alerts

- Validate that alert was generated by detection rule
- Check for your test indicators in alerts
- Close alerts and any tickets/cases that were created
- CI/CD pipeline job success/failure

Example code: [https://github.com/threat-punter/detection-as-code-example/blob/main/detections_cli/__main__.py#L74](https://github.com/threat-punter/detection-as-code-example/blob/main/detections_cli/__main__.py#L74)
Deploying changes to the SIEM

Changes are pushed to the SIEM after code is merged into the main branch

Syncing rule metadata to GitLab

- After changes are deployed to SIEM
- Pipeline job pulls latest rules from SIEM and commits updated metadata to rule config file in GitLab project

```
63  type: MULTI_EVENT
64 + okta_administrator_role_assigned_to_non_admin_user_account:
65 +   alerting: true
66 +   archive_time: null
67 +   archived: false
68 +   create_time: '2024-02-08T23:39:19.682863Z'
69 +   enabled: true
70 +   id: ru_340b3f6d-916a-4365-b6c0-b63cd5deb265
71 +   resource_name: projects/[LOCATION_ID]/locations/us/instances/[INSTANCE_ID]
72 +   revision_create_time: '2024-02-08T23:39:19.682863Z'
73 +   revision_id: v_1707435559_682863000
74 +   run_frequency: HOURLY
75 +   type: SINGLE_EVENT
76 okta_new_api_token_created:
77 +   alerting: false
78 +   archive_time: null
... ...
```
Modifying rules

- Detection Engineer creates a branch and pull request with proposed changes
- Tests succeed
- Peer review & approval obtained
- Changes are merged to the main branch
- Rule changes are deployed to SIEM
Auditing for rule changes

- 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
Benefits of centralized detection management

- Auditors might ask for proof that you have a detection implemented (and that its tested)
  - For example, detections related to data loss prevention or SWIFT compliance
- Purple Teaming – Offensive team can analyze detections and look for ways to evade them
- Code repository is searchable
  - Can quickly check if you have a rule for an attack technique
Key takeaways
Which organizations can benefit from adopting DaC?

Yes
- Large orgs with complex, dynamic IT environment and lots of normalized security data available
- Auditing & change management needed for detective security controls
- Security budget for Detection Engineers and required engineering expertise
- Modern security tools (manage content via API)

No
- Small orgs with simple, static IT environment
- Limited security budget
- Not much security data available for analysis
- Small (or no dedicated) security team
- Security tools with no support for integration
- Partnering with an MSSP may be a good fit (they’re likely using DaC to manage rules across multiple customers)
Advantages of adopting Detection-as-Code

● Increased collaboration around rule development and sharing in the community
  ○ A group of practitioners with unique insights working together will result in more accurate and effective rules
● More control over changes to detections
● Automated testing
  ○ Reduced risk of introducing false positives/negatives
  ○ Provides confidence that your logging, detection, and alerting is working
  ○ Helps identify issues quickly before misses occur
Useful resources

- Detection-as-Code
  - My blog series and example code for getting started: [1](#) and [2](#)
  - 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
  - Detection-as-Code panel — Julie Agnes Sparks, Jackie Bow, Jessica Rozhin, Louis Barrett

- Detection Engineering
  - Detection Engineering Weekly — Zack Allen
  - Practical Threat Detection Engineering — Megan Roddie, Jason Deyalsingh, Gary J. Katz

- Free rules: Google, Elastic, Splunk
Thank you

David French
Staff Adoption Engineer, Google Cloud
@threatpunter
Q&A
Acknowledgement

Thanks to the following people for their contributions to the security community and/or providing me with valuable feedback:

Kyle Bailey, Patrick Bareiss, Adam Cole, James Black, Anton Chuvakin, Dan Dye, Serhat Gülbetekin, Jose Hernandez, Dave Herrald, Justin Ibarra, Chris Long, Dan Lussier, Vishwanath Mantha, Christopher Martin, Anton Ovrutsky, Julie Agnes Sparks, John Stoner, John Tuckner, Wade Wells, Ross Wolf