Rolling your own
Detections as Code
Ft. Elastic Security

Mika Ayenson : @stryker0x
Justin Ibarra : @br0k3ns0und

HELLO! I'M...

Mika Ayenson, Ph.D.
Senior Security Research Engineer @ Elastic

@stryker0x

HELLO! I'M...

Justin Ibarra,
Threat Research and Detection
Engineering Lead @ Elastic

@br0k3ns0und

TABLE OF CONTENTS.

01 What is DaC and why it's needed?

 Perhaps you've heard of Infrastructure as Code (IaC)?! DaC is the close relative!

Target Audience!

Security Analysts

Aide responding rapidly to emerging threats.

Detection Engineers

Streamline detection logic development, testing, and deployment.

Security Team Leads

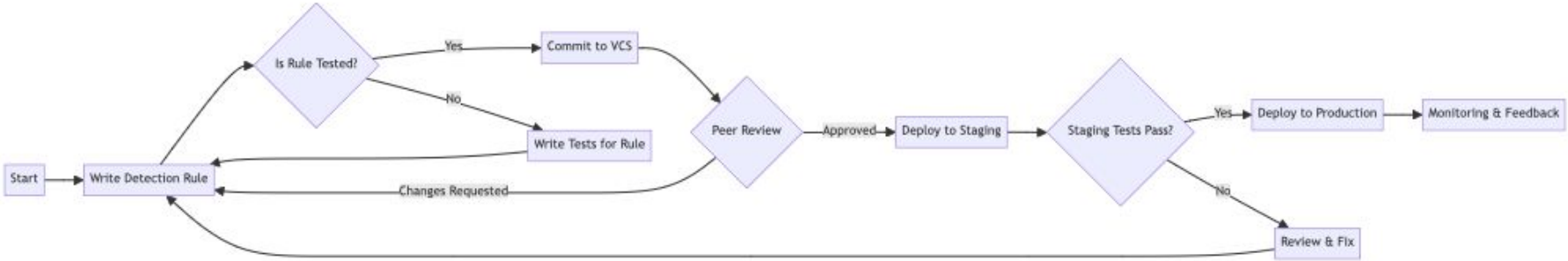
Seeking to implement best practices for rule version control, auditing, and quality assurance.

DevOps Engineers

Integrating security practices into CI/CD pipelines, aiming for a more cohesive and automated approach.

IT Security Architects

Exploring ways to incorporate as-code principles into security operations.



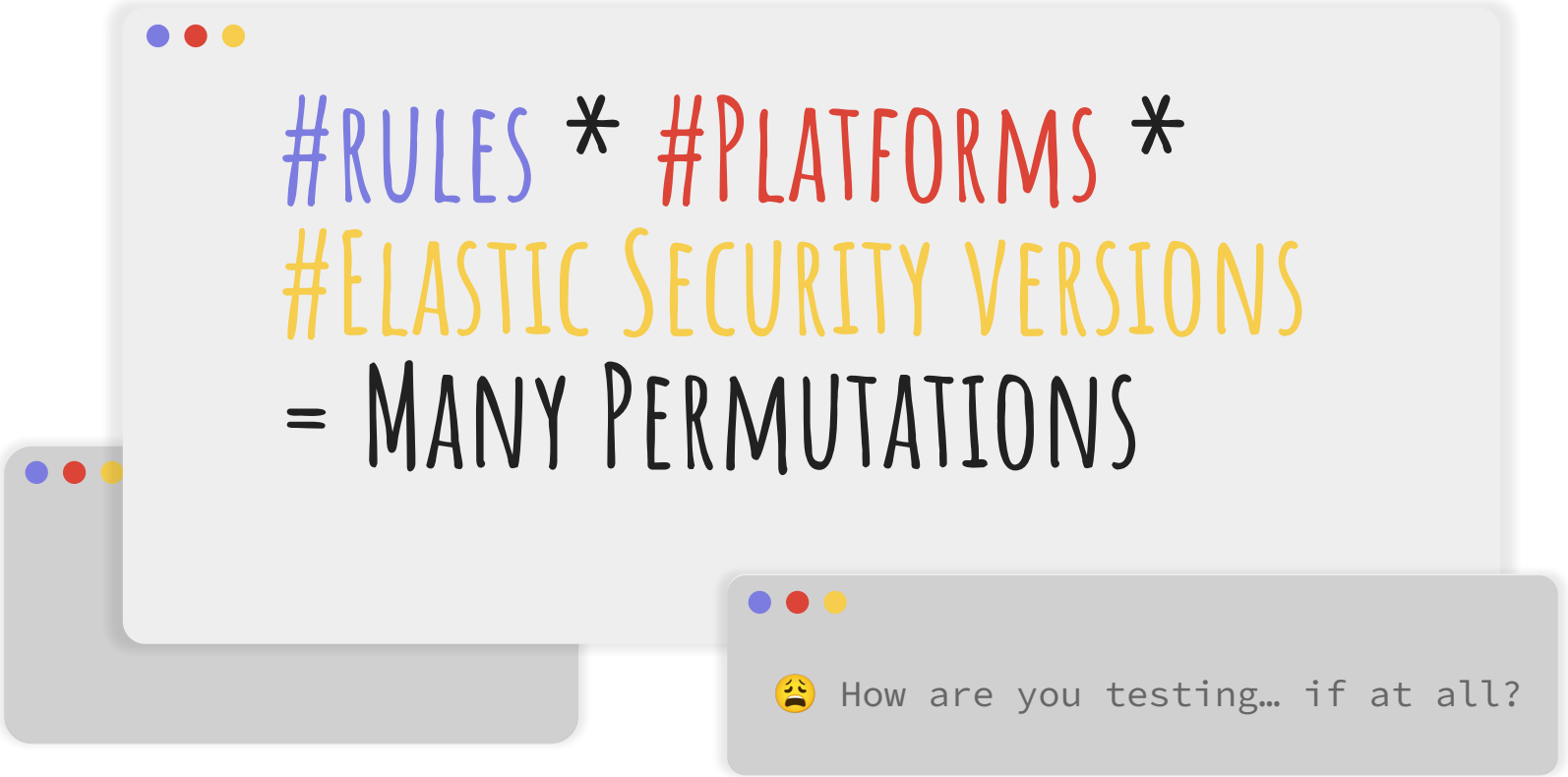
It's more than just a CI/CD workflow

 DaC: A methodology that applies software development practices to the creation and management of security detection rules, enabling automation, version control, testing, and collaboration in the development & deployment of security detections.

But WHY do we Need this?

```
<ul>
  <li>Ever-Growing Rule Sets</li>
  <li>Broader Adoption of Automation</li>
  <li>Drive Security Team Towards Maturity</li>
  <li>Expanding Threat Landscape</li>
  <li>Compliance and Governance</li>
</ul>
```

We encourage you to share your use-case!


#Rules * #Platforms *
#Elastic Security versions
= Many Permutations
😩 How are you testing… if at all?

Timeline of How the concept evolved

2014-2016

Early mentions may have been considered as codifying security detections.

2017-2019

Growth in interest evolved (e.g. RTA, ART) into automated detection logic internal workflows.

2020-2023

Test frameworks emerge and adoption as companies begin to showcase DaC.

2024-Present

Widespread adoption and advertisement of DaC how-to-guides blogs.

Future

DaC capabilities fully implemented within company security solution offerings.

02 High Level components, and concepts.

 When unpacking the essential elements, navigating through the processes, and defining the scope, we found that there is no one single option.

Multiple approaches for multiple users

User A

“As an Enterprise, I need to manage multiple air-gapped dev/prod spaces.”

User B

“As an MSSP I need to manage multiple customers’ dev/prod clusters with different rulesets.”

User C

“As a limited SMB, I need to automate as much as possible.”

Hierarchy and Lexicon of Concepts

- Core components
  - Sub-components
    - SC Options
  - CC Options
- Governance models

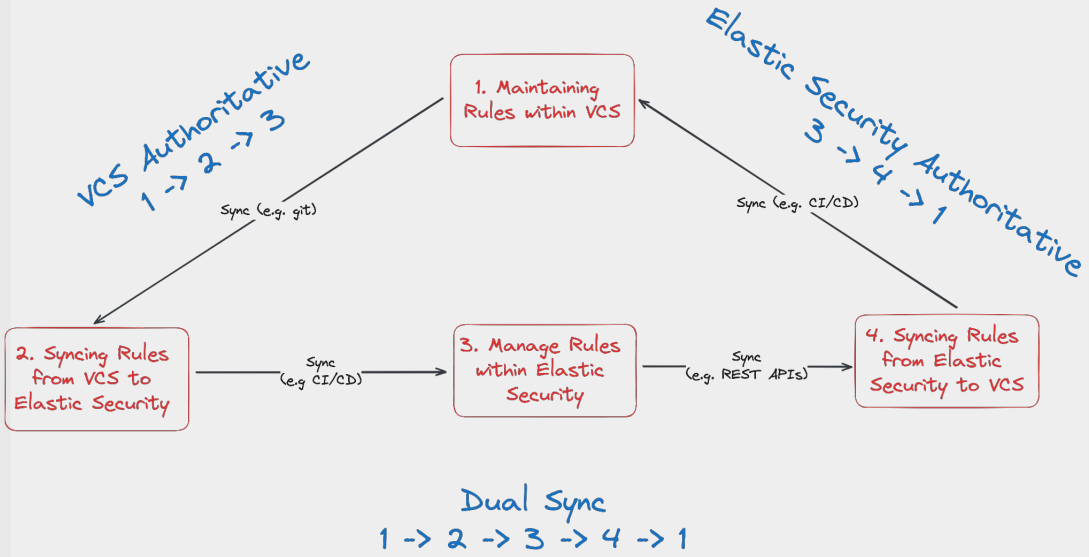
Hierarchy

What are the concepts and components and how do they relate to each other?

Lexicon

Consistent verbiage and nomenclature enables simpler collaboration and planning.

Core COMPONENTS Flow by Governance



Core components

Sub-components

Options

Core components

- Maintaining rules within a Version Control System (VCS)
- Syncing rules *from* VCS *to* their respective platform
- Managing rules *within* the platform
- Syncing rules *from* the platform *to* VCS

Hierarchy

- Core components
  - Sub-components
    - SC Options
  - CC Options
- Governance models

SUB-COMPONENTS AND OPTIONS

- Maintaining rules within VCS
  - Rule schema validation
    - Local repo dataclass
    - Remote Kibana REST API
  - Detection logic validation
    - Local EQL/KQL lib validation
    - Remote Kibana REST API
  - ...
- Syncing from VCS to platform
  - ...

Hierarchy

- Core components
  - Sub-components
    SC Options
  - CC Options
- Governance models

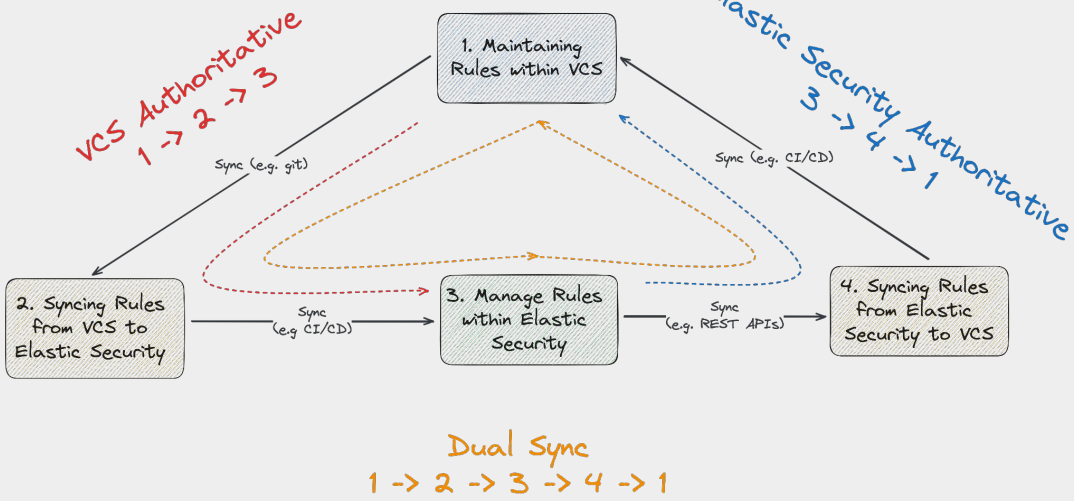
Governance Models

- VCS as authoritative
- Platform as authoritative
- Dual sync between VCS and the platform

Hierarchy

- Core components
  - Sub-components
    - SC Options
  - CC Options
- Governance models

Core COMPONENTS Flow by Governance



SYNCING OPTIONS

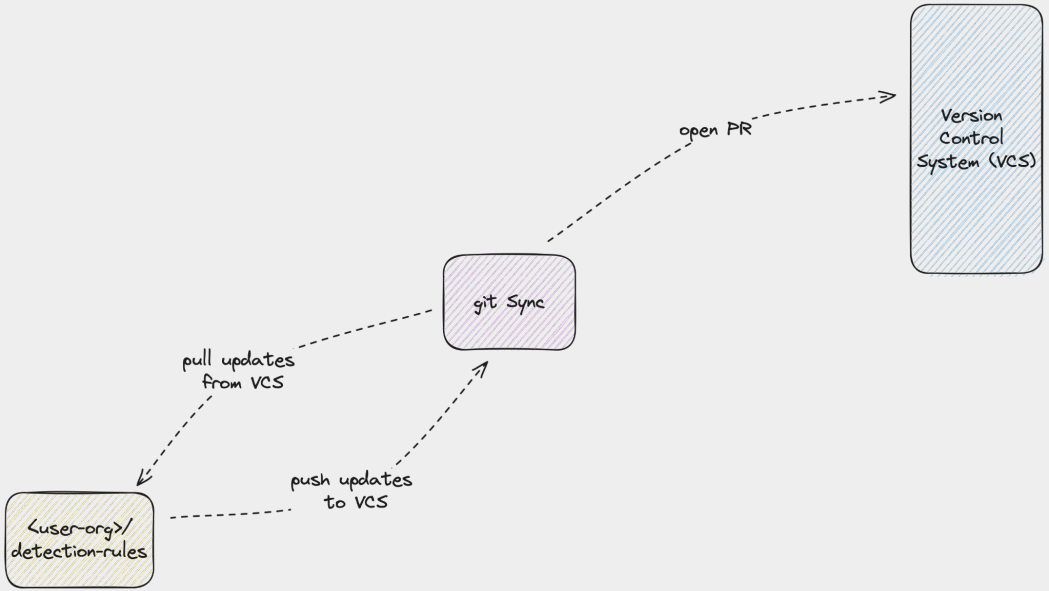
git

REST API

CI CD

RULE MANAGEMENT OPTIONS

CC: Maintaining Rules within VCS



DESCRIPTION

Covers creating and managing rules as code locally and using version control tools like git to Sync to the VCS.

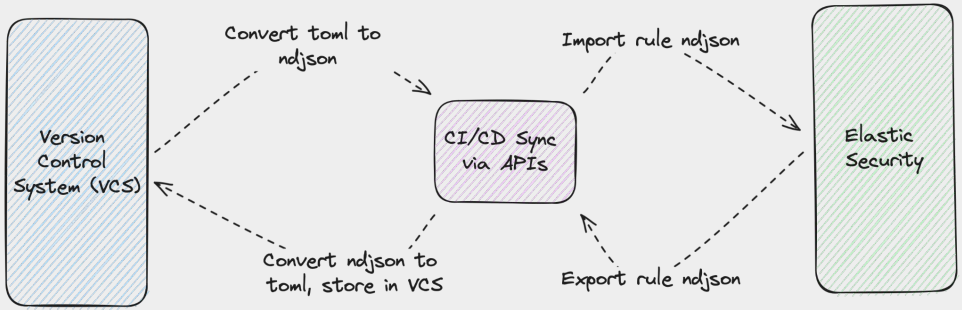
Requirements

- Dedicated repo to store detection rules and collaborate
- Local schema and query validation tools

RULE MANAGEMENT OPTIONS

- Directly create, modify, and manage rule files locally
- Manually push/pull rules to VCS for backup/version control

CC: SYNCING RULES FROM VCS TO THE PLATFORM



DESCRIPTION

Covers the automated or manual processes of deploying or updating rules in Elastic Security from VCS.

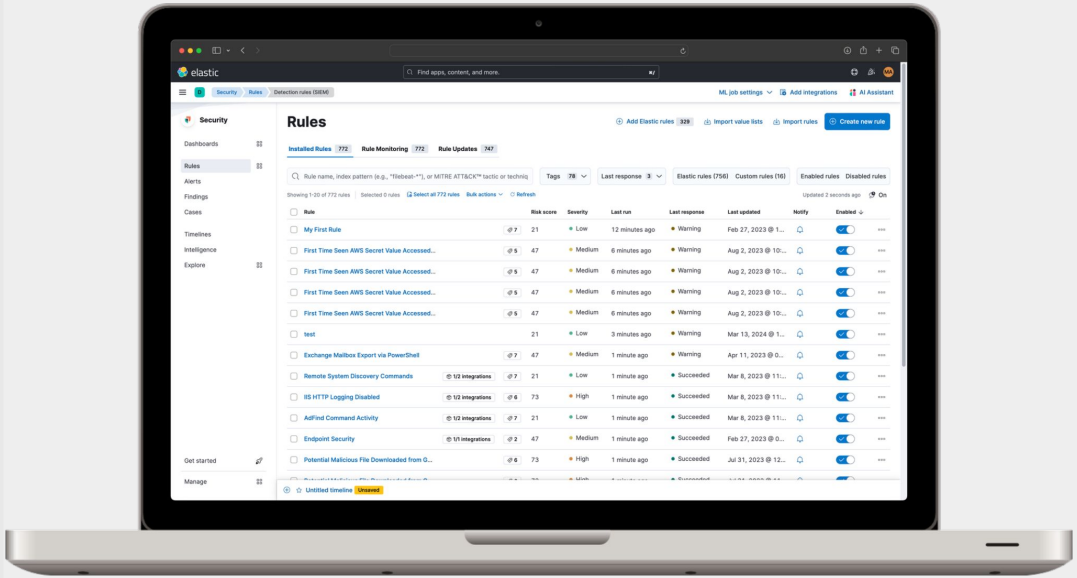
Requirements

- API access to Elastic Stack
- Authentication credentials
- CI/CD pipeline (optional)

RULE MANAGEMENT OPTIONS

- Import rules into Elastic Security using CLI or API
- Configure CI/CD for automated syncing

CC: Managing rules within Your Platform



DESCRIPTION

Focuses on creating, testing, and managing rules directly in Elastic Security, while considering backup and versioning strategies.

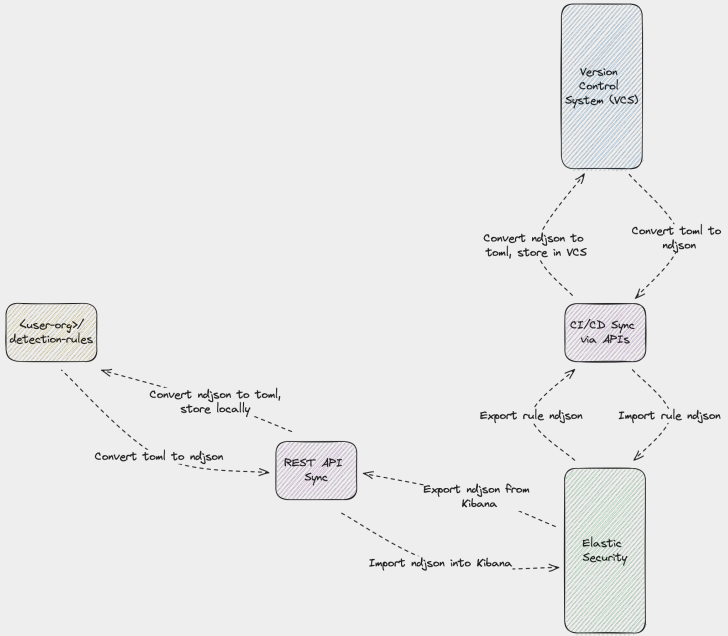
Requirements

- Elastic Security access with permissions
- Knowledge of Elastic Security's UI

RULE MANAGEMENT OPTIONS

- Directly create, modify, and manage rules in Elastic Security
- Manually export rules for backup/version control

CC: Syncing rules from Your Security Solution to VCS



DESCRIPTION

Describes exporting and versioning rules from Elastic Security back into VCS for tracking and collaboration.

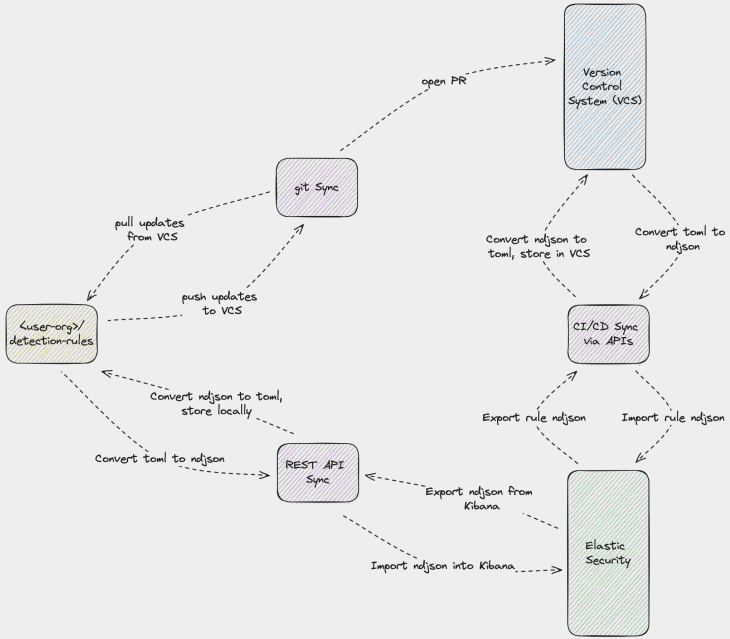
Requirements

- Scripting for API interaction
- Authentication
- CI/CD setup for automation (optional)

RULE MANAGEMENT OPTIONS

- Export rules using Detection Engine API
- Commit exported rules into VCS
- Use CI/CD workflows to automate the process

GM: Dual Sync Between VCS and Platform



DESCRIPTION

Highlights a hybrid approach that ensures rules are synchronized and up-to-date in both Elastic Security and VCS.

Requirements

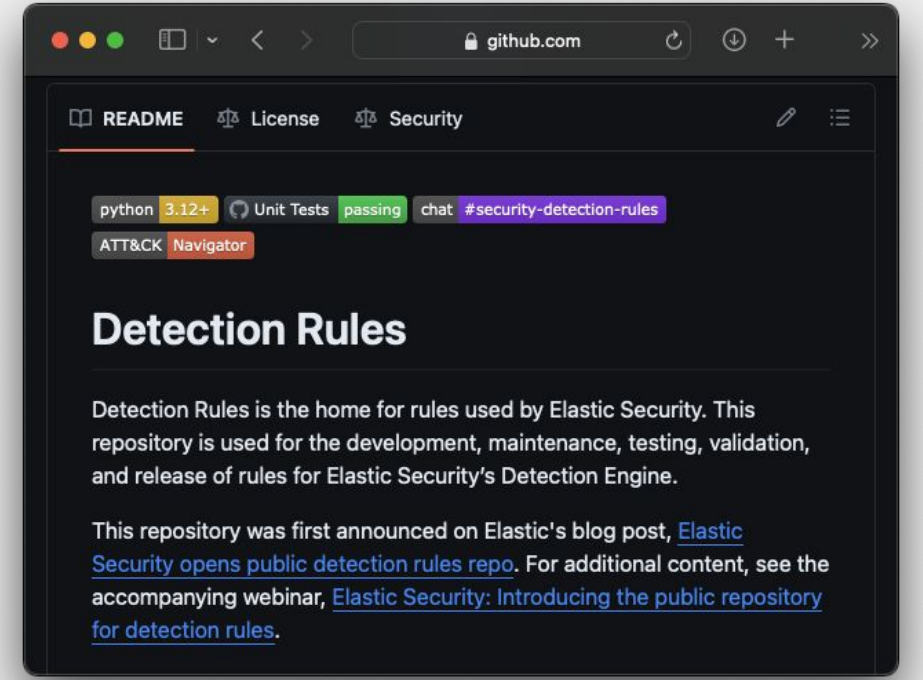
- Setup for bidirectional syncing
- Authentication
- Access
- Automation tools/scripts

RULE MANAGEMENT OPTIONS

- Establish sync process for both directions
- Automate sync using CLI, API, and CI/CD
- Regularly review and reconcile discrepancies

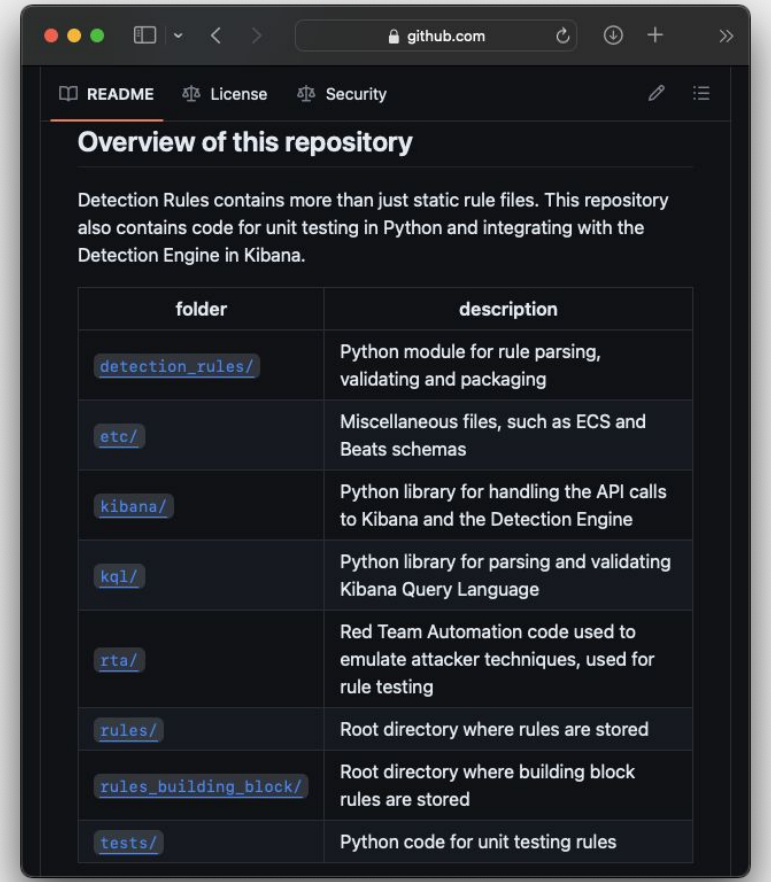
03 Bias to leverage detection-rules

 We preference proven practices and spotlight the 'detection-rules' repository as the cornerstone of effective DaC methodologies, but why?



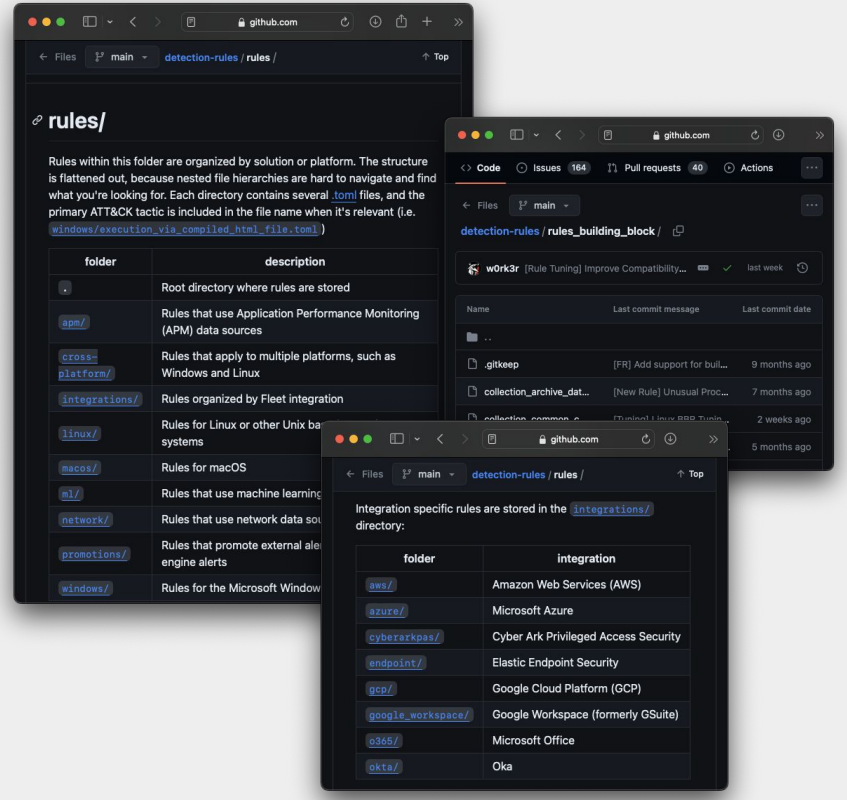
GITHUB/ELASTIC/ DETECTION-RULES

 First, here's a primer on the github.com/elatic/detection-rules repo, features, and CLI w/DaC context.



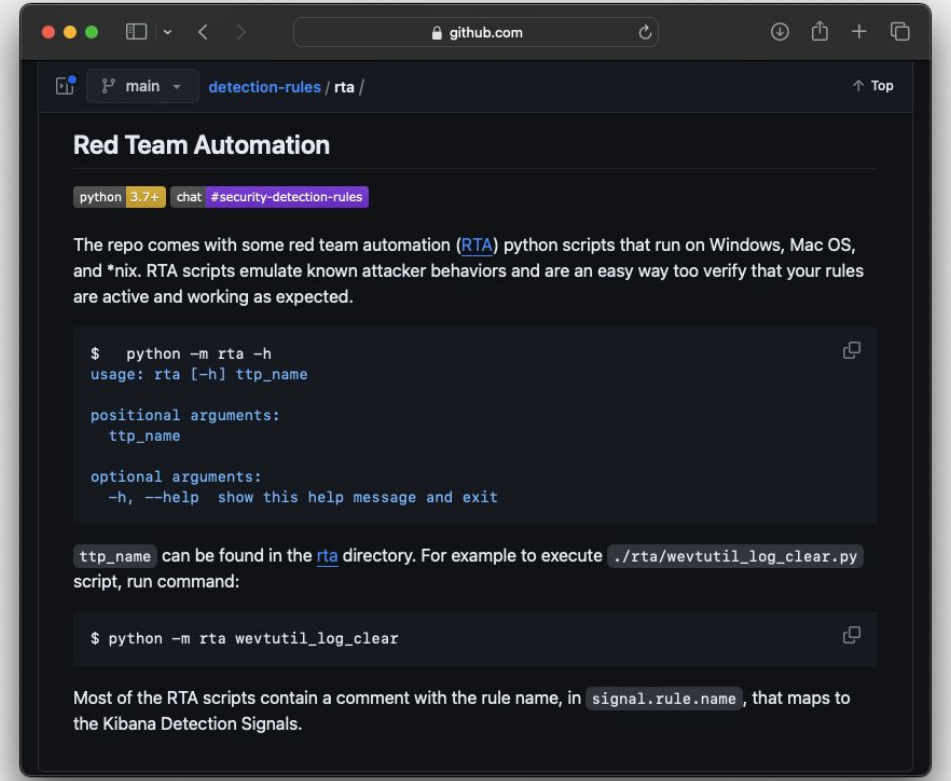
Repo Structure Rules and DAC Rule Management

 We store our rule management and testing Python logic next to our Prebuilt rules with entry points in our unit test and the CLI



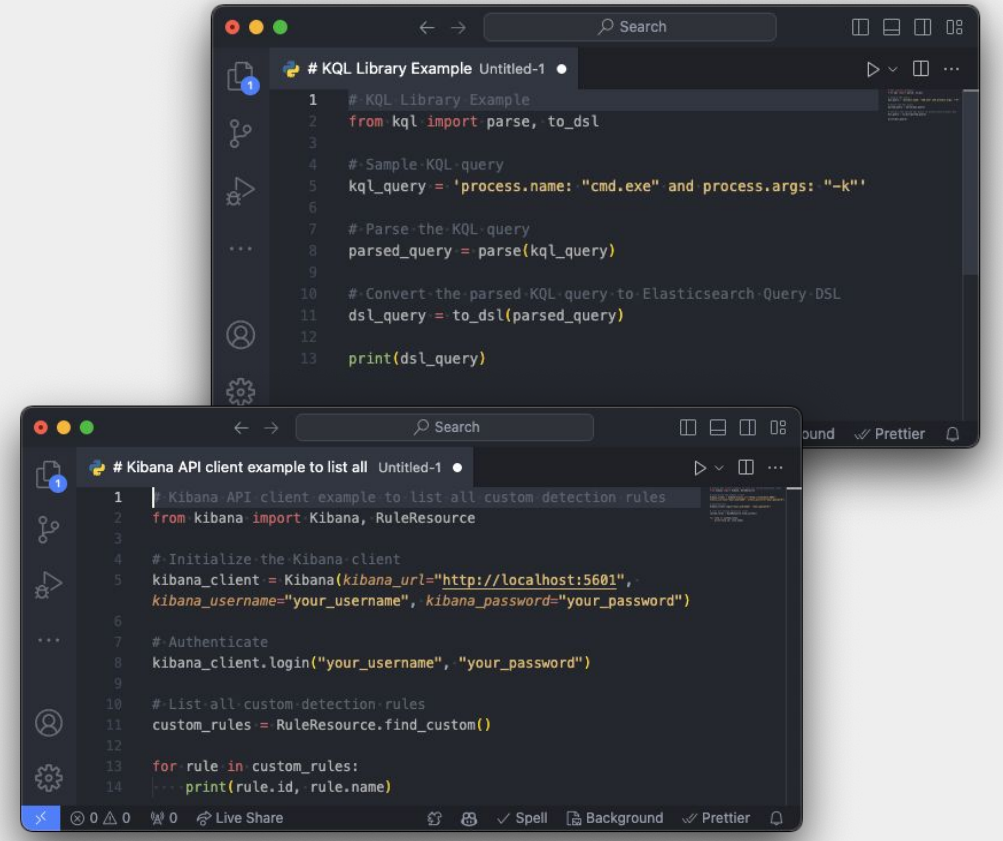
Available Behavior and Building Block Prebuilt Detections

 Our prebuilt rules contain endpoint and integration specific detections; some backed by building block rules. Users can place their existing rules in a CUSTOM_DIR, which is ingested by the rule loader.



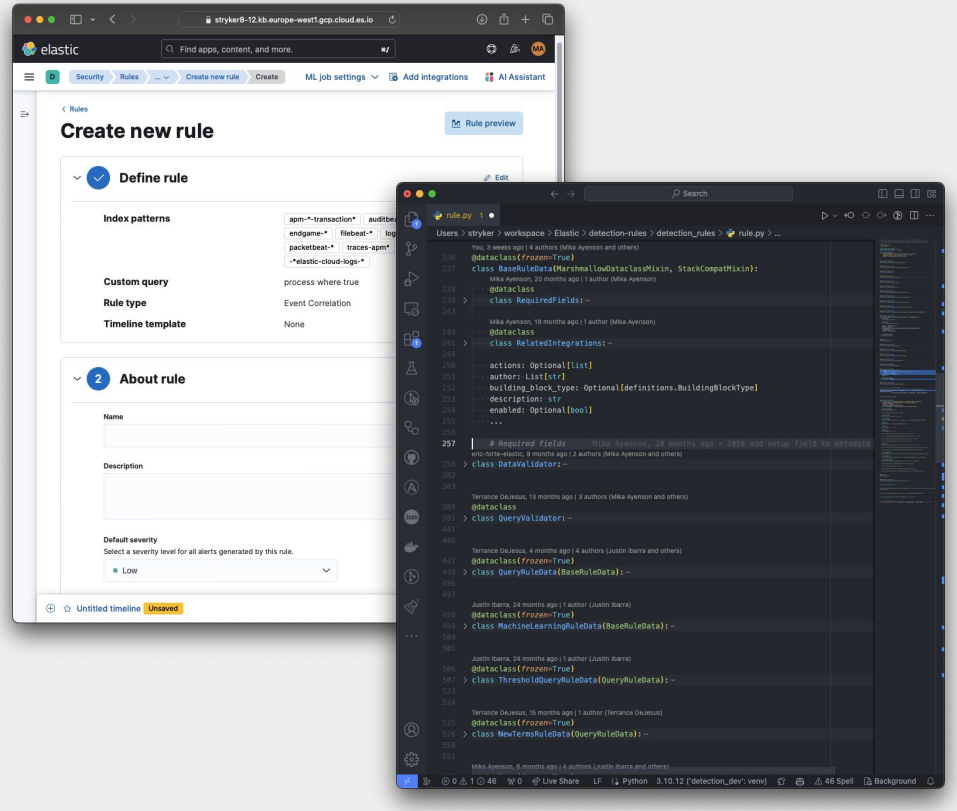
Red Team Automation

 RTAs have been around since Endgame days. We now maintain a set of endpoint RTAs within the detection-rules repo.



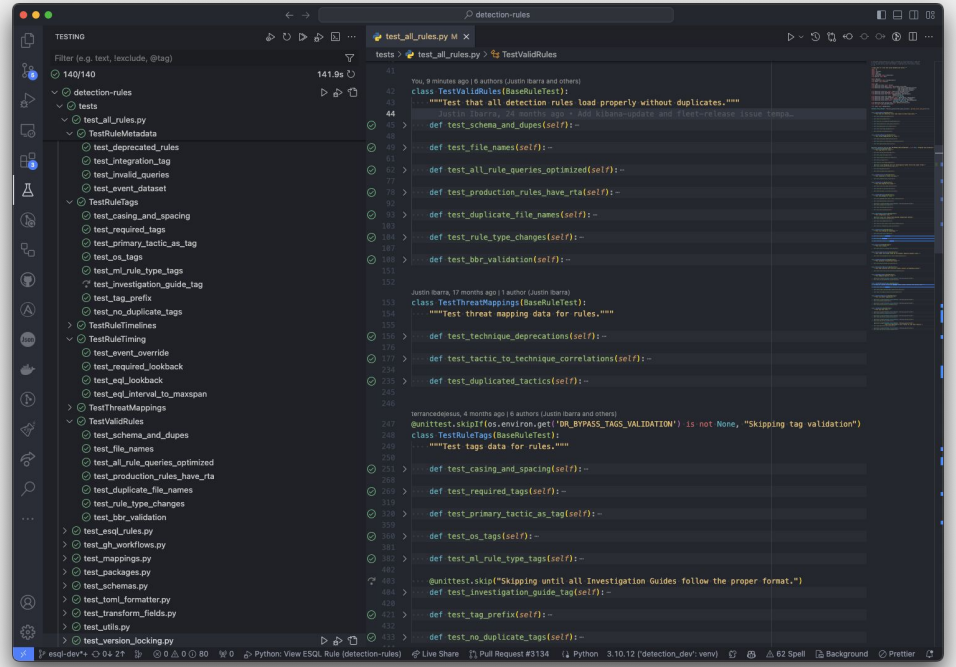
KQL and Kibana python Libraries

 We've decoupled these two Python libraries to be installed as independent third-party packages. Note: No pypi support.



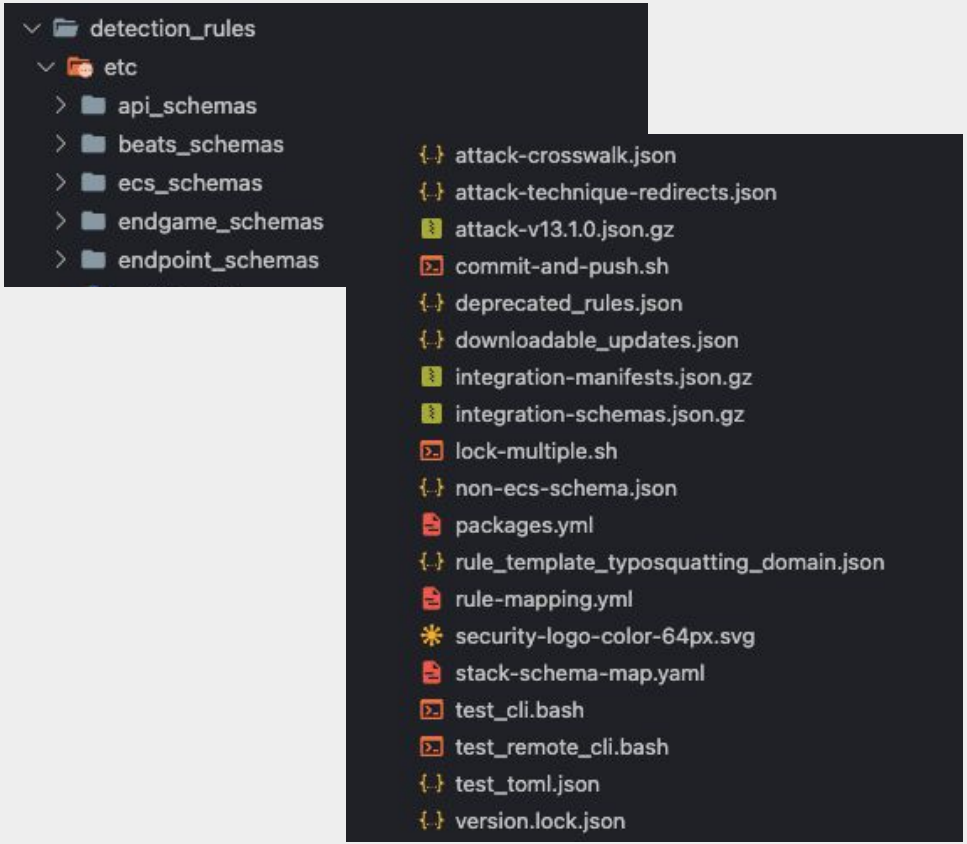
Dataclass and Marshmallow Schema Validation

 Many of the query languages and rule fields are supported to enable local stack-independent validation.



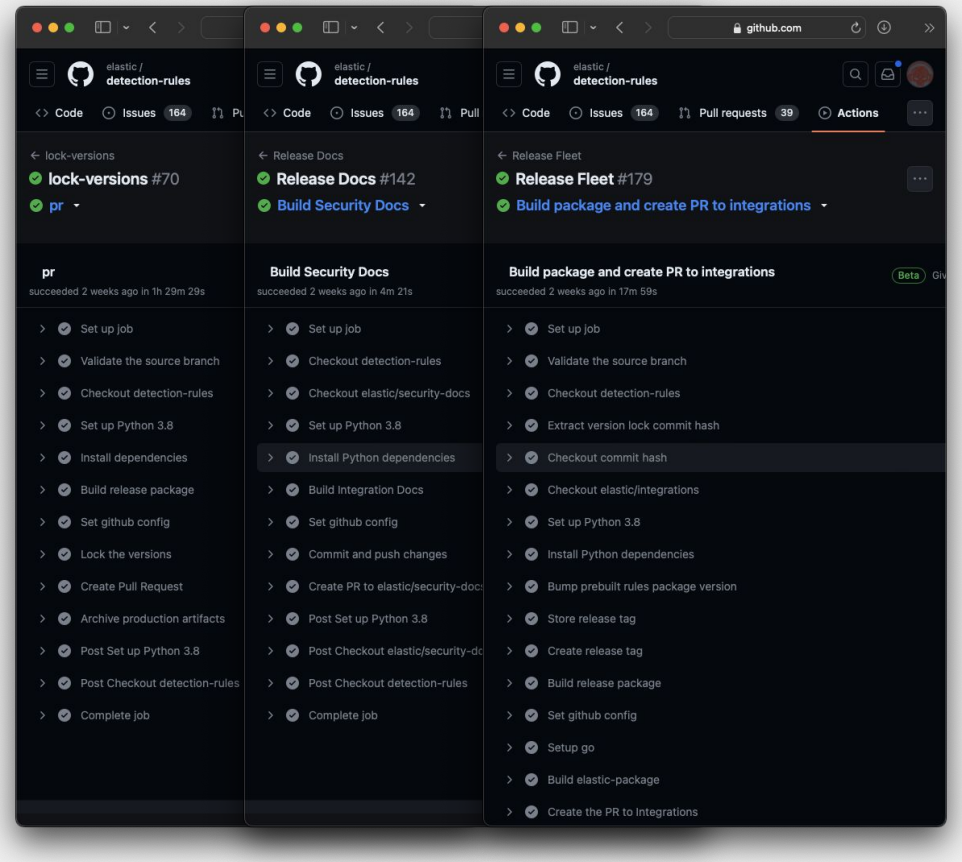
Unit testing and Query Validation

 Out of the box, there is query syntax and semantic validation. Also, it provides unit tests that follow best practices.



Miscellaneous Files for Custom Configuration

 We maintain different files to manage and configure how the rules are versioned and tested.



How we use the CLI internally

 From the beginning, the CLI has served as the core entry point to our CI/CD version and release pipelines. We've recently exposed more of this functionality for others to use!

04 Quickstart E2E Reference alpha Example

 Let’s see a use case where the user wants to implement DaC from scratch.

OS Prerequisites to Follow

Python

Download and install the Python 3.12+ version.

Git

Download and install the latest version of Git.

GitHub

Create a GitHub account if one does not already exist.

SSH Keys

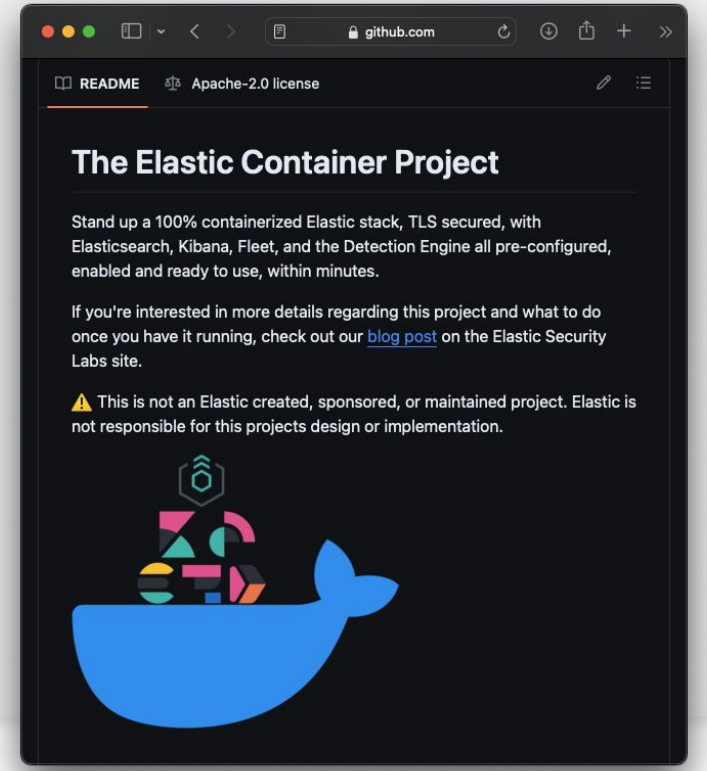
Optionally configure connecting to GitHub with SSH.

Access

Permissions to manage and configure GitHub Action secrets.

 Note: If using an alternative VCS, you will need to translate the principles. Remember this is just one way out of many.

Setup Elastic Security with ECP



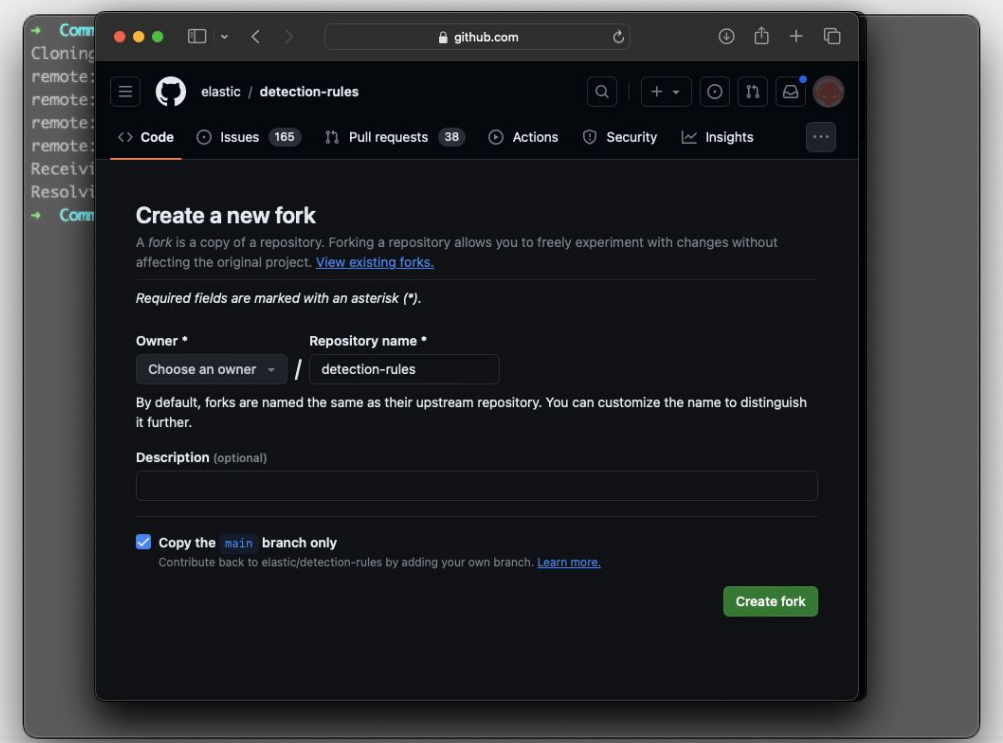
Task

Optionally deploy Elastic Security using the ECP to get up and running quickly.

Steps

1. Navigate to https://github.com/peasead/elastic-container.git
2. Install the prerequisites
3. Follow the instructions to deploy ECP with docker

Fork & Clone Repo



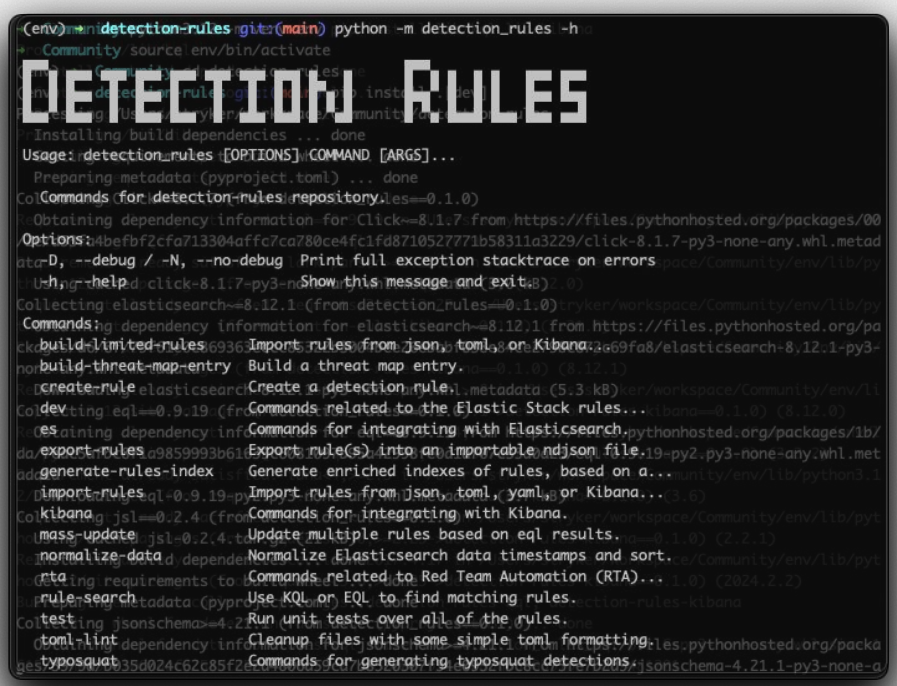
Task

Fork and clone the Elastic detection-rules repo to start managing custom rules with the CLI provided.

Steps

1. Navigate to https://github.com/elastic/detection-rules/fork
2. Choose an owner
3. Click Create Fork
4. Navigate to the forked repo
5. Click Copy url to clipboard
6. Open terminal
7. Run: `git clone git@github.com:<repo>/detection-rules.git`

Install Python Dependencies



Task

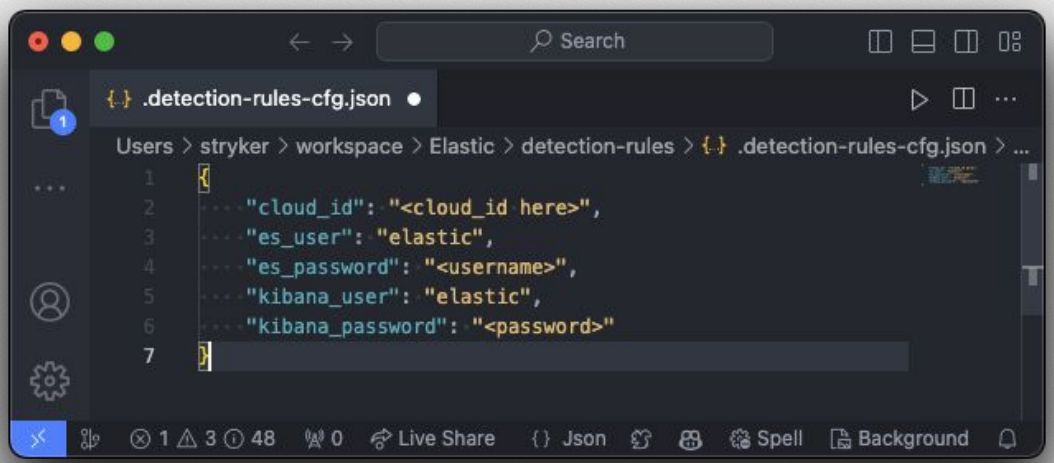
Within the terminal, install the Python dependencies required to use the CLI and test to make sure it's available.

Steps

1. Run: `python -m venv env`
2. Run: `source env/bin/activate`
3. Run: `cd detection-rules`
4. Run: `pip install .[dev]`
5. Run: `pip install lib/kql lib/kibana`

Optionally use the `make` command provided with the Makefile to create the virtual environment and install dependencies.

Configure Remote Authentication



Task

Create an auth config locally to connect to Elastic Security with the CLI.

Steps

1. Create a file in the root of the repo called .detection-rules-cfg.json
2. Supply username, password, and either elasticsearch_url or cloud_id
3. Test the connection

Create & Configure Custom Dir

```
# detection-rules config file

files:
  deprecated_rules: deprecated_rules.json
  packages: packages.yml
  stack_schema_map: stack-schema-map.yaml
  version_lock: version.lock.json
# directories:
  # actions_dir: actions
  # exceptions_dir: exceptions

# to set up a custom rules directory, copy this file to the root of the custom rules directory, which is set
#   using the environment variable DETECTION_RULES_DIR
#   example structure:
#     custom-rules
#       ├── _config.yaml
#       ├── example_rule_1.toml
#       ├── example_rule_2.toml
#       └── etc
#           ├── deprecated_rules.json
#           ├── packages.yml
#           ├── stack-schema-map.yaml
#           └── version.lock.json
#       └── actions
##          ├── action_1.toml
##          ├── action_2.toml
#       └── exceptions
##          ├── exception_1.toml
##          ├── exception_2.toml
#
#   update custom-rules/_config.yaml with:
#     deprecated_rules: etc/deprecated_rules.json
#     packages: etc/packages.yml
#     stack_schema_map: etc/stack-schema-map.yaml
#     version_lock: etc/version.lock.json
#
#   the paths in this file are relative to the custom rules directory (DETECTION_RULES_DIR/)
#
# Refer to each original source file for purpose and proper formatting
#

# testing:
#   config: etc/example_test_config.yaml

# This points to the testing config file (see example under detection_rules/etc/example_test_config.yaml)
# This can either be set here or as the environment variable `DETECTION_RULES_TEST_CONFIG`, with precedence
#   going to the environment variable if both are set. Having both these options allows for configuring testing on
#   prebuilt Elastic rules without specifying a rules _config.yaml.
#
# If set in this file, the path should be relative to the location of this config. If passed as an environment variable,
#   it should be the full path
```

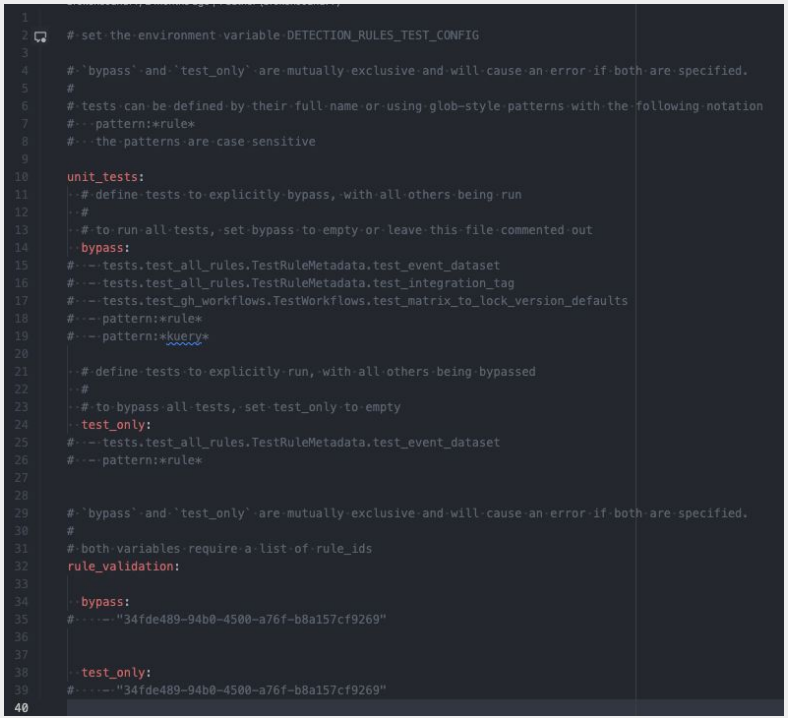
Task

Specify the custom rules folder, initialize the default config files for schema validation, and set the CUSTOM_RULES_DIR.

Steps

1. Run: `python -m detection_rules dac init --custom_dir <directory name>`
2. Run: `export CUSTOM_RULES_DIR=<directory name>`
3. Edit the `_config.yaml` for additional customization (e.g action list, exception list, testing config path, schema, etc.)

Configure Unit Testing



```
#·set·the·environment·variable·DETECTION_RULES_TEST_CONFIG

#·`bypass`·and·`test_only`·are·mutually·exclusive·and·will·cause·an·error·if·both·are·specified.
#
#·tests·can·be·defined·by·their·full·name·or·using·glob-style·patterns·with·the·following·notation
#···pattern:*rule*
#···the·patterns·are·case·sensitive

unit_tests:
··#·define·tests·to·explicitly·bypass,·with·all·others·being·run
··#
··#·to·run·all·tests,·set·bypass·to·empty·or·leave·this·file·commented·out
··bypass:
#·····-·tests.test_all_rules.TestRuleMetadata.test_event_dataset
#·····-·tests.test_all_rules.TestRuleMetadata.test_integration_tag
#·····-·tests.test_gh_workflows.TestWorkflows.test_matrix_to_lock_version_defaults
#·····-·pattern:*rule*
#·····-·pattern:*kuery*

··#·define·tests·to·explicitly·run,·with·all·others·being·bypassed
··#
··#·to·bypass·all·tests,·set·test_only·to·empty
··test_only:
#·····-·tests.test_all_rules.TestRuleMetadata.test_event_dataset
#·····-·pattern:*rule*


#·`bypass`·and·`test_only`·are·mutually·exclusive·and·will·cause·an·error·if·both·are·specified.
#
#·both·variables·require·a·list·of·rule_ids
rule_validation:

··bypass:
#·····-·"34fde489-94b0-4500-a76f-b8a157cf9269"


··test_only:
#·····-·"34fde489-94b0-4500-a76f-b8a157cf9269"
```

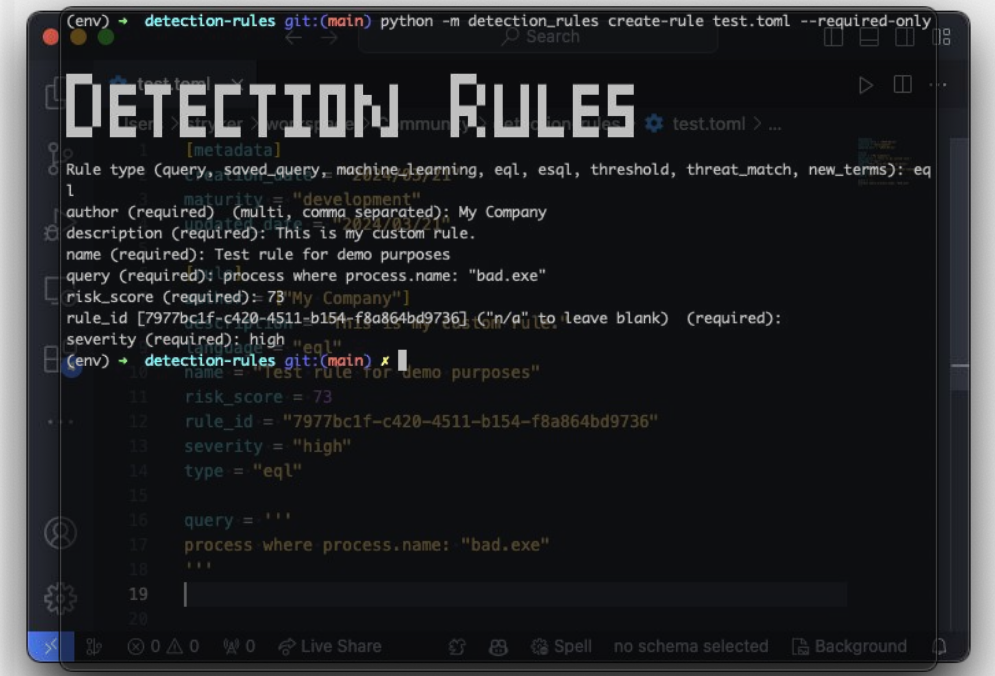
Task

Configure specific unit tests to bypass or test_only. Additional select specific rules to skip or test_only. Default executes all.

Steps

1. Review the prebuilt unit tests within `detection_rules/tests/` to opt-out/opt-in
2. Optionally modify the test config file `etc/example_test_config.yaml` within the CUSTOM_RULES_DIR to specify specific test conditions

Create TOML Rule



Task

Create a custom detection rule TOML file and store within the CUSTOM_RULES_DIR.

Steps

1. Use the interactive CLI to create a rule. Run: `python -m detection_rules create-rule test.toml --required-only`
2. Visually review the created file

Alternatively copy an existing prebuilt rule as a template and modify the values.

Create Action List

```toml
[metadata]
creation_date = "2024-02-21"
rule_id = "5d1e96c6-1ee8-4f19-9416-1d8d81428f59"
rule_name = "Example Rule Name"
updated_date = "2024-02-22"
deprecation_date = "2025-01-01"      # optional
comments = "This is an example action list" # optional
maturity = "beta"                    # optional

[[actions]]
action_type_id = ".email"
group = "default"
params.message = "Action triggered: Example Rule Name"
id = "action_001"                    # optional
frequency = { "throttle": "5m" }     # optional

[[actions]]
action_type_id = ".slack"
group = "default"
params.message = "Some other notification"
```

Task

Optionally configure action lists if managing in TOML files independent from the detection rule logic.

Steps

1. Modify the `CUSTOM_RULES_DIR/_config.yaml` to specify the `action_dir` if not supplied on `dac init`
2. Create an action list TOML file in the actions directory mapped to the rules

Create Exception List

```toml
[metadata]
creation_date = "2024-02-21"
rule_id = "5d1e96c6-1ee8-4f19-9416-1d8d81428f59"
rule_name = "Example Rule Name"
updated_date = "2024-02-22"
comments = "This is an example exception list."
maturity = "development"

[[exceptions]]
description = "Example exception container"
list_id = "exception_list_01"
name = "Sample Exception List"
namespace_type = "single"
tags = ["tag1", "tag2"]
type = "detection"

  [[exceptions.items]]
  description = "Exception item description"
  list_id = "item_list_01"
  name = "Exception Item Name"
  namespace_type = "single"
  tags = ["exception_item_tag1"]
```

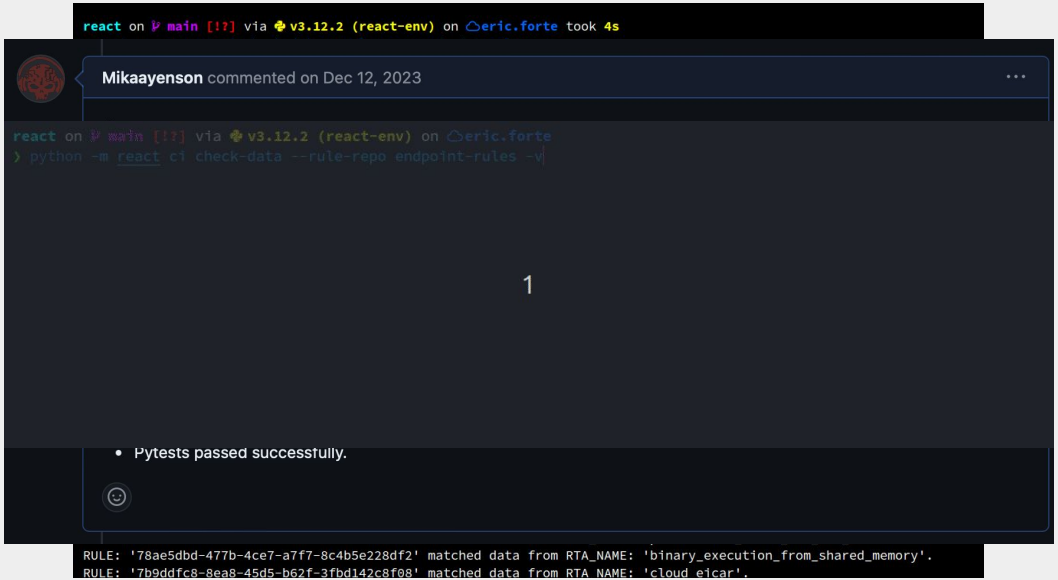
Task

Optionally configure exception lists if managing in TOML files independent from the detection rule logic.

Steps

1. Modify the `CUSTOM_RULES_DIR/_config.yaml` to specify the `exceptions_dir` if not supplied on `dac init`
2. Create an exception list TOML file in the exceptions directory mapped to the rules

Efficacy and Functional Testing Opportunities



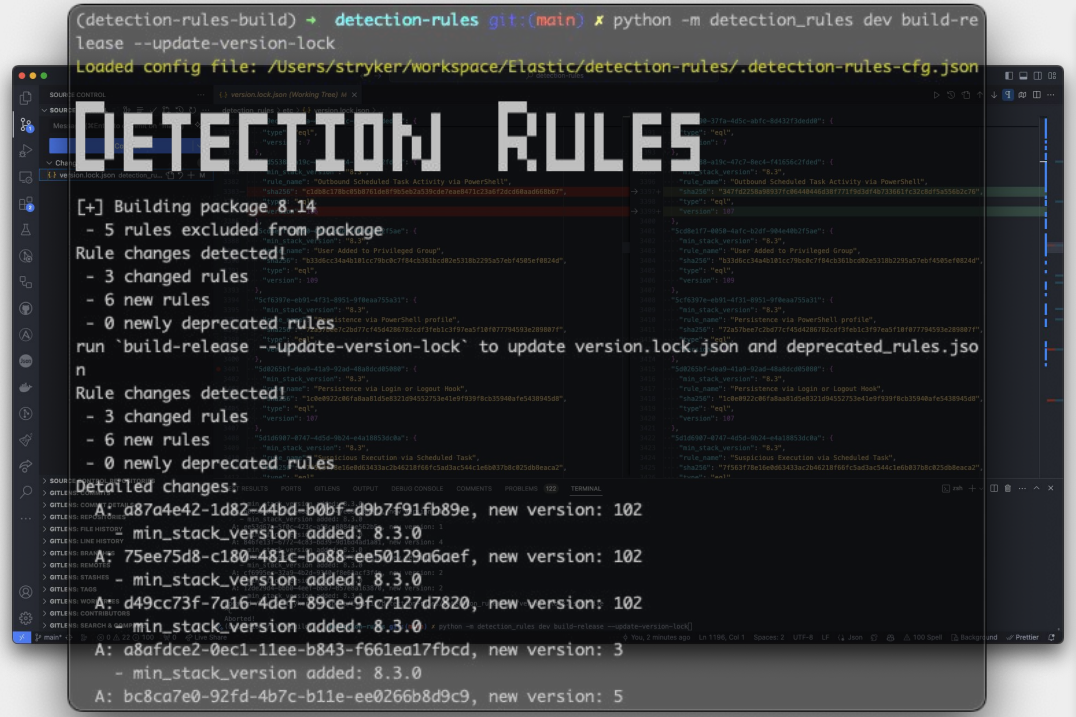
Task

Prior to opening a PR to track custom rules in VCS, perform testing and validation.

Steps

1. Run: `CUSTOM_RULES_DIR=custom-rules python -m detection_rules test`
2. Test the query within Elastic Security to check telemetry

Rule Versioning Strategy



Task

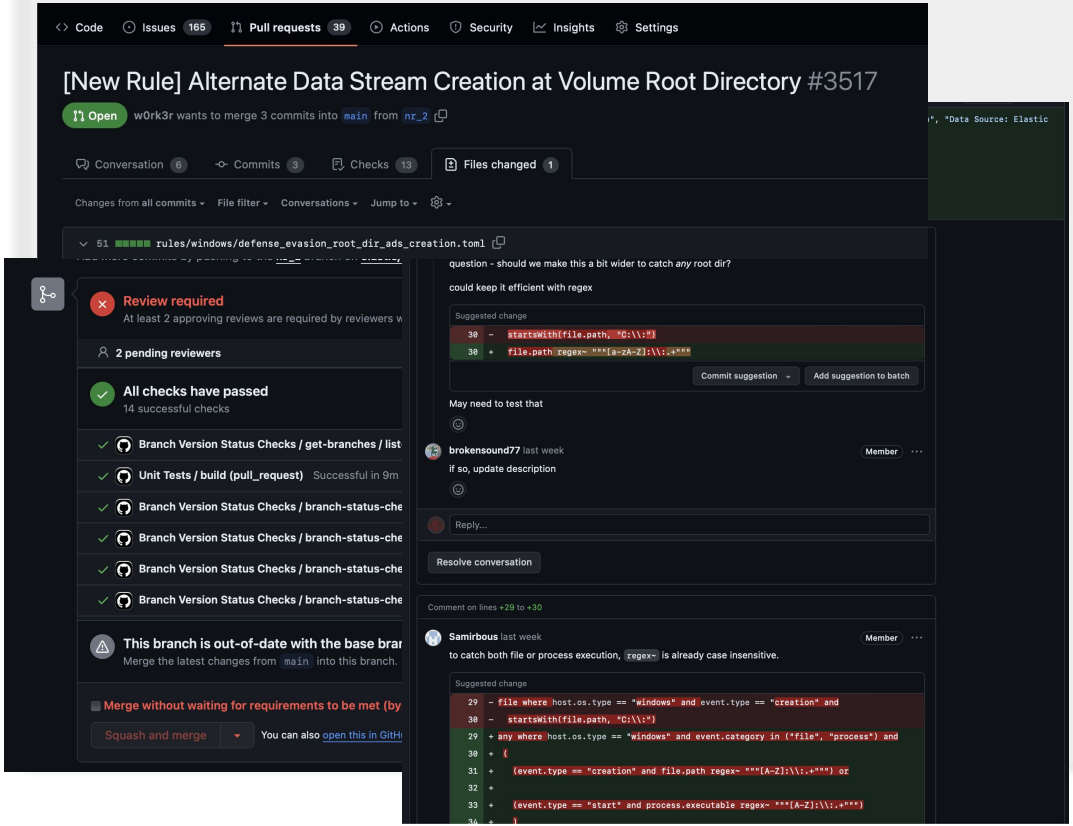
Determine the best versioning strategy either using either:

- a) Kibana `revision` field managed by the detection engine
- b) CLI version lock strategy

Steps

1. Prior to publishing production rules Run: `python -m detection_rules dev build-release --update-version-lock`
2. Commit the version.lock

Traditional PR Review Picasso



Task

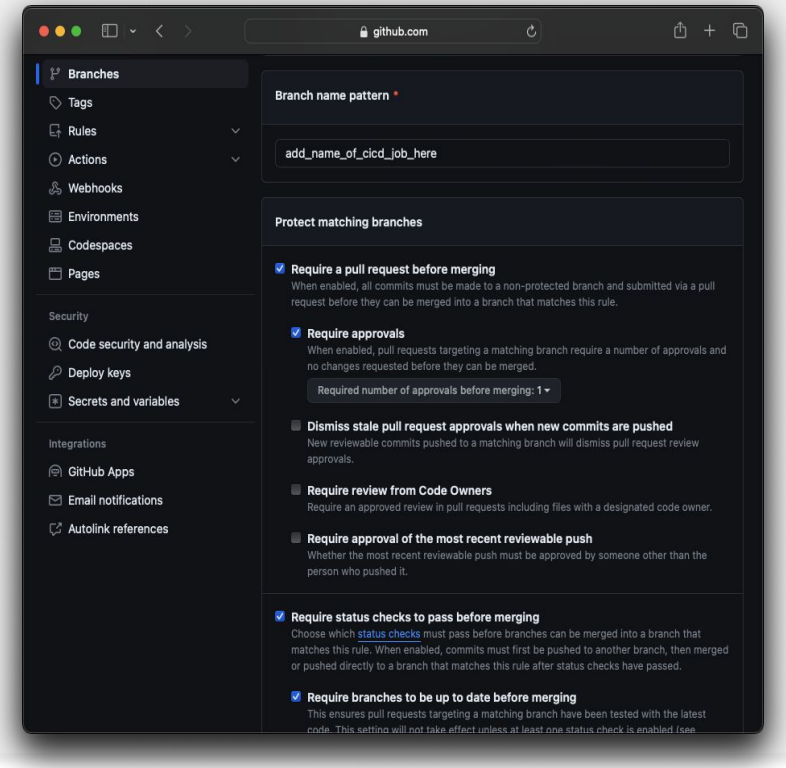
Collaborate with the internal team on the new and tuned rules to review / improve detection rules.

Steps

1. Create a PR to your forked repo and follow traditional PR best practices
2. Review rule metadata
3. Ensure query best detects the threat
4. When unit tests pass, merge!

👋 If you'd like to contribute a good rule upstream, Create a PR from a Fork!

Configure CICD & Branch Protections



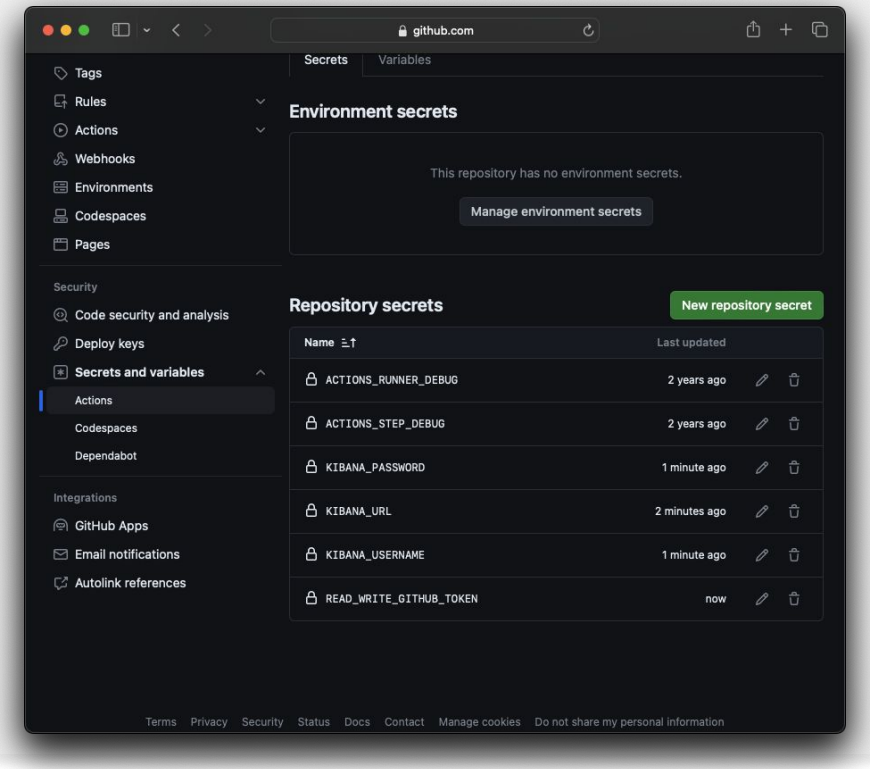
Task

Enforce branch protection policies requiring CI/CD workflows pass before allowing merges so only validated changes are deployed.

Steps

1. Create a new branch protection rule for the main branch
2. Under "Require status checks to pass before merging", select the CI/CD workflows related to rule syncing
3. Apply the branch protection rule and test by creating a new PR to the main branch

Configure Branch Secrets and Variables



Task

Add GitHub Action secrets and variables to open PRs, commit changes, import/export Elastic Security rules in GitHub Actions.

Steps

1. Add GitHub secrets for KIBANA_URL, KIBANA_USER, KIBANA_PASSWORD, and READ_WRITE_GITHUB_TOKEN
2. Add a GitHub variable for CUSTOM_RULES_DIR

Optionally defer testing to Kibana using the built in CLI to test rule responses.

Create CICD Per-PR Sync Options

```yaml
name: CI/CD Per-PR Sync Workflow

on:
  pull_request:
    branches: [ "*" ]
    types: [opened, synchronize, reopened, labeled, unlabeled]

jobs:
  pr-check:
    runs-on: ubuntu-latest
    env: ...

    steps:
    - name: Checkout Repository ...

    - name: Set up Python 3.12 ...

    - name: Install Dependencies ...

    - name: Run Unit Tests ...

    - name: Check for Telemetry Review Label ...

    - name: Telemetry Check (Optional) ...

    - name: Remove Telemetry Review Label ...
```

Task

Configure GitHub Actions validate and test each time a Pull Request is created or updated, promoting early detection of issues.

Steps

1. Create a GitHub action workflow .github/workflows/pr-sync.yml workflow
2. Use the `on: pull_request:` trigger
3. Monitor the PR for successful deployment and validate rule functionality in the test environment

Create Manual Dispatch Sync Options

```yaml
name: Manual Dispatch Sync Workflow

on:
  workflow_dispatch:
    inputs: ...

jobs:
  manual-dispatch-sync:
    runs-on: ubuntu-latest
    env: ...

    steps:
    - name: Checkout Repository ...

    - name: Set up Python 3.12 ...

    - name: Install Dependencies ...

    - name: Export and Import Rules if flag is true ...

    - name: Update Version Lock ...

    - name: Create Pull Request ...

    - name: Commit Directly to Main ...
```

Task

Create on-demand detection rules sync to Elastic Security, giving teams the control to push updates as needed.

Steps

1. Define a .github/workflows/manual-sync.yml
2. Use the workflow_dispatch: event
3. Use GitHub Actions UI to manually trigger the workflow and validate rule synchronization and versions

Create Scheduled Sync Options

```yaml
name: Scheduled Sync Workflow

on:
  schedule:
    # Schedule to run at a specific time, e.g., at 01:00 every day
    - cron: '0 1 * * *'

jobs:
  scheduled-sync:
    runs-on: ubuntu-latest
    env:
      CUSTOM_RULES_DIR: ${{ secrets.CUSTOM_RULES_DIR }}

    steps:
    - name: Checkout Repository ⋯

    - name: Set up Python 3.12 ⋯

    - name: Install Dependencies ⋯

    - name: Export Rules from Kibana ⋯

    - name: Import Rules to Local Repo ⋯

    - name: Update Version Lock ⋯

    - name: Check for Existing Sync PR ⋯

    - name: Create or Update Pull Request ⋯

    - name: Commit to Existing PR Branch ⋯
```

Task

Create scheduled syncs to pull the Elastic Security rules, ensuring consistent alignment without manual intervention.

Steps

1. Create a .github/workflows/scheduled-pull.yml GitHub Action file
2. Use the on: schedule: trigger to define the frequency of updates, such as nightly or weekly pulls
3. Periodically review sync PRs and commit history for updates

Create Push to Production Sync Options

```yaml
name: Push to Production Sync

on:
  push:
    branches:
      - main
    paths:
      - 'version.lock.json'
  workflow_dispatch:

jobs:
  sync-to-production:
    runs-on: ubuntu-latest
    env: ...

    steps:
    - name: Checkout Repository ...

    - name: Set up Python 3.12 ...

    - name: Install Dependencies ...

    - name: Export Rules from Kibana ...

    - name: Import Rules to Local Repo ...

    - name: Verify Version Lock Consistency ...

    - name: Export Rules from Repo to NDJSON ...

    - name: Import Rules to Kibana from NDJSON ...
```

Task

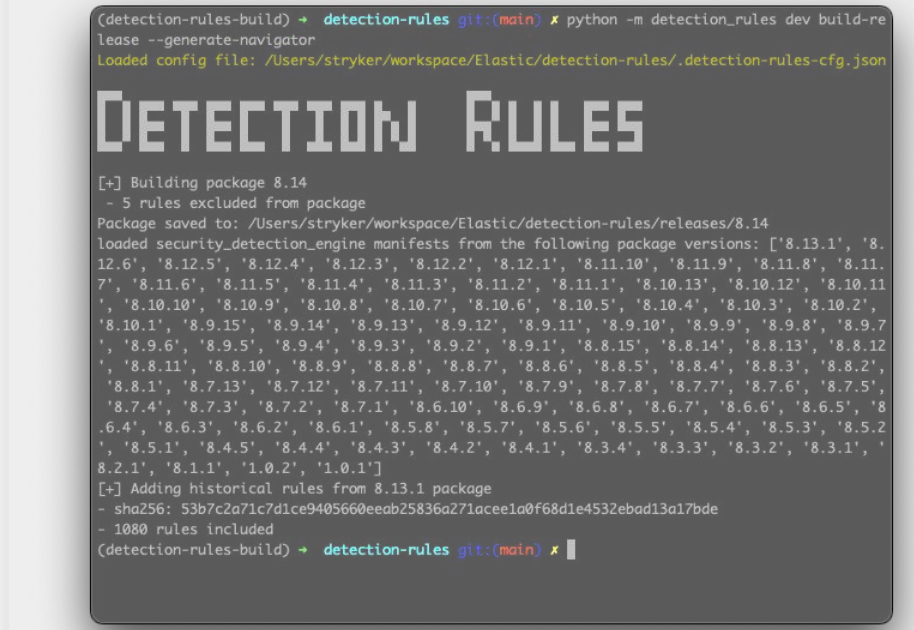
Create a workflow to deploy detection rules to Elastic Security upon new commits into the main branch.

Steps

1. Create a .github/workflows/sync-to-prod.yml GitHub Action
2. In the workflow, use the on: push: branches: [main] trigger
3. Add a step to verify version lock file updates.
4. Test the workflow by merging a test rule into the main branch

05 Go Deeper with Advanced Features

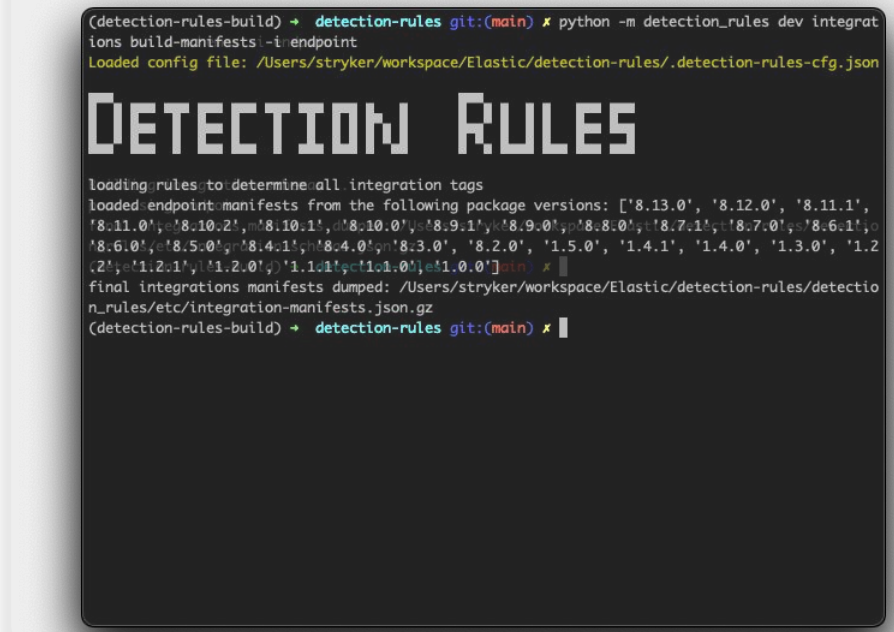
 Exploring the depths of DaC capabilities within the CLI to enhance your detection strategies.



Build Release

This will build a release package that includes MITRE summary information, changelog, and export the rules into an NDJSON. Useful when comprehensively packaging the ruleset.

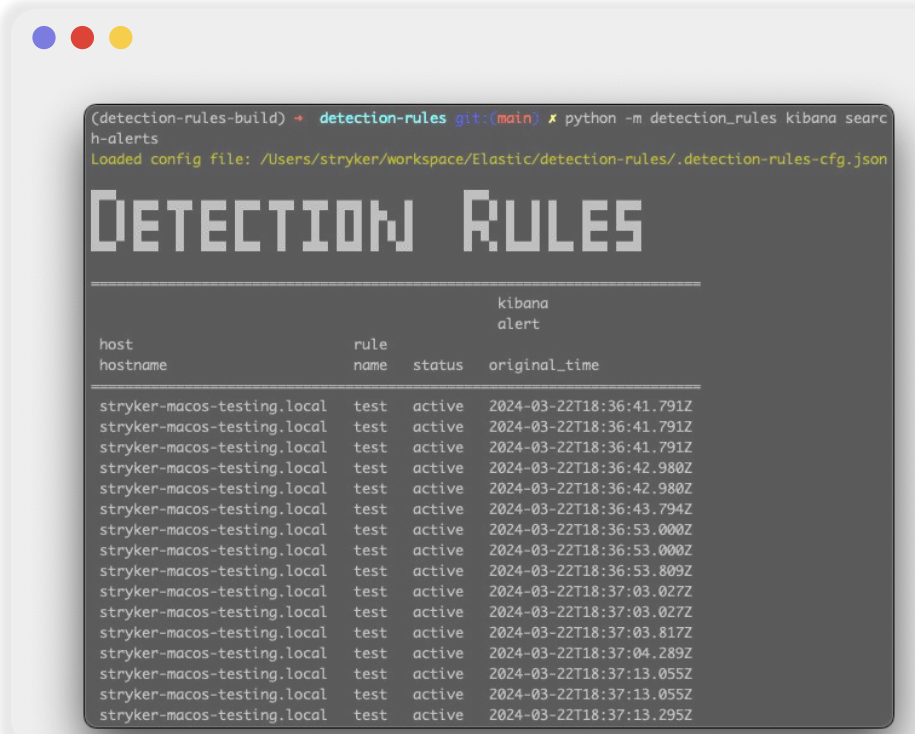
Example Run: `python -m detection_rules dev build-release --generate-navigator`
Example Run: `python -m detection_rules dev build-release --update-version-lock`
Example Run: `make release` (Note: Building a package takes several minutes)



Integration Schemas

These commands will update the integration manifest and integration schemas that are used to validate query fields. Useful when validating custom rules against new integration schemas.

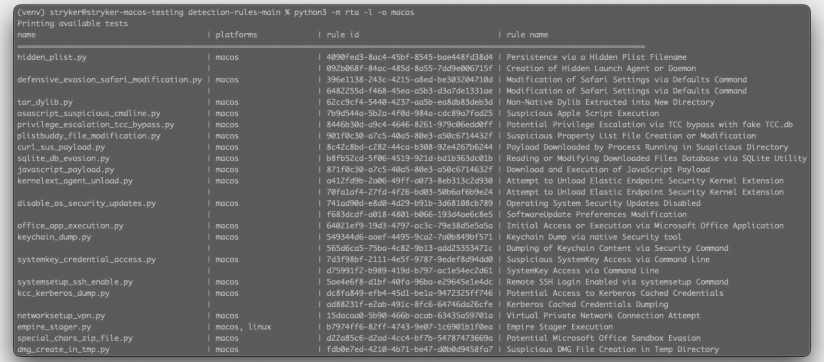
Example Run: `python -m detection_rules dev integrations build-manifests -i endpoint`
Example Run: `python -m detection_rules dev integrations build-schemas -i endpoint`



Search for Alerts

This command will search for alerts generated over a period of time. Useful when programmatically testing detections against adversarial activity.

Example Run: **python -m detection_rules kibana search-alerts**



```
(venv) stryker@stryker-macos-testing detection-rules-main % python3 -m rta -l -o macos
Printing available tests
name                                    | platforms             | rule id                                | rule name
----------------------------------------+-----------------------+----------------------------------------+-----------------------------------------------------------------------------------
hidden_plist.py                         | macos                 | 4090fed3-8ac4-45bf-8545-bae448fd38d4 | Persistence via a Hidden Plist Filename
                                        |                       | 092b068f-84ac-485d-8a55-7dd9e006715f | Creation of Hidden Launch Agent or Daemon
defensive_evasion_safari_modification.py | macos                 | 396e1138-243c-4215-a8ed-be303204710d | Modification of Safari Settings via Defaults Command
                                        |                       | 6482255d-f468-45ea-a5b3-d3a7de1331ae | Modification of Safari Settings via Defaults Command
tar_dylib.py                            | macos                 | 62cc9cf4-5440-4237-aa5b-ea8db83deb3d | Non-Native Dylib Extracted into New Directory
osascript_suspicious_cmdline.py         | macos                 | 7b9d5544a-5b2a-4f0d-984a-cdc89a7fad25 | Suspicious Apple Script Execution
privilege_escalation_tcc_bypass.py      | macos                 | 8446b30d-e9c4-4646-8261-979c06edd0ff | Potential Privilege Escalation via TCC bypass with fake TCC.db
plistbuddy_file_modification.py         | macos                 | 901f0c30-a7c5-40a5-80e3-a50c671432ff | Suspicious Property List File Creation or Modification
curl_sus_payload.py                     | macos                 | 8c42c8bd-c282-44ca-b308-92e4267b6244 | Payload Downloaded by Process Running in Suspicious Directory
sqlite_db_evasion.py                    | macos                 | b8fb52cd-5f06-4519-921d-bd1b363dc01b | Reading or Modifying Downloaded Files Database via SQLite Utility
javascript_payload.py                   | macos                 | 871f0c30-a7c5-40a5-80e3-a50c671432ff | Download and Execution of JavaScript Payload
kernelext_agent_unload.py               | macos                 | a412fd9b-2a06-49ff-a073-8eb313c2d930 | Attempt to Unload Elastic Endpoint Security Kernel Extension
                                        |                       | 70fa1af4-27fd-4f26-bd03-50b6af6b9e24 | Attempt to Unload Elastic Endpoint Security Kernel Extension
disable_os_security_updates.py          | macos                 | 741ad90d-e8d0-4d29-b91b-3d68108cb789 | Operating System Security Updates Disabled
                                        |                       | f683dcdf-a018-4801-b066-193d4ae6c8e5 | SoftwareUpdate Preferences Modification
office_app_execution.py                 | macos                 | 64821ef9-19d3-4797-ac3c-79e36d5e5a5a | Initial Access or Execution via Microsoft Office Application
keychain_dump.py                        | macos                 | 549344d6-aaef-4495-9ca2-7a0b849bf571 | Keychain Dump via native Security tool
                                        |                       | 565d6ca5-75ba-4c82-9b13-add25353471c | Dumping of Keychain Content via Security Command
systemkey_credential_access.py          | macos                 | 7d3f98bf-2111-4e5f-9787-9edef8d94dd0 | Suspicious SystemKey Access via Command Line
                                        |                       | d75991f2-b989-419d-b797-ac1e54ec2d61 | SystemKey Access via Command Line
systemsetup_ssh_enable.py               | macos                 | 5ae4e6f8-d1bf-40fa-96ba-e29645a1e4dc | Remote SSH Login Enabled via systemsetup Command
kcc_kerberos_dump.py                    | macos                 | dc8fa849-efb4-45d1-be1a-9472325ff746 | Potential Access to Kerberos Cached Credentials
                                        |                       | ad88231f-e2ab-491c-8fc6-64746da26cfe | Kerberos Cached Credentials Dumping
networksetup_vpn.py                     | macos                 | 15dacaa0-5b90-466b-acab-63435a59701a | Virtual Private Network Connection Attempt
empire_stager.py                        | macos, linux          | b7974ff6-82ff-4743-9e07-1c6901b1f0ea | Empire Stager Execution
special_chars_zip_file.py               | macos                 | d22a85c6-d2ad-4cc4-bf7b-54787473689a | Potential Microsoft Office Sandbox Evasion
dmg_create_in_tmp.py                    | macos                 | fdb0e7ed-4210-4b71-be47-d0b0d9458fa7 | Suspicious DMG File Creation in Temp Directory
```

Executing RTAs

These commands will list and execute red team automation python scripts that run on Windows, MacOS, and *nix. Useful to emulate adversarial activity.

Example Run: `python -m rta -l`
Example Run: `python -m rta -n eicar`

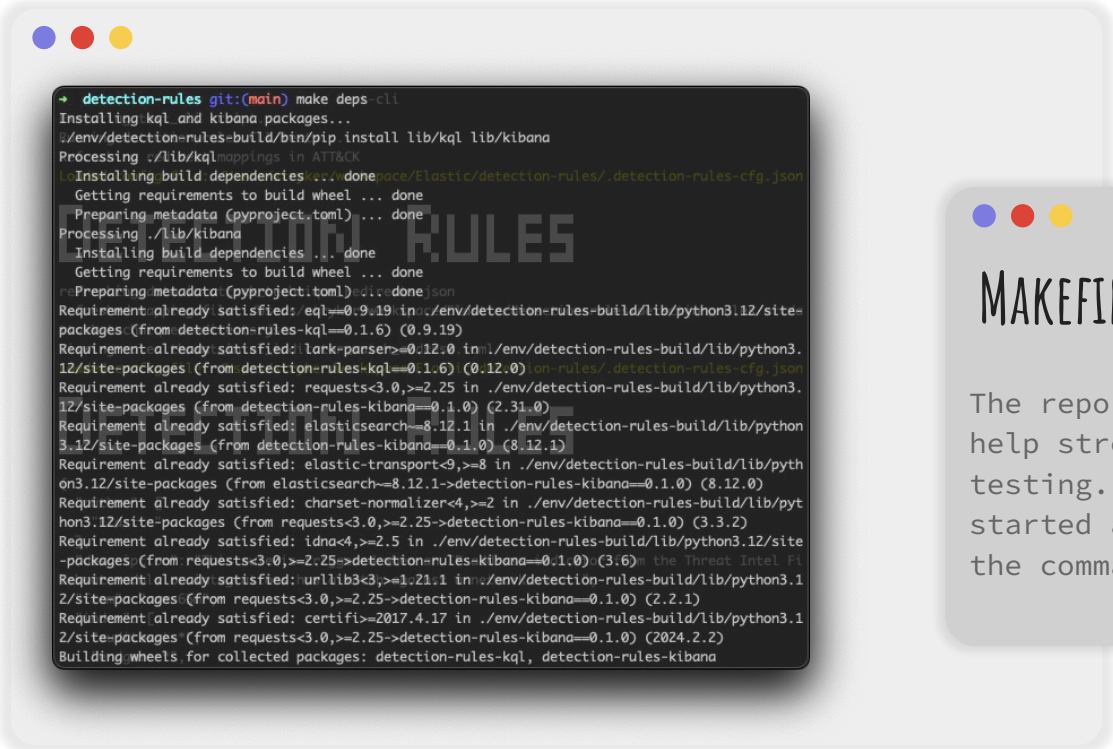
Collect Elastic Events

This command will collect events from Elasticsearch. Useful to collect while testing adversarial activity (e.g. RTAs).

Example Run:

```
python -m detection_rules es collect-events 3a2437df-bed6-4d6a-b390-16f27548f340 -i "logs-endpoint.*"
```

(Note: The UUID is the **host.id** field of the endpoint)



Makefile

The repo includes a Makefile to help streamline installation and testing. Useful for getting started and testing out some of the commands.

Example Run: **make deps**
Example Run: **make test-cli** (Note: Will generate several files)

06 Conclusion and Questions

 We encourage you early adopters to test out our Alpha DaC capabilities and provide feedback!

Releasing Resources today to help you Start Rolling

Reference Doc

Check out the Reference doc for pros/cons of different approaches.

DaC Use Cases

Check out the DaC-use-cases GitHub repo for example approaches.

Need Pointers?

Feel free to reach out on Elastic's community slack #security-rules-dac channel.

Connect + Community + Contributing

Do you have any questions?

Mika Ayenson
@stryker0x

Justin Ibarra
@br0k3ns0und