

# ROLLING YOUR OWN DETECTIONS AS CODE

FT. ELASTIC SECURITY



`<p>Mika Ayenson : @stryker0x</p>`

`<p>Justin Ibarra : @br0k3ns0und</p>`



HELLO! I'M...

<p>

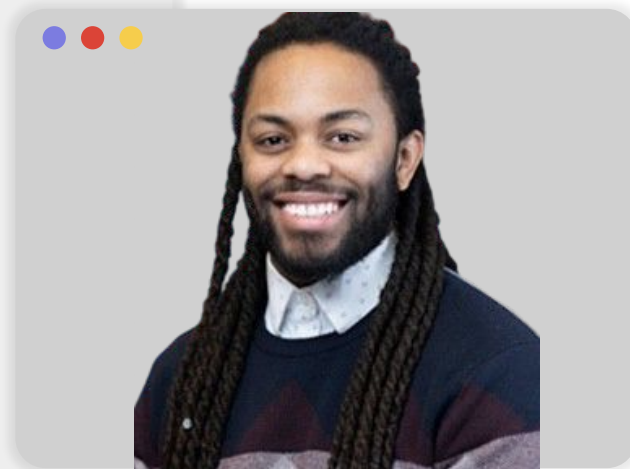
**Mika Ayenson**, Ph.D.

Senior Security Research Engineer @  
Elastic

</p>



@stryker0x



HELLO! I'M...

<p>

**Justin Ibarra,**

Threat Research and Detection  
Engineering Lead @ Elastic

</p>



@br0k3ns0und



# TABLE OF CONTENTS.



01

What it is DaC  
and Why it's  
Needed



02

High Level  
Components,  
Workflows, and  
Delineation.



03

Bias to Leverage  
detection-rules.



04

Quickstart **E2E**  
**Reference** Example



05

*Slides only*  
Go Deeper with  
Advanced Features



06

**Conclusion** and  
Questions





# 01 WHAT IS DAC AND WHY IT'S NEEDED?

<p> Perhaps you've heard of [Infrastructure as Code](#) (IaC)?! DaC is the close relative! </p>

# 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.



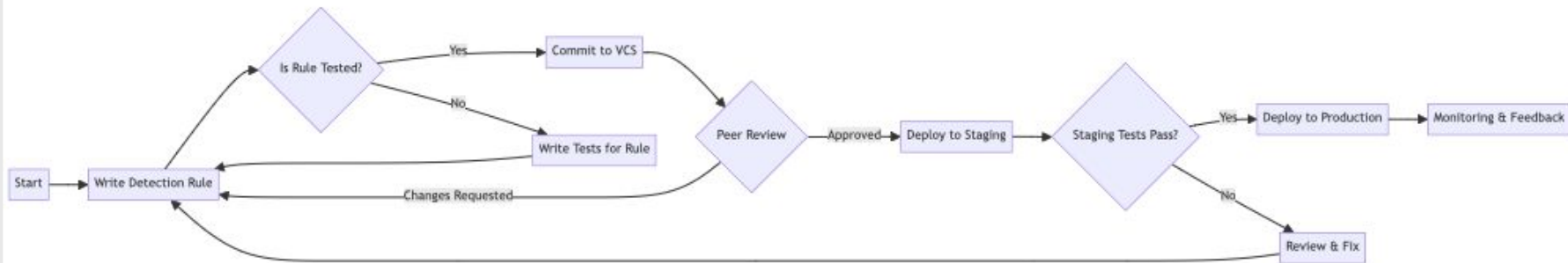
## DevOps Engineers

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

`<p> 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.</p>`



# 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>



#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.

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

# 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



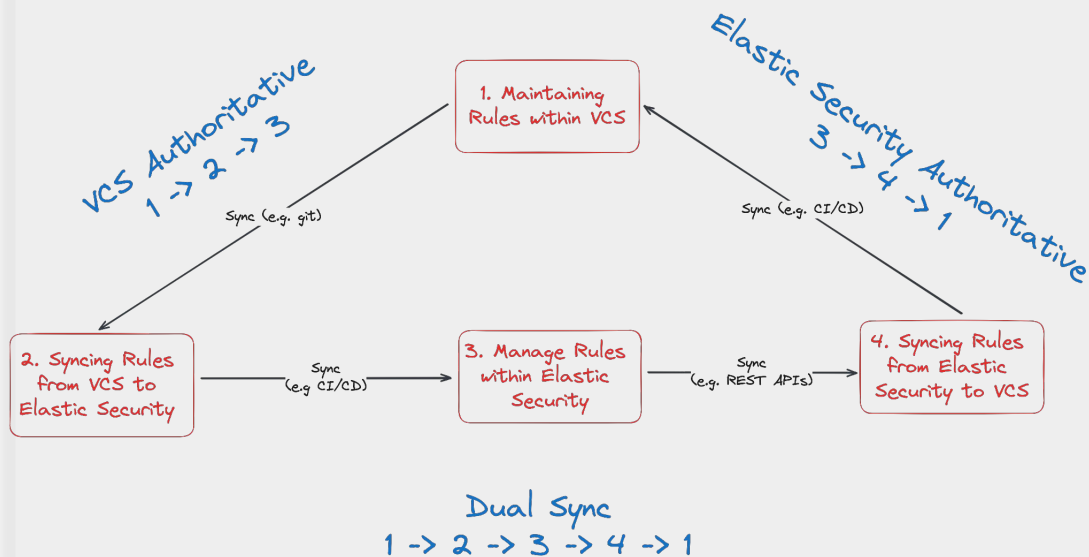
## Hierarchy

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

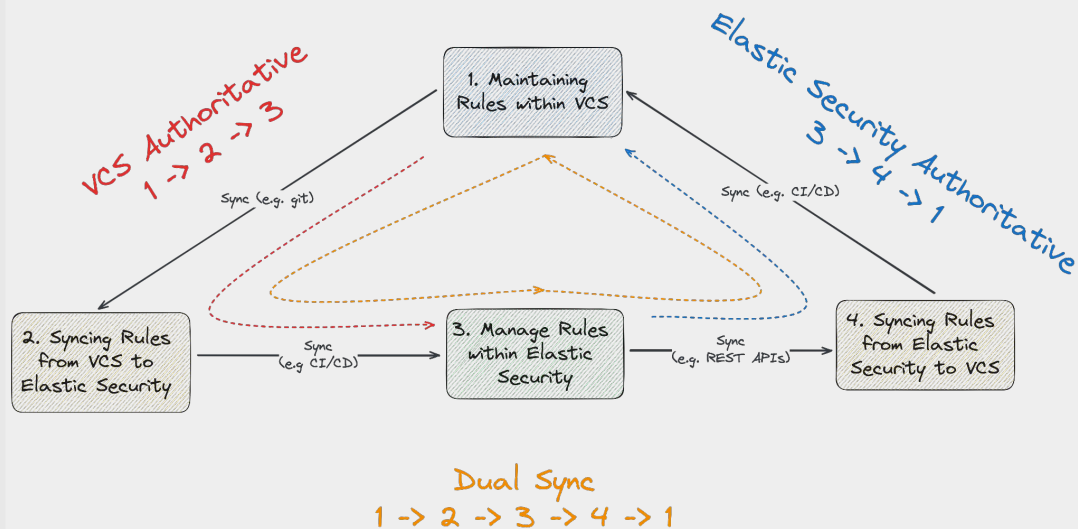
# GOVERNANCE MODELS

- 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



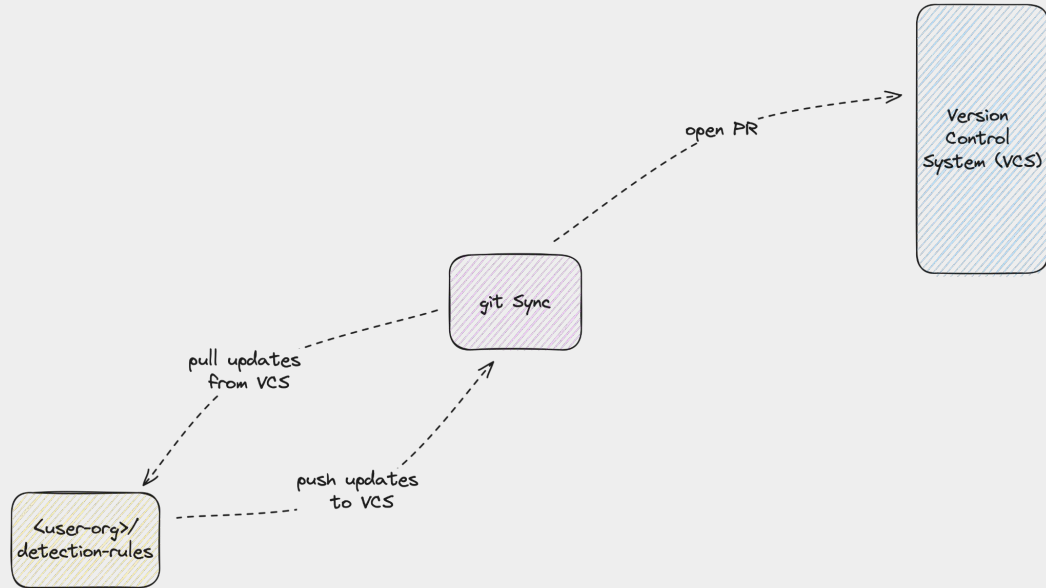
## SYNCING OPTIONS



## 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.

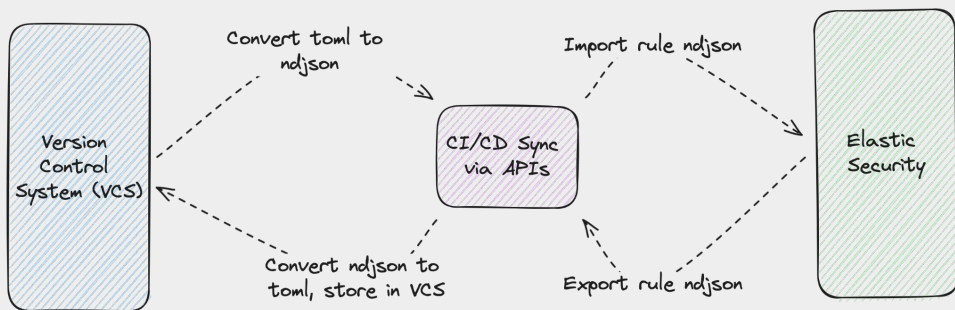
## Requirements

- 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

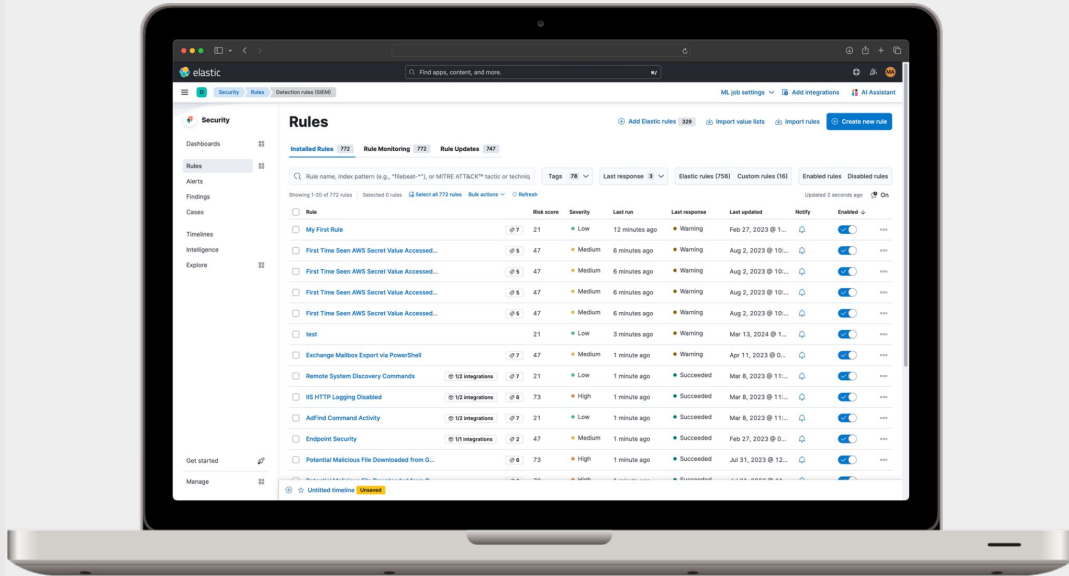
- API access to Elastic Stack
- Authentication credentials
- CI/CD pipeline (optional)

## 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.

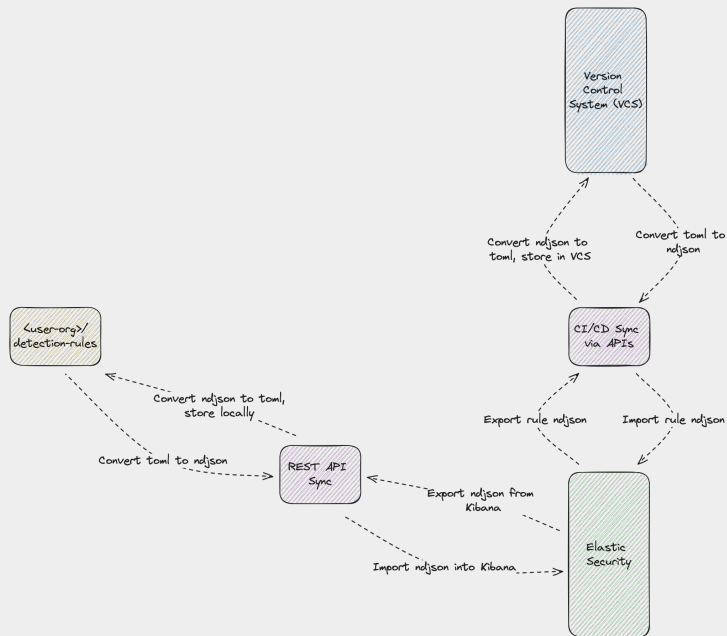
## Requirements

- 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.

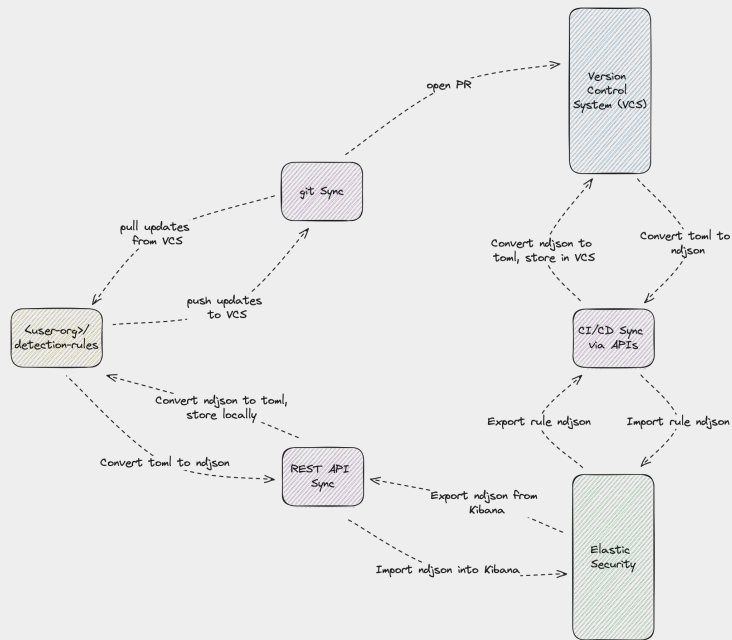
## Requirements

- 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

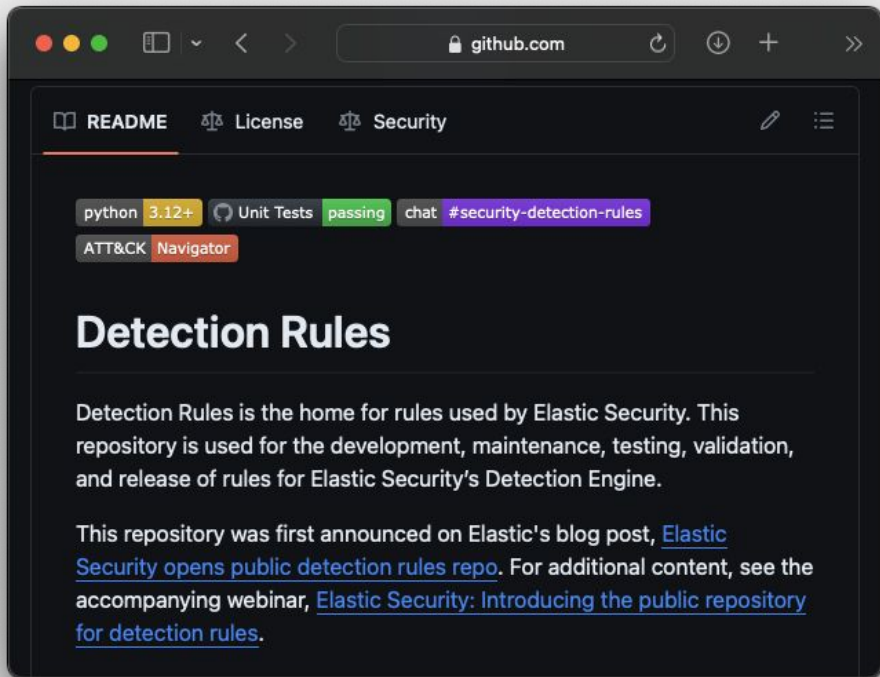
## RULE MANAGEMENT OPTIONS

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



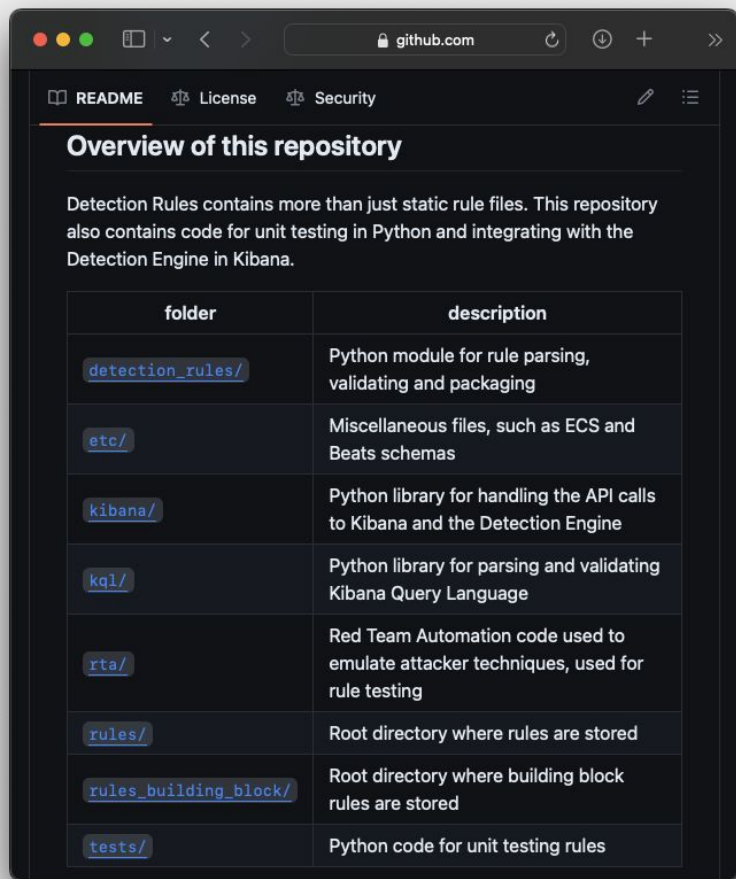
# 03 BIAS TO LEVERAGE DETECTION-RULES

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



# GITHUB/ELASTIC/ DETECTION-RULES

<p> First, here's a primer on the [github.com/elastic/detection-rules](https://github.com/elastic/detection-rules) repo, features, and CLI w/DaC context. </p>



The screenshot shows a GitHub repository overview page. At the top, there are navigation links for 'README', 'License', and 'Security'. The main heading is 'Overview of this repository'. Below this, a paragraph explains that the repository contains more than just static rule files, including Python code for unit testing and integration with the Detection Engine in Kibana. A table follows, listing various folders and their descriptions.

folder	description
<a href="#">detection_rules/</a>	Python module for rule parsing, validating and packaging
<a href="#">etc/</a>	Miscellaneous files, such as ECS and Beats schemas
<a href="#">kibana/</a>	Python library for handling the API calls to Kibana and the Detection Engine
<a href="#">kql/</a>	Python library for parsing and validating Kibana Query Language
<a href="#">rta/</a>	Red Team Automation code used to emulate attacker techniques, used for rule testing
<a href="#">rules/</a>	Root directory where rules are stored
<a href="#">rules_building_block/</a>	Root directory where building block rules are stored
<a href="#">tests/</a>	Python code for unit testing rules

# REPO STRUCTURE RULES AND DAC RULE MANAGEMENT

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

# AVAILABLE BEHAVIOR AND BUILDING BLOCK PREBUILT DETECTIONS

<p> 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.</p>

The image displays three overlapping screenshots of a GitHub repository. The top-left screenshot shows the 'rules/' directory with a table of folders and their descriptions. The top-right screenshot shows the 'rules\_building\_block/' directory with a list of files and their commit history. The bottom-center screenshot shows the 'integrations/' directory with a table of integration-specific folders and their corresponding services.

folder	description
.	Root directory where rules are stored
<a href="#">apm/</a>	Rules that use Application Performance Monitoring (APM) data sources
<a href="#">cross-platform/</a>	Rules that apply to multiple platforms, such as Windows and Linux
<a href="#">integrations/</a>	Rules organized by Fleet integration
<a href="#">linux/</a>	Rules for Linux or other Unix based systems
<a href="#">macos/</a>	Rules for macOS
<a href="#">ml/</a>	Rules that use machine learning
<a href="#">network/</a>	Rules that use network data sources
<a href="#">promotions/</a>	Rules that promote external alerting engine alerts
<a href="#">windows/</a>	Rules for the Microsoft Windows operating system

Integration specific rules are stored in the [integrations/](#) directory:

folder	integration
<a href="#">aws/</a>	Amazon Web Services (AWS)
<a href="#">azure/</a>	Microsoft Azure
<a href="#">cyberarkpas/</a>	Cyber Ark Privileged Access Security
<a href="#">endpoint/</a>	Elastic Endpoint Security
<a href="#">gcp/</a>	Google Cloud Platform (GCP)
<a href="#">google_workspace/</a>	Google Workspace (formerly GSuite)
<a href="#">o365/</a>	Microsoft Office
<a href="#">okta/</a>	Okta



The screenshot shows a GitHub repository page for 'Red Team Automation' on the 'main' branch. The repository is located at 'detection-rules / rta /'. The page title is 'Red Team Automation'. Below the title, there are tags for 'python 3.7+' and 'chat #security-detection-rules'. The main content is a README file with the following text:

The repo comes with some red team automation (RTA) python scripts that run on Windows, Mac OS, and \*nix. RTA scripts emulate known attacker behaviors and are an easy way too verify that your rules are active and working as expected.

```
$ python -m rta -h
usage: rta [-h] ttp_name

positional arguments:
  ttp_name

optional arguments:
  -h, --help  show this help message and exit
```

`ttp_name` can be found in the `rta` directory. For example to execute `./rta/wevtutil_log_clear.py` script, run command:

```
$ python -m rta wevtutil_log_clear
```

Most of the RTA scripts contain a comment with the rule name, in `signal.rule.name`, that maps to the Kibana Detection Signals.

# RED TEAM AUTOMATION

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



# KQL AND KIBANA PYTHON LIBRARIES

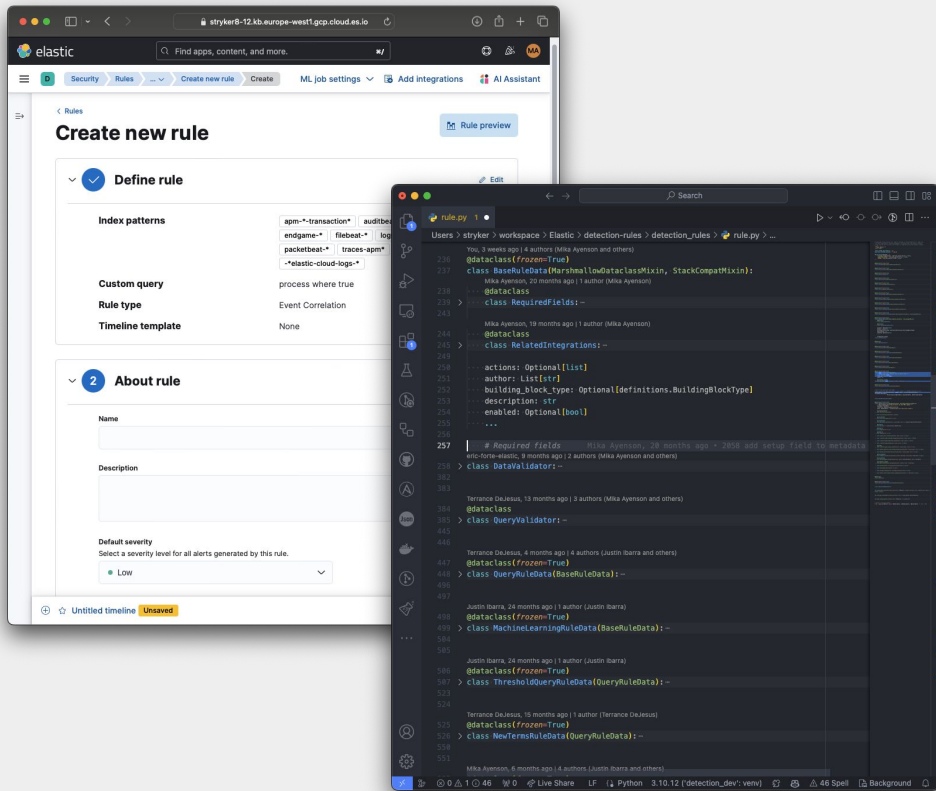
```
# KQL Library Example Untitled-1
1 # KQL Library Example
2 from kql import parse, to_dsl
3
4 # Sample KQL query
5 kql_query = 'process.name: "cmd.exe" and process.args: "-k"'
6
7 # Parse the KQL query
8 parsed_query = parse(kql_query)
9
10 # Convert the parsed KQL query to Elasticsearch Query DSL
11 dsl_query = to_dsl(parsed_query)
12
13 print(dsl_query)
```

```
# Kibana API client example to list all Untitled-1
1 # Kibana API client example to list all custom detection rules
2 from kibana import Kibana, RuleResource
3
4 # Initialize the Kibana client
5 kibana_client = Kibana(kibana_url="http://localhost:5601",
6 kibana_username="your_username", kibana_password="your_password")
7
8 # Authenticate
9 kibana_client.login("your_username", "your_password")
10
11 # List all custom detection rules
12 custom_rules = RuleResource.find_custom()
13
14 for rule in custom_rules:
15     print(rule.id, rule.name)
```

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

LET'S GO A LITTLE  
BIT DEEP...ER





# DATACLASS AND MARSHMALLOW SCHEMA VALIDATION

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

The image shows a code editor window with a dark theme. On the left is a file explorer showing a project structure for 'detection-rules'. The main editor area displays Python code for unit tests. The code includes a base class `TestValidRules(BaseRuleTest)` and a subclass `TestRuleTags(BaseRuleTest)`. The `TestValidRules` class has several test methods: `test_schema_and_dupes`, `test_file_names`, `test_all_rule_queries_optimized`, `test_production_rules_have_rta`, `test_duplicate_file_names`, `test_rule_type_changes`, and `test_bbr_validation`. The `TestRuleTags` class has methods: `test_technique_deprecations`, `test_tactic_to_technique_correlations`, and `test_duplicated_tactics`. The `TestThreatMappings` class has methods: `test_technique_deprecations`, `test_tactic_to_technique_correlations`, and `test_duplicated_tactics`. The `TestValidRules` class also has a `test_esol_rules.py` file. The code is well-commented and includes docstrings. The editor interface includes a search bar, a file explorer, and a status bar at the bottom.

# UNIT TESTING AND QUERY VALIDATION

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



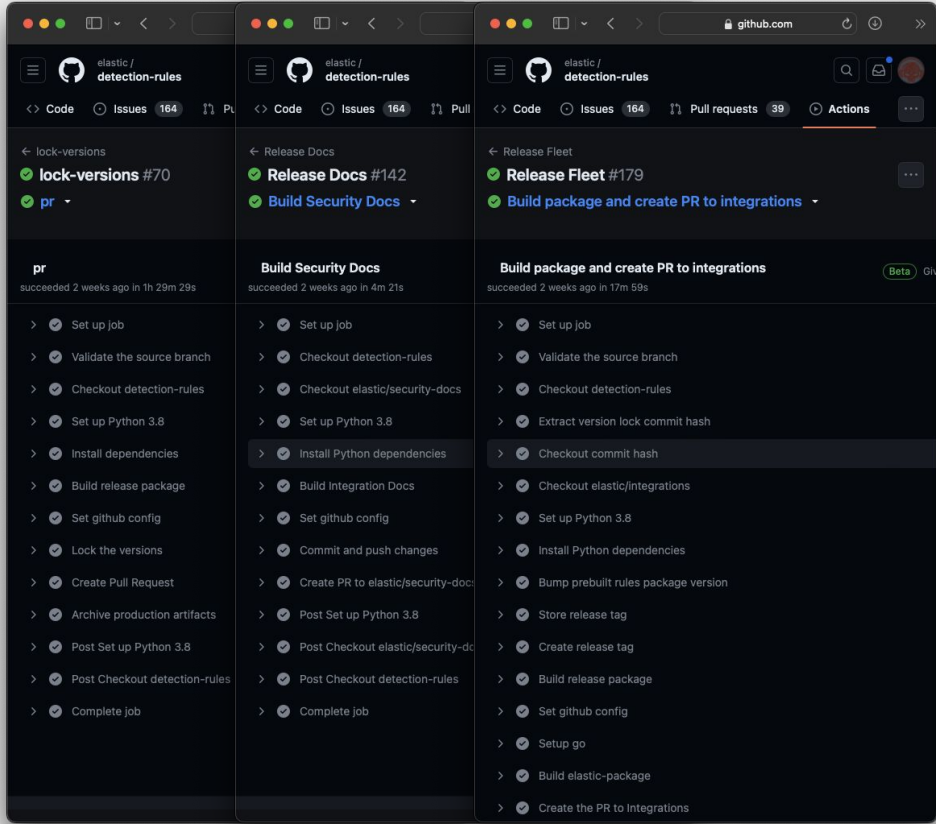
```

  detection_rules
  etc
  api_schemas
  beats_schemas
  ecs_schemas
  endgame_schemas
  endpoint_schemas
  attack-crosswalk.json
  attack-technique-redirects.json
  attack-v13.1.0.json.gz
  commit-and-push.sh
  deprecated_rules.json
  downloadable_updates.json
  integration-manifests.json.gz
  integration-schemas.json.gz
  lock-multiple.sh
  non-ecs-schema.json
  packages.yml
  rule_template_typosquatting_domain.json
  rule-mapping.yml
  security-logo-color-64px.svg
  stack-schema-map.yml
  test_cli.bash
  test_remote_cli.bash
  test_toml.json
  version.lock.json

```

# MISCELLANEOUS FILES FOR CUSTOM CONFIGURATION

<p> We maintain different files to manage and configure how the rules are versioned and tested. </p>



# HOW WE USE THE CLI INTERNALLY

<p> 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! </p>



# 04 QUICKSTART E2E REFERENCE EXAMPLE

`<p>` Let's see a use case where the user wants to implement DaC from scratch.`</p>`



LET'S GO ALPHA!



LEG



# 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.

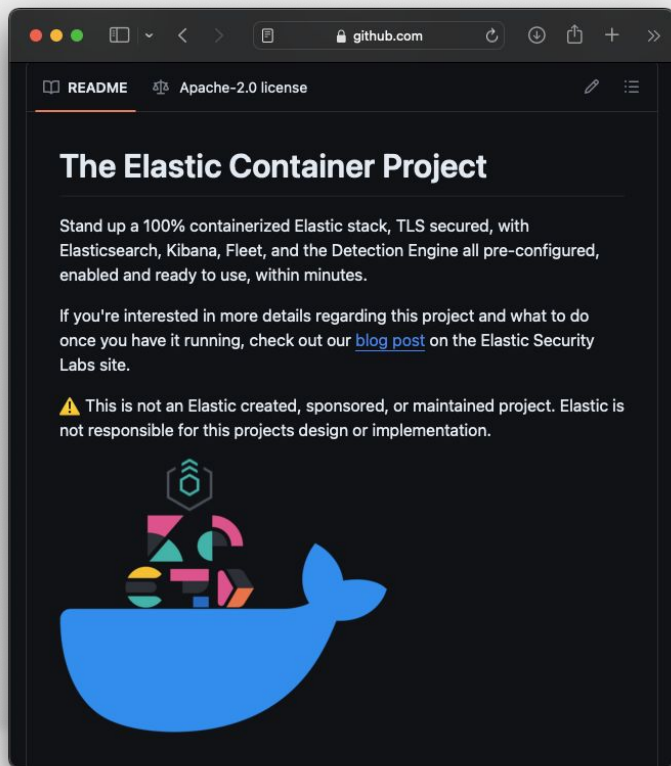


## Access

Permissions to manage and configure GitHub Action [secrets](#).

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

# SETUP ELASTIC SECURITY WITH ECP



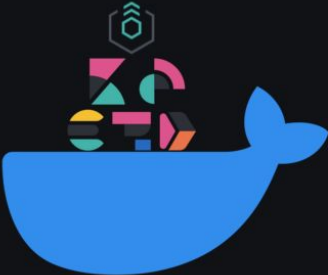
README Apache-2.0 license

## The Elastic Container Project

Stand up a 100% containerized Elastic stack, TLS secured, with Elasticsearch, Kibana, Fleet, and the Detection Engine all pre-configured, enabled and ready to use, within minutes.

If you're interested in more details regarding this project and what to do once you have it running, check out our [blog post](#) on the Elastic Security Labs site.

⚠ This is not an Elastic created, sponsored, or maintained project. Elastic is not responsible for this projects design or implementation.



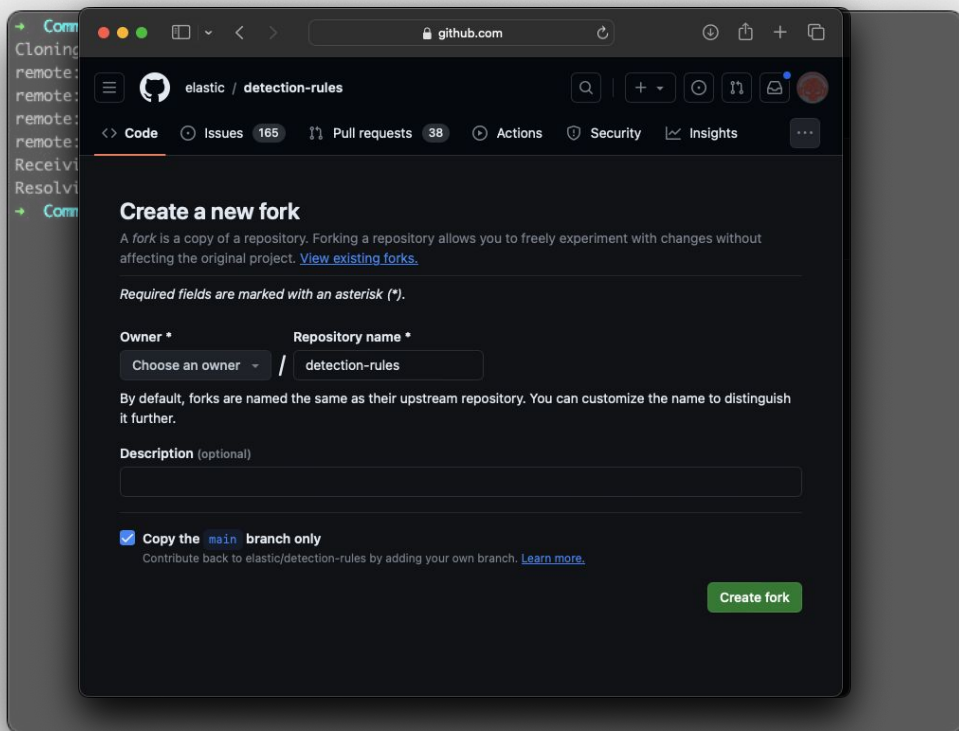
## Task

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

```
(env)minidetection-rules git:(main) python -m detection_rules -h
Community source env/bin/activate
Usage: detection_rules [OPTIONS] COMMAND [ARGS]...
Preparing metadata (pyproject.toml) ... done
Commands for detection_rules repository:
  click==8.1.0
  Obtain dependency information for Click==8.1.7 from https://files.pythonhosted.org/packages/00
Options:
  -D, --debug / -N, --no-debug Print full exception stacktrace on errors
  -h, --help Show this message and exit.
Collecting elasticsearch==8.12.1 (from detection_rules==0.1.0)
Commands:
  build-limited-rules Import rules from json, toml, or Kibana...
  build-threat-map-entry Build a threat map entry.
  create-rule Create a detection rule.
  dev Installing dependency information for elasticsearch==8.12.1 from https://files.pythonhosted.org/pa
  export-rules Export rule(s) into an importable ndjson file.
  generate-rules-index Generate enriched indexes of rules, based on a...
  import-rules Import rules from json, toml, yaml, or Kibana...
  kibana Commands for integrating with Kibana.
  mass-update Update multiple rules based on eql results.
  normalize-data Normalize Elasticsearch data timestamps and sort.
  requirements Requirements related to Red Team Automation (RTA)...
  rule-search Use KQL or EQL to find matching rules.
  test Run unit tests over all of the rules.
  toml-lint Cleanup tests with some simple toml formatting.
  typosquat Commands for generating typosquat detections.
```

## DETECTION RULES

### Task

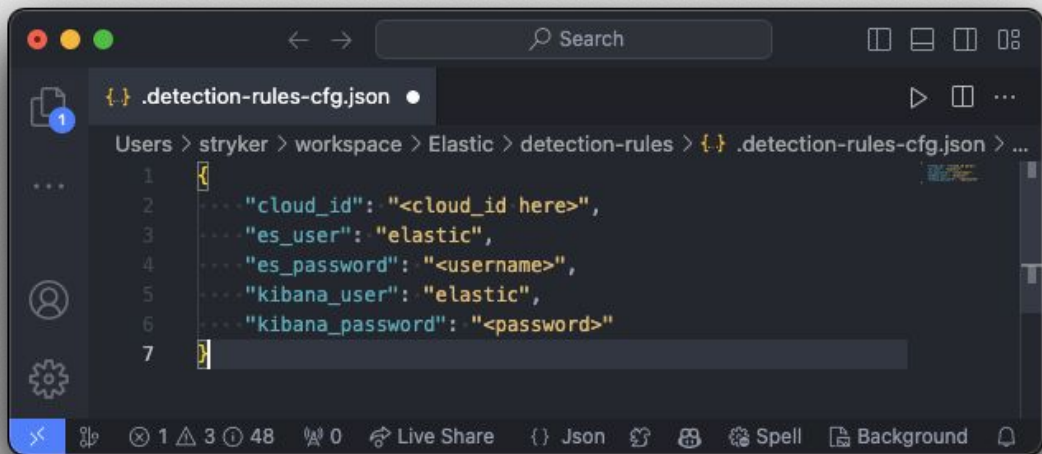
Within the terminal, install the Python dependencies required to use the CLI and test to make sure it's available.

### 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



```
Users > stryker > workspace > Elastic > detection-rules > .detection-rules-cfg.json > ...
1
2  "cloud_id": "<cloud_id here>",
3  "es_user": "elastic",
4  "es_password": "<username>",
5  "kibana_user": "elastic",
6  "kibana_password": "<password>"
7
```

## 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

```
1 # detection-rules config file
2
3 files:
4   deprecated_rules: deprecated_rules.json
5   packages: packages.yml
6   stack_schema_map: stack-schema-map.yaml
7   version_lock: version.lock.json
8 # directories:
9 #   actions_dir: actions
10 #   exceptions_dir: exceptions
11
12 # to set up a custom rules directory, copy this file to the root of the custom rules directory, which is set
13 # using the environment variable DETECTION_RULES_DIR
14 # example structure:
15 # --- custom-rules
16 # --- | _config.yaml
17 # --- | example_rule_1.toml
18 # --- | example_rule_2.toml
19 # --- | etc
20 # --- | deprecated_rules.json
21 # --- | packages.yml
22 # --- | stack-schema-map.yaml
23 # --- | version.lock.json
24 # --- | actions
25 # --- | | action_1.toml
26 # --- | | action_2.toml
27 # --- | exceptions
28 # --- | | exception_1.toml
29 # --- | | exception_2.toml
30 #
31 # update custom-rules/_config.yaml with:
32 # deprecated_rules: etc/deprecated_rules.json
33 # packages: etc/packages.yml
34 # stack_schema_map: etc/stack-schema-map.yaml
35 # version_lock: etc/version.lock.json
36 #
37 # the paths in this file are relative to the custom rules directory (DETECTION_RULES_DIR)
38 #
39 # Refer to each original source file for purpose and proper formatting
40 #
41 # testing:
42 # config: etc/example_test_config.yaml
43
44 # This points to the testing config file (see example under detection_rules/etc/example_test_config.yaml)
45 # This can either be set here or as the environment variable 'DETECTION_RULES_TEST_CONFIG', with precedence
46 # going to the environment variable if both are set. Having both these options allows for configuring testing on
47 # prebuilt Elastic rules without specifying a rules _config.yaml.
48 #
49 # If set in this file, the path should be relative to the location of this config. If passed as an environment variable
50 # 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

```
1
2 # set the environment variable DETECTION_RULES_TEST_CONFIG
3
4 # 'bypass' and 'test_only' are mutually exclusive and will cause an error if both are specified.
5 #
6 # tests can be defined by their full name or using glob-style patterns with the following notation
7 # -- pattern:*rule*
8 # -- the patterns are case sensitive
9
10 unit_tests:
11 --# define tests to explicitly bypass, with all others being run
12 --#
13 --# to run all tests, set bypass to empty or leave this file commented out
14 --# bypass:
15 # -- tests.test_all_rules.TestRuleMetadata.test_event_dataset
16 # -- tests.test_all_rules.TestRuleMetadata.test_integration_tag
17 # -- tests.test_gh_workflows.TestWorkflows.test_matrix_to_lock_version_defaults
18 # -- pattern:*rule*
19 # -- pattern:*kuery*
20
21 --# define tests to explicitly run, with all others being bypassed
22 --#
23 --# to bypass all tests, set test_only to empty
24 --# test_only:
25 # -- tests.test_all_rules.TestRuleMetadata.test_event_dataset
26 # -- pattern:*rule*
27
28
29 # 'bypass' and 'test_only' are mutually exclusive and will cause an error if both are specified.
30 #
31 # both variables require a list of rule_ids
32 rule_validation:
33
34 --# bypass:
35 # ---- "34fde489-94b0-4500-a76f-b8a157cf9269"
36
37
38 --# test_only:
39 # ---- "34fde489-94b0-4500-a76f-b8a157cf9269"
40
```

## 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

```
(env) → detection-rules git:(main) python -m detection_rules create-rule test.toml --required-only
```

## DETECTION RULES

```
[metadata]
Rule type (query, saved_query, machine_learning, eql, esql, threshold, threat_match, new_terms): eql
maturity = "development"
author (required) (multi, comma separated): My Company
description (required): This is my custom rule.
name (required): Test rule for demo purposes
query (required): process where process.name: "bad.exe"
risk_score (required): 73 "My Company"
rule_id [7977bc1f-c420-4511-b154-f8a864bd9736] ("n/a" to leave blank) (required):
severity (required): high "eql"
type = "eql"

name = "Test rule for demo purposes"
risk_score = 73
rule_id = "7977bc1f-c420-4511-b154-f8a864bd9736"
severity = "high"
type = "eql"

query = '''
process where process.name: "bad.exe"
'''
```

## 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

```
[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

```
[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

```
react on v main [!?] via v3.12.2 (react-env) on Eric.forte took 4s
```

Mikaayenson commented on Dec 12, 2023

```
react on v main [!?] via v3.12.2 (react-env) on Eric.forte
> python -m react ci check-data --rule-repo endpoint-rules -v
```

1

- Pytests passed succesfully.

```
RULE: '78ae5dbd-477b-4ce7-a7f7-8c4b5e228df2' matched data from RTA_NAME: 'binary_execution_from_shared_memory'.
RULE: '7b9ddfc8-8ea8-45d5-b62f-3fbd142c8f08' matched data from RTA_NAME: 'cloud_eicar'.
```

## 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

```
(detection-rules-build) → detection-rules git:(main) x python -m detection_rules dev build-re
lease --update-version-lock
Loaded config file: /Users/stryker/workspace/Elastic/detection-rules/.detection-rules-cfg.json
```

## DETECTION RULES

```
[+] Building package 8.14
- 5 rules excluded from package
Rule changes detected!
- 3 changed rules
- 6 new rules
- 0 newly deprecated rules
run 'build-release --update-version-lock' to update version.lock.json and deprecated_rules.js
n
Rule changes detected!
- 3 changed rules
- 6 new rules
- 0 newly deprecated rules
Detailed changes:
A: a87a4e42-1d82-44bd-b0bf-d9b7f91fb89e, new version: 102
min_stack_version added: 8.3.0
A: 75ee75d8-c180-481c-ba88-ee50129a6aef, new version: 102
- min_stack_version added: 8.3.0
A: d49cc73f-7a16-4def-89ce-9fc7127d7820, new version: 102
min_stack_version added: 8.3.0
A: a8afdce2-0ec1-11ee-b843-f661ea17fbcd, new version: 3
- min_stack_version added: 8.3.0
A: bc8ca7e0-92fd-4b7c-b11e-ee0266b8d9c9, new version: 5
```

## 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

The screenshot displays a GitHub Pull Request (PR) titled "[New Rule] Alternate Data Stream Creation at Volume Root Directory #3517". The PR is open and shows 39 pull requests, 165 issues, and 3 actions. The PR is currently in a state where a review is required, with 2 pending reviewers. The PR description includes a question: "question - should we make this a bit wider to catch any root dir? could keep it efficient with regex". The PR also shows a suggested change to the file `rules/windows/defense_evasion_root_dir_ads_creation.toml`. The change is a diff showing the addition of a regex pattern to the `file.path` field. The PR also shows a comment from `brokensound77` asking "if so, update description" and a comment from `Samirbous` asking "to catch both file or process execution, `regex` is already case insensitive". The PR also shows a comment from `Samirbous` asking "to catch both file or process execution, `regex` is already case insensitive". The PR also shows a comment from `Samirbous` asking "to catch both file or process execution, `regex` is already case insensitive".

Code

Issues 165

Pull requests 39

Actions

Security

Insights

Settings

## [New Rule] Alternate Data Stream Creation at Volume Root Directory #3517

Open wOrk3r wants to merge 3 commits into `main` from `pr_2`

Conversation 6

Commits 3

Checks 13

Files changed 1

Changes from all commits

File filter

Conversations

Jump to

51 `rules/windows/defense_evasion_root_dir_ads_creation.toml`

Review required

At least 2 approving reviews are required by reviewers with write permissions

2 pending reviewers

All checks have passed

14 successful checks

Branch Version Status Checks / get-branches / list

Unit Tests / build (pull\_request) Successful in 9m

Branch Version Status Checks / branch-status-check

Branch Version Status Checks / branch-status-check

Branch Version Status Checks / branch-status-check

Branch Version Status Checks / branch-status-check

This branch is out-of-date with the base branch

Merge without waiting for requirements to be met (by Squash and merge)

You can also open this in GitHub

question - should we make this a bit wider to catch any root dir?  
could keep it efficient with regex

Suggested change

```
30 -   startsWith(file.path, "C:\\\\")
30 +   file.path regex= "^(A-Z):\\\\.*"
```

Commit suggestion

Add suggestion to batch

May need to test that

brokensound77 last week Member

if so, update description

Reply...

Resolve conversation

Comment on lines +29 to +30

Samirbous last week Member

to catch both file or process execution, `regex` is already case insensitive.

Suggested change

```
29 - file where host.os.type == "windows" and event.type == "creation" and
30 -   startsWith(file.path, "C:\\\\")
29 + any where host.os.type == "windows" and event.category in ("file", "process") and
30 +   {
31 +     (event.type == "creation" and file.path regex= "^(A-Z):\\\\.*") or
32 +     (event.type == "start" and process.executable regex= "^(A-Z):\\\\.*")
33 +   }
```

## Task

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

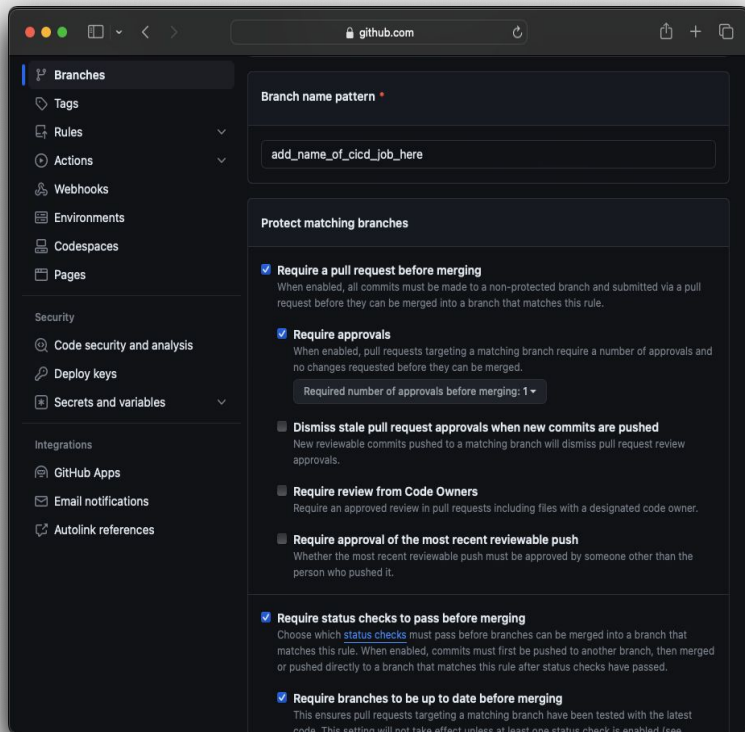
## Steps

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



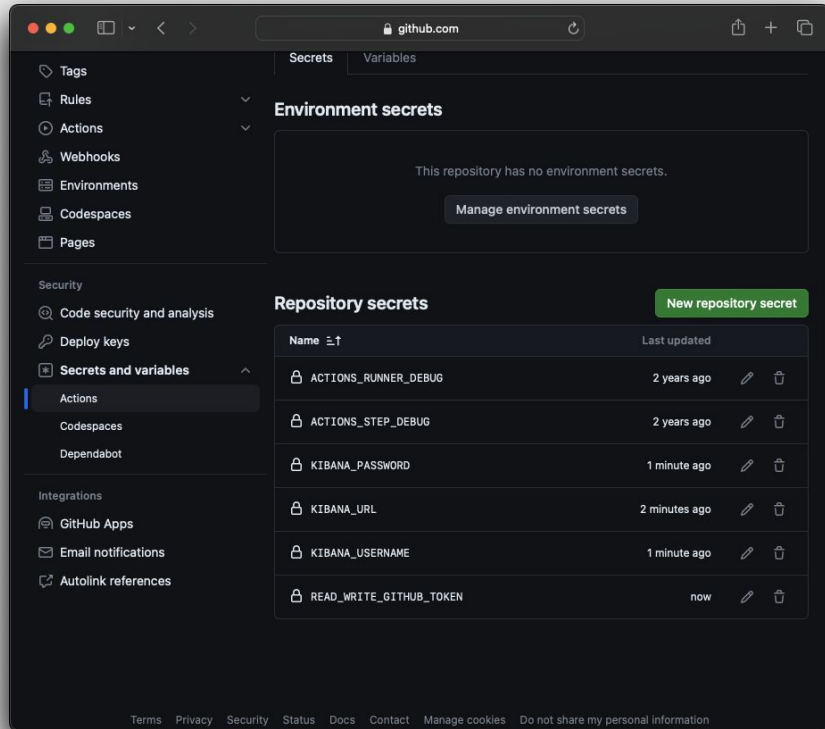
## Task

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



The screenshot shows the GitHub repository settings page for 'Secrets and variables'. The left sidebar contains navigation options: Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages, Security (Code security and analysis, Deploy keys, Secrets and variables), and Integrations (GitHub Apps, Email notifications, Autolink references). The 'Secrets and variables' section is expanded, showing 'Environment secrets' (empty) and 'Repository secrets' (containing a table of secrets).

Name	Last updated
ACTIONS_RUNNER_DEBUG	2 years ago
ACTIONS_STEP_DEBUG	2 years ago
KIBANA_PASSWORD	1 minute ago
KIBANA_URL	2 minutes ago
KIBANA_USERNAME	1 minute ago
READ_WRITE_GITHUB_TOKEN	now

## 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

```
1 name: CI/CD Per-PR Sync Workflow
2
3 on:
4   pull_request:
5     branches: [ "*" ]
6     types: [ opened, synchronize, reopened, labeled, unlabeled ]
7
8 jobs:
9   pr-check:
10    runs-on: ubuntu-latest
11    env: --
12
13    steps:
14    > - name: Checkout Repository --
15
16    > - name: Set up Python 3.12 --
17
18    > - name: Install Dependencies --
19
20    > - name: Run Unit Tests --
21
22    > - name: Check for Telemetry Review Label --
23
24    > - name: Telemetry Check (Optional) --
25
26    > - name: Remove Telemetry Review Label --
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58 > - name: Remove Telemetry Review Label --
59
60
61
62
63
64
65
66
67
68
69
70
```

## 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

```
1 name: Manual Dispatch Sync Workflow
2
3 on:
4   workflow_dispatch:
5   > inputs: --
6
7
8
9
10
11 jobs:
12   manual-dispatch-sync:
13     runs-on: ubuntu-latest
14   > env: --
15
16
17
18   > steps:
19     - name: Checkout Repository --
20
21
22     - name: Set up Python 3.12 --
23
24
25     - name: Install Dependencies --
26
27
28     - name: Export and Import Rules if flag is true --
29
30
31     - name: Update Version Lock --
32
33
34     - name: Create Pull Request --
35
36
37     - 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

```
1 name: Scheduled Sync Workflow
2
3 on:
4   - schedule:
5     - # Schedule to run at a specific time, e.g., at 01:00 every day
6     - cron: '0 1 * * *'
7
8 jobs:
9   - scheduled-sync:
10    - runs-on: ubuntu-latest
11    - env:
12      - CUSTOM_RULES_DIR: ${ secrets.CUSTOM_RULES_DIR }
13
14    - steps:
15      - name: Checkout Repository--
16
17      - name: Set up Python 3.12--
18
19      - name: Install Dependencies--
20
21      - name: Export Rules from Kibana--
22
23      - name: Import Rules to Local Repo--
24
25      - name: Update Version Lock--
26
27      - name: Check for Existing Sync PR--
28
29      - name: Create or Update Pull Request--
30
31      - 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

```
1 name: Push to Production Sync
2
3 on:
4   - push:
5     - branches:
6       - main
7     - paths:
8       - 'version.lock.json'
9     - workflow_dispatch:
10
11 jobs:
12   - sync-to-production:
13     - runs-on: ubuntu-latest
14     - env: ~
15
16     - steps:
17       - name: Checkout Repository ~
18       - name: Set up Python 3.12 ~
19       - name: Install Dependencies ~
20       - name: Export Rules from Kibana ~
21       - name: Import Rules to Local Repo ~
22       - name: Verify Version Lock Consistency ~
23       - name: Export Rules from Repo to NDJSON ~
24       - 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

<p> Exploring the depths of DaC capabilities within the CLI to enhance your detection strategies. </p>

```
(detection-rules-build) + detection-rules git:(main) * python -m detection_rules dev build-release --generate-navigator  
Loaded config file: /Users/stryker/workspace/Elastic/detection-rules/.detection-rules-cfg.json
```

# DETECTION RULES

```
[+] Building package 8.14  
- 5 rules excluded from package  
Package saved to: /Users/stryker/workspace/Elastic/detection-rules/releases/8.14  
loaded security_detection_engine manifests from the following package versions: ['8.13.1', '8.12.6', '8.12.5', '8.12.4', '8.12.3', '8.12.2', '8.12.1', '8.11.10', '8.11.9', '8.11.8', '8.11.7', '8.11.6', '8.11.5', '8.11.4', '8.11.3', '8.11.2', '8.11.1', '8.10.13', '8.10.12', '8.10.11', '8.10.10', '8.10.9', '8.10.8', '8.10.7', '8.10.6', '8.10.5', '8.10.4', '8.10.3', '8.10.2', '8.10.1', '8.9.15', '8.9.14', '8.9.13', '8.9.12', '8.9.11', '8.9.10', '8.9.9', '8.9.8', '8.9.7', '8.9.6', '8.9.5', '8.9.4', '8.9.3', '8.9.2', '8.9.1', '8.8.15', '8.8.14', '8.8.13', '8.8.12', '8.8.11', '8.8.10', '8.8.9', '8.8.8', '8.8.7', '8.8.6', '8.8.5', '8.8.4', '8.8.3', '8.8.2', '8.8.1', '8.7.13', '8.7.12', '8.7.11', '8.7.10', '8.7.9', '8.7.8', '8.7.7', '8.7.6', '8.7.5', '8.7.4', '8.7.3', '8.7.2', '8.7.1', '8.6.10', '8.6.9', '8.6.8', '8.6.7', '8.6.6', '8.6.5', '8.6.4', '8.6.3', '8.6.2', '8.6.1', '8.5.8', '8.5.7', '8.5.6', '8.5.5', '8.5.4', '8.5.3', '8.5.2', '8.5.1', '8.4.5', '8.4.4', '8.4.3', '8.4.2', '8.4.1', '8.3.4', '8.3.3', '8.3.2', '8.3.1', '8.2.1', '8.1.1', '1.0.2', '1.0.1']  
[+] Adding historical rules from 8.13.1 package  
- sha256: 53b7c2a71c7d1ce9405660eeab25836a271ace1a0f68d1e4532ebad13a17bde  
- 1080 rules included  
(detection-rules-build) + detection-rules git:(main) * |
```

## 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)**

```
(detection-rules-build) + detection-rules git:(main) * python -m detection_rules dev integrations build-manifests -i endpoint
Loaded config file: /Users/stryker/workspace/Elastic/detection-rules/.detection-rules-cfg.json
```

## DETECTION RULES

```
loading rules to determine all integration tags
loaded endpoint manifests from the following package versions: ['8.13.0', '8.12.0', '8.11.1', '8.11.0', '8.10.2', '8.10.1', '8.10.0', '8.9.1', '8.9.0', '8.8.0', '8.7.1', '8.7.0', '8.6.1', '8.6.0', '8.5.0', '8.4.1', '8.4.0', '8.3.0', '8.2.0', '1.5.0', '1.4.1', '1.4.0', '1.3.0', '1.2.2', '1.2.1', '1.2.0', '1.1.1', '1.1.0', '1.0.0']
final integrations manifests dumped: /Users/stryker/workspace/Elastic/detection-rules/etc/integration-manifests.json.gz
(detection-rules-build) + detection-rules git:(main) * |
```

## 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`

```
(detection-rules-build) + detection-rules git:(main) x python -m detection_rules kibana search-  
h-alerts  
Loaded config file: /Users/stryker/workspace/Elastic/detection-rules/.detection-rules-cfg.json
```

## DETECTION RULES

host	rule		kibana
hostname	name	status	alert
			original_time
stryker-macos-testing.local	test	active	2024-03-22T18:36:41.791Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:41.791Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:41.791Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:42.980Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:42.980Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:43.794Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:53.000Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:53.000Z
stryker-macos-testing.local	test	active	2024-03-22T18:36:53.809Z
stryker-macos-testing.local	test	active	2024-03-22T18:37:03.027Z
stryker-macos-testing.local	test	active	2024-03-22T18:37:03.027Z
stryker-macos-testing.local	test	active	2024-03-22T18:37:03.817Z
stryker-macos-testing.local	test	active	2024-03-22T18:37:04.289Z
stryker-macos-testing.local	test	active	2024-03-22T18:37:13.055Z
stryker-macos-testing.local	test	active	2024-03-22T18:37:13.055Z
stryker-macos-testing.local	test	active	2024-03-22T18:37:13.295Z

## 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`**



```
(env) 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 | 4000fed3-8ac4-45bf-8545-bae448fd3864 | Persistence via a Hidden Plist Filename
defensive_evasion_safari_modification.py | macos | 092d084f-84ac-485d-8a55-76d9e90715f | Creation of Hidden Launch Agent or Daemon
defensive_evasion_safari_modification.py | macos | 3960113f-433c-4223-a8ee-bc03304738d | Modification of Safari Settings via Defaults Command
defensive_evasion_safari_modification.py | macos | 6482253d-f468-45ea-a503-d3d7d613310e | Modification of Safari Settings via Defaults Command
tar_dylib.py | macos | 62cc9c74-5440-4237-a050-ea5db33de33d | Non-Native Dylib Extracted into New Directory
execscript_suspicious_cmdline.py | macos | 7092549a-302a-4f06-984a-eca807f6a25 | Suspicious Apple Script Execution
privilege_escalation_tcc_bypass.py | macos | 8446036d-9c4d-4646-8d61-979c80d8df | Potential Privilege Escalation via TCC bypass with Fake TCC.db
plist_buddy_file_modification.py | macos | 901f0c30-a7c5-40a5-8063-a50c6714432f | Suspicious Property List File Creation or Modification
curl_sun_payload.py | macos | 8c42c86d-c282-44ca-b388-924e267b6244 | Payload Downloaded by Process Running in Suspicious Directory
sqlite_db_evasion.py | macos | 049322cd-9706-4529-921e-8a13353c491b | Reading or Modifying Downloaded Files Database via SQLite Utility
javascript_payload.py | macos | 871f0c30-a7c5-40a5-8063-a50c6714632f | Download and Execution of JavaScript Payload
kernel_ext_agent_unload.py | macos | 0412f09b-2a06-49ff-a073-8eb313c2a930 | Attempt to Unload Elastic Endpoint Security Kernel Extension
704810f4-27f6-4f26-8083-5066f069e24 | macos | 704810f4-27f6-4f26-8083-5066f069e24 | Attempt to Unload Elastic Endpoint Security Kernel Extension
711a2901-e8a0-4a29-991b-3d68188cb789 | macos | 711a2901-e8a0-4a29-991b-3d68188cb789 | Operating System Security Updates Disabled
disable_os_security_updates.py | macos | f683cdf-0818-4801-b066-1936a0a6c8e5 | SoftwareUpdate Preferences Modification
office_app_execution.py | macos | 04021f93-19d3-479f-ac3c-79e386da565a | Initial Access or Execution via Microsoft Office Application
keychain_dump.py | macos | 5493446d-aaef-4e89-9a0c-7a0b489f971 | Keychain Dump via native Security tool
keychain_dump.py | macos | 5650cac5-75ba-4c82-9a13-add25353471c | Dumping of Keychain Content via Security Command
systemkey_credential_access.py | macos | 7d3f988f-2111-4e5f-9787-9ede78294a0b | Suspicious SystemKey Access via Command Line
systemkey_credential_access.py | macos | 073991f2-3089-413d-b397-act848e2c261 | SystemKey Access via Command Line
systemsetup_ssh_enable.py | macos | 5ae4e6f8-d1bf-40fa-96ba-e29645e1e46c | Remote SSH Login Enabled via systemsetup Command
kcc_kerberos_dump.py | macos | dc8fa949-efb4-45d1-ba1a-9472323f7746 | Potential Access to Kerberos Cached Credentials
kcc_kerberos_dump.py | macos | 0a88231f-e2ab-491c-8f6b-04746d026cfe | Kerberos Cached Credentials Dumping
networksetup_vpn.py | macos | 1350ac0b-3a9b-466a-ac0b-63433c07981a | Virtual Private Network Connection Attempt
empire_stager.py | macos, linux | b79747fe-82ff-4743-9e07-1c090101f0ea | Empire Stager Execution
special_chars_zip_file.py | macos | d22a856b-d2ad-4c04-bf7b-54787473669a | Potential Microsoft Office Sandbox Evasion
dmg_create_in_tmp.py | macos | f8c67ed-421b-4071-be47-080b9958f07 | 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`

```
(detection-rules-build) → detection-rules git:(main) ✗ python -m detection_rules es collect-events A9ADA181-ABC3-55F3-BDCB-FBD666D47FDF
Loaded config file: /Users/stryker/workspace/Elastic/detection-rules/.detection-rules-cfg.json
```

## DETECTION RULES

```
Press any key once detonation is complete ...
787 events saved to: /Users/stryker/workspace/Elastic/detection-rules/collections/A9ADA181-ABC3-55F3-BDCB-FBD666D47FDF/20240322T133116L/endpoint.ndjson
(detection-rules-build) → detection-rules git:(main) ✗
```



## 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)

```
+ detection-rules git:(main) make deps cli
Installing kql and kibana packages...
~/env/detection-rules-build/bin/pip install lib/kql lib/kibana
Processing ./lib/kql
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Processing ./lib/kibana
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: eql==0.9.19 in ./env/detection-rules-build/lib/python3.12/site-packages (from detection-rules-kql==0.1.6) (0.9.19)
Requirement already satisfied: lark-parser>=0.12.0 in ./env/detection-rules-build/lib/python3.12/site-packages (from detection-rules-kql==0.1.6) (0.12.0)
Requirement already satisfied: requests<3.0,>=2.25 in ./env/detection-rules-build/lib/python3.12/site-packages (from detection-rules-kibana==0.1.0) (2.31.0)
Requirement already satisfied: elasticsearch==8.12.1 in ./env/detection-rules-build/lib/python3.12/site-packages (from detection-rules-kibana==0.1.0) (8.12.1)
Requirement already satisfied: elastic-transport<9,>=8 in ./env/detection-rules-build/lib/python3.12/site-packages (from elasticsearch==8.12.1->detection-rules-kibana==0.1.0) (8.12.0)
Requirement already satisfied: charset-normalizer<4,>=2 in ./env/detection-rules-build/lib/python3.12/site-packages (from requests<3.0,>=2.25->detection-rules-kibana==0.1.0) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in ./env/detection-rules-build/lib/python3.12/site-packages (from requests<3.0,>=2.25->detection-rules-kibana==0.1.0) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in ./env/detection-rules-build/lib/python3.12/site-packages (from requests<3.0,>=2.25->detection-rules-kibana==0.1.0) (2.2.1)
Requirement already satisfied: certifi>=2017.4.17 in ./env/detection-rules-build/lib/python3.12/site-packages (from requests<3.0,>=2.25->detection-rules-kibana==0.1.0) (2024.2.2)
Building wheels for collected packages: detection-rules-kql, detection-rules-kibana
```

## 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

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

# 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



THANK YOU!

