



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

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.



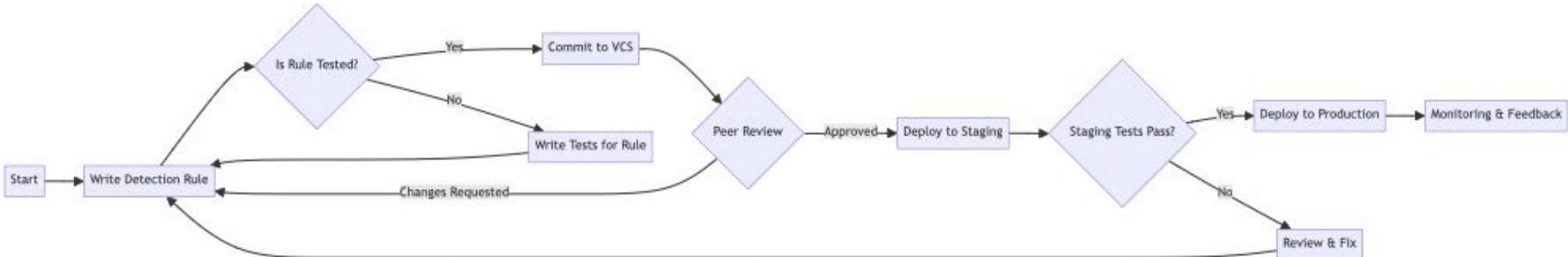
DevOps Engineers

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.



BUT WHY DO WE NEED THIS?

Ever-Growing Rule Sets

Broader Adoption of Automation

Drive Security Team Towards Maturity

Expanding Threat Landscape

Compliance and Governance



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.

<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

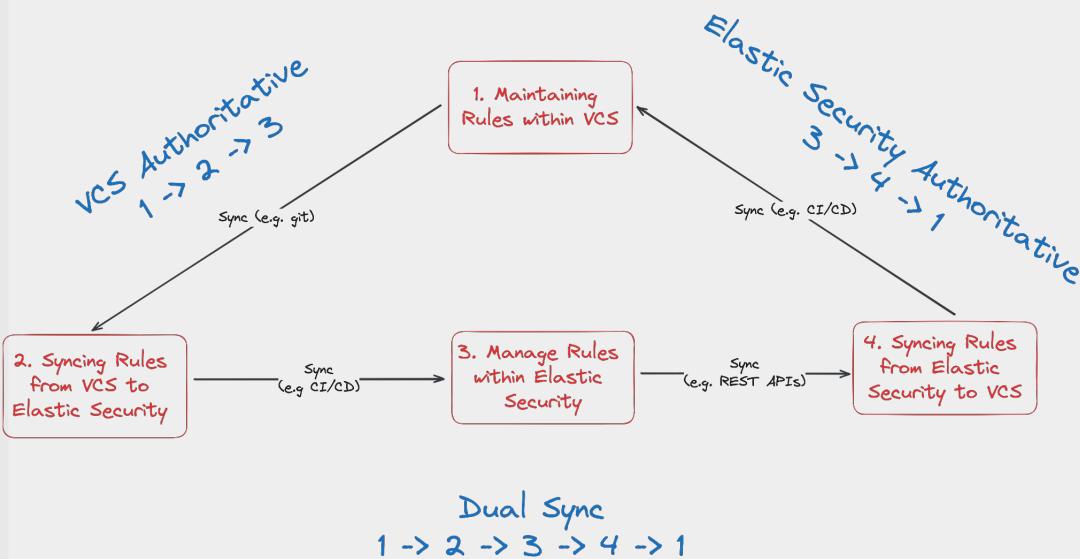
What are the concepts and components and how do they relate to each other?

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

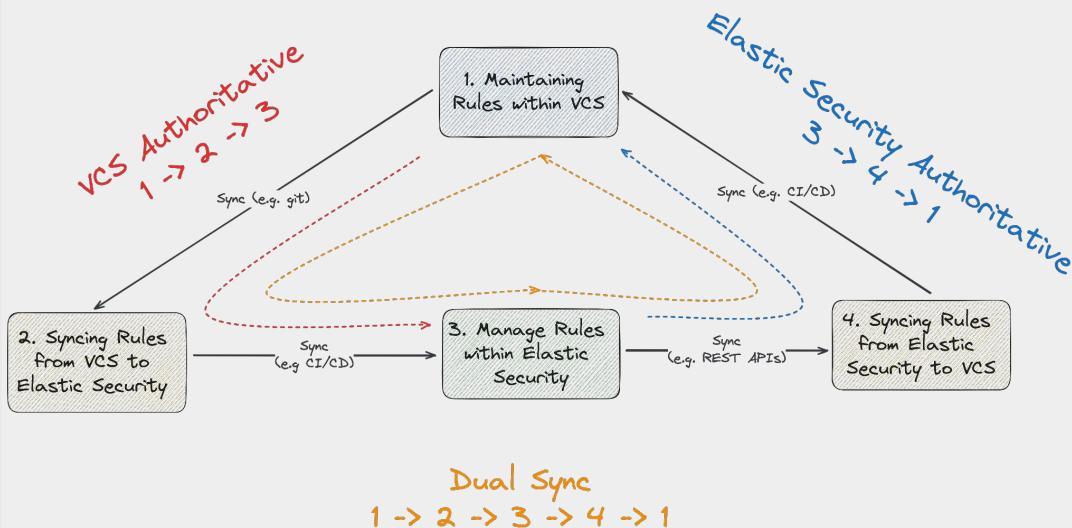
- VCS as authoritative
- Platform as authoritative
- Dual sync between VCS and the platform



Hierarchy

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

CORE COMPONENTS FLOW BY GOVERNANCE



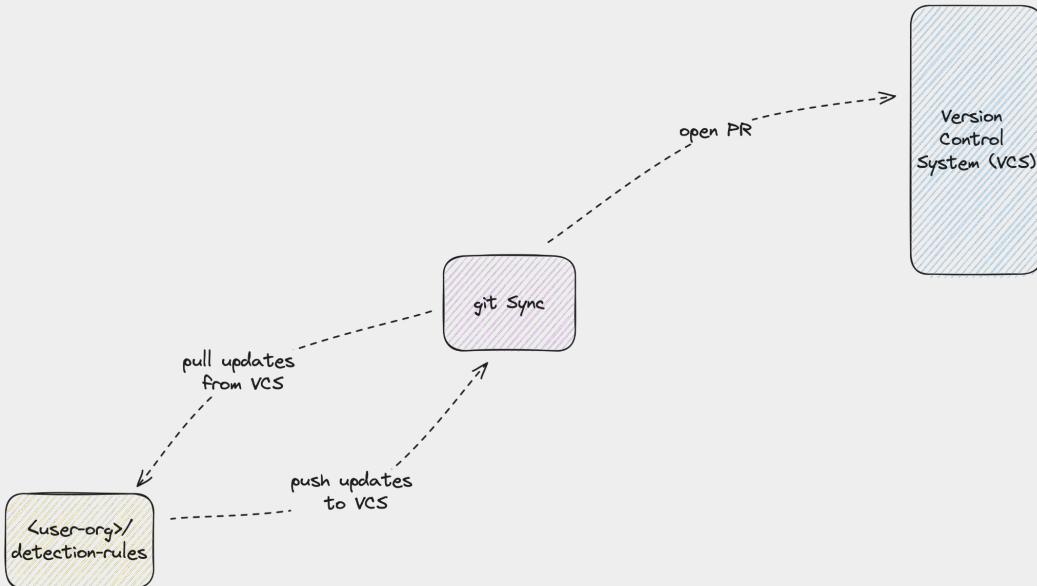
SYNCING OPTIONS



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

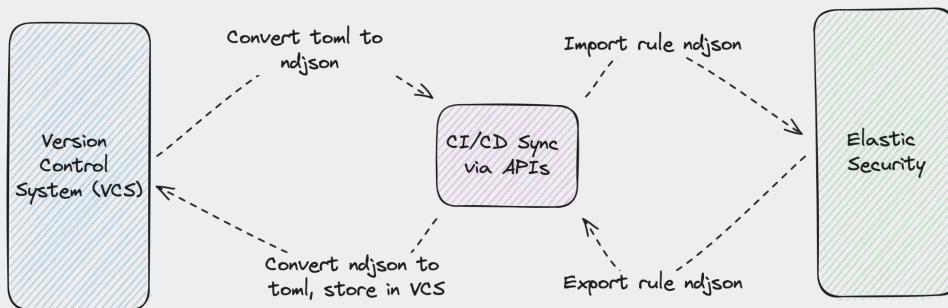
- Dedicated repo to store detection rules and collaborate
- Local schema and query validation tools

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.

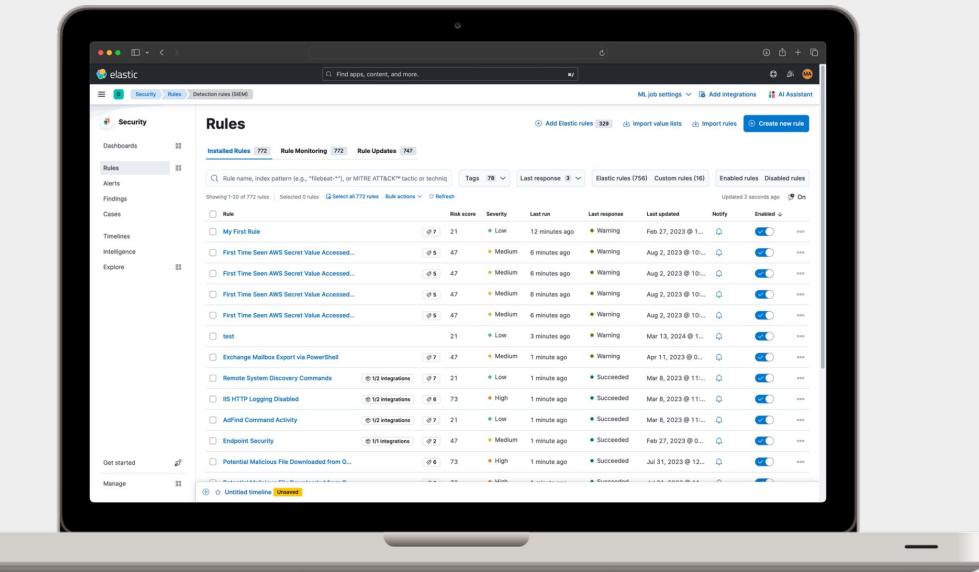
Requirements

- 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.

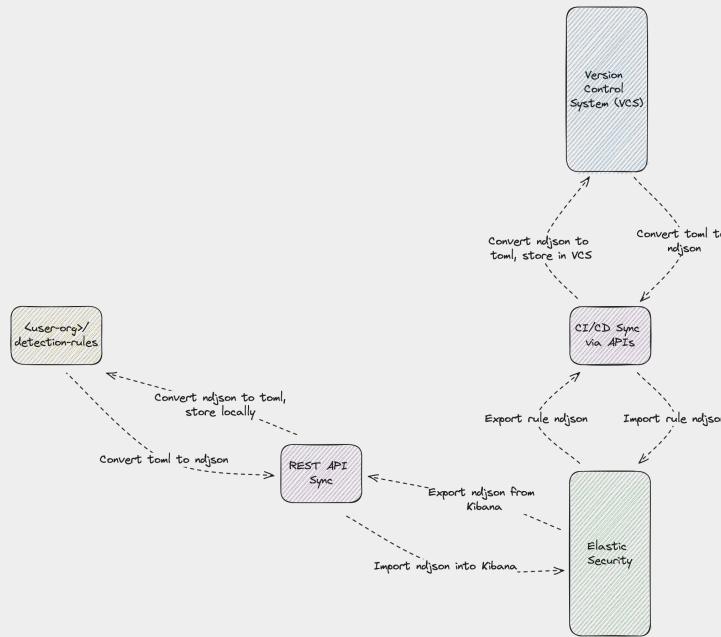
Requirements

- 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.

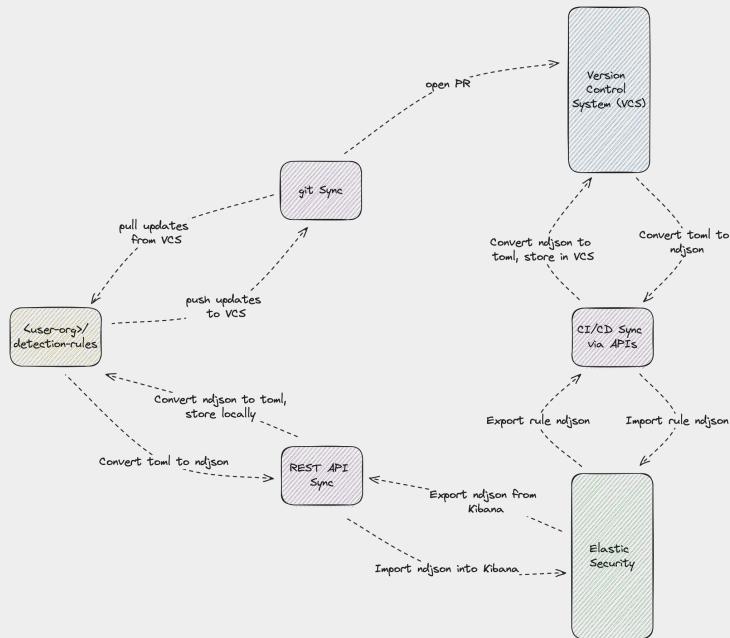
Requirements

- 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.com

README License Security

python 3.12+ Unit Tests passing chat #security-detection-rules

ATT&CK Navigator

Detection Rules

Detection Rules is the home for rules used by Elastic Security. This repository is used for the development, maintenance, testing, validation, and release of rules for Elastic Security's Detection Engine.

This repository was first announced on Elastic's blog post, [Elastic Security opens public detection rules repo](#). For additional content, see the accompanying webinar, [Elastic Security: Introducing the public repository for detection rules](#).

GITHUB/ELASTIC/ DETECTION-RULES

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



github.com

README **License** **Security**

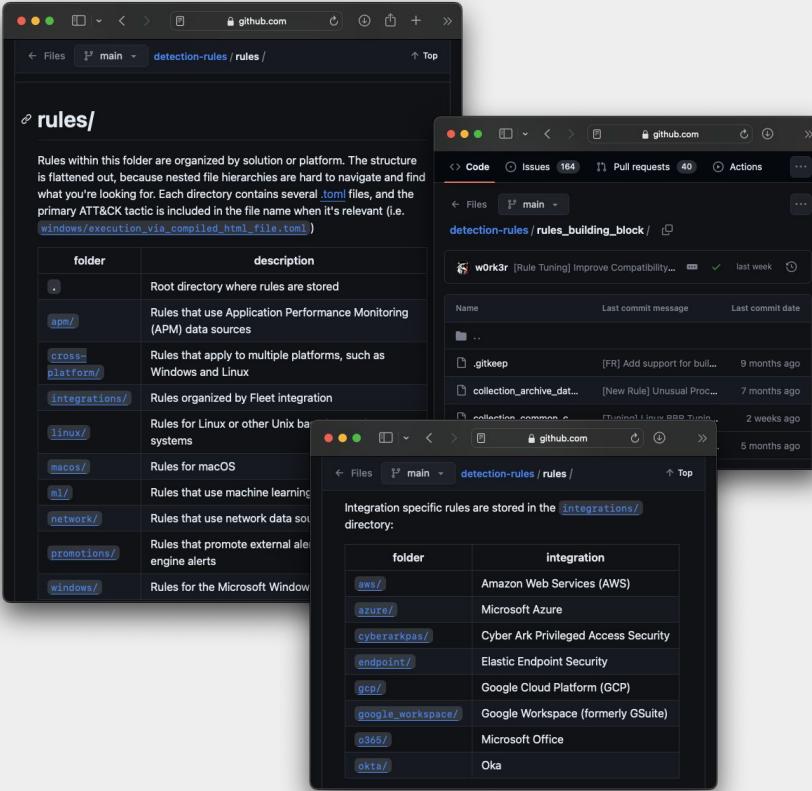
Overview of this repository

Detection Rules contains more than just static rule files. This repository also contains code for unit testing in Python and integrating with the Detection Engine in Kibana.

folder	description
<code>detection_rules/</code>	Python module for rule parsing, validating and packaging
<code>etc/</code>	Miscellaneous files, such as ECS and Beats schemas
<code>kibana/</code>	Python library for handling the API calls to Kibana and the Detection Engine
<code>kql/</code>	Python library for parsing and validating Kibana Query Language
<code>rta/</code>	Red Team Automation code used to emulate attacker techniques, used for rule testing
<code>rules/</code>	Root directory where rules are stored
<code>rules_building_block/</code>	Root directory where building block rules are stored
<code>tests/</code>	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 screenshot shows a GitHub repository page for 'detection-rules / rta'. The title is 'Red Team Automation'. Below the title, there are buttons for 'python 3.7+' and 'chat #security-detection-rules'. A text block states: '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.' Below this, a terminal session shows the usage of the 'rta' script:

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

positional arguments:
  ttp_name

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

A note below the terminal session says: '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
```

At the bottom, a note states: '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>



The image shows two side-by-side code editors. The top editor has a dark theme and displays a script titled '# KQL Library Example Untitled-1'. It contains the following Python code:

```
# KQL Library Example
from kql import parse, to_dsl

# Sample KQL query
kql_query = 'process.name: "cmd.exe" and process.args: "-k"'

# Parse the KQL query
parsed_query = parse(kql_query)

# Convert the parsed KQL query to Elasticsearch Query DSL
dsl_query = to_dsl(parsed_query)

print(dsl_query)
```

The bottom editor also has a dark theme and displays a script titled '# Kibana API client example to list all Untitled-1'. It contains the following Python code:

```
# Kibana API client example to list all custom detection rules
from kibana import Kibana, RuleResource

# Initialize the Kibana client
kibana_client = Kibana(kibana_url="http://localhost:5601",
                      kibana_username="your_username", kibana_password="your_password")

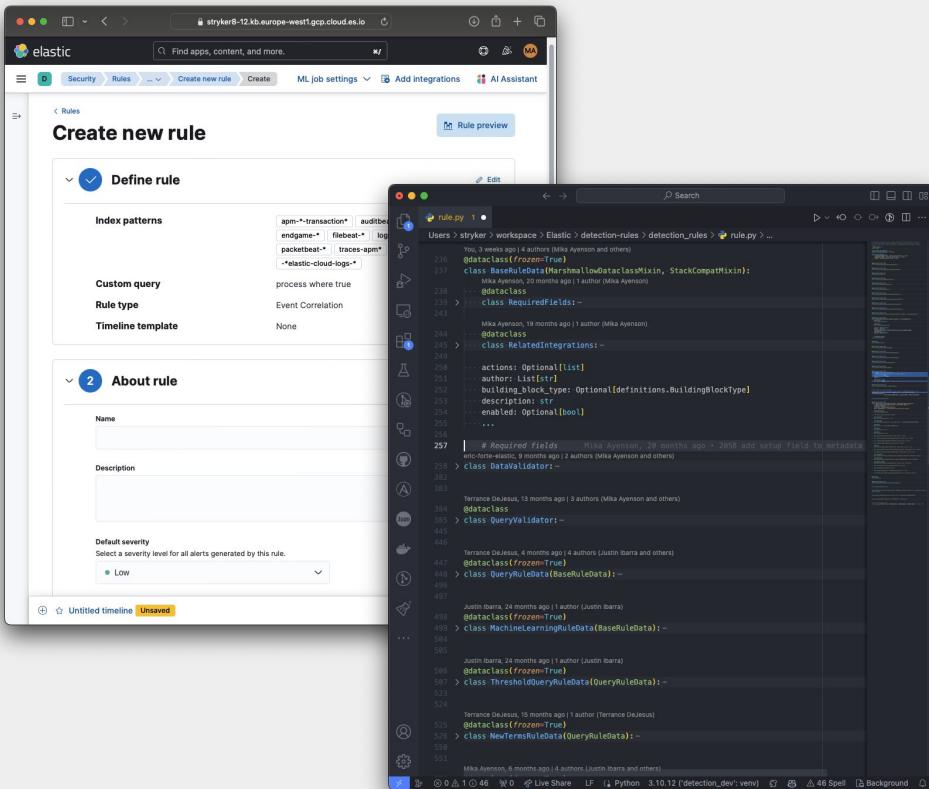
# Authenticate
kibana_client.login("your_username", "your_password")

# List all custom detection rules
custom_rules = RuleResource.find_custom()

for rule in custom_rules:
    print(rule.id, rule.name)
```

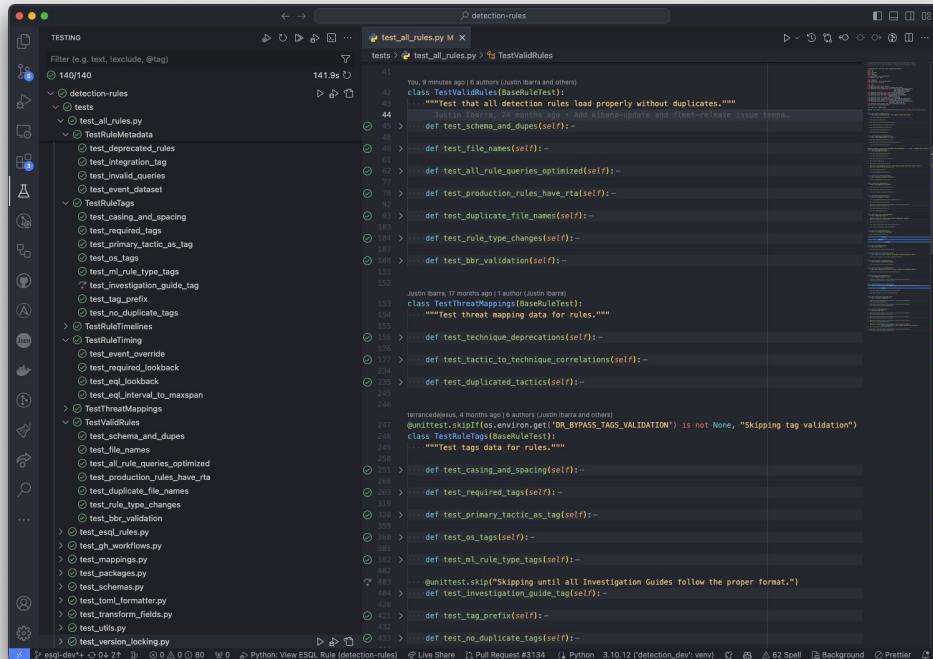
KQL AND KIBANA PYTHON LIBRARIES

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



DATACLASS AND MARSHMALLOW SCHEMA VALIDATION

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



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.yaml
        [ ] test_cli.bash
        [ ] test_remote_cli.bash
        { } test_toml.json
        { } version.lock.json
```

MISCELLANEOUS FILES FOR CUSTOM CONFIGURATION

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



The image shows three side-by-side screenshots of GitHub Actions pipelines for the `elastic/detection-rules` repository. Each screenshot displays a different stage of the CI/CD process:

- Left Screenshot:** Shows the initial pipeline stages: `lock-versions`, `pr`, and `Build Security Docs`. The `pr` stage succeeded 2 weeks ago in 1h 29m 29s. The `Build Security Docs` stage succeeded 2 weeks ago in 4m 21s.
- Middle Screenshot:** Shows the `Release Docs` stage, which includes `Release Fleet` and `Build Security Docs`.
- Right Screenshot:** Shows the `Release Fleet` stage, which includes `Build package and create PR to integrations`.

In all three screenshots, the pipeline steps are listed on the left, and the status of each step is indicated by a green checkmark and a small circular icon.

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 ALPHA EXAMPLE

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

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



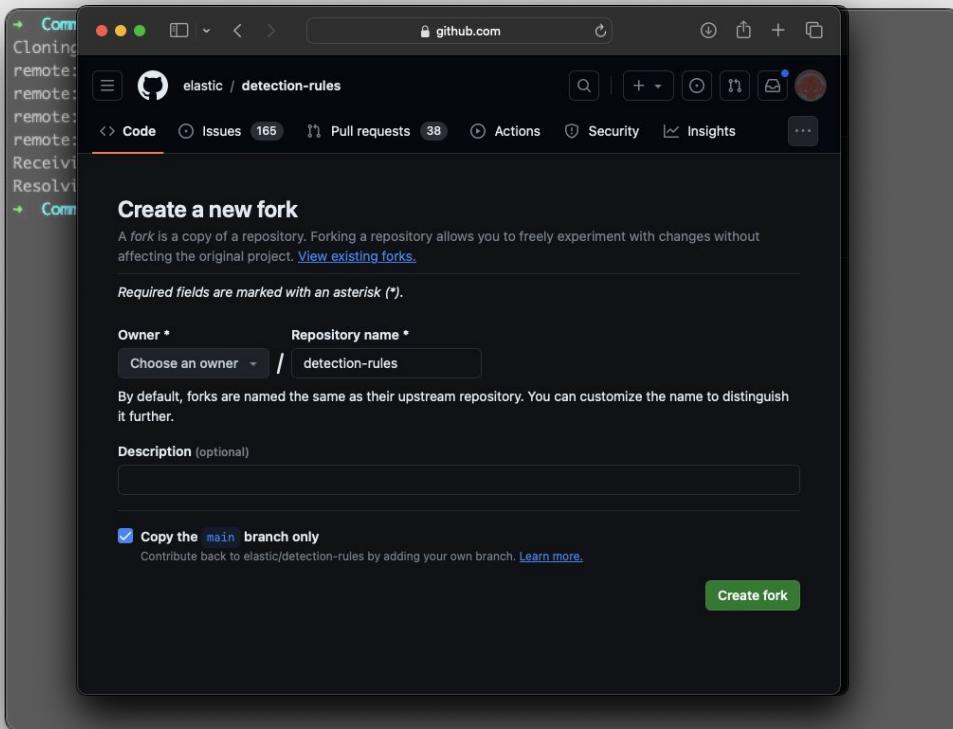
Task

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

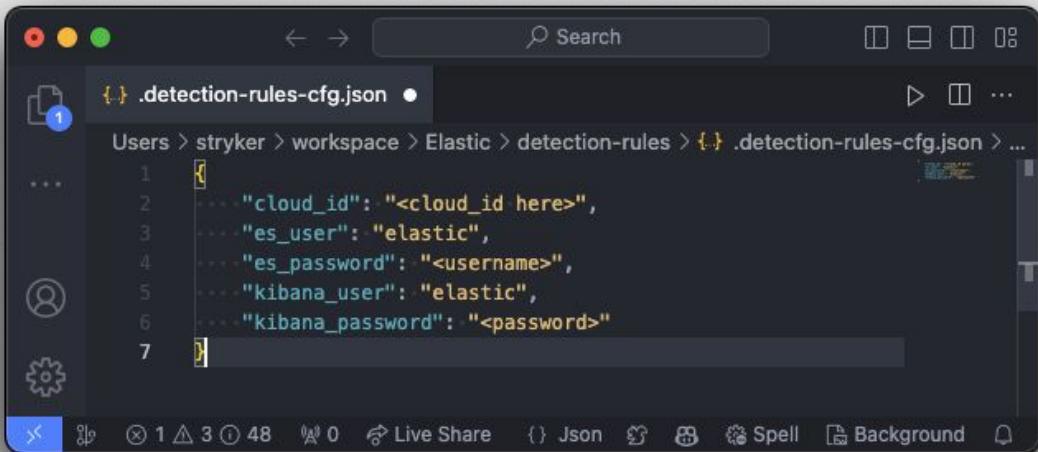
Within the terminal, install the Python dependencies required to use the CLI and test to make sure it's available.

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



```
{ } .detection-rules-cfg.json ●  
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 } ]  
...
```

Live Share {} Json ⚙️ 🔍 Spell Background

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



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
[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
rule_id [7977bc1f-c420-4511-b154-f8a864bd9736] ("n/a" to leave blank) (required):
severity (required): high
(env) + detection-rules git:(main) x
10 name = "Test rule for demo purposes"
11 risk_score = 73
12 rule_id = "7977bc1f-c420-4511-b154-f8a864bd9736"
13 severity = "high"
14 type = "eql"
15
16 query = ''
17 process where process.name: "bad.exe"
18 ...
19
20 |
```

The terminal shows the execution of the command `python -m detection_rules create-rule test.toml --required-only`. The output displays a TOML configuration file named `test.toml` with the following content:

```
[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
rule_id [7977bc1f-c420-4511-b154-f8a864bd9736] ("n/a" to leave blank) (required):
severity (required): high
(env) + detection-rules git:(main) x
10 name = "Test rule for demo purposes"
11 risk_score = 73
12 rule_id = "7977bc1f-c420-4511-b154-f8a864bd9736"
13 severity = "high"
14 type = "eql"
15
16 query = ''
17 process where process.name: "bad.exe"
18 ...
19
20 |
```

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 ✘ main [!?] via ✘ v3.12.2 (react-env) on ⚡eric.forte took 4s

Mikaayenson commented on Dec 12, 2023

react on ✘ main [!?] via ✘ v3.12.2 (react-env) on ⚡eric.forte

> python -m react ci check-data --rule-repo endpoint-rules -v

• Pytests passed successfully.

RULE: '78ae5dbd-477b-4ce7-a7f7-8c4b5e228df2' matched data from RTA_NAME: 'binary_execution_from_shared_memory'.
RULE: '7b9ddfc8-8ea8-45d5-b62f-3fd142c8f0b' 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) ✘ python -m detection_rules dev build-release --update-version-lock
Loaded config file: /Users/stryker/workspace/Elastic/detection-rules/.detection-rules-cfg.json
[+] 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.json
n
Rule changes detected!
- 3 changed rules
- 6 new rules
- 0 newly deprecated rules
Detailed changes:
A: a87a4e42-1d82-4bd8-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-89fc-9fc7127d7820, new version: 102
    min_stack_version added: 8.3.0
A: a8afdc2-0ec1-1lee-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 shows a GitHub pull request interface for a file named `rules/windows/defense_evasion_root_dir_ads_creation.toml`. The pull request is titled "[New Rule] Alternate Data Stream Creation at Volume Root Directory #3517". The commit message is "w0rk3r wants to merge 3 commits into `main` from `mr_2`". The pull request has 165 issues, 39 pull requests, and 1 action. It includes security, insights, and settings options.

The main content area shows a diff between two versions of the file. A red circle highlights a section of code where the reviewer suggests changing the regex pattern to catch any root directory. The suggested change is:

```
30 - startsWith(file.path, "C:\\;")  
30 + file.path regex= "[a-zA-Z]:\\;.*"
```

Below the code, there's a note: "May need to test that". A comment from user `brokenSound77` says: "if so, update description". Another comment from user `Samirous` says: "to catch both file or process execution, `regex=` is already case insensitive." The pull request has 2 pending reviewers and all checks have passed.

At the bottom, there are buttons for "Merge without waiting for requirements to be met (by [])" and "Squash and merge". A note says "You can also open this in GitHub".

Task

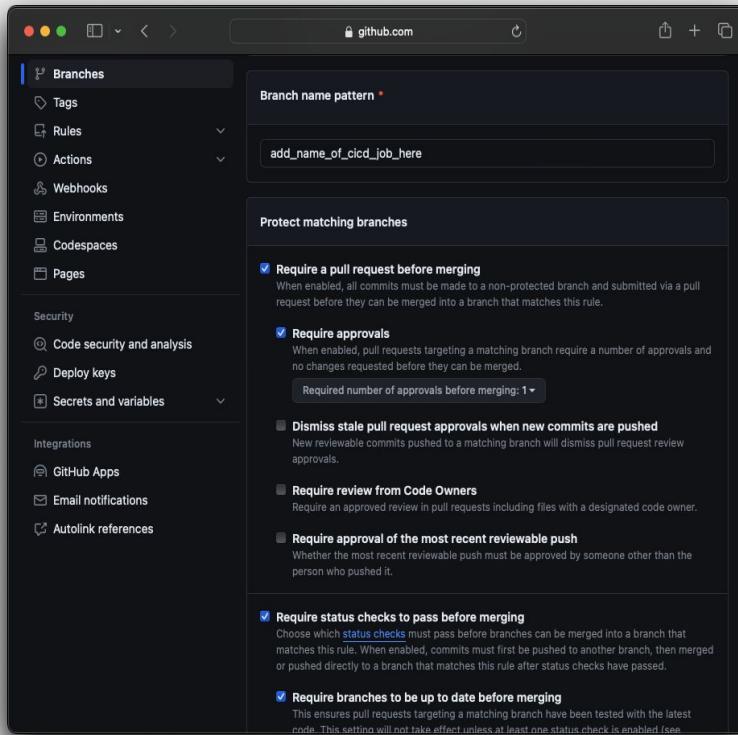
Collaborate with the internal team on the new and tuned rules to review / improve detection rules.

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



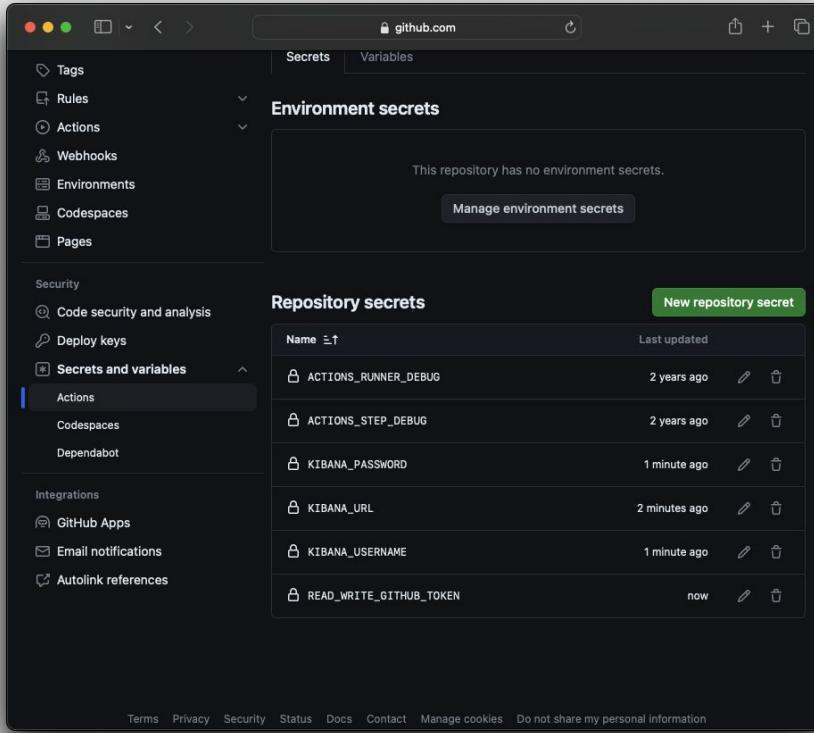
Task

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

```
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
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  jobs:
8    manual-dispatch-sync:
9      runs-on: ubuntu-latest
10     env:
11
12     steps:
13       - name: Checkout Repository
14
15       - name: Set up Python 3.12
16
17       - name: Install Dependencies
18
19       - name: Export and Import Rules if flag is true
20
21       - name: Update Version Lock
22
23       - name: Create Pull Request
24
25       - 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
19       - name: Set up Python 3.12
20
21       - name: Install Dependencies
22
23       - name: Export Rules from Kibana
24
25       - name: Import Rules to Local Repo
26
27       - name: Verify Version Lock Consistency
28
29       - name: Export Rules from Repo to NDJSON
30
31       - 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-production.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: 53b7c2a7c7d1ce9405660eeab25836a271acee1a0f68d1e4532ebad13a17bde
- 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 integrat  
ions 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.  
'2.0', '1.1.2.1', '1.1.2.0', '1.1.1.0', '1.1.0.0', '1.0.1.0', '1.0.0.0']  
final integrations manifests dumped: /Users/stryker/workspace/Elastic/detection-rules/detectio  
n_rules/etc/integration-manifests.json.gz  
(detection-rules-build) + detection-rules git:(main) ✘
```

DETECTION RULES

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) ✘ python -m detection_rules kibana search-alerts
Loaded config file: /Users/stryker/workspace/Elastic/detection-rules/.detection-rules-cfg.json
```

DETECTION RULES

host hostname	rule name	kibana status	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`

name	platforms	rule_id	rule_name
hidden_plist.py	macos	409f9ed3-8af4-43bf-834d-bae4ff6f8e84	Persistence via a Hidden Plist File Name
		0926bbaf-84cc-485d-bb55-7d98d900675f	Creation of Hidden Launch Agent or Daemon
defensive_evasion_safari_modification.py	macos	39ae1138-243c-4215-a8ed-be303204e70d	Modification of Safari Settings via Defaults Command
		6482255d-f468-45e5-a0b3-d5a7de13310d	Modification of Safari Settings via Defaults Command
tar_dylib.py	macos	62c5cf4-5440-4237-8505-ea0d83de83d2	Non-Native Dylib Extracted Into New Directory
osqueryi_dylib_extractor_c cmdline.py	macos	72d5d44c-502a-4f80-984c-cd85a0592d25	Suspicious Apple Script Execution
privilege_elevation_tcc_bypass.py	macos	14485a2a-3a00-4a00-9300-030000000000	Privileged Application Executed via TCC bypass with fake TCC.db
plist_studdy_file_modification.py	macos	9019fc30-07c5-4002-9001-030000000000	Suspicious Property List File Creation or Modification
plistsus_payload.py	macos	8c4242cd-c282-44cc-b308-92426765624	Payload Downloaded by Process Running in Suspicious Directory
sqlite_db_evasion.py	macos	b8f52cd-5f1a-453d-921d-bd1b353d6c03	Reading or Modifying Downloaded Files Database via SQLite Utility
javascript_payload.py	macos	874010c0-07c5-40c5-8801-030000000000	Download and Execution of JavaScript Payload
kernelext_agent_unload.py	macos	9d10a25b-4029-4001-9072-0e0310310000	Attempts to Unload Elastic Configuration Security Kernel Extension
disable_os_security_updates.py	macos	741ad90d-e840-4d20-b91b-3d68138c0783	Operating System Security Updates Disabled
office_app_execution.py	macos	f633ccdf-0018-4801-b066-1934de6e8c83	SoftwareUpdate Preferences Modification
keychain_dump.py	macos	649219f9-19d3-4797-03c3-79c36369e505	Initial Access or Execution via Microsoft Office Application
systemkey_credential_a_access.py	macos	56934ab6-00f7-4400-9000-700000000000	Keychain Access via Native Shellcode
systemsetup_ssh_enable.py	macos	15303a2a-4e23-4913-9013-030000000000	Enabling of SSH via SystemSetup Command
kcc_kerberos_dump.py	macos	7d593fbf-2111-4e57-9787-0eade9a944d0	Suspicious SystemKey Access via Command Line
networksetup_vpn.py	macos	d7593f12-b593-4194-b797-ac1e4ec4c261	SystemKey Access via Command Line
empire_stager.py	macos, linux	50a4ef18-d1f7-401c-98bc-e20645e1e4dc	Remote SSH Login Enabled via systemsetup Command
special_chars_zip_file.py	macos	15dd0ca0-502d-4669-ac0b-6143e599701a	Potential Access to Kerberos Cached Credentials
dns_create_in_tmp.py	macos	b73974f6-82ff-4743-9e07-1c6901b1f0ea	Fileless Empire Stager Credentials Dumping

Example Run: `python -m rta -l`

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

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.



```
(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/mappings in ATT&CK
└── Installing build dependencies
    └── donepace/Elastic/detection-rules/.detection-rules-cfg.json
        ├── 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) in the Threat Intel API
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
```

Example Run: `make deps`

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

MAKEFILE

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



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