EclecticIQ Qradar App

Technical Design Document

Document History

Version	Description of version/changes	Author	Date
1.0	Initial Design Document	Pranav	02-02-2022

Review History

Version	Comments	Reviewed By	Date
1.0		Raghavendra	07-02-2022

Introduction & Background

This document provides information about the Qradar application developed for EclecticIQ.

The purpose of this Technical Design Document is to provide an architectural overview of the EclecticIQ Qradar Application.

EclecticIQ Intelligence Platform

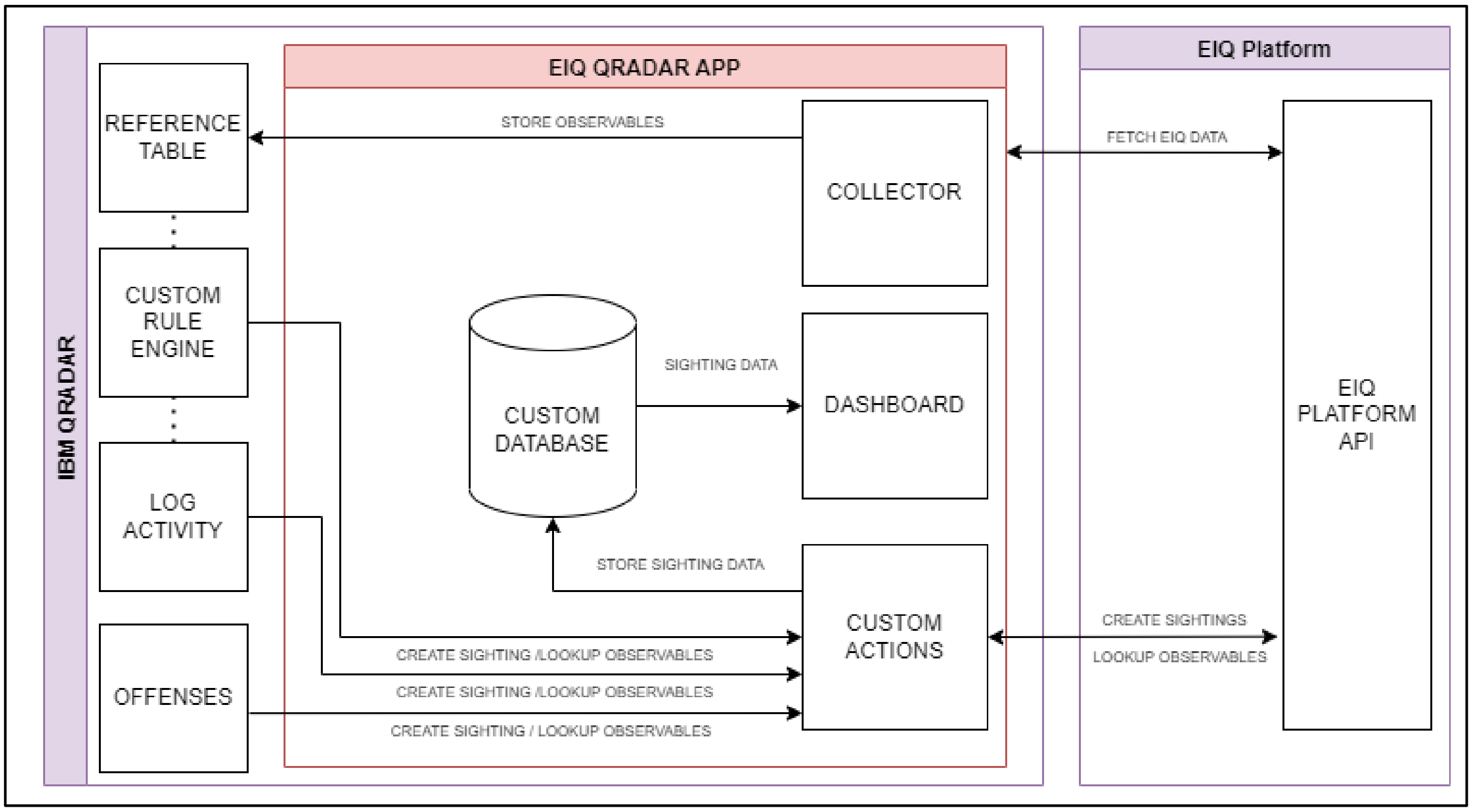
- EclecticIQ Intelligence Center is the threat intelligence solution that unites machine-powered threat data processing and dissemination with human-led data analysis without compromising analyst control, freedom or flexibility.
- Intelligence Center consolidates vast amounts of internal and external structured and unstructured threat data in diverse formats from open sources, commercial suppliers, and industry partnerships. This data becomes a collaborative, contextual intelligence source of truth.
- EIQ data processing pipeline ingests, normalizes, transforms, and enriches this incoming threat data into a complex, and flexible data structure. Next, our technology optimizes and prioritizes this data to help identify the most critical threats more rapidly.
- For total flexibility, Intelligence Center disseminates intelligence as reports for stakeholders or as machine-readable feeds that integrate with third-party controls to improve detection, hunting, and response.
- Intelligence Center offers cloud-like scalability and cost-effectiveness within your trusted environment.

Scope

- The EIQ app for Qradar will collect the observables data from the EIQ platform and store it in Qradar reference tables.
- Users will be provided an option for sighting creation by right clicking on the events in the Log Activity and Offenses tab of Qradar.
- Users will be provided with an option to attach the custom action to create sighting. This action can be attached while creating custom rules in Qradar.
- Users will be provided an option to lookup observables by right clicking on the events in the Log Activity and Offenses tab of Qradar.
- Dashboard will be provided with below three widgets
    - Sightings by time (histogram)
    - Sighting by Confidence (bar chart)
    - Sighting by type count (bar chart)

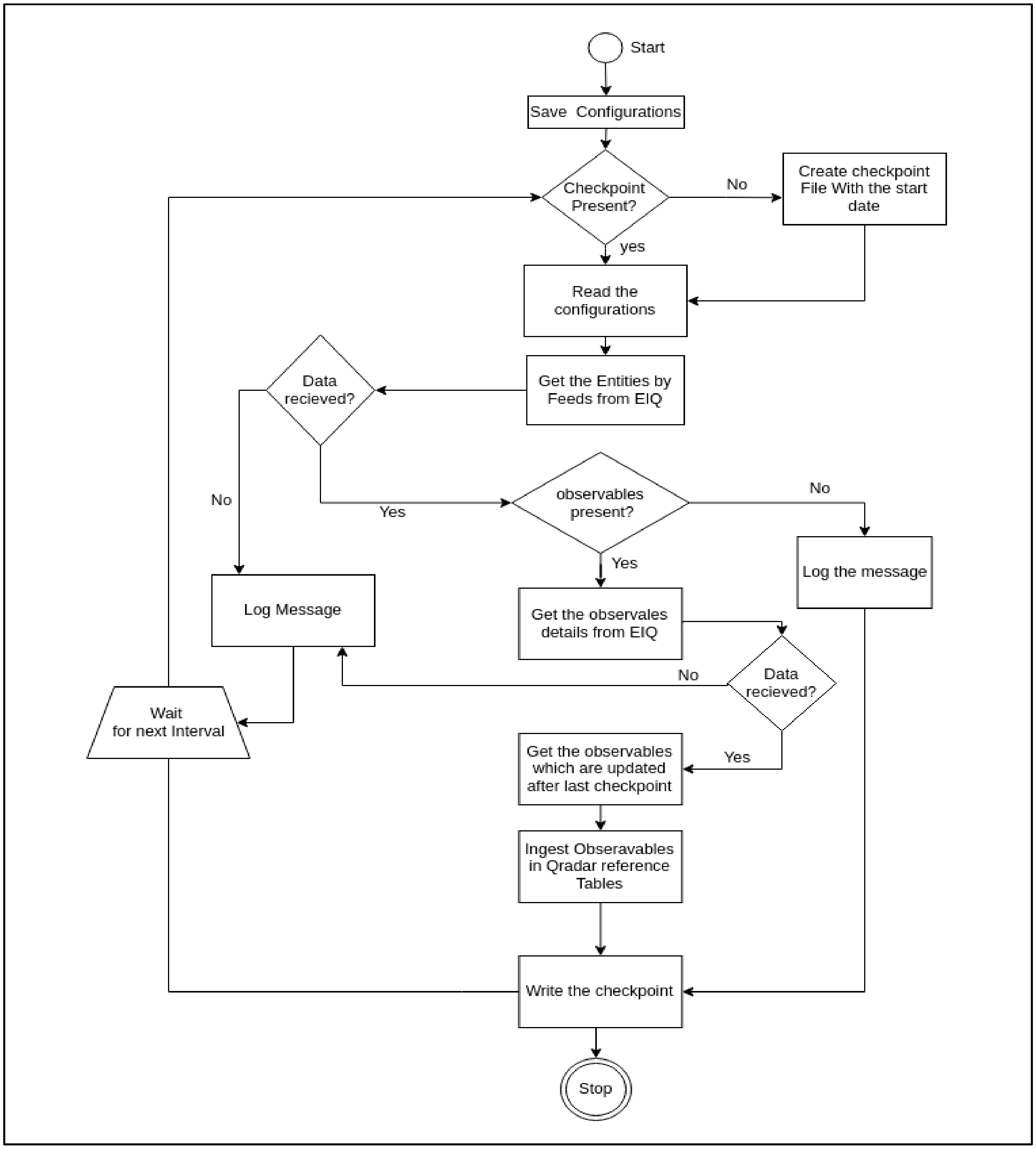
Architecture

Architecture Diagram

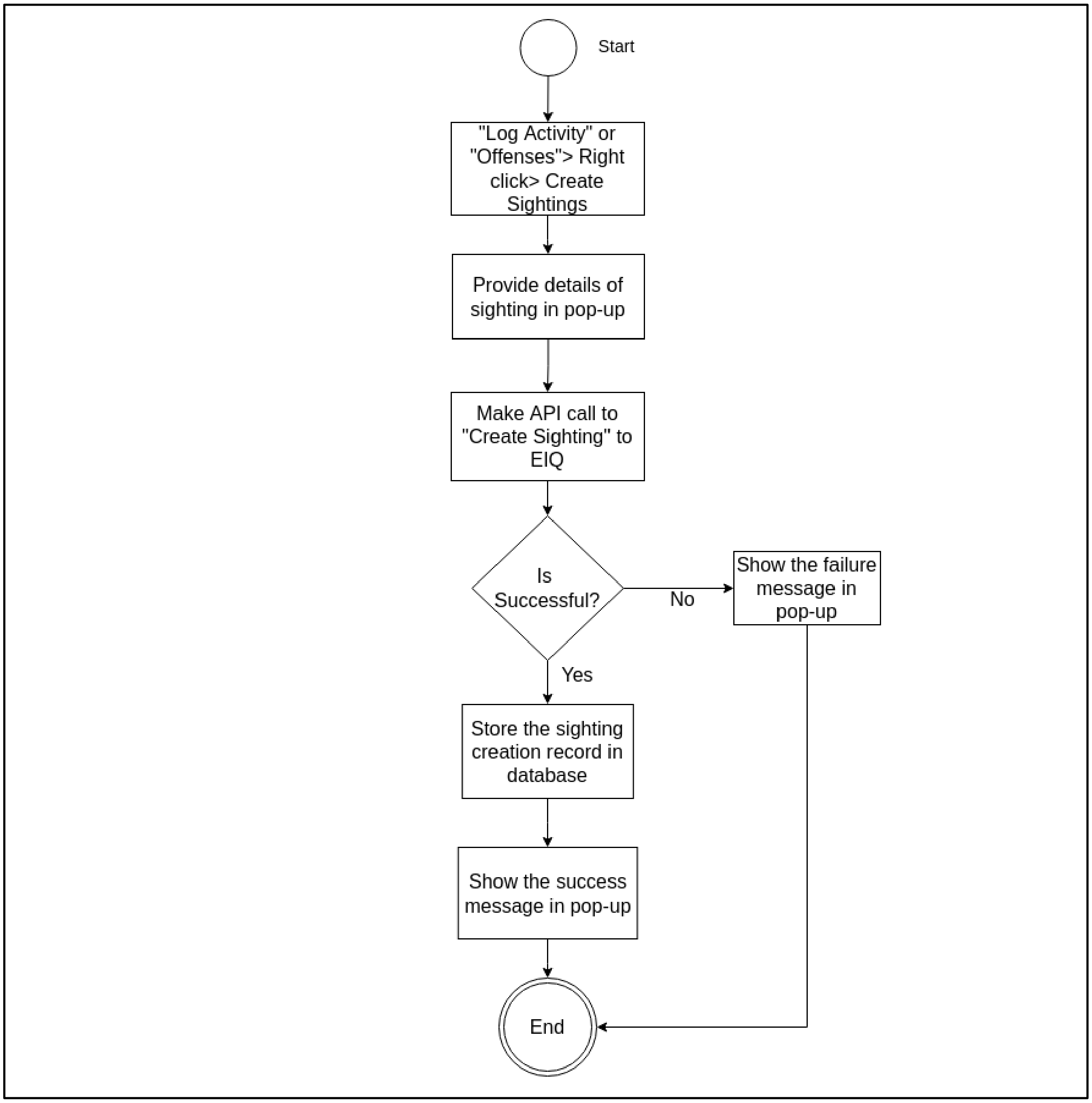


Process flow

EIQ Observables Collection

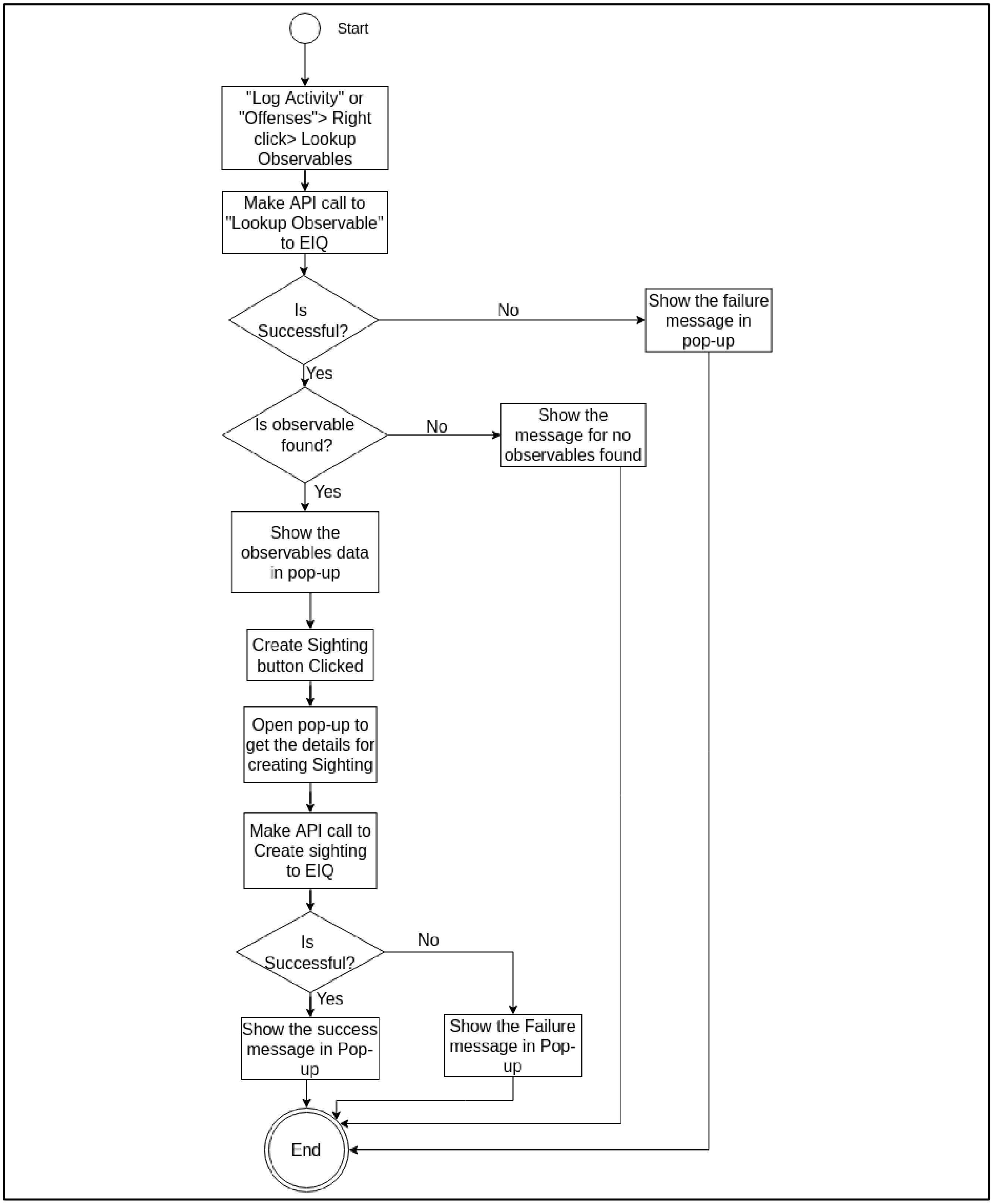

Start
Save Configurations
Checkpoint Present?
No
Create checkpoint File With the start date
yes
Read the configurations
Get the Entities by Feeds from EIQ
Data recieved?
No
Yes
observables present?
No
Yes
Log the message
Log Message
Get the observales details from EIQ
Data recieved?
No
Wait for next Interval
Yes
Get the observables which are updated after last checkpoint
Ingest Obseravables in Qradar reference Tables
Write the checkpoint
Stop

Create Sighting Flow



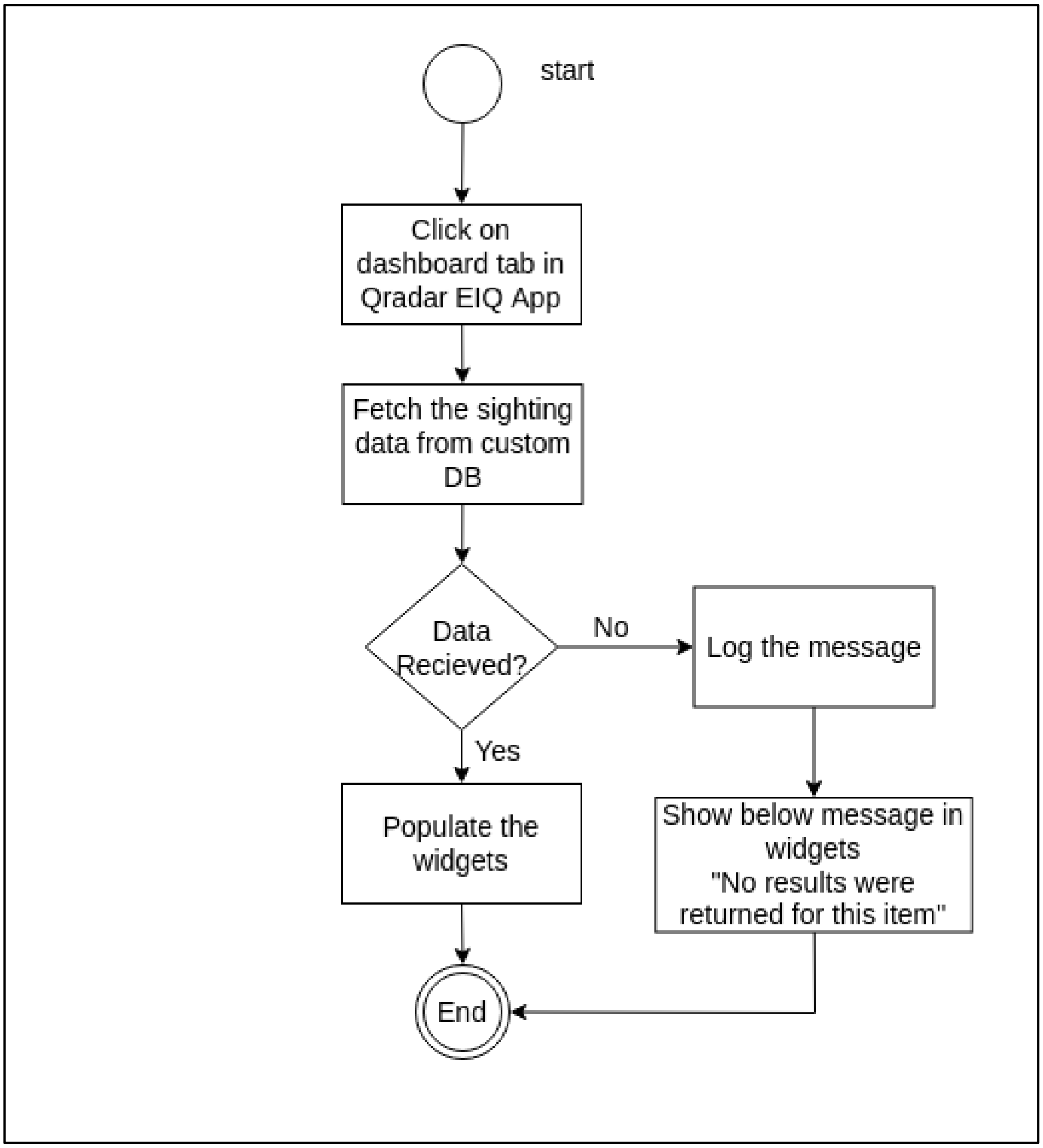
Start
"Log Activity" or "Offenses"> Right click> Create Sightings
Provide details of sighting in pop-up
Make API call to "Create Sighting" to EIQ
Is Successful?
No
Show the failure message in pop-up
Yes
Store the sighting creation record in database
Show the success message in pop-up
End

Lookup Observables Flow

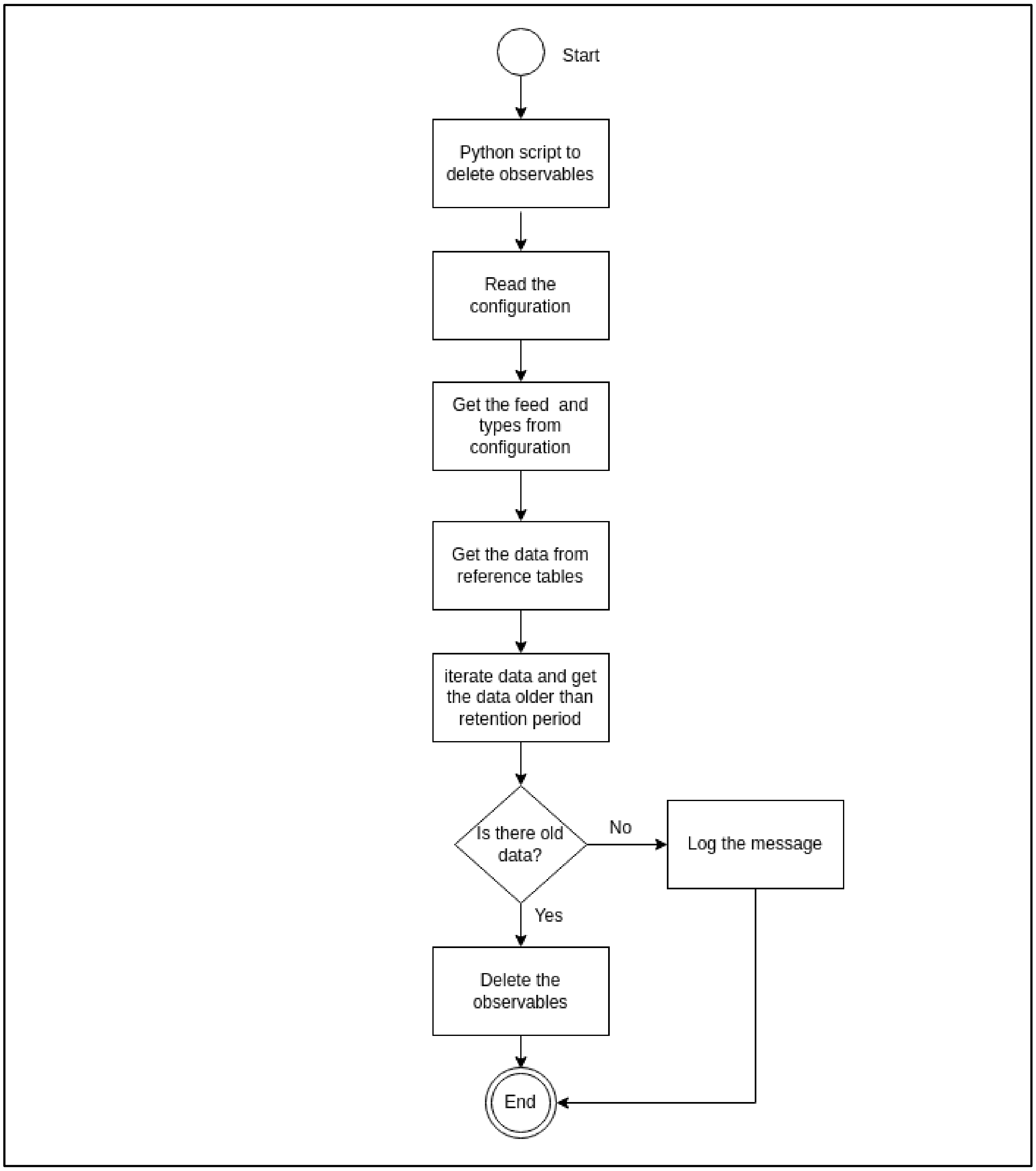


Start
"Log Activity" or "Offenses"> Right click> Lookup Observables
Make API call to "Lookup Observable" to EIQ
Is Successful?
No
Show the failure message in pop-up
Yes
Is observable found?
No
Show the message for no observables found
Yes
Show the observables data in pop-up
Create Sighting button Clicked
Open pop-up to get the details for creating Sighting
Make API call to Create sighting to EIQ
Is Successful?
No
Yes
Show the success message in Pop-up
Show the Failure message in Pop-up
End

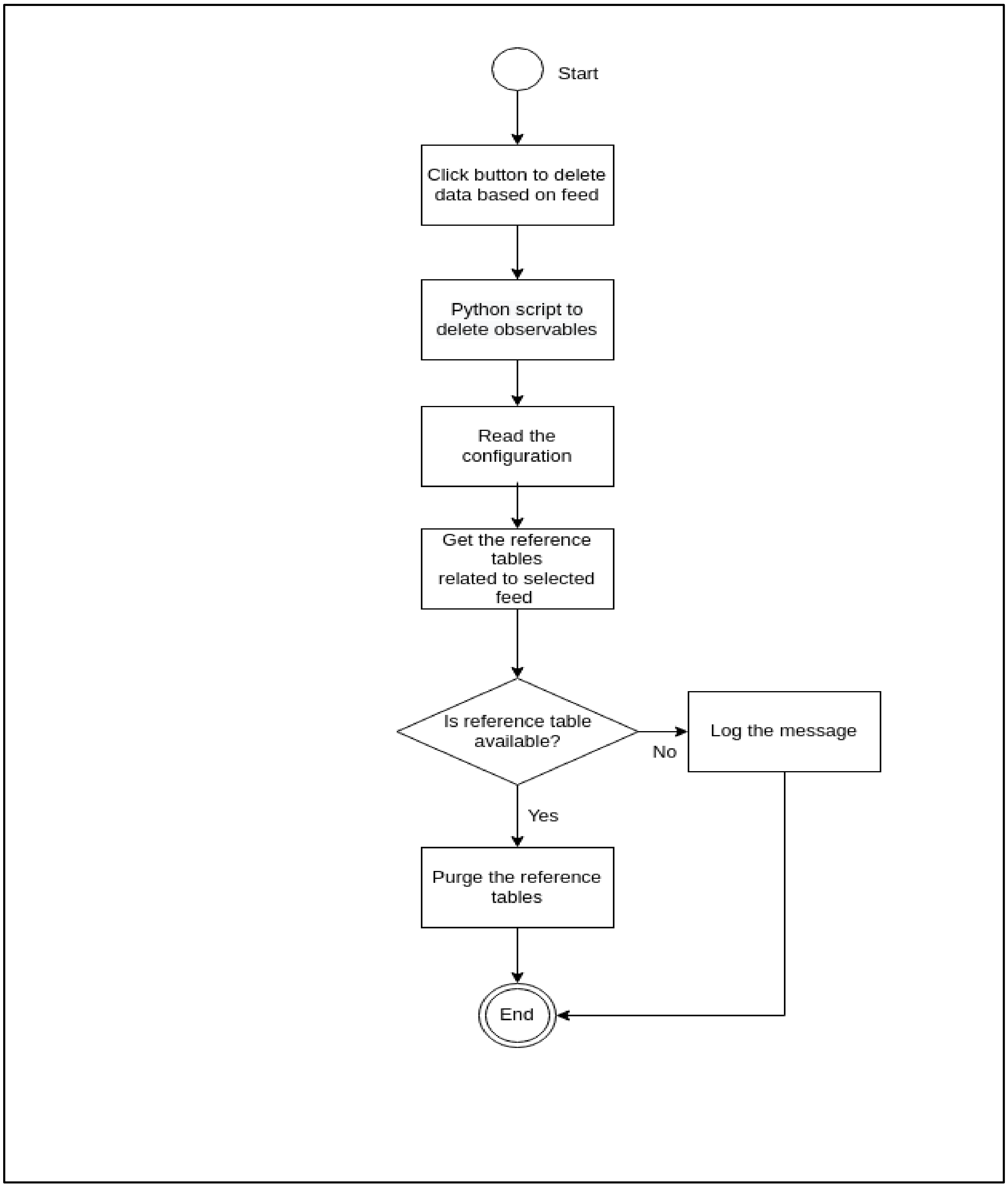
Dashboard Flow


start
Click on dashboard tab in Qradar EIQ App
Fetch the sighting data from custom DB
Data Recieved?
No
Log the message
Yes
Populate the widgets
Show below message in widgets "No results were returned for this item"
End

Scheduled Observable Deletion Flow (based on retention period given in setup)


Start
Python script to delete observables
Read the configuration
Get the feed and types from configuration
Get the data from reference tables
iterate data and get the data older than retention period
Is there old data?
No
Log the message
Yes
Delete the observables
End

Manual Observable Deletion Flow (based on button click on setup page)


Start
Click button to delete data based on feed
Python script to delete observables
Read the configuration
Get the reference tables related to selected feed
Is reference table available?
No
Log the message
Yes
Purge the reference tables
End

Prerequisites

1. QRadar On-prem version 7.3.3 Patch 6. Reference: Qradar Installation Guide
2. EclecticIQ Intelligence Center 2.12.
3. Outgoing feeds should be set up on EclecticIQ Intelligence Center.
4. API key generated from EIQ platform.

Working

EIQ Observables Collection

1. After the installation, the user will have to provide details below in the setup page and click the "Test connection" button.
   Once the connection is successful, the user will have to click on the "Save" button to save the configuration.
   a. Name (Required)
   b. Host (Required)
   c. API key (Required)
   d. Qradar Security token (Required)
2. After saving the credentials, the user will have to enter the below details and click on save.
   a. Select one or more outgoing feeds to collect observables data. (Required)
   b. Select observable time to live. Default (Required)
   c. Select first run backfill historical data time. (Required)
   d. Select the observable types to ingest. (Required)
   e. Select the interval to collect the observables. (Required)
3. Setup page will show a message that shows the last poll date for Observable ingestion.
4. Once the above steps are complete, the collector will start the scheduler with a job to collect the data with the interval mentioned in the setup. The entities will be collected by each feed from the EIQ "Get entities API".
   a. On Successful connection:
      i. From the above-collected entities, the app will make a list of all the unique observables
         1. If Observables are present
            a. For each observable from the list, API calls will be made to EIQ "Get observables API".
            b. On successful Connection:
               i. From all collected observables, the collector will find all the observables which are the latest and will ingest those in qradar reference tables.
               ii. The checkpoint will be stored with the latest updated observable time. This checkpoint will be used in the next data collection schedule.
            c. On failure status response:

            i. The Connector will log the response received and end the process and will wait for the next schedule
        d. On failure to connect
            i. The connector will retry the connection for the number of times provided in configurations. (If no configuration is provided, it will retry by default 3 times).
            ii. On successive retry failure, the connector will log and end the process and will wait for the next schedule.
    2. If Observables are not present
        a. Log the message the no observables found
        b. The checkpoint will be stored with the current time. This checkpoint will be used in the next data collection schedule.

b. On failure status response:
    i. The Connector will log the response received and end the process and will wait for the next schedule.
c. On failure to connect
    i. The connector will retry the connection for the number of times provided in configurations. (If no configuration is provided, it will retry by default 3 times).
    ii. On successive retry failure, the connector will log the message and end the process and will wait for the next schedule.

Create Sighting (Manually)

1. Users can go into the “Log Activity” or “Offenses” tab, open any log and can right click on (IP/URL/Domain/Hash/Email) and click on “Create Sighting”. A pop-up window will appear to ask for the below details. Clicking on save will create sightings in the EIQ platform with provided details.
    a. Sighting Value: Value which is clicked
    b. Sighting type: Type of sighting. Possible values: ip, domain, url, email, hash
    c. Sighting title: Title of sighting
    d. Sighting description: Description of sighting
    e. Sighting confidence: Confidence of sighting. Possible values: low, medium, high
    f. Sighting tags delimited by a comma: Any tags to attach with sighting

Create Sightings (Automatic based on custom rules)

Users can create the custom rules from “Offenses”>” Rules”>” Action”. While creating these rules user will be able to attach the custom action to create sightings on EIQ. When these rules are fired, the app will make the call to EIQ “create sighting“ API to create the sighting from the event.

Lookup Observables

Users can go into the logs received and can right click on (IP/URL/Domain/Hash/Email) and click on “Lookup Observables”. A pop-up window will appear with details about observables associated with clicked value and there will be a button as below:

a. Create Sighting: Clicking on this button pop-up to ask details about sighting will open and an API call will be made to the EIQ platform to create a sighting of clicked value.

Dashboard

Once the sighting is created from Qradar either by custom rule or manually by the user, a record of that will be stored in a custom DB in the app. Dashboard will be populated using stored sighting data. Below three widgets will be loaded.

1. Sightings by time (histogram)
2. Sighting by Confidence (bar chart)
3. Sighting by type count (bar chart)

Manual Observable Deletion Flow

Deletion of observables will be performed when the user de-selects the outgoing feed and clicks the button to delete the data collected from that feed in the setup page. While the user will click on this option, the app will make an API call to Delete reference tables of qradar and all tables created for that feed will be deleted along with the schema. Once deleted recovery will not be possible

Scheduled Observable Deletion Flow

In the configuration screen, users will be able to select the retention period. One scheduled script will be running in the backend every day to remove the data which was updated before the retention period.

Storage of Observables

Reference tables will be created per feed once the user saves the configuration. For example, if there are 3 feeds selected as feed1, feed2, feed3 and 3 observables types are selected as ip, url, domain, then a total of 9 tables will be created with names following format as `eiq_<feed_name>_<type>`. In feed_name only alphanumeric characters will be taken and spaces will be converted to underscore(“_”).

API

EIQ APIs

The application will use EIQ’s V1 public API endpoints

Note: Rate limit of EIQ APIs is 600 requests per minute, or 10 per second.

Get Entities

This API will be used to fetch the entities from EIQ by feeds.

Attribute	**Property**
Method	GET
Endpoint	/api/beta/entities
Content-type	application/json
Authorization	Basic API key

Request URI Parameter

Parameter	**How to Provide**	**Value**
limit	User-configurable	Number of records in a page. This will be sent as a query parameter
offset	Counter by collector module	Index of the record from where to start fetching data
filter[“last_updated_at”]	Checkpoint / User-configurable	Start time from which to fetch the data. Should be in YYYY-MM-DDThh:mm:ss.sssZ format.
filter[“outgoing_feeds”]	User-configurable	Outgoing feeds that the user will configure in the setup page. In case of multiple feeds, add this parameter multiple times in request

Sample response

```
{
  "count": 10,
  "data": [
    {
      "attachments": [],
      "created_at": "2021-10-22T07:49:40.792159+00:00",
      "data": {
        "confidence": "unknown",
        "id": "{http://www.alienvault.com}ExploitTarget-6e31075e-4339-5aa5-9bd9-854bf85dde64",
        "producer": {
          "description": "Pulse Creator",
          "identity": "AlienVault",
          "references": [
            "https://www.trendmicro.com/en_us/research/21/k/Squirrelwaffle-Exploits-ProxyShell-and-
ProxyLogon-to-Hijack-Email-Chains.html",
            "https://www.trendmicro.com/content/dam/trendmicro/global/en/research/21/k/squirrelwaffle-
exploits-proxyshell-and-proxylogon-vulnerabilities-in-microsoft-exchange-to-hijack-email-chains/IOCs-
squirrelwaffle-exploits-proxyshell-and-proxylogon-microsoft-exchange-vulnerabilities-to-hijack-email-
chains.txt"
          ],
          "roles": [
            "Initial Author"
          ],
          "type": "information-source"
        },
        "timestamp": "2021-11-22T14:56:37+00:00",
        "title": "CVE-2021-31207"
      },
      "datasets": [
        "https://ic-playground.eclecticiq.com/api/beta/datasets/42"
      ],
      "id": "c6055673-cd32-4f3b-bd8c-5f4bb9d49957",
      "incoming_feed": "https://ic-playground.eclecticiq.com/api/beta/incoming-feeds/4",
      "last_updated_at": "2021-12-13T15:47:50.383566+00:00",
      "meta": {
        "attacks": [],
        "estimated_observed_time": "2021-11-22T14:56:37+00:00",
        "estimated_threat_end_time": null,
        "estimated_threat_start_time": "2021-11-22T14:56:37+00:00",
        "half_life": 182,
            "source_reliability": "C",
            "tags": [
               "Aviation",
               "Government"
            ],
            "taxonomies": [],
            "tlp_color": null
         },
         "observables": [
            "https://ic-playground.eclecticiq.com/api/beta/observables/3714542"
         ],
         "outgoing_feeds": [
            "https://ic-playground.eclecticiq.com/api/beta/outgoing-feeds/6"
         ],
         "relevancy": 0.7544026209775505,
         "sources": [
            "https://ic-playground.eclecticiq.com/api/beta/sources/ce87c556-0abd-46ca-80ca-03bb748bb751"
         ],
         "type": "exploit-target"
      }
],
  "limit": 10,
  "offset": 0,
  "total_count": 12
}
```

Get Observables

This API will be used to fetch the observables from EIQ by observable id.

Attribute	**Property**
Method	GET
Endpoint	/api/beta/observables/{id}
Content-type	application/json
Authorization	Basic API key

Request URI Parameter

No parameters will be required for this API call.

Sample Response

```
{
  "data": {
    "created_at": "2021-10-22T07:49:40.822732+00:00",
    "entities": [
      "https://ic-playground.eclecticiq.com/api/beta/entities/c6055673-cd32-4f3b-bd8c-5f4bb9d49957",
      "https://ic-playground.eclecticiq.com/api/beta/entities/b6fbfc2b-76c9-4d38-bce5-229feb686305"
    ],
    "id": 3714542,
    "last_updated_at": "2021-10-22T07:49:40.764095+00:00",
    "meta": {
      "maliciousness": "safe"
    },
    "sources": [
      "https://ic-playground.eclecticiq.com/api/beta/sources/ce87c556-0abd-46ca-80ca-03bb748bb751",
      "https://ic-playground.eclecticiq.com/api/beta/sources/cbdd9b26-2d97-411a-b5fc-bf5f7a93604a"
    ],
    "type": "cve",
    "value": "2021-31207"
  }
}
```

Create Sighting

This API will be used to create sightings in EIQ.

Attribute	Property
Method	POST
Endpoint	/api/beta/entities/
Content-type	application/json
Authorization	Basic API key

Request Body Parameter

Parameter	How to Provide	Value
data.type	Fixed value	Fixed value: “eclecticiq-sighting”
data.data.id	User Configurable/ taken from code	Unique ID of the sighting
data.data.confidence	User-configurable	Confidence of the sighting. Possible values: none, unknown, low, medium, high Default value: unknown
data.data.title	User-configurable	Title of the sighting. Example: Qradar detected <IP> ipv4
data.data.description	User-configurable	Outgoing feeds that the user will configure in the setup page. In case of multiple feeds, add this parameter multiple times in request
data.data.timestamp	Current time	Current time when the request is made

Sample Response

```
{
  "data": {
    "attachments": [],
    "created_at": "2022-02-04T08:53:44.367493+00:00",
    "data": {
      "confidence": "high",
      "description": "Created by QRadar Offense  ID = XXXXXX",
      "id": "test",
      "timestamp": "2021-10-27T06:50:37.856622+00:00",
      "title": "QRadar Sighting of  1.1.1.1"
    },
    "datasets": [],
    "id": "14f5016b-915d-4d7a-a611-4be13ae3d576",
    "incoming_feed": null,
    "last_updated_at": "2022-02-04T08:53:44.284661+00:00",
    "meta": {
      "attacks": [],
      "estimated_observed_time": "2022-02-04T08:53:44.367493+00:00",
      "estimated_threat_end_time": null,
      "estimated_threat_start_time": "2021-10-27T06:50:37.856622+00:00",
      "half_life": 182,
      "source_reliability": "A",
      "taxonomies": [],
      "tlp_color": null
    },
    "observables": [],
    "outgoing_feeds": [],
    "relevancy": 0.6832803062020821,
    "sources": [
      "https://ic-playground.eclecticiq.com/api/beta/sources/9a479225-37d1-4dae-9554-172eeccea193"
    ],
    "type": "eclecticiq-sighting"
  }
}
```

Lookup Observables

This API will be used to fetch the observables from EIQ by observable id.

Attribute	Property
Method	GET
Endpoint	/api/beta/observables
Content-type	application/json
Authorization	Basic API key

Request URI Parameter

Parameter	How to Provide	Value
filter[type]	User Configurable	Type from: ipv4, domain, uri, hash-md5, email
filter[value]	User Configurable	Value to search

Sample Response

```
{
  "count": 1,
  "data": [
    {
      "created_at": "2021-11-10T12:32:43.377113+00:00",
      "entities": [
        "https://ic-playground.eclecticiq.com/api/beta/entities/320f9c4b-be8a-40a3-9fd7-aaafa951f474"
      ],
      "id": 4377849,
      "last_updated_at": "2021-11-10T12:32:43.310217+00:00",
      "meta": {
        "maliciousness": "medium"
      },
      "sources": [
        "https://ic-playground.eclecticiq.com/api/beta/sources/f533dc6b-98e7-4bdd-8326-570093f0e191"
      ],
        "type": "ipv4",
        "value": "178.175.124.168"
      }
   ],
   "limit": 10,
   "offset": 0,
   "total_count": 1
}
```

Qradar APIs

Create Reference Table

This API will be used to create a reference table in Qradar

Attribute	**Property**
Method	POST
Endpoint	/api/reference_data/tables
Content-type	application/json
Authorization	Basic SEC token

Request URI Parameter

Parameter	**How to Provide**	**Value**
name	Taken from code	Name of reference table in format “eiq_<feed_name>_<type>”. Type can be any one of ip,email,domain,url or hash.
key_name_types	Taken from code	Fields to store in table. Sample [{"key_name": "created_at", "element_type": "ALNIC"}, {"key_name": "last_updated_at", "element_type": "ALNIC"}, {"key_name": "maliciousness", "element_type": "ALNIC"}, {"key_name": "value", "element_type": "ALNIC"}]
outer_key_label	Taken from code	Static “eiq_value”
element_type	Taken from code	Type if outerkey label. Default: ALN

Sample response

```
{
  "timeout_type": "UNKNOWN",
  "number_of_elements": 0,
  "creation_time": 1644409416837,
  "name": "eiq_feed1_ip",
  "key_name_types": {
     "maliciousness": "ALNIC",
     "last_updated_at": "ALNIC",
     "created_at": "ALNIC",
     "Id": "ALNIC",
     "value": "ALNIC"
  },
  "element_type": "ALNIC",
  "key_label": "test_eiq"
}
```

Get Reference Table

This API will be used to get the data of reference table

Attribute	Property
Method	GET
Endpoint	/api/reference_data/tables/<table_name>
Content-type	application/json
Authorization	Basic SEC token

Request URI Parameter

Parameter	How to Provide	Value
limit	Taken from code	Number of records in a page. This will be sent as a query parameter
offset	Taken from code	Index of the record from where to start fetching data
fields	Taken from code	Which fields to return

Sample response

```
{
   "timeout_type": "UNKNOWN",
   "number_of_elements": 8,
   "data": {
      "6481301": {
         "created_at": {
            "last_seen": 1644244673374,
            "first_seen": 1644244673374,
            "source": "reference data api",
            "value": "2022-02-07t12:15:49.900671+00:00"
         },
         "last_updated_at": {
            "last_seen": 1644244673374,
            "first_seen": 1644244673374,
            "source": "reference data api",
            "value": "2022-02-07t12:15:49.9006+00:00"
         },
         "maliciousness": {
            "last_seen": 1644244673374,
            "first_seen": 1644244673374,
            "source": "reference data api",
            "value": "unknown"
         },
         "value": {
            "last_seen": 1644244673374,
            "first_seen": 1644244673374,
            "source": "reference data api",
            "value": "http://dapp-walletauth.com/connect"
         },
        "id": {
            "last_seen": 1644244673374,
            "first_seen": 1644244673374,
            "source": "reference data api",
            "value": "id"
         }
      }
      },
   "creation_time": 1644244131349,
   "name": "eiq_feed1_ip",
   "key_name_types": {
        "maliciousness": "ALNIC",
        "last_updated_at": "ALNIC",
        "created_at": "ALNIC",
        "Id": "ALNIC",
        "value": "ALNIC"
    },
    "element_type": "ALNIC",
    "key_label": "eiq_value"
}
```

Add data to Reference Table

This API will be used to add the data of reference table

Attribute	Property
Method	POST
Endpoint	/api/reference_data/tables/bulk_load/<table_name>
Content-type	application/json
Authorization	Basic SEC token

Request Body Parameter

Parameter	How to Provide	Value
Payload	Observable response	Sample: {"6481302": {"created_at": "2023-02-07T12:15:49.900671+00:00","last_updated_at": "2022-02-07T12:15:49.9006+00:00","maliciousness": "unknown","value": "http://dapp-walletauth.com/connect", “feed”: feed1}}

Sample response

```
{
  "timeout_type": "UNKNOWN",
  "number_of_elements": 8,
  "creation_time": 1644244131349,
  "name": "test_pranav_table",
  "element_type": "ALNIC"
}
```

Delete Reference Table

This API will be used to delete the reference table

Attribute	**Property**
Method	Delete
Endpoint	/reference_data/tables/{name}
Content-type	application/json
Authorization	Basic SEC token

Request Body Parameter

Parameter	**How to Provide**	**Value**
purge_only	From code	Only purge the table and keep the schema. Default: False

Sample response

```
{
  "timeout_type": "UNKNOWN",
  "number_of_elements": 8,
  "creation_time": 1644244131349,
  "name": "test_pranav_table",
  "element_type": "ALNIC"
}
```

Delete Reference Table Data

This API will be used to delete the data from reference table

Attribute	Property
Method	Delete
Endpoint	/reference_data/tables/{name}/{outer_key}/{inner_key}
Content-type	application/json
Authorization	Basic SEC token

Request Body Parameter

Parameter	How to Provide	Value
value	From reference table data fetched by code	Value of the field

Sample response

```
{
  "timeout_type": "UNKNOWN",
  "number_of_elements": 8,
  "creation_time": 1644244131349,
  "name": "test_pranav_table",
  "element_type": "ALNIC"
}
```

Logging

To track the normal flow of execution of the application as well as any of the issues occurring during the process execution, logging is enabled.

Qpylib provides its logging mechanism, the collector will use this default logging mechanism to log the messages depending on the level mentioned in the table below.

The collector will follow the below guidelines:

Log Level	**Description**
Critical	Very severe error events might cause the application to terminate.
Error	Error events of considerable importance that will prevent normal program execution, but might still allow the application to continue running.
Warning	Potentially harmful situations of interest to end-users or system managers that indicate potential problems.
Info	Informational messages that might make sense to end-users and system administrators, and highlight the progress of the application.
Debug	Relatively detailed tracing used by application developers. The exact meaning of the three debug levels varies among subsystems.

Retry Mechanism

The sac_requests library module allows sending requests to any third-party application. It has a retry mechanism inbuilt that will be triggered by the library module itself if there is any connection error or request timeout error.
By default, **sac_request** retries **max 3 (by default)** times until it returns and if all the retries failed, it will raise an exception that will be logged.

Params	**Fields**	**Type**	**Description**
timeout	Optional	int	Request timeout in seconds. **Default 10 seconds.**
max_retry	Optional	int	A maximum number of retries to be performed. **Default 3**
retry_interval	Optional	int	Several seconds to wait between two consecutive retries. **Default 10 seconds.**
verify_ssl	Optional	bool	Whether the request should verify the SSL certificate or not. **Default False**

Exception Handling

In exception handling, we will be using Python **built-in exceptions**.

The critical operation which can raise an exception will be placed inside the try clause. The code that handles the exception will be written in the except clause.

We will be choosing what operations to perform once we have caught the exception. We will try to catch specific exceptions in the code.

Checkpoint

Checkpoint is a mechanism to store and maintain the DateTime of the last fetched observable. This will help in retrieving the fresh observables always without having any duplicate observables fetched earlier.

Post-deployment of the Qradar app, the user has to provide the start time in the setup page from which the observables should be fetched, this is a one-time activity. The collector makes use of the environment to store the end time of the last executed process to identify it as the start time for the next API call on each interval. Checkpoint value will be stored per feed.

Pagination

The pagination mechanism enables retrieval of multiple pages of response for the same parameters from the API.
For achieving the pagination we will pass below parameters:

1. Limit : Maximum number of items to be returned
2. Offset: Retrieve results starting from the specified (zero-based) index

Initially the offset will be zero and the limit will be the size of data to retrieve in a single page. While in the consecutive calls we will increase the offset by limit to get the next set of data. The loop will be exited when the number of events returned is less than limit or it is zero.

Static type checking and linting

- Use Flake8 for linting and static type analysis to ensure code quality.
- Use SonarQube to find out and fix any “code smells”, “duplicate codes”, “security hotpots” etc.

Unit Test and Code coverage

We will use the pytest framework to write the unit tests.

- We will be covering all the positive and negative scenarios in the Unit Tests.
- We will make sure the code coverage is more than 80% for the code that Sacumen has written.
- To know the code coverage, we are going to use the framework “coverage”, which can be installed through the pip.

Inhouse modules

With the experience of developing multiple connectors in python, Sacumen has developed a few of the python modules in-house that we have found are commonly used across all the connectors. These modules will help in reducing the development cycle since they have certain functionalities that they provide out of the box with few configurations and little or no modifications.

The list of modules is as follows:

Name	**Description**
sac_requests	This module is developed on top of python's requests module. It helps in sending requests to any external API and internally manages retries on failure, SSL certificate verification, etc.

Deployment Process

Open QRadar Extensions Management on the Admin tab to install the App on QRadar.

1. Click the **Admin** tab
2. In the **System Configuration** section, click **Extensions Manager**.
    - Click **Add**.
    - In the Add a New Extension window, click **Browse** to find the app.
    - Select **Install immediately**, and then click **Add**.
    - Click **Install**.
      The app appears in the Extensions Management window after it is installed.

Certification

Technology Partners can submit QRadar extensions to the IBM Security App Exchange portal to start the review process. For more information refer: Publishing your extension

Before submitting the application for IBM review, manifest.json should be added and the package should be signed. For more information refer: How to get your extension package ready for validation