Quokka

Quokka's Wiz integration Instructions

Contents

Quokka

Pre-requisites:

The Quokka integration script requires Python 3.x and the Requests package version 2.32.5

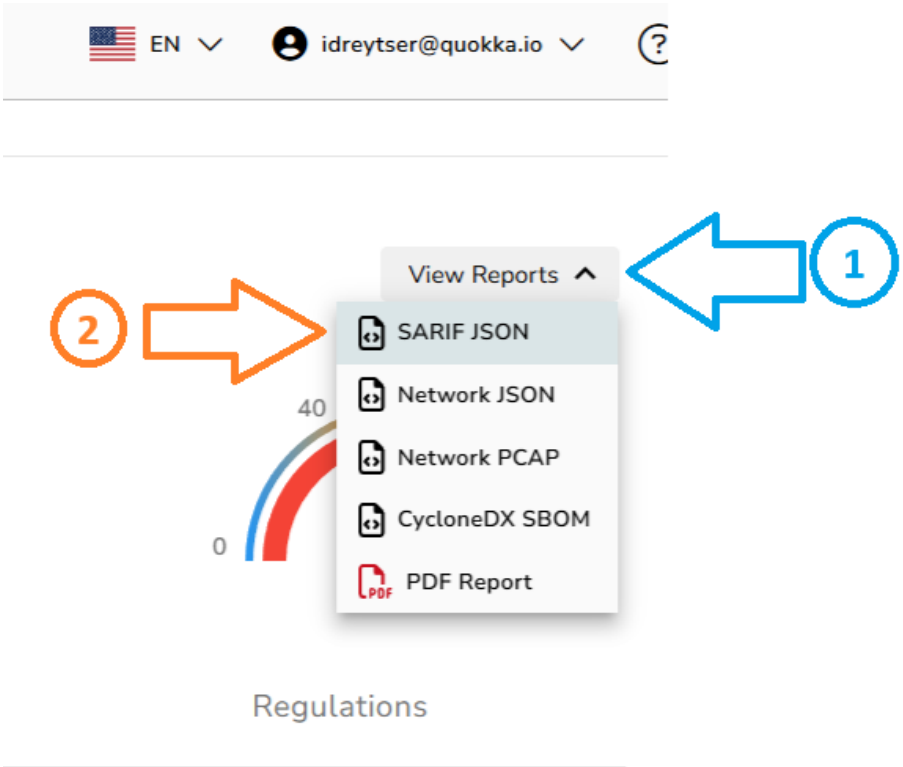
Required Parameters:

1. The Quokka SARIF file, which is produced by the analysis and can be downloaded via UI or API
2. The Wiz Client ID, Client Secret, GraphQL endpoint and Auth URL are required
3. For each scan, the following information is required
    a. Datasource ID (uniquely identifies all scans for this mobile app)
    b. The source control Repository URL (Github, Gitlab, etc)
    c. The source control branch

Typical Workflow:

This integration shall be called once every 24 hours on all relevant codebases with Quokka scans. For each codebase, download the latest results. This can be done within the Quokka UI, or via API submissions:

a. Within the UI, select the New Quokka Report. From the top-right click the View Reports pulldown, then select the SARIF JSON:

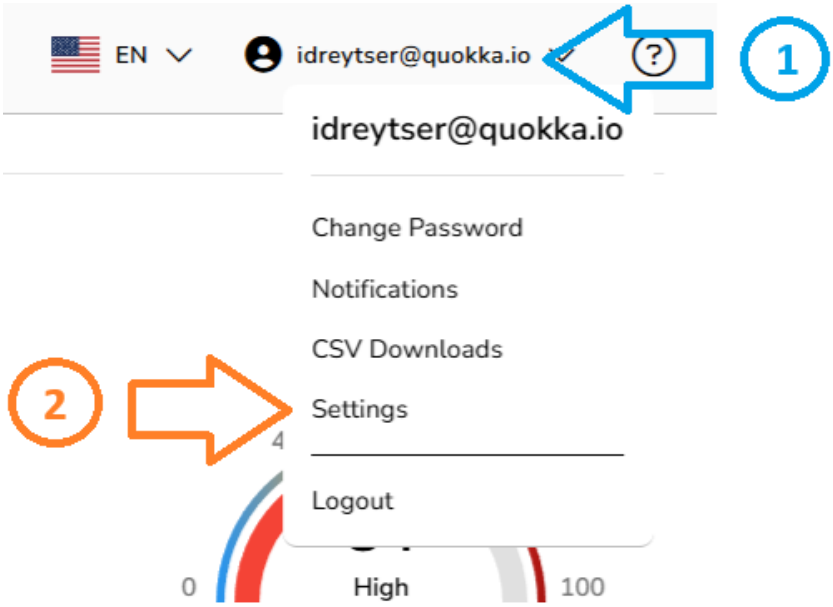


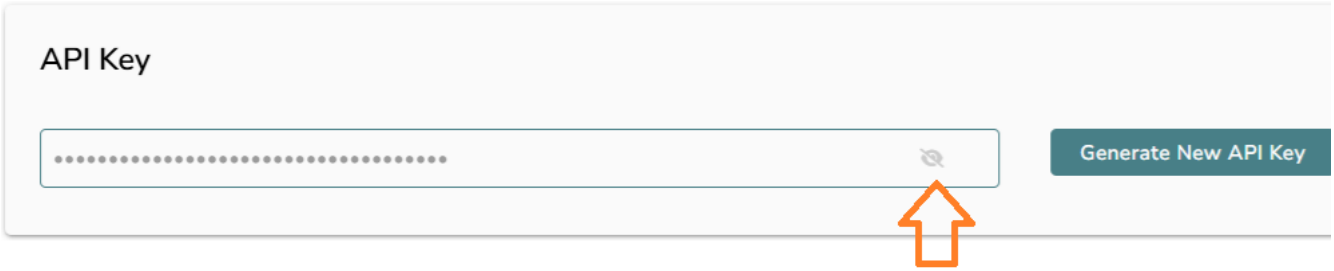
b. Using the APIs or submissions scripts (such as Github Actions or the Gitlab Integration), retrieve the SARIF JSON using the following API route:

API GET Request endpoint: https://api.kryptowire.com
GET Parameters: uuid=**<the UUID of the scan>**&regeneratePDF=false&key=**<The Quokka API key>**

NOTE: Code for a reference Python Script to submit a binary to Quokka for analysis and retrieve the SARIF file is provided at Appendix 1 to this document.

To obtain the API key, click on the user's email at the top right of the UI, then click Settings. From the Settings screen click the view icon to show the API key:

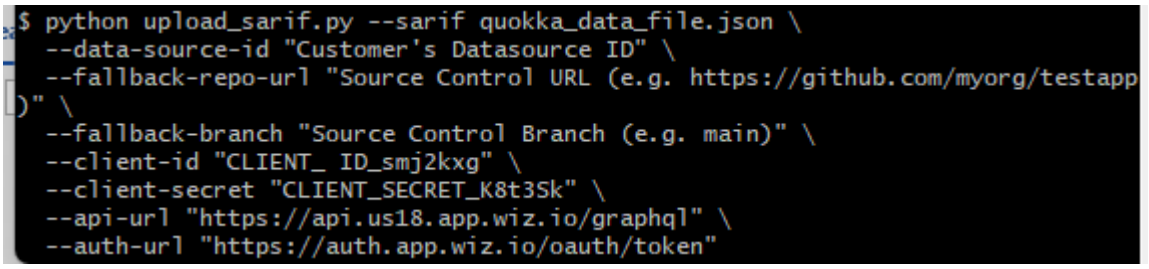




Quokka

Running the Integration Script:

Once the SARIF JSON is downloaded, run the integration script with the required parameters. Pay close attention to the Datasource parameter – it **MUST BE** consistent for each codebase or app being analyzed. Subsequent scans of the same app **MUST** be mapped to the same Datasource!



```
$ python upload_sarif.py --sarif quokka_data_file.json \
  --data-source-id "Customer's Datasource ID" \
  --fallback-repo-url "Source Control URL (e.g. https://github.com/myorg/testapp
)" \
  --fallback-branch "Source Control Branch (e.g. main)" \
  --client-id "CLIENT_ ID_smj2kxg" \
  --client-secret "CLIENT_SECRET_K8t3Sk" \
  --api-url "https://api.us18.app.wiz.io/graphql" \
  --auth-url "https://auth.app.wiz.io/oauth/token"
```

The script will error without all required parameters

FAQ

1. How often should we run this integration? Should it run on every build?
   A: This integration is intended to run once every 24 hours.

2. What if the Datasource changes between scans
   A: The Datasource is the primary key for ensuring that findings are tracked scan over scan. Make sure the Datasource is consistently applied.

3. What if we are scanning app store binaries?
   A: Find out the branch name from which store binaries are created, and use that branch, i.e. “main” or similar.

4. What if we need help?
   A: Contact support@quokka.io and our excellent Support team will help you out!

Quokka

Appendix 1: Sample Submission Python Script

This is a sample Python script to submit a binary to Quokka for analysis and retrieve the SARIF file when it is finished.

```
import click
import requests
import time
import os
from requests import ConnectionError
from datetime as dt

@click.command()
@click.option('--api_key', prompt='API Key', help='Your API key for the Kryptowire API.')
@click.option('--app_filename', default=None, help='App binary filename (required if uuid is not provided).')
@click.option('--uuid', default=None, help='UUID (required if app_filename is not provided).')

def submit_app(api_key, app_filename, uuid):
  if not app_filename and not uuid:
    raise click.UsageError('You must provide either app_filename or uuid.')

  base_url = "https://api.kryptowire.com/api"
  submit_url = f"{base_url}/submit"
  results_url = f"{base_url}/results/sarif"
  if not uuid:
    # Determine platform based on file extension
    thePlatform = "android" if app_filename.lower().endswith(".apk") else "ios"

    # Submit the app binary
    files = {
      'app': open(app_filename, 'rb'),
    }
    params = {'key': api_key, 'platform': thePlatform }
    print("Submitting app...")
    response = requests.post(submit_url, data=params, files=files)
    response.raise_for_status()
    uuid = response.json().get('uuid')
    print(f"Submitted app with UUID: {uuid}")

  # Poll the API for the results
  elapsed_time = 0
```

```
    while True:
        try:
            response = requests.get(f"{results_url}?uuid={uuid}&regeneratePDF=false&key={api_key}")
        except ConnectionError:
            click.echo(f'Connection error. Retrying...{elapsed_time}s')
            break
        if response.status_code == 404:
            click.echo(f'UUID: {uuid}. Artifacts not ready yet, waiting...{elapsed_time}s')
        else:
            response.raise_for_status()
            content_disposition = response.headers.get('content-disposition')
            if content_disposition:
                filename = content_disposition.split('filename=')[1].strip('"')
                timestamp = dt.datetime.now().strftime('%m-%d_%H-%M-%S')
                filename = f"{timestamp}-{filename}"  # Include date in filename
                with open(os.path.join(filename), 'wb') as f:
                    f.write(response.content)
                click.echo(f'Download complete. File saved as {filename}.')
                break
        time.sleep(20)
        elapsed_time += 20

if __name__ == '__main__':
    submit_app()
```