Randori Recon API Documentation

This document includes information on how to access, authenticate, and interact with the Randori Recon V1 API available to Randori Design Partners.

If you have questions or comments about the Randori API or any other matter, please reach out to:

Randori Contact	Email	Randori Slack Name
Ian Lee	ilee@randori.com	@ian
Michael Daganhardt	mdaganhardt@randori.com	@mdaganhardt
Francisco Donoso	fran@randori.com	@fran

Additionally, the Randori team is always available via the `#design_partners` slack channel.

Randori Recon API v1

This is V1 of the Randori Recon API, designed to support the retrieval of entity records discovered by Randori's platform during automated, continous reconnaissance.

This guide will go over the available operations, proper syntax, possible responses and formats, as well as highlight specific use cases.

The Randori API is based on the openAPI 3.0 specification, and accordingly supports API client generation in a wide range of languages (python, ruby, go, curl, PHP, etc.)

Currently, the Randori API supports read-only requests, though we will support additional operations to allow for data input in the future.

Viewing the API

The Randori Recon V1 API spec is avaialble to users who are currently logged into the Randori Platform UI at https://alpha.randori.io.

Once logged in, visit the following URL to download a copy of the Randori API spec in JSON format:

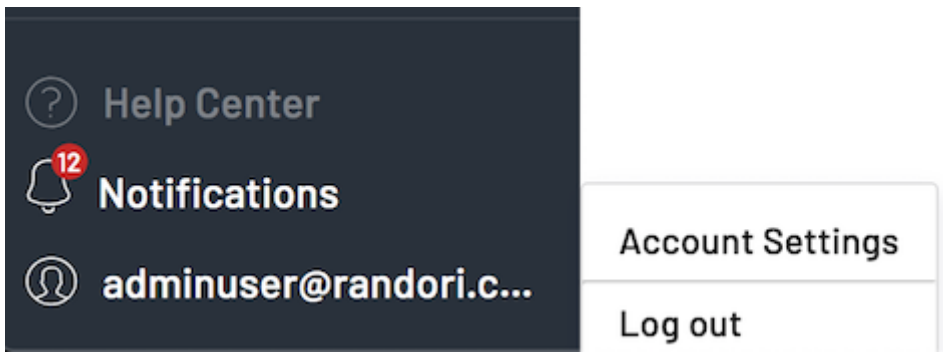
https://alpha.randori.io/openapi

Generating a Randori API key

Users will need an API token in order to interact with the Randori API. In order to generate a token, users need to have the administrator role within their Randori Organization. If you need a token, but cannot create one, ask your organization's Randori Admin to provide you with one.

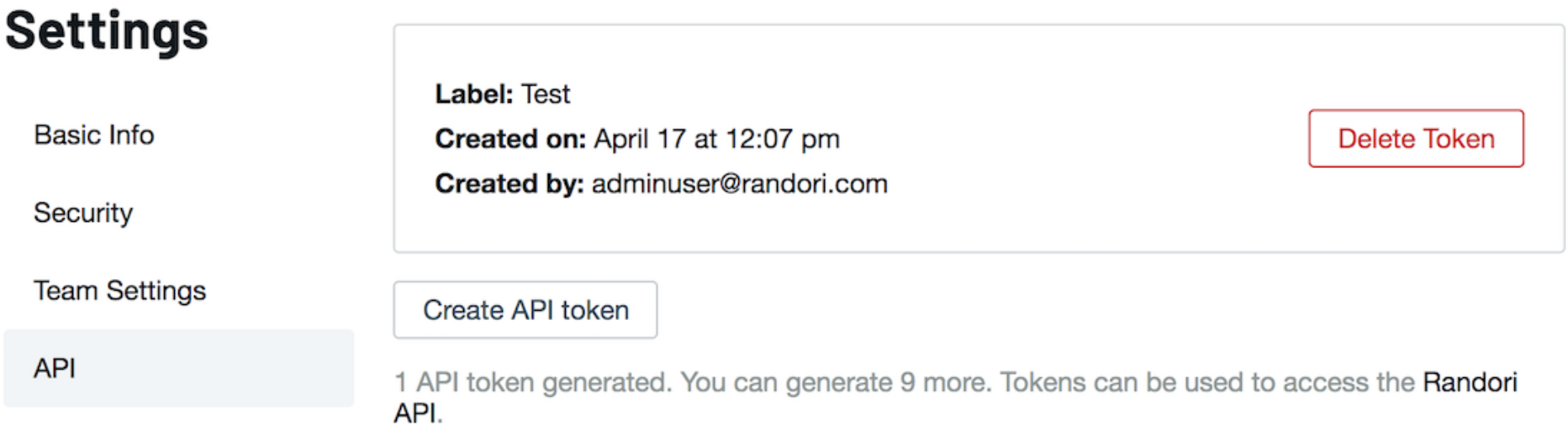
Navigating to account settings

If you are an administrator, you can access the token generation screen by logging into the Randori Platform UI, clicking on your name (bottom left of screen) and selecting "account settings"



Creating and managing tokens

Once you are in the "account settings" screen, nagivate to the API option and generate a new token. You will also be able to manage existing tokens, but you **cannot** view the API keys associated with those tokens.



Generate a query in jQuery querybuilder format

Once you have parsed the api spec into a program such as Openapi-ui or Postman, you will notice that the complex query field (q) needs to be converted to a based 64 encoded string in order to interact with the API. This field is used to apply advanced filters to the query (asset criticality, ports, protocols, etc). The "offset", "limit", and "sort" fields are appended to the API Endpoint url as a plaintext string. Users can create their JSON Query object, convert to base64 and append to the request url.

This is an example JQuery Querybuilder:

```
{
    "condition": "AND",
    "rules": [
        {
            "field": "table.target_temptation",
            "operator": "greater_or_equal",
            "value": 15  # medium or higher target temptation
        }
    ]
}
```

After generating the query, you will need to encode it into base64. Using the above query as an example, the `q` HTTP parameter will be sent as follows (please note `\` added in order to render correctly)

```
eyJjb25kaXRpb24iOiAiQU5EIiwgInJ1bGVzIjogW3siZmllbGQiOiAidGFibGUudGFyZ2V0X3RlbXB0YXRpb24iLCAib3\

BlcmF0b3IiOiAiZ3JlYXRlcl9vcl9lcXVhbCIsICJ2YWx1ZSI6IDE1fV19
```

Downloading and Viewing the Randori API spec

Downloading the Randori API spec

The following section of the document describes how to view the Randori API spec via a OpenAPI UI docker container.

The Randori Recon V1 API spec is avaialble to users who are currently logged into the Randori Platform UI at https://alpha.randori.io. Once logged in, visit the following URL to download a copy of the Randori API spec in JSON format:

https://alpha.randori.io/openapi

Interacting with the Randori API using OpenAPI UI

***Note*: The OpenAPI UI project is not maintained or supported by Randori. This is simply an example.**

In order to use the instructions below you will need to download and install docker.

Once you have downloaded and installed docker, follow the instructions below to interact with the Randori API spec via OpenAPI UI.

```
# download the OpenAPI docker container
docker pull swaggerapi/swagger-ui
# Start the container
docker run --rm -p 8080:8080 -d --name swagger -e API_URL=http://localhost:8080/randori-api.json \
swaggerapi/swagger-ui
```

```
# Copy our downloaded API spec into the container
docker cp randori-api.json swagger:/usr/share/nginx/html/randori-api.json
```

Once the container is running and the Randori API spec is copied into the container, simply visit the folllowing URL in your web browser:

http://localhost:8080

Examples:

The examples below also requires that you have an environment variable named `RANDORI_API_KEY`. In order to *securely* store your API key as an environment variable without exposting the key in your shell (command line) history please use the following commands (available in bash or unix-like shells)

```
# Silently read the variable without exposting it in bash history
read -s RANDORI_API_KEY

export RANDORI_API_KEY
```

Simple command line demo using curl:

Here, we are querying information on a specific service from the command line and piping the JSON formatted response to python's JSON tool.

```
# Use a variable that includes the service UUIDs
service_ids="7dec2bc5-c1f7-4531-910b-eda3990ffe12,6f17e9db-91df-45d6-b215-19a1a5fa0e9b"
# Use that variable in a one line curl example

curl -s "https://alpha.randori.io/recon/api/v1/service/$service_ids" \
-H "accept: application/json, text/plain" -H "Authorization: $RANDORI_API_KEY" | python -m json.tool
```

Python Example:

The python example script below assumes that you've already generated an API key as described in the *Generating a Randori API key* section of this document.

The script below also requires that you have an environment variable named `RANDORI_API_KEY`. In order to *securely* store your API key as an environment variable without exposting the key in your shells history.

```
# Silently read the variable without exposting it in bash history
read -s RANDORI_API_KEY

export RANDORI_API_KEY
```

Example Python Script

```
#!/usr/bin/python3

"""
Example API module for exporting Randori data to CSV for all medium or higher
confidence and medium or higher target tempation assets.

Prior to running this code, ensure the environment variable
RANDORI_API_KEY is set to the path to the API file.
"""

import requests
import json
import os
import csv
from base64 import b64encode

# Get the Randori Recon API token from environment variables
api_token = os.getenv("RANDORI_API_KEY")

if api_token is None:
```

```
    print("Missing environment variable RANDORI_API_KEY.")
    print("Please export RANDORI_API_KEY=$RANDORI_API_TOKEN first.")
    exit(1)
elif len(api_token) < 100:
    print("This token appears too short. Please ensure the entire token"
          "added as an environment variable.")
    exit(1)

RANDORI_PLATFORM_URL = "https://alpha.randori.io/"

# Randori Recon entity types (excluding service)

endpoints = ['recon/api/v1/ip',
             'recon/api/v1/hostname',
             'recon/api/v1/network',
             'recon/api/v1/target',
             'recon/api/v1/service']


# Construct headers which send our API token
headers = {'Authorization': api_token,
           "Content-Type": "application/json"}

# Construct our complex query in JQuery querybuilder
query = {
    "condition": "AND",
    "rules": [
        {
            "field": "table.target_temptation",
            "operator": "greater_or_equal",
            "value": 15  # medium or higher target temptation
        }
    ]
}

# We need the query to be a string in order to base64 encode it easily
query = json.dumps(query)

# Randori expects the 'q' query string to be base64 encoded
query = b64encode(query.encode())


def generate_csv(endpoint, csvname):

    # Max value is 2000
    limit = 0  # initially we may set the limit to 0 to get the total records
    offset = 0  # initial number of records to skip.
    all_entities = []

    params = {
        'q': query,
        'offset': offset,
        'limit': limit,
        'sort': '-target_temptation'  # sort from highest to lowest
    }

    # Construct the full endpoint URL for use by requests
    url = RANDORI_PLATFORM_URL + endpoint
    response = requests.get(url, params=params, headers=headers)

    """
    Results for these queries have this form:
    {
      "count": 0,
      "data": [ { ... },  ... ],
      "offset": 0,
      "total": 0
    }
    """

    # check the server response code for success
    if response.status_code != 200:
        print(f"The request for {url} failed! The server returned: "
              f"{response.status_code}")

    # the response is returned as json data
    result = response.json()
    if 'total' not in result or 'data' not in result:
        print("The server returned an unexpected reponse.")
        exit(1)
```

```
    # get the total number of records in the set to display to the user
    total_records = result['total']

    entity = endpoint.split('/')[-1]  # for the example endpoints, the last
    # component of the path is the type of entity we are querying.
    print(f"Requesting data about {total_records} {entity}s from Randori")

    if total_records == 0:
        print(f'No records for {entity}, skipping')
        return

    # make further requests for records 10 at a time. Max value is 2000
    # Please update this value
    params['limit'] = 10

    while offset <= total_records:
        try:
            # make the request for the next 10 records
            response = requests.get(
                url,
                params=params,
                headers=headers)
            # verify the result, if we ever get a non-200 response
            # something has gone wrong and we should stop making requests
            if response.status_code != 200:
                print("The server returned an unexpected reponse. "
                      f"We got HTTP status code: {response.status_code}")
                exit(1)
            result = response.json()
            records = result['data']
            if result['count'] == 0:
                break
            for record in records:
                all_entities.append(record)
            offset = offset + result['count']
            params['offset'] = offset

        except Exception as e:
            print(f'exception was {e}')

    # finally, write out the data to a CSV file
    with open(csvname, 'w') as csvfile:
        # Dynamically get names of columns based on dictionary keys
        columns = all_entities[0].keys()
        writer = csv.DictWriter(csvfile, fieldnames=columns)
        writer.writeheader()
        for entity in all_entities:
            writer.writerow(entity)
    print(f"Completed writing {csvname}")


for endpoint in endpoints:
    entity = endpoint.split('/')[-1]
    csvname = entity + '_randori_export.csv'
    generate_csv(endpoint, csvname)
```