

FINAL REPORT – CHALLENGE

Presented by Herón Arteaga Hernández.
May 22, 2023.

Index

<i>Documentation of the research and implementation</i>	<i>2</i>
Objective	2
Technologies Used	2
Background.....	2
Step-by-Step Guide	3
Requirements:	3
<i>Reproduction steps (Dockerfile and any additional files, if applicable)</i>	<i>4</i>
File thousand_app.py.....	4
<i>Bonus challenge</i>	<i>6</i>
File Dockerfile	6
<i>Vulnerability report or mitigation strategies.....</i>	<i>7</i>
<i>Links to the GitHub repository, Docker Hub images, or one-liners to recreate the project.</i>	<i>10</i>

Documentation of the research and implementation

Objective

The objective of this challenge is to demonstrate skills, abilities, and knowledge in container technologies, API handling, automation and DevOps, and vulnerability detection and management in these environments.

Technologies Used

- Docker: Used to build and manage containers. It allows packaging the application and its dependencies into an isolated and portable environment.
- Python: Used as the programming language to interact with the ThousandEyes API and make the corresponding API calls.
- ThousandEyes API: Used to make API calls to the ThousandEyes API and retrieve the list of tests configured in the ThousandEyes application.
- Logging: Used to print outputs in log format, which provides more detailed information obtained from the ThousandEyes API.

Additionally, for the additional objective of retrieving data from another ThousandEyes trial account, the Gmail alias functionality can be used to create multiple trial accounts and make API calls using those accounts.

Background

ThousandEyes is a network monitoring and performance platform that offers an API (Application Programming Interface) to allow developers and users to access platform data and functionality programmatically.

The ThousandEyes API provides a set of endpoints and methods that enable various operations such as obtaining information about monitoring agents, configuring network tests, accessing performance metrics, retrieving event and alert data, and more.

Some of the capabilities that can be achieved with the ThousandEyes API include:

Creation and management of network tests: It is possible to configure and monitor network performance and availability tests such as ping tests, link speed tests, route tests, web page tests, among others.

Collection of performance data: The API allows retrieving performance data and metrics from monitoring agents distributed in different geographical locations. This includes information about latency, packet loss, jitter, availability, and other key performance indicators.

Presented by Herón Arteaga Hernández.

May 22, 2023.

Access to historical and real-time data: Historical and real-time data on network performance can be accessed to analyze trends, identify issues, and make informed decisions in network infrastructure management.

Automation and programming: The API enable task automation and integration with other tools and systems by programming API requests and responses.

The ThousandEyes API uses token-based authentication, where an authentication token is provided to make API calls on behalf of an authorized user.

Step-by-Step Guide

Requirements:

1. Choose the programming language you will use to build the application inside the container. In this project, we will use Python.
2. Have Docker installed.
3. Create a new directory where you will work on the project.
4. Create the XXXX.py file inside the project directory. You can use a text editor to create this file and add the necessary code to make the ThousandEyes API calls. Save the file in the project directory.

```
thousand_app.py
1 import requests
2 import logging
3 import json
4
5 # Define authentication credentials
6 API_USERNAME = 'heron.artega.h@gmail.com'
7 API_TOKEN = 'pntb1ied1mm0vdpjrlpolj4nyu5iht7'
8
9 # Configure logging
10 logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
11
12 # Make API call with the current credentials
13 response = requests.get('https://api.thousandeyes.com/v6/tests.json', auth=(API_USERNAME, API_TOKEN))
14
15 resp_body = response.json()
16
17 # API Response
18 if response.status_code == 200:
19     for test in resp_body['test']:
20         created_by = test['createdBy']
21         test_id = test['testId']
22         test_type = test['type']
23         print(f'CreatedBy: {created_by}')
24         print(f'TestID: {test_id}')
25         print(f'Type: {test_type}')
26
27 # Data from another ThousandEyes trial account
28 trial_email = 'mr.emilio.olguin@gmail.com'
29 trial_auth_token = 'qgz263clac68qfrbv77gyj9cczejqv0'
30 logging.info(f'Attempting to retrieve data from trial account: {trial_email}')
31
32 # API call with trial account credentials
33 trial_auth = (trial_email, trial_auth_token)
34 trial_response = requests.get('https://api.thousandeyes.com/v6/tests.json', auth=trial_auth)
35
36 if trial_response.status_code == 200:
37     trial_data = trial_response.json()
38     logging.info(f'List of tests from trial account:')
39     for test in trial_data['test']:
40         logging.info(f'Test ID: {test['testId']}, Test Name: {test['testName']}')
41 else:
42     logging.error(f'Error calling API with trial account: {trial_response.status_code}')
43
```

5. Open a terminal and navigate to the created directory.
6. Create a new file named "Dockerfile" in that directory using your favorite text editor.
7. Open the Dockerfile in your text editor and start writing the instructions.

```

1 # Use a Python base image
2 FROM python:3.11
3
4 # Set working directory
5 WORKDIR /app
6
7 # Copy the code into the container
8 COPY thousand_app.py .
9
10 # Install dependencies
11 RUN pip install requests
12
13 # Specify the command to run when the container starts
14 CMD [ "python", "thousand_app.py" ]

```

8. Build the Docker image.

```

crypto@CryptoMacBook-Pro Heron %
crypto@CryptoMacBook-Pro Heron % docker build -t thousandeyes-heron-artega-container

```

9. Once the image building is complete, you can run a container based on that image.

```

crypto@CryptoMacBook-Pro Heron % docker run thousandeyes-heron-artega-container
2023-05-20 07:49:09,966 - INFO - Attempting to retrieve data from trial account: mr.emilio.olguin@gmail.com
2023-05-20 07:49:10,508 - INFO - List of tests from trial account:
2023-05-20 07:49:10,508 - INFO - Test ID: 3736561, Test Name: azulcrema.com.mx
2023-05-20 07:49:10,508 - INFO - Test ID: 3736556, Test Name: circuloderma.com.mx
2023-05-20 07:49:10,508 - INFO - Test ID: 3736560, Test Name: Azulcrena
2023-05-20 07:49:10,508 - INFO - Test ID: 3736554, Test Name: Circulo_Derma
2023-05-20 07:49:10,508 - INFO - Test ID: 3736558, Test Name: CirculoDerma_PageLoad
CreatedBy: Heron Arteaga (heron.artega.h@gmail.com)
TestId: 3736561
Type: dns-server
CreatedBy: Emilio Olguin (mr.emilio.olguin@gmail.com)
TestId: 3736556
Type: dns-server
CreatedBy: Heron Arteaga (heron.artega.h@gmail.com)
TestId: 3736560
Type: http-server
CreatedBy: Emilio Olguin (mr.emilio.olguin@gmail.com)
TestId: 3736554
Type: http-server
CreatedBy: Emilio Olguin (mr.emilio.olguin@gmail.com)
TestId: 3736558
Type: page-load

```

10. This will execute the container based on the image and display the result of the ThousandEyes API call.

Reproduction steps (Dockerfile and any additional files, if applicable)

File thousand_app.py

1. Import the necessary modules: requests, logging, and json.

```

import requests
import logging
import json

```

2. Define the authentication credentials to access the ThousandEyes API. In this case, we use the variables API_USERNAME and API_TOKEN.

```
# Define authentication credentials
API_USERNAME = 'heron.arteaga.h@gmail.com'
API_TOKEN =
```

3. Configure the logging with the logging level set to INFO and the message format.

```
# Configure logging
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s - %(message)s')
```

4. Make an API call using the provided authentication credentials. We use the requests.get method to send a GET request to the URL https://api.thousandeyes.com/v6/tests.json. The authentication credentials are passed through the auth parameter as a tuple (API_USERNAME, API_TOKEN).

```
# Make API call with the current credentials
response = requests.get('https://api.thousandeyes.com/v6/tests.json', auth=(API_USERNAME, API_TOKEN))
```

5. Store the API response in the variable response.

```
# Make API call with the current credentials
response = requests.get('https://api.thousandeyes.com/v6/tests.json', auth=(API_USERNAME, API_TOKEN))
```

6. Extract the response body in JSON format using the.json() method of the response.

```
resp_body = response.json()
```

7. If the response status code is 200 (success), we iterate over the list of tests in resp_body['test'].

```
# API Response
if response.status_code == 200:
    for test in resp_body['test']:
```

8. For each test, extract and display relevant information such as the creator, test ID, and type.

```
        created_by = test['createdBy']
        test_id = test['testId']
        test_type = test['type']
        print("CreatedBy:", created_by)
        print("TestId:", test_id)
        print("Type:", test_type)
```

Bonus challenge

- Next, define another ThousandEyes trial account using the variables `trial_email` and `trial_auth_token`.

```
# Data from another ThousandEyes trial account
trial_email = 'mr.emilio.olauin@gmail.com'
trial_auth_token =
```

- Log an info message using the logging module to indicate that we're attempting to retrieve data from the trial account.

```
logging.info(f'Attempting to retrieve data from trial account: {trial_email}')
```

- Make another API call using the provided trial account credentials. Similar to before, we use the `requests.get` method and pass the authentication credentials through the `auth` parameter.

```
# API call with trial account credentials
trial_auth = (trial_email, trial_auth_token)
trial_response = requests.get('https://api.thousandeyes.com/v6/tests.json', auth=trial_auth)
```

- Check if the response status code is 200.

```
if trial_response.status_code == 200:
```

- If it is, convert the response to JSON format and use the logging module to log information about the tests in the trial account.

```
trial_data = trial_response.json()
logging.info('List of tests from trial account:')
for test in trial_data['test']:
    logging.info(f"Test ID: {test['testId']}, Test Name: {test['testName']}")
```

- If the status code is not 200, we log an error message using the logging module.

```
else:
    logging.error(f'Error calling API with trial account: {trial_response.status_code}')
```

This code makes an API call to the ThousandEyes API to retrieve a list of tests configured in a specific account and then attempts to retrieve data from another trial account using different authentication credentials. The results are printed to the console or logged to the logging system, depending on how the logging level is configured.

File Dockerfile

- The Dockerfile starts with the `FROM` instruction, which specifies the base image to use for the container. In this case, it uses the Python 3.11 base image.

```
1 # Use a Python base image
2 FROM python:3.11
3
```

- The WORKDIR instruction sets the working directory inside the container to /app. This is where the subsequent commands will be executed.

```
4 # Set working directory
5 WORKDIR /app
```

- The COPY instruction copies the file thousand_app.py from the local machine (the context where the Docker build command is executed) to the /app directory inside the container.

```
7 # Copy the code into the container
8 COPY thousand_app.py .
```

- The RUN instruction is used to execute a command during the build process. In this case, it runs the pip install requests command inside the container to install the requests library, which is a dependency for the application.

```
10 # Install dependencies
11 RUN pip install requests
```

- Finally, the CMD instruction specifies the command to run when the container starts. It sets the entry point to python and the argument to thousand_app.py, which is the main Python script of the application.

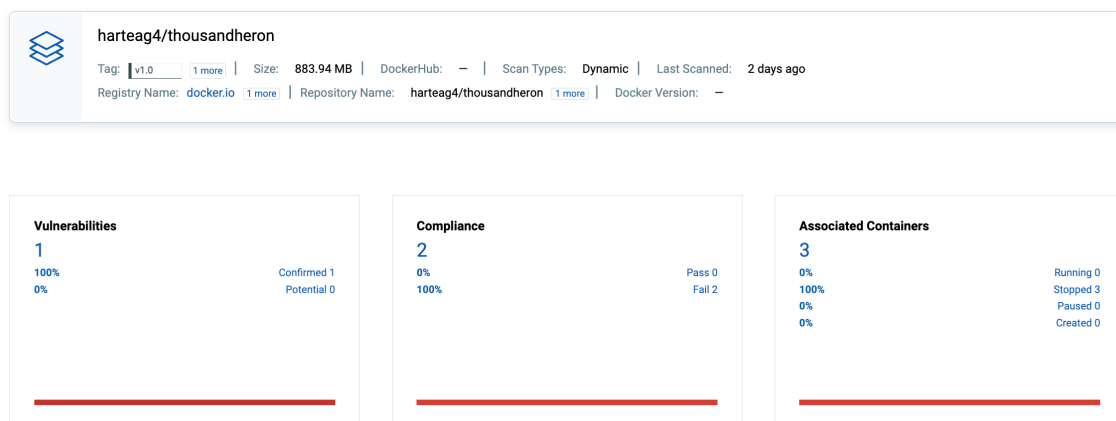
```
13 # Specify the command to run when the container starts
14 CMD [ "python", "thousand_app.py" ]
```

Vulnerability report or mitigation strategies

Here is a vulnerability report generated by the Qualys tool.

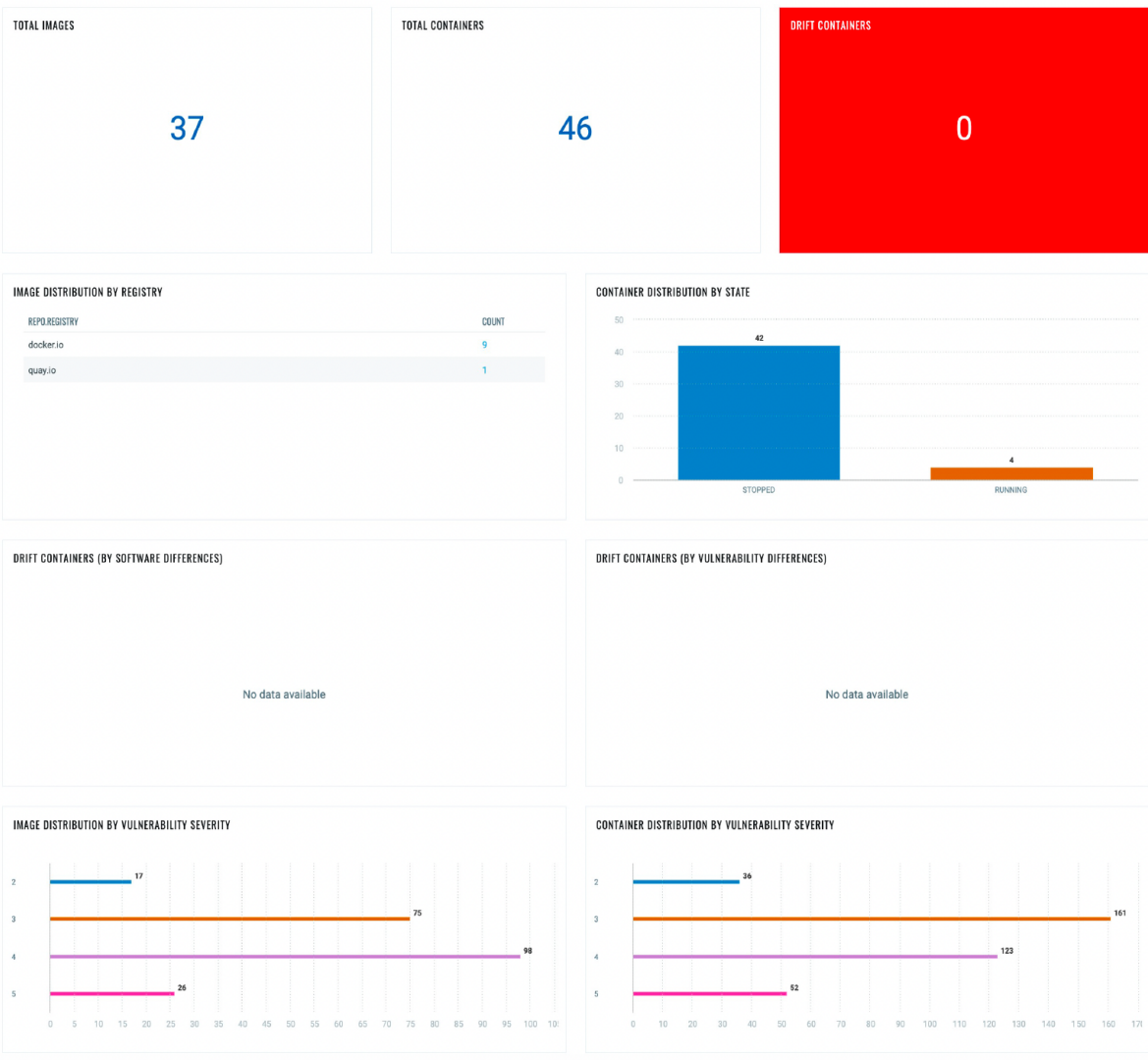
Summary

Quick Summary of the Image




Presented by Herón Arteaga Hernández.
May 22, 2023.

As you can see, there is only one critical vulnerability identified in the container generated for this project. However, the report includes vulnerabilities detected in all the computer images used.



Specifically for the topic of this project, the detected vulnerability is **Debian Security Update for postgresql-13 (DSA 5401-1)**.

General Information



Debian Security Update for postgresql-13 (DSA 5401-1)
CVE: [CVE-2023-2454](#) [\[more\]](#)
Published Date: 7 days ago 07:31 AM
Severity: ■■■■■

Identification	CVSS Summary	Vulnerability Analysis
QID: 181779	CVSSv2 Base: 5.4	Exploitability: 0
Category: Debian	CVSSv2 Temporal: 4	Patches: 1
Modified Date: -	CVSSv3.1 Base: 8.6	Malwares: 0
Discovery Method: AUTHENTICATED	CVSSv3.1 Temporal: 7.5	
Authentication: UNIX_AUTH	Access Vector: Adjacent Network	
Supported Apps: VM, CA-Linux Agent	Vendor Reference: DSA 5401-1	

Impact

Successful exploitation of this vulnerability could lead to a security breach or could affect integrity, availability, and confidentiality.

Solution

Refer to Debian security advisory [DSA 5401-1](#) for updates and patch information.

After analyzing the recommendations and conducting research on this vulnerability. To fix this vulnerability, it is necessary to update the postgresql-13 version to the recommended version **13.11-0+deb11u1**.

[SECURITY] [DSA 5401-1] postgresql-13 security update

To: debian-security-announce@lists.debian.org
Subject: [SECURITY] [DSA 5401-1] postgresql-13 security update
From: Moritz Muehlenhoff <jmm@debian.org>
Date: Thu, 11 May 2023 16:36:32 +0000
Message-id: <ZF0ZkF1zcPUiyach@seger.debian.org>
Reply-to: debian-security-announce-request@lists.debian.org

-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA512

Debian security Advisory DSA-5401-1 security@debian.org
<https://www.debian.org/security/> Moritz Muehlenhoff
May 11, 2023 <https://www.debian.org/security/faq>

Package : postgresql-13
CVE ID : CVE-2023-2454 CVE-2023-2455

Two security issues were found in PostgreSQL, which may result in privilege escalation or incorrect policy enforcement.

For the stable distribution (bullseye), these problems have been fixed in version 13.11-0+deb11u1.

We recommend that you upgrade your postgresql-13 packages.

For the detailed security status of postgresql-13 please refer to its security tracker page at:
<https://security-tracker.debian.org/tracker/postgresql-13>

Further information about Debian Security Advisories, how to apply these updates to your system and frequently asked questions can be found at: <https://www.debian.org/security/>

Mailing list: debian-security-announce@lists.debian.org

-----BEGIN PGP SIGNATURE-----

iQIzBAEBCgAdFiEEtuYvPRKsOE1cDakFEMKttsN8TjYFAMrdF+kACgkQEMKttsN8
TjZnMA/+OW6jzVbc8StDbvz+I23baEOZiDuXYQ+LxMTB1XQMU52AsZqOOVNHwkc
wG2DtZRNWVPuFtfJOREnpjDMviqR+/yBjL1Ou4jH68V8EFTv1I9jk70AmOGcL4
mYSr66iawcIpsQF05mv/BcMREI5raSNT/tFDtIBRFQLEYMwLfnBfprN/DGzn5x/p
4Wn/7AH7a0GuZ569YaEpgMo4eJ3g9wEsG8wgDMU1eNXT67BH2qRt0h/9MyZbnNVV
/b0XKX+0w1DROad9JSToSeQX11J/8U99DPLZE20RzrFrcvUiyjUxdkYBDHnOT
W/uWkNBTjucE72RasMn8XtygUA/XIik8jtbTLSSysyoSb00NNbKo74HCwKOLtZyoR
84SEd5IknZRWfmszL0wZj8qy3VwrPes2mgfFQYWXGPrjPFJmksWU0fmuLD8rt7L
02XSPoWbM47FNA+12LKqFeb39wocXo/D46G2QX0c6gIm9oeYyglWApMVBqRkDsp
s/Fv1kBeyTzq7MGUEWjpNiJaHSna3j893kr7cfnTcBe50gKe/7yijVf7bSoGmcv
Lv1b6zqQ7K5V/Mv8he9k6eInvpiIzEDTCL+BI3tXde3BO/TMTfpgpSXOAizIB+PX
Vz6TzW07b1MzNKhK16YqVpWt7tMtRefLdzg4CMA3QMrvzv8zPKU=
=D+6d

-----END PGP SIGNATURE-----

Presented by Herón Arteaga Hernández.
May 22, 2023.

Links to the GitHub repository, Docker Hub images, or one-liners to recreate the project.

<https://hub.docker.com/r/harteag4/thousandheron/tags>

<https://github.com/Harteag4H/Challenge-ThousandEyes->