12-15-2022

Docker Container Image – Vulnerability Scanning

Joseph U. Ohaeche

Docker Container Image – Vulnerability Scanning

Joseph U. Ohaeche
Grand Valley State University, School of Computing, and Information Systems
Michigan, USA
Email: ohaechej@mail.gvsu.edu

***Abstract*— The technology landscape for container adoption has greatly evolved over the years from the first known Unix U7 container concept introduced in 1979 to the most utilized docker container concept which emerged in 2013.**

Docker container image is essentially a lightweight, standalone executable software package with capabilities to run an application. It is important to know that container images become containers when deployed, and simultaneously docker container images become docker containers when deployed on Docker Engine.

This project paper aims, evaluates, and presents a methodology useful in vulnerability scanning of docker container images and suggests possible fixes based on OWASP and CVE standards before being deployed or made live.

Results from this project show the importance of utilizing Docker Container Registry mapped with continuous integration and continuous deployment (CI/CD) pipeline for early detection of vulnerabilities in a docker image to help eliminate or reduce enterprise/organization data or security breaches.

***Index Terms*— Docker Container Images, Vulnerability, Google Container Registry, GitHub Action**

I. INTRODUCTION

Docker founded by Solomon Hykes is a PaaS product that utilizes O.S virtualization to deliver software in packages called a container. The inception of Docker pushed back on virtual machines as this offers an even more efficient and faster-containerized platform for enterprises and has rapidly been adopted by various cloud services out there.

Google is one of the many big tech firms to have adopted and partnered with Docker to come up with the Google/Docker Registry and the whole idea behind this was to launch Google Container Registry (which is a private docker registry running on Google Cloud Service)

Images on Google Container Registry can be accessed easily from managed VMs, Google Container Engine, non-Google cloud services, or individual personal machines making it easily versatile to be adopted by various organizations or software developers. Google Container Registry ***(GCR)*** implements a Docker protocol so that users can push and pull images directly with Docker clients.

Docker Registry API is a protocol that enables and facilitates the distribution of images to the Docker Engine thereby giving cloud services e.g., Google Cloud, AWS, etc., open-source O.S e.g., Linux; the platform to utilize Docker Container Registry to create and deploy container images to various cloud services/protocols.

Enterprises, businesses, etc. utilizing this service make it a hot debate to question the security it provides to container images being deployed or made live by these enterprises or businesses to their clients. Developers who utilize these services are prone to making mistakes that can lead to different types of vulnerabilities being exploited by agents with malicious intent to commit cyber-attacks ranging from ransomware to data breaches.

The four major areas to be conscious of when reviewing Docker security are namely:

- Kernel Namespace
- Control Groups
- Docker Daemon Attack-Surface
- Linux Kernel Capabilities

Our focus on vulnerability scanning will be addressing the above-mentioned areas of Docker Security.

To address the concerns of docker security, I will be demonstrating the automation of a program built with Python Programming Language and integrated into GitHub Action for more efficient script automation of vulnerability scanning in container images.

GitHub is simply an open-source internet hosting platform for software development, and I will be utilizing this platform for my program automation using GitHub Action. GitHub Action is a continuous integration and continuous delivery (CI/CD) platform that allows software developers to automate their program builds, and test for issues before deploying or making their pipeline live. You can learn more about the Components of GitHub Action and its workflow.

This is an ongoing project. In this paper, I will introduce and demonstrate how GitHub Action is integrated with Google Container Registry for proprietary image vulnerability scanning and suggests fixes based on OWASP and CVE standards.

II. ENVIRONMENT SETUP

❖ *Google Cloud Platform – Environment Setup*

A basic Google Workspace Account gives you access to the Google Cloud and Google Container Registry. To gain full access to the Google Cloud Platform to create container images, or build virtual machine projects, we will need to opt into a Tier Account (for the purpose of the project we opted for the Free GCP Tier Account, which gives $300 worth of free credit).

A GCP Tier Account typically requires the below information to be activated:

1. **Account Information** (Location, Organization GCP Needs)
2. **Identity Verification** (Contact Details)
3. **Accepting** the GCP Terms and Conditions.

After the successful setup of the GCP Account, this gives full access to the Container Registry, under Container Registry navigation we need to do the following.

- **Create GCP Project:** (two projects namely ***"Capstone - Vulnerable Image and Capstone - Good Image"*** were created for the purpose of vulnerability scanning and detection demonstration in container images)

Fig 1: Figure displaying 2 Container Images (Vulnerable & Good Image)

- **Enable Container Registry API**
- **Enable Container Analysis API**

For adequate Access and Identity Pool Management, which will be useful for authentication and connection to public platforms and other cloud service platforms, there are certain authentication settings that need to be properly set up for security reasons. Under the IAM and Admin navigation, the following actions were performed.

- **Service Account Creation** (Create a service account with appropriate permission for container image registry and container image analysis

- **Workload Identity Pool Creation** (Identity Pool manages and organizes external identities such as Azure, AWS, GitHub, etc. for CI/CD automation) Under this feature IAM gives the option to the System Administrator to grant access to valid federated identities in the identity pool

- **Identity Provider Secure Connection** (For this project, the GitHub platform was utilized for CI/CD automation, GitHub authenticates using the OpenID Connect (OIDC). OIDC is an additional layer of open authentication protocol on the OAuth framework. It's utilized to allow single sign-on (SSO) to access third-party OpenID Providers e.g., GitHub (being an OpenID Provider is utilizing the OIDC to access and connect to Google Container Registry Platform)

- **Provider Mapping Configuration** (Set adequate attributes and claims from providers to be easily accessible on IAM navigation. ***["google.subject=assertion.sub,attribute.actor=assertion.actor,attribute.repository=assertion.repository"]***

- **Access** (Service account holder or System Administrator allows valid pool identities access to resources in Google Cloud)

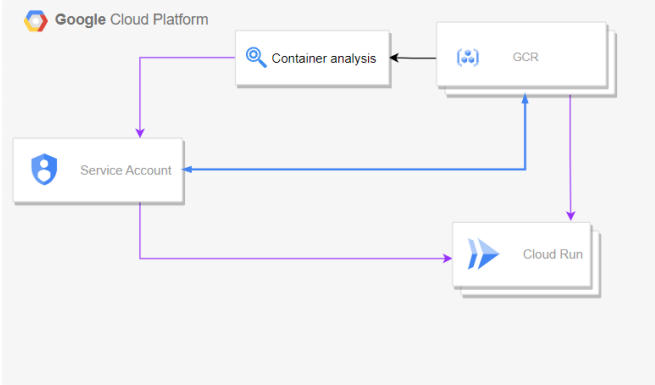


Fig 2: Figure displays the Google Cloud Platform (GCP) Environment Interaction

❖ *GitHub Action Platform – Environment Setup*

A basic GitHub Account grants you access to the GitHub Action feature required for the CI/CD automation

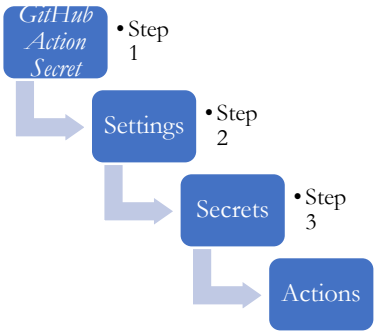
Once signed into the GitHub account, we navigate to the GitHub Repository and created a new Repository ***(named Proprietary Image Scanning – Vulnerability Extraction)***.

For security concerns to enable secure authentication management of the ***Workload Identity Federation Provider (WIF_Provider)*** and

Workload Identity Federation Service Account (WIF_Service_Account) integration from Google Cloud to GitHub, the *WIF_Provider* and *WIF_Service_Account* were made secret.

WIF_Provider – "make secret"

WIF_Service_Account – "make secret"



After setting WIF_Provider and WIF_Service_Account to secret we then go ahead to create our workflow **"*Scan and Deploy to Cloud Run*"** and insert our YAML Script which will be utilized to run job automation on GitHub.

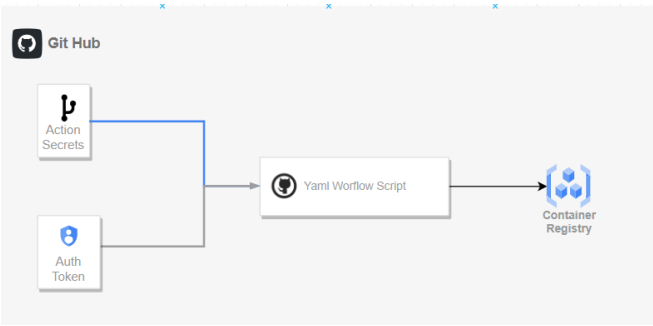


Fig 3: Figure displaying GitHub Environment Interaction

III. VULNERABILITY SCANNING AUTOMATION

After successfully verifying and authenticating to Google Cloud Platform from the inserted YAML script, we utilized a Python Script to handle and access the Google Cloud Platform Resource (Google Container Registry) for vulnerability scanning. The following script was used to authenticate GCP.

```yaml
- name: Google Auth
  id: auth
  uses: 'google-github-actions/auth@v0'
  with:
    token_format: 'access_token'
    workload_identity_provider: '${{ secrets.WIF_PROVIDER }}'
    service_account: '${{ secrets.WIF_SERVICE_ACCOUNT }}' #
```

Fig 4: Figure displaying Authentication Token Generation in GitHub Action

The essence of the Python script is to handle GCP – API by authenticating, and after successful authentication grants access to GCR then scans and checks for vulnerabilities in the proprietary image of the GCR, if a vulnerability is detected our Python script throws an exception error and halts deployment but if the proprietary image of the GCR is free from vulnerabilities then it continues to the next phase of job run and deployment. In addition, our vulnerability scanning, and extraction process helps to classify critical levels of vulnerability detected.

```
container_analysis_url = "https://containeranalysis.googleapis.com/v1/projects/${{ env.PROJECT_ID }}/occurrences:vulnerabilitySummary"

headers = {
  "Content-Type": "application/json",
  "Authorization": token
}


response = requests.request(
  "GET",
   url,
   headers=headers
)
```

Fig 5: Figure displays the handling of Container Analysis API

The following packages were installed to handle the API request.

- **Requests:** HTTP Python Library
- **JSON:** Standard text-based format for structured data representation on Web Apps

```
print(json.dumps(json.loads(response.text), sort_keys=True, indent=4, separators=(",", ": ")))

result_dict = json.loads(response.text)

if len(result_dict) == 0:
    print("good to deploy")
else:
    results = result_dict['counts']
    for i in results:
     print (i['severity'], i['totalCount'])
    raise Exception("vulnurabilities found")
```

Fig 6: The figure displays the logical process of either deploying image or throwing an error exception if vulnerabilities are detected

The Python script after handling the Container Analysis API for vulnerabilities, and upon the developer patching any detected vulnerabilities, runs the Python script automation again to confirm container image is totally free from further vulnerabilities afterward if no vulnerabilities are found, the Python script successfully deploys Container Image to whatever cloud platform or service the image is needed in production.

```yaml
- name: Deploy to Cloud Run
  id: deployrun
  run: |-
       gcloud beta run jobs create ${{ env.SERVICE }} \
       --image ${{ env.GCR }}/${{ env.PROJECT_ID }}/update_img:v1 \
       --project ${{ env.PROJECT_ID }} \
       --set-env-vars SLEEP_MS=10000 \
       --set-env-vars FAIL_RATE=0.5 \
       --max-retries 5 \
       --region ${{ env.REGION }}
```

Fig 7: The figure displays the process of deploying a Cloud Run Job if the image is free from vulnerabilities

❖ ***Python Script – Functionality***

Library packages imported were utilized by the Python Script to better handle the Container Analysis API, GitHub Action generates an authentication token which will be stored as a variable in conjunction with the Container Analysis URL to access the scanned vulnerabilities in the cloud platform.

Excerpts from the Python Script for authentication are shown below:

```
# This code uses the 'requests' library:
# http://docs.python-requests.org
import requests
import json
token = "Bearer ${{ steps.auth.outputs.access_token }}"
container_analysis_url = https://containeranalysis.googleapis.com/v1/projects/${{ env.PROJECT_ID }}/occurrences:vulnerabilitySummary
```

Handling the API:

An API has a header in which values can be passed to get outputs in the desired formats like JSON or XML. For this project JSON format was used. Output is generated in JSON format, whilst ensuring the header contains the required authentication token to create a dictionary of headers utilized to get a response.

Response:

When handling an API, a GET or POST method is to be specified for API handling. The GET method was utilized in this case scenario. We specify the URL for the GET method to retrieve the required information and then pass the HTTP Header in the request. Output is stored in the "response variable" and returned in JSON format.

Excerpts from the Python Script for handling API are shown below:

```
headers = {
"Content-Type":
"application/json",
"Authorization": token
}
response = requests.request(
"GET",
container_analysis_url,
headers=headers
)
```

Main Logic:

With the use of the JSON library, we loaded the JSON data/output into a dictionary afterward we loop through the Python script to detect vulnerabilities and their severities accordingly. The script function is to raise an exception error and halt the deployment process if any vulnerability is found in the entire project.

Excerpts from the Python Script for vulnerability detection are shown below:

```
print(json.dumps(json.loads(response.text),
sort_keys=True, indent=4, separators=(",", ": ")))
result_dict = json.loads(response.text)
if len(result_dict) == 0:
print("good to deploy")
else:
results = result_dict['counts']
for i in results:
print (i['severity'], i['totalCount'])
raise Exception("vulnerabilities found")
```

IV. USE CASES

This project's proof of concept can be adopted and implemented in the real-time tech industry for various Startup Firms concerned with Cloud Architecture Security.

1. A tech startup develops a healthcare SaaS application to be hosted onto a cloud service. It is paramount to ensure that the production environment is free from any vulnerabilities to reduce attack surfaces and assure stakeholders their product strictly adheres to HIPAA policy compliance.

This tech startup comes up with a staging project where they can leverage this project concept, and simply add an additional step to the YAML Script to push an image to production if it is free of vulnerability after it goes through the vulnerability scanning test; if not free from vulnerabilities, the Python script automation triggers an exception error causing deploying the application into production to halt giving the software developers a window frame to fully patch vulnerabilities found before they can move forward with rolling out their SaaS application.

Below is the YAML modification that can be implemented after the Python script step.

- -name:push to prod
  run:|-
  docker push gcr.io/$PROJECT_ID/$SERVICE_NAME:$GITHUB_SHA

2. Another use case scenario is concerned with an organization or enterprise with complex microservice applications spread across a multi-cluster and multi-cloud environment or a hybrid setup including an on-premises data center. This project can be leveraged to implement a secure solution to deploy workloads in a large Kubernetes Cluster Environment whilst adhering to security compliances like "***PCI DSS" or "Fed Ramp"***

Asides from the two use cases mentioned above, I'm certain this isn't just limited to the above-mentioned use cases, but different organizations, enterprises or startups concerned with cloud security can leverage, adopt and implement the vulnerability scanning concept focused on in this project.

❖ ***GOAL***

- The major advantage of this process is simply the fact this is Cloud Agnostic, as we are handling the entire process through the API rather than using a specific Cloud Software Development Kit (SDK). A good example shown in this project is how we managed to authenticate with Google Cloud but allow the option to authenticate with any other cloud service provider or even on-premises using OpenID Connect (OIDC).
- This process can be utilized to scan the entire project for vulnerabilities or even a single image, all that is needed is to change the resource URL accordingly.
- This process is not program specific, in the world of microservices, different microservices are written in different programming languages. In our example, we have demonstrated this capability on a Golang Image, but it can still be used to scan any other type of container image.
- Compared to VirusTotal which can scan and find vulnerabilities mostly in code, our solution can scan vulnerabilities in existing Docker OS. This comes with a huge advantage as security testing is mostly focused on code/software. One can easily change the project or image to be scanned easily by inserting the project or image name into the required field on the YAML Script.

IV. RESULTS

As noted earlier, for demonstration purposes, two images `(Capstone – Good Image and Capstone – Vulnerable Image)` were utilized to demonstrate the functionality of our Python script detecting vulnerabilities and halting deployment of the image into production, while if a good image it successfully allows deployment of the image into production.

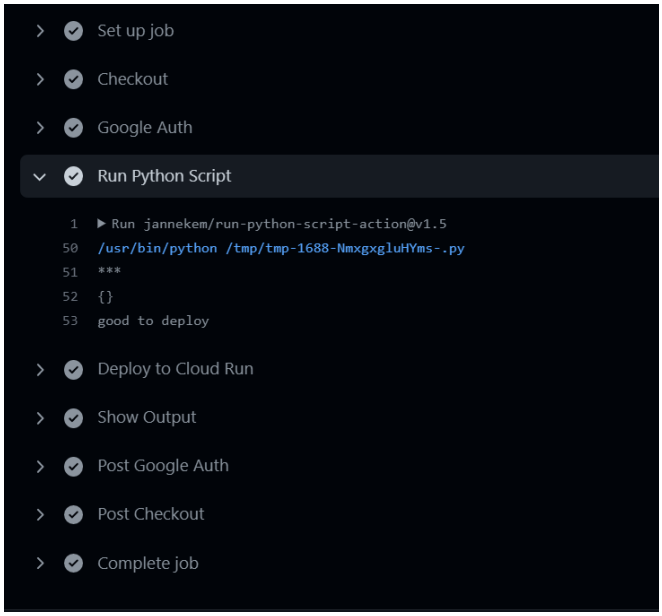


Fig 8: Image of Successful Deployment of Container Image without vulnerabilities

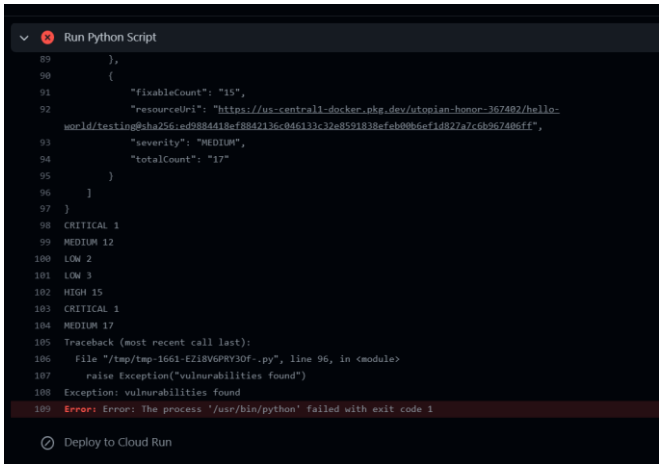


Fig 9: Image displaying python script detecting vulnerabilities and their severity

Fig 10: Image displaying scan results based on the severity

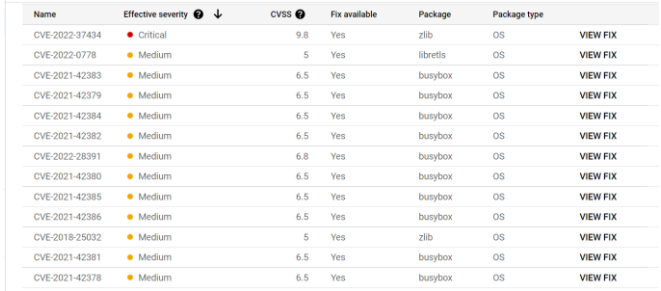


Fig 11: Image displaying more details on vulnerabilities and suggested fix

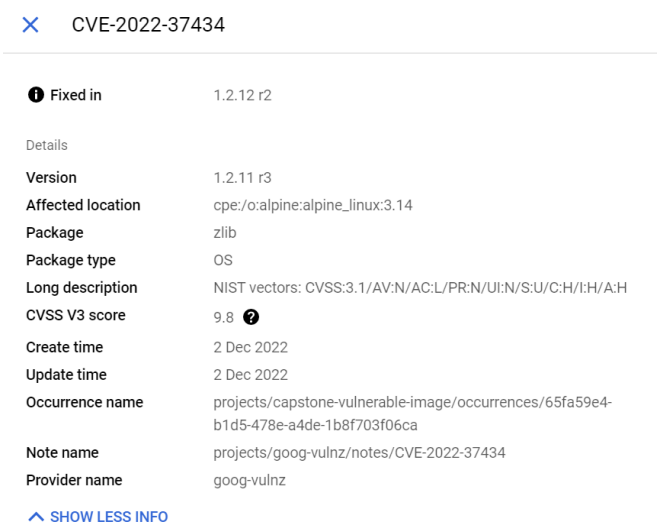


Fig 12: Image highlighting a vulnerability severity and suggested fix based on CVE standards

V. CONCLUSION AND FUTURE WORK

This is an ongoing project that currently focuses on the integration of various components/resources already provided by cloud service providers to ensure security compliance is met accordingly. We have introduced the use of Python script for CI/CD automation for vulnerability scanning in proprietary images, which is useful in detecting and classifying detected vulnerability based on severity to the container image purpose which will help with critical decision making on either deploying or halt deploying the container to production. In the near future, more work will be focused on continual vulnerability scanning and extraction while the container image is deployed and in production. It is also important to note the need to implement the Principle of Least Privilege (PoLP) when assigning access based on role to service account holders. As demonstrated by our result, detected vulnerabilities can be fixed by matching these vulnerabilities to the OWASP and CVE standards that suggest a possible fix for recorded vulnerabilities, this way it helps IT Managers or businesses to better prepare and handle container images being deployed into production to reduce attack surfaces for exploits by bad actors out there.

ACKNOWLEDGMENT

Special appreciation to Prof. Xinli Wang and to the anonymous reviewers. Their comments played a helpful role in improving the soundness of this project. This work is in partial fulfillment of my Cybersecurity Graduate Program at the School of Computing and Information Systems – Grand Valley State University.

REFERENCES

[1] Python GH-Action. Last retrieved from https://github.com/jannekem/run-python-script-action on November 23, 2022.

[2] JSON Output – Import. Last retrieved from https://docs.python.org/3/library/json.html on November 23, 2022.

[3] GitHub Action – Google Cloud Setup in Github. Last retrieved from https://github.com/google-github-actions/setup-gcloud on November 23, 2022.

[4] Enabling Keyless Authentication from GitHub Action. Last retrieved from https://cloud.google.com/blog/products/identity-security/enabling-keyless-authentication-from-github-actions on November 22, 2022.

[5] Deploy to Cloud Run (Google Cloud Platform). Last retrieved from https://cloud.google.com/run/docs/quickstarts/deploy-container on November 22, 2022.

[6] Golang Docker Official Image. Last retrieved from https://hub.docker.com/_/golang/tags on November 20, 2022.

[7] Google Cloud Platform Container Analysis API. Last retrieved from https://cloud.google.com/container-analysis/docs/container-analysis on November 20, 2022.