Product Requirements and Technical Tasks Document

Prepared by: Kranthi Kumar Chimata

Date: 23-04-2025

Task 1: Vulnerability Insights Dashboard

1. Objective

The goal of the Vulnerability Insights Dashboard is to help security professionals and developers identify vulnerabilities in container images (e.g., Docker images) and fix them based on their severity. This will help improve the security of deployed applications and prevent potential breaches.

2. Target Users

- Security Analysts: They need a way to find and fix vulnerabilities quickly.

- Developers: They need to ensure that the images they use in production are secure and free from known vulnerabilities.

3. User Stories

- As a security engineer, I need to see a list of container images and their vulnerabilities so that I can identify which images need immediate attention.

- As a developer, I want to filter images by severity (Critical, High, Low) so I can focus on fixing the most important vulnerabilities first.

- As a user, I want to click on an image and see detailed information about the vulnerabilities so I can understand the problem and know how to fix it.

4. Key Features

- Dashboard View: Lists container images with number of vulnerabilities by severity.

- Filters: Filter images by severity to prioritize critical issues.

- Detailed Image View: Shows breakdown of vulnerabilities, including severity, description, and fix suggestions.

- Fix Suggestions: Provides recommendations or links for remediation where available.

5. Success Metrics

- Time to Fix Critical Vulnerabilities: Measure how long it takes to fix high-severity issues.

- Number of Vulnerabilities Fixed: Track the volume of resolved issues.

- User Satisfaction: Gather user feedback (via surveys, forms, or interviews) to assess the tool's helpfulness and usability.

Task 2: Kubernetes Security Scan

Objective

You are required to set up a **local Kubernetes cluster** (using **Minikube**, **K3s**, or **Kind**), and then perform a security scan on the cluster using a tool like **Kubescape** or any other security scanning tool.

Steps

Step 1: Set Up Minikube (Kubernetes Cluster)

- First, install **Minikube** on your system. You can follow the official installation instructions from the Minikube website.
- Once you have Minikube installed, you can start your local Kubernetes cluster by running the following command in your terminal:

```
minikube start
```

- Once you have Minikube installed, you can start your local Kubernetes cluster by running the following command in your terminal:

Step 2: Install Kubescape

Kubescape is a tool that checks if your Kubernetes setup has any security problems. You need to install it so you can scan your Minikube Kubernetes cluster for vulnerabilities.

- Use the command below to install Kubescape:

```
curl -s
https://raw.githubusercontent.com/armosec/kubescape/master/install.sh |
sudo bash
```

Step 3: Run the Security Scan

Now that Kubescape is installed, you will use it to scan your Kubernetes setup for security issues.

- In your terminal, run the following command:

```
kubescape scan framework nsa
```

Explanation: This command tells Kubescape to scan your Kubernetes setup using the **NSA framework**, which checks for security best practices.

Kubescape will scan your local Kubernetes cluster for security vulnerabilities and provide the findings.

Step 4: Export the Scan Results

Once the scan is completed, save the results to a **JSON format** file which is what the company asked for

- To save the results, run this command:

```
kubescape scan framework nsa --format json > findings.json
```

Explanation: This command saves the security scan results into a **JSON file** called `findings.json`.

Step 5: Submit the findings.json file

- Once you have the **`findings.json`** file, attach it to your submission.

- Ensure that the file is properly named as `findings.json`.

Summary

1. **Set up Minikube** to run a local Kubernetes cluster.
2. **Install Kubescape** to scan your cluster for security issues.

3. **Run the scan** with the NSA framework.
4. **Export the results** to a JSON file using:
   `kubescape scan framework nsa --format json > findings.json`
5. **Submit the findings.json file**.

Example Command for Kubescape:

Here's an example of how to run the scan and export the results:

```
# Install Kubescape

curl -s
https://raw.githubusercontent.com/armosec/kubescape/master/install.sh
| sudo bash

# Run the security scan

kubescape scan framework nsa

# Export the results in JSON format

kubescape scan framework nsa --format json > findings.json
```

Task 3: GoLang Program and Kubernetes Deployment

Objective

Build a GoLang web app that displays the current date and time. Containerize the app using Docker, push the image to DockerHub, deploy it on Kubernetes with 2 replicas, and expose it to the internet.

Steps :

1. **GoLang Web App**
   A simple GoLang program that shows the current date and time in a browser.

   Code used (`main.go`):

```
package main

import (
        "fmt"
```

```
        "net/http"
        "time"
)

func handler(w http.ResponseWriter, r *http.Request) {
        fmt.Fprintf(w, "Current Date and Time: %s", time.Now().Format("2006-01-02
15:04:05"))
}

func main() {
        http.HandleFunc("/", handler)
    http.ListenAndServe(":8080", nil)
}
```

Dockerfile

Created a **`Dockerfile`**:

```
FROM golang:1.18
WORKDIR /app
COPY . .
RUN go build -o main .
CMD ["./main"]
```

Build & Push Image

- Built the Docker image:

  ```
  docker build -t kranthi1176/go-time-app .
  ```

- Pushed the image to DockerHub:

  ```
  docker push kranthi1176/go-time-app
  ```

Kubernetes Deployment

Deployment file (`deployment.yaml`):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: go-time-app
spec:
  replicas: 2
  selector:
      matchLabels:
      app: go-time-app
  template:
      metadata:
      labels:
      app: go-time-app
      spec:
      containers:
      - name: go-time-container
      image: kranthi1176/go-time-app
      ports:
      - containerPort: 8080
```

Exposed the App with a Service

Service file (`service.yaml`):

```
apiVersion: v1
kind: Service
metadata:
  name: go-time-service
spec:
  selector:
      app: go-time-app
  ports:
      - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```

Command to Apply

- Applied using:

```
kubectl apply -f deployment.yaml
kubectl apply -f service.yaml
```

  - If using Minikube:

```
minikube service go-time-service
```

Result / Output

- App displays current date and time in the browser

- Runs with 2 replicas on Kubernetes

- Exposed to the internet using LoadBalancer (via Minikube or cloud setup)