**Product Requirements Document**

**Title: Product Requirement and Low-Fidelity Wireframes**

**1. Overview**

This document details the requirements for a security product designed to scan container images and report vulnerabilities. The product aims to assist users in identifying vulnerable container images, evaluating the severity of these vulnerabilities, and prioritizing the remediation of critical issues.

**2. Objectives**

* Provide a clear overview of vulnerabilities in container images.
* Enable users to quickly identify and prioritize images requiring immediate attention.
* Ensure the interface is user-friendly and intuitive.

**3. User Stories**

1. **As a user**, I need a dashboard to view an overview of all container images and their vulnerabilities.
2. **As a user**, I need to filter container images based on vulnerability severity.
3. **As a user**, I need to see detailed information about each vulnerability.
4. **As a user**, I need to receive alerts for critical and high vulnerabilities.
5. **As a user**, I need to track the status of vulnerability fixes.

**4. Functional Requirements**

1. **Dashboard View**: Display an overview of all container images and their vulnerability statuses.
2. **Image List View**: Show a detailed list of container images, including the number of vulnerabilities and their severities.
3. **Filtering and Sorting**: Provide options to filter and sort images by severity, name, and scan date.
4. **Vulnerability Details**: Present detailed information for each vulnerability, including severity, description, and recommended actions.
5. **Notifications**: Send alerts for critical and high vulnerabilities.
6. **Fix Tracking**: Track the status of vulnerability remediation efforts.

**5. Non-Functional Requirements**

1. **Performance**: Ensure the system handles large repositories efficiently.
2. **Usability**: Design an intuitive and easy-to-navigate interface.
3. **Scalability**: Ensure the system can scale to accommodate growing numbers of images and vulnerabilities.

**6. User Interface**

* **Dashboard**: Summarizes the total number of images, vulnerabilities by severity, and recent alerts.
* **Image List**: A tabular view displaying each image, the number of vulnerabilities, their severity, and the last scan date.
* **Image Details**: Detailed view of vulnerabilities for each image, including severity, descriptions, and recommended fixes.
* **Notifications**: Panel showing recent alerts for critical and high vulnerabilities.
* **Filter Options**: Dropdowns and checkboxes for filtering images by severity, name, and date.

**Low-Fidelity Wireframes**

Low-fidelity wireframes are essential in visualizing the layout and functionality of the user interface without focusing on detailed design elements. Here are the wireframes for the main screens of the product.

**1. Dashboard**

**Description**: The dashboard gives a snapshot of the security status of all container images, displaying key metrics and visual summaries.

**Elements**:

* **Total Images Count**: A prominent display of the total number of container images.
* **Vulnerabilities by Severity**: A visual summary (e.g., bar or pie chart) showing the distribution of vulnerabilities by severity.
* **Recent Alerts**: A list of the latest alerts for critical and high vulnerabilities.
* **Quick Access Links**: Buttons for quick navigation to key actions like "View All Images" and "Scan New Image".

**2. Image List View**

**Description**: This view lists all container images with details about their vulnerabilities, allowing users to sort and filter as needed.

**Elements**:

* **Image Name**: The name or identifier of the container image.
* **Last Scan Date**: The date the image was last scanned.
* **Number of Vulnerabilities**: A count of vulnerabilities for each image, broken down by severity.
* **Filter and Sort Options**: Tools for filtering and sorting images by various criteria.
* **Search Bar**: A search input for quickly finding specific images.

**3. Image Details View**

**Description**: This view provides comprehensive details about the vulnerabilities found in a specific container image.

**Elements**:

* **Image Name**: The name or identifier of the container image.
* **Vulnerability List**: Detailed information on each vulnerability, including severity, description, and recommended fixes.
* **Fix Status**: Indicators showing the status of remediation efforts.
* **Action Buttons**: Options to mark vulnerabilities as fixed or to rescan the image.

**4. Notifications Panel**

**Description**: Displays recent alerts for critical and high vulnerabilities, allowing users to quickly access detailed information.

**Elements**:

* **Alert List**: Each alert includes the image name, vulnerability severity, and a brief description.
* **Link to Details**: Directs users to the detailed view of the vulnerability.

**5. Filter Options**

**Description**: These options allow users to filter the image list based on various criteria to quickly find images with specific vulnerabilities.

**Elements**:

* **Dropdowns for Severity**: Filters to select vulnerabilities by severity (Critical, High, Medium, Low).
* **Image Name**: Filter based on the name of the container image.
* **Date**: Filter based on the date the image was last scanned.
* **Checkboxes**: For selecting multiple filters simultaneously.
* **Apply and Reset Filters Buttons**: To apply the selected filters or reset them to default.

**Development Action Items**

1. **Database Design**: Create a database schema to store container images, vulnerabilities, and their relationships.
2. **API Development**: Develop APIs to fetch container images, retrieve vulnerability data, and update the fix status.
3. **Frontend Development**: Build the user interface based on the provided wireframes.
4. **Notification System**: Implement a system to send real-time notifications for critical and high vulnerabilities.
5. **Filtering and Sorting**: Develop filtering and sorting functionality on the frontend to allow users to organize images based on various criteria.
6. **Performance Optimization**: Ensure the system performs efficiently even with large volumes of data.
7. **Scalability Testing**: Conduct scalability testing to ensure the system can handle an increasing number of images and vulnerabilities.

### Case Study: Kubernetes Security Scan Using Kubescape

#### Introduction

In this case study, I will walk you through the process of setting up a local Kubernetes cluster using Minikube and performing a security scan with the Kubescape tool. The primary objective is to identify potential security vulnerabilities within the cluster and document the findings in a JSON file. This exercise not only demonstrates the process but also emphasizes the importance of continuous security scanning in maintaining a secure Kubernetes environment.

#### Background

As container orchestration becomes more prevalent, securing Kubernetes clusters is paramount. Kubernetes manages containerized applications across a cluster of machines, ensuring that they run efficiently and reliably. However, like any system, it is susceptible to security risks. Tools like Kubescape help identify these risks by scanning the cluster and providing detailed reports on vulnerabilities and misconfigurations.

#### Task

The task involves:

1. Installing a local Kubernetes cluster using Minikube.
2. Using Kubescape to perform a security scan on the cluster.
3. Generating and documenting the findings in a JSON file.

#### Deliverables

A JSON file containing the Kubernetes security findings.

#### Step-by-Step Process

##### 1. Installing Minikube

**Minikube** is a tool that allows you to run Kubernetes locally. It creates a single-node Kubernetes cluster on your local machine, which is perfect for development and testing purposes.

**Installation Steps:**

1. **Install Minikube**:
   * For Windows, use the installer available here.
   * For macOS, use Homebrew:

brew install minikube

* + For Linux, download the latest Minikube release and install it:

https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

sudo install minikube-linux-amd64 /usr/local/bin/minikube

1. **Start Minikube**:
   * Open a terminal and run:

minikube start

* + This command initializes a single-node Kubernetes cluster on your local machine.

##### 2. Installing Kubescape

**Kubescape** is an open-source tool that performs Kubernetes cluster scans to identify vulnerabilities and misconfigurations.

**Installation Steps:**

1. **Download and Install Kubescape**:
   * Run the following command to install Kubescape:

https://raw.githubusercontent.com/armosec/kubescape/master/install.sh | /bin/bash

1. **Verify Installation**:
   * Ensure Kubescape is installed correctly by checking the version:

kubescape version

##### 3. Performing a Security Scan with Kubescape

1. **Run the Security Scan**:
   * Execute the following command to scan your Minikube cluster:

kubescape scan --submit

* + This command will scan the cluster and print the results to the terminal.

1. **Save the Findings to a JSON File**:
   * To save the findings to a JSON file, use:

kubescape scan --format json --output results.json

* + This command generates a file named results.json containing the security findings.

##### 4. Analyzing the JSON Output

Once the scan is complete, open the results.json file to review the findings. Here’s an example of what the JSON output might look like:

json

{

"clusterName": "minikube",

"namespace": "default",

"controls": [

{

"id": "C-001",

"name": "Control 1",

"description": "Ensure that the API server is secure.",

"severity": "High",

"status": "Failed",

"findings": [

{

"resource": "kube-apiserver",

"message": "API server is accessible over HTTP."

}

]

},

{

"id": "C-002",

"name": "Control 2",

"description": "Ensure that etcd is secure.",

"severity": "Medium",

"status": "Passed",

"findings": []

}

]

}

This output includes critical details such as the cluster name, namespace, control IDs, descriptions, severity levels, status, and specific findings.

#### Summary and Conclusion

In this case study, we successfully set up a local Kubernetes cluster using Minikube and performed a security scan with Kubescape. The findings were documented in a JSON file, which provided a comprehensive overview of the security posture of the cluster.

Continuous security scanning is crucial for maintaining a secure Kubernetes environment. By regularly scanning clusters for vulnerabilities and misconfigurations, organizations can proactively address potential security issues before they are exploited.

#### Action Items

Based on the findings, the following action items are recommended:

1. **Address High Severity Issues**: Immediate attention should be given to high-severity vulnerabilities, such as securing the API server.
2. **Review and Fix Medium Severity Issues**: Although less critical, medium severity issues should also be addressed to enhance overall security.
3. **Implement Continuous Scanning**: Establish a routine for regular security scans to ensure the cluster remains secure over time.
4. **Educate and Train Team**: Ensure that the development and operations teams are aware of best practices for securing Kubernetes clusters.

This exercise has not only provided insights into the current security status of the Kubernetes cluster but also highlighted the importance of ongoing security assessments in a dynamic environment.

### Problem Statement 3 (Technical): Hosting a GoLang Program with Docker and Kubernetes

#### Step #1: Create a GoLang Program and Host on GitHub

1. **Create the GoLang Program:** Create a simple GoLang web application that displays the current date and time.

// main.go

package main

import (

"fmt"

"net/http"

"time"

)

func currentDateTime(w http.ResponseWriter, r \*http.Request) {

fmt.Fprintf(w, "Current Date and Time: %s", time.Now().Format(time.RFC1123))

}

func main() {

http.HandleFunc("/", currentDateTime)

fmt.Println("Starting server on port 8080...")

http.ListenAndServe(":8080", nil)

}

1. **Create a Dockerfile:** Write a Dockerfile to containerize the GoLang application.

# Dockerfile

FROM golang:1.18-alpine

WORKDIR /app

COPY . .

RUN go build -o main .

EXPOSE 8080

CMD ["./main"]

1. **Host the Code on GitHub:**
   * Create a new repository on GitHub.
   * Push the GoLang code and Dockerfile to the repository.

git init

git remote add origin https://github.com/yourusername/your-repo.git

git add .

git commit -m "Initial commit"

git push -u origin master

1. **Build and Push the Docker Image to DockerHub:**
   * Build the Docker image and push it to DockerHub.

docker build -t yourusername/your-repo:latest .

docker login

docker push yourusername/your-repo:latest

#### Step #2: Deploy the Container with 2 Replicas to Kubernetes

1. **Create Kubernetes Deployment YAML:** Write a deployment YAML file to deploy the application with 2 replicas.

# deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: datetime-app

spec:

replicas: 2

selector:

matchLabels:

app: datetime-app

template:

metadata:

labels:

app: datetime-app

spec:

containers:

- name: datetime-app

image: yourusername/your-repo:latest

ports:

- containerPort: 8080

1. **Deploy the Application to Kubernetes:**
   * Apply the deployment YAML to create the deployment in your Kubernetes cluster.

kubectl apply -f deployment.yaml

#### Step #3: Expose the App to the Internet

1. **Create a Service to Expose the Application:** Write a service YAML file to expose the application to the internet.

# service.yaml

apiVersion: v1

kind: Service

metadata:

name: datetime-service

spec:

type: LoadBalancer

selector:

app: datetime-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

1. **Apply the Service Configuration:**
   * Apply the service YAML to create a LoadBalancer service in your Kubernetes cluster.

kubectl apply -f service.yaml

1. **Get the External IP:**
   * After the service is created, get the external IP to access the application.

kubectl get services datetime-service

The external IP will be listed under the EXTERNAL-IP column. Use this IP to access your application from the internet.

#### Resources

* **Qwiklabs:** <https://www.qwiklabs.com/>
* **Play with Kubernetes:** <https://labs.play-with-k8s.com/>

By following these steps, we can create a simple GoLang web application that displays the current date and time, containerize it using Docker, deploy it to a Kubernetes cluster with 2 replicas, and expose it to the internet.