The document contains three problem statements for a Product Manager. Here are the solutions for each problem statement:

Problem Statement 1: Product Requirement and Low-Fidelity Wireframes

Background/Task: A security product requires scanning container images and showing users the findings. Container images contain applications with their dependencies, and these components might have known vulnerabilities. Users need to understand which container images have vulnerabilities and how severe they are. If there are any critical or high vulnerabilities, they need to fix them and identify which images need to be fixed. Users have thousands of images in their repository.

Deliverables:

1. **Product Requirements Document:**
    - **User Stories:**
        - As a user, I want to scan container images to identify vulnerabilities.
        - As a user, I want to see the severity of vulnerabilities in container images.
        - As a user, I want to filter images based on vulnerability severity.
        - As a user, I want to prioritize fixing critical and high vulnerabilities.
    - **Functional Requirements:**
        - The system should allow users to upload and scan container images.
        - The system should display a list of scanned images with their vulnerability status.
        - The system should categorize vulnerabilities by severity (e.g., critical, high, medium, low).
        - The system should provide detailed information about each vulnerability.
        - The system should allow users to filter and sort images based on vulnerability severity.
    - **Non-Functional Requirements:**
        - The system should handle large repositories with thousands of images efficiently.

        - The system should provide real-time scanning and reporting.
        - The system should ensure data security and privacy.
    - **Wireframes:**
        - **Dashboard:** Overview of scanned images, summary of vulnerabilities.
        - **Image Details Page:** Detailed view of vulnerabilities for a selected image.
        - **Filter/Sort Options:** Options to filter and sort images based on severity.
2. **Low-Fidelity Wireframes:**
    - **Dashboard:** !Dashboard Wireframe
    - **Image Details Page:** !Image Details Wireframe
    - **Filter/Sort Options:** !Filter/Sort Wireframe
3. **Development Action Items:**
    - Set up a container image scanning tool.
    - Develop the user interface for uploading and scanning images.
    - Implement the backend logic for scanning and reporting vulnerabilities.
    - Create APIs for fetching and displaying vulnerability data.
    - Develop filtering and sorting functionalities.
    - Ensure scalability and performance optimization.

Problem Statement 2: Kubernetes Security Scan

Background/Task: Install a local K8s cluster (such as Minikube, K3s, Kind, etc.) and use a tool such as Kubescape (or any other tool) to scan for findings and send the list of the findings.

Deliverables:

- **JSON File Containing the K8s Findings:**
    - Install Minikube:

      minikube start
    - Install Kubescape:

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

- Scan the cluster:

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

- The results.json file will contain the findings.

Problem Statement 3: Technical Task

Step #1: Create a GoLang Program

- **GoLang Program:**

```
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().For
mat(time.RFC1123))
}

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

- **Push to DockerHub:**

  - Create a Dockerfile:

```
FROM golang:alpine
WORKDIR /app
COPY . .
RUN go build -o main .
```

  - Build and push the Docker image:

    bash docker build -t yourusername/date-time-app .

    docker push yourusername/date-time-app

Step #2: Deploy the Container with 2 Replicas to K8s

- **K8s Deployment YAML:**

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

Step #3: Expose the App to the Internet

K8s Service YAML:

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