Product Requirements Document (PRD)

For Problem Statement 1

1. Introduction

- **Title:** Container Image Vulnerability Scanner
- **Objective:** To provide a solution for scanning container images for known vulnerabilities and helping users identify and fix critical and high vulnerabilities in their image repository.

2. Background

- **Problem:** Container images contain applications and dependencies that might have known vulnerabilities. Users need a way to scan these images, understand which ones have vulnerabilities, and prioritize fixes based on the severity of these vulnerabilities.
- **Target Users:** DevOps teams, Security teams, and Developers who manage containerized applications.

3 . User Stories

- **As a DevOps engineer**, I want to quickly detect which images in my repository have critical vulnerabilities so I can prioritize fixing them.
- **As a security lead**, I need to ensure all images meet our security standards and fix critical issues before deploying applications.
- **As a developer**, I want an easy way to see the steps needed to fix any vulnerabilities found in my container images

User Goals

- **Identify vulnerabilities:** Users want to quickly and accurately detect vulnerabilities within their container images.
- **Prioritize risks:** Users need to prioritize vulnerabilities based on their severity and potential impact. (e.g., critical, high, medium, low).
- **Understand remediation steps:** Users should be able to easily understand the recommended steps to address identified vulnerabilities.
- **Manage large-scale deployments:** Users with thousands of images require a scalable solution to efficiently scan and manage their container image security.

4. Functional Requirements

- **Image Scanning**: The system should scan container images for known vulnerabilities.
- **Severity Classification**: The system should classify vulnerabilities into categories (critical, high, medium, low).
- **Dashboard**: A dashboard to display the scan results, including the number of vulnerabilities and their severity.
- **Filtering and Sorting**: Users should be able to filter and sort images based on vulnerability severity.
- **Notifications**: The system should send notifications for critical vulnerabilities.
- **Reporting**: Generate reports summarizing the scan results.

5. Non-Functional Requirements

- **Performance**: The system should be able to scan thousands of images efficiently.
- **Usability**: The interface should be intuitive and easy to navigate.
- **Security**: Ensure that the scanning process does not introduce new vulnerabilities.

6. Technical Requirements

- **Integration**: Integrate with popular container registries (e.g., Docker Hub, AWS ECR).
- **Scalability**: The system should handle large repositories with thousands of images.
- **API**: Provide an API for integrating with CI/CD pipelines.

7. Prioritization of Requirements (MVP vs. Future Enhancements)

- **MVP**: Image scanning, severity classification, dashboard, filtering and sorting, notifications for critical vulnerabilities, API integration.
- **Future Enhancements**: Additional integrations (e.g., private registries), deeper remediation insights, advanced reporting and analytics, and machine learning-based vulnerability predictions.

8. Risk Considerations

- **False Positives/Negatives**: Ensure the vulnerability detection mechanism is fine-tuned to avoid false positives that may cause unnecessary alarms or false negatives that may overlook critical issues.
- **Performance Bottlenecks**: For large repositories, scanning thousands of images may cause bottlenecks. Implementing distributed scanning processes will mitigate this risk.
- **User Fatigue**: Too many notifications or overly complex dashboards could overwhelm users. Ensure the user interface is simple and notifications are actionable.

Development Action Items

1. **Define Scanning Algorithms:**
    - Identify and integrate vulnerability databases.
    - Develop algorithms for vulnerability detection and severity assessment.
2. **Develop User Interface:**
    - Create wireframes and UI mockups.
    - Develop frontend components for dashboards, lists, and detailed views.
3. **Backend Development:**
    - Develop APIs for scanning images and retrieving results.
    - Implement data storage solutions.
4. **Integration:**
    - Integrate with CI/CD tools and container registries.
5. **Testing:**
    - Test scanning accuracy and performance.
    - Conduct usability testing for the user interface.
6. **Documentation:**
    - Prepare user documentation and system integration guides.

Conclusion

This PRD and development plan aim to deliver a robust, user-friendly, and scalable solution for scanning container images, identifying vulnerabilities, and helping users prioritize and fix critical security risks. The development action items provide a clear roadmap to building the solution, with key focus areas like performance, usability, and integration.

For Problem Statement 2

Kubescape was chosen to perform a comprehensive security scan on the Kubernetes cluster due to its alignment with security frameworks like NSA and CIS. The findings in **results.json** highlight key vulnerabilities and misconfigurations within the cluster, along with remediation steps.