Problem Statement 1:

Title: Container Image Vulnerability Scanner

1. Overview

The Container Image Vulnerability Scanner is designed to help DevOps and security teams easily identify and fix security issues within their containerized applications. By scanning container images for vulnerabilities, it provides clear, actionable insights so teams can prioritize and address critical risks quickly. With a user-friendly dashboard, it makes managing security across large repositories simple. Integrated with existing DevOps workflows, the scanner ensures that container images are kept secure and up to date, giving teams peace of mind by automatically monitoring for new vulnerabilities and alerting them when action is needed.

1. Objectives

* Identify vulnerabilities in container images and display them in a user-friendly dashboard.
* Prioritize remediation by highlighting critical and high-severity vulnerabilities.
* Enable real-time notifications for critical vulnerabilities, reducing response time.
* Streamline DevOps workflows with actionable insights, fixing guidance and automated scanning.

1. User Stories

* As a DevOps Engineer, I want to scan thousands of container images in my repository for vulnerabilities so that I can identify and address critical issues.
* As a Security Manager, I need to receive real-time alerts when critical vulnerabilities are detected, so I can take immediate action.
* As an IT Administrator, I want to see a dashboard that summarizes vulnerability trends and allows me to filter by severity and fix availability.
* As a Software Engineer, I need to know if fixes for vulnerabilities are available to speed up remediation.

1. Features and Functional Requirements

Feature 1: Vulnerability Scanning

* Description: The system scans container images in the repository for known vulnerabilities using up-to-date CVE (Common Vulnerabilities and Exposures) databases.
* Functional Requirements:
  + The system must scan container images and list vulnerabilities based on severity (Critical, High, Medium, and Low).
  + Users can initiate manual scans or schedule periodic scans.
  + Vulnerability data should be continuously updated from external sources (CVE databases).
* Feature 2: Dashboard
  + Description: A centralized view where users can monitor the vulnerability status of all container images.
  + Functional Requirements:
    - The dashboard must display total scanned images, vulnerabilities grouped by severity (Critical/High/Medium/Low), and the percentage of images affected.
    - The dashboard must include a heatmap to visually represent the distribution of vulnerabilities.
    - A table should list each container image with details such as Image Name, Last Scan Date, Severity of Vulnerabilities, Fix Availability, and Rescan button.
    - Users can filter the list by severity, image name, date, or fix availability.
* Feature 3: Detailed Vulnerability Report
  + Description: A detailed report for each container image, showing the vulnerabilities, severity, and available fixes.
  + Functional Requirements
    - The report must show a table with the following fields: CVE ID, Component Affected, Severity, Description, and Fix Availability.
    - Users should be able to filter the vulnerabilities within the report by severity and fix availability.
    - Users can trigger rescans or view detailed remediation steps for each vulnerability.
* Feature 4: Alerts and Notifications
  + Description: Real-time alerts for critical vulnerabilities and scheduled reports.
  + Functional Requirements:
    - The system should send notifications (email or in-app) for critical vulnerabilities detected during a scan.
    - Users can set up alerts based on severity levels (e.g., only for Critical/High vulnerabilities).
    - Scheduled daily/weekly reports summarizing vulnerability trends should be available.
* Feature 5: Fix Availability Indicator
  + Description: Indicate whether a fix is available for a detected vulnerability.
  + Functional Requirements:
    - Vulnerabilities listed should include a “Fix Available” indicator, highlighting if a patch or upgrade is possible.
    - The product must provide recommendations for remediation, such as updating a specific dependency or library.

1. Non-Functional Requirements (NFRs)

* Performance: The product should scan container images with an average time of less than 30 seconds per image.
* Scalability: The system must handle repositories with up to 10,000 container images and maintain performance standards.
* Security: All data transfers and access to the vulnerability scanner should be encrypted (e.g., using TLS) and require user authentication.
* Usability: The user interface should be intuitive and easy to navigate, with minimal training required for DevOps engineers and security professionals.

1. Success Metrics/KPIs

* Scan Performance: 90% of container images should be scanned in under 30 seconds.
* Adoption: 80% of users should be able to identify and remediate critical vulnerabilities within 24 hours.
* Fix Rate: Reduce the average vulnerability remediation time by 30% within the first quarter.
* User Engagement: At least 75% of users should opt-in for real-time notifications and weekly summary reports.

1. Assumptions

* Users will have basic knowledge of container image management and vulnerability handling.
* The system will integrate with common container registries (e.g., Docker Hub, Amazon ECR).
* The CVE database used for vulnerability detection is regularly updates and maintained by external provider.

1. Dependencies

* CVE Database: Requires regular synchronization with an external CVE database for accurate vulnerability identification.
* Container Registries: Integration with container registries (Docker Hub, Amazon ECR, etc.)
* Notification Services: Integration with email and notification services for sending alerts to users.

1. Timeline/Milestones

* Week 1: Design low-fidelity wireframes and gather feedback from stakeholders.
* Week 4: Develop a beta version for internal testing with real-time scanning and dashboard features.
* Week 8: Finalize integration with container registries and external CVE databases.
* Week 12: Full product launch with vulnerability scanning, alerts, and remediation features.

1. Risks

* Outdated CVE Data: If the CVE database is not updated regularly, scan results could be inaccurate. Mitigation: Ensure regular synchronization with the CVE database.
* Performance Bottlenecks: Scanning large repositories could lead to slow performance. Mitigation: Optimize the backend to handle parallel scans and reduce response times.
* False Positives: Incorrect vulnerability identification could lead to unnecessary fixes. Mitigation: Implement a mechanism to manually validate vulnerabilities before suggesting fixes.

1. Stakeholders

* Product Manager: Responsible for overseeing the product’s development and ensuring alignment with business goals.
* Engineering Team: Responsible for implementing the vulnerability scanner, dashboard, and notification features.
* UX Designers: Responsible for creating the user interface and ensuring a seamless user experience.
* DevOps Team: Provides feedback on functionality and ensures the scanner integrates with existing workflows.