Problem Statement 1:

Title: 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 all these components might have known vulnerabilities.

As a user:

I need to understand which container images have vulnerabilities and how severe they are.

If there are any critical or high vulnerabilities, I need to fix them and thus need to identify which images have to be fixed.

I have thousands of images in my repository.

Help us build a product requirements/wireframe that can help users solve the above problems.

Deliverables:

Create a Product Requirements document for the above.

Create low-fidelity wireframes for the user interface for this product.

(Bonus/Optional Task) Identify development action items that can be discussed with the development team

Product Requirements Document: Container Image Vulnerability Management

1. Introduction

- **1.1 Purpose:** This document outlines the requirements for a product that helps users identify and manage vulnerabilities in container images. It will serve as a guide for the design and development of the product.
- **1.2 Goals:**
- Enable users to quickly assess the security posture of their container images.
- Provide clear information about vulnerabilities, including severity levels.
- Facilitate the identification of images that require remediation.
- Support the management of a large number of images.

2. Target Audience

- DevOps Engineers
- Security Engineers
- System Administrators
- Software Developers

3. Problem Statement

Users need an efficient way to:

- Understand which container images have vulnerabilities and their severity.
- Identify which images need to be fixed to mitigate critical or high-severity vulnerabilities.
- Manage and prioritize vulnerability remediation across a large repository of container images.

4. Proposed Solution

The product will provide a comprehensive solution for container image vulnerability management, with the following key features:

- **Vulnerability Scanning:** Automated scanning of container images for known vulnerabilities.
- **Vulnerability Reporting:** Clear and concise reports detailing identified vulnerabilities, including severity levels (Critical, High, Medium, Low), CVEs, and affected packages.
- **Image Prioritization:** Tools to help users prioritize which images to address first, based on vulnerability severity, image usage, or other criteria.
- **Filtering and Search:** Ability to filter and search for images based on various parameters, such as repository, tag, vulnerability severity, or scan status.
- **User Interface:** An intuitive and user-friendly interface for viewing scan results, managing images, and generating reports.
- **Integration:** Integration with existing container registries (e.g., Docker Registry, Harbor, AWS ECR) and CI/CD pipelines.
- **Notifications:** Ability to configure alerts and notifications for new vulnerabilities or changes in vulnerability status.

5. Functional Requirements

- **5.1 Image Scanning:**
- The system shall be able to scan container images from various registries.
- Scans should be performed automatically upon image push to a registry (if integrated) or on a scheduled basis.
- The system shall use a vulnerability database (e.g., CVE database, vendor advisories) to identify vulnerabilities.
- The system shall provide options to configure scan settings, including which vulnerabilities to scan for.
- **5.2 Vulnerability Data:**
- For each vulnerability, the system shall display:
- CVE ID
- Severity level (Critical, High, Medium, Low)
- Description of the vulnerability
- Affected package(s) and version(s)
- Recommended remediation steps
- Link to the vulnerability source (e.g., NVD)
- The system shall provide a way to update the vulnerability database.
- **5.3 Image Management:**
- The system shall display a list of scanned images, including:
- Image name and tag
- Registry

- Scan status (e.g., scanned, not scanned, error)
- Number of vulnerabilities by severity
- Last scanned date
- Users shall be able to filter the list of images by:
- Registry
- Tag
- Scan status
- Vulnerability severity
- Date range
- Users shall be able to search for images by name or tag.
- Users should be able to sort images
- **5.4 Reporting:**
- The system shall generate reports on image vulnerabilities.
- Reports should include a summary of vulnerabilities by severity.
- Reports should be exportable in various formats (e.g., PDF, CSV, JSON).
- **5.5 User Authentication and Authorization:**
- The system shall support user authentication to control access.
- The system shall support role-based access control (RBAC) to define user permissions.
- **5.6 System Configuration:**
- The system shall allow administrators to configure:
- Registry connections
- Scanning schedules
- Notification settings
- User roles and permissions
- Vulnerability database updates
- **5.7 API:**
- The system shall provide an API for integrating with other tools and systems.

6. Non-Functional Requirements

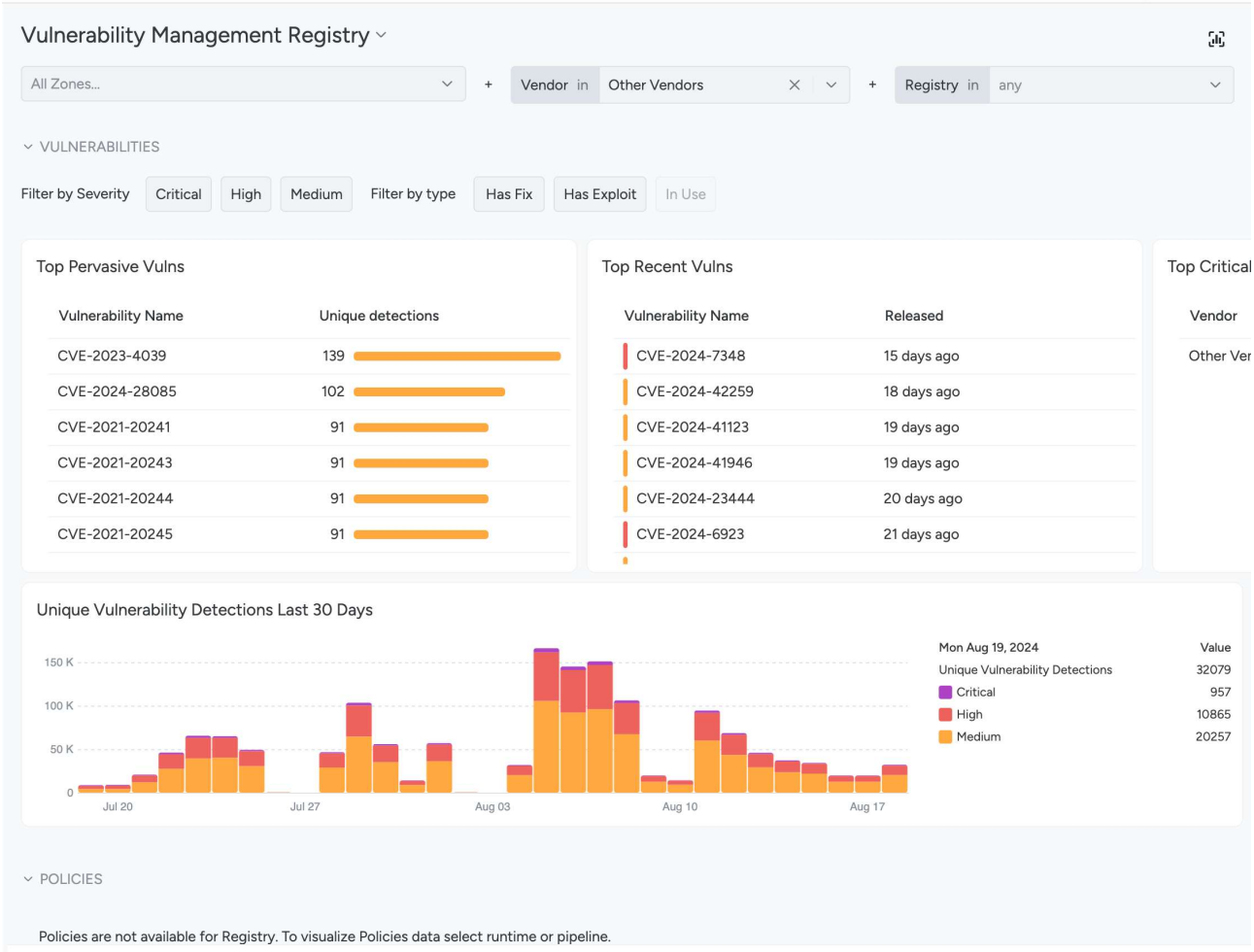
- **6.1 Performance:**
- The system shall be able to scan images quickly and efficiently.
- The system shall be able to handle a large number of images.
- The UI shall be responsive and provide a smooth user experience.
- **6.2 Scalability:**
- The system shall be scalable to support a growing number of images and users.
- **6.3 Reliability:**
- The system shall be reliable and provide accurate vulnerability information.

- **6.4 Security:**
- The system shall be secure and protect sensitive data.
- The system shall follow security best practices.
- **6.5 Usability:**
- The system shall be easy to use and understand.
- The UI shall be intuitive and user-friendly.

7. Low-Fidelity Wireframes

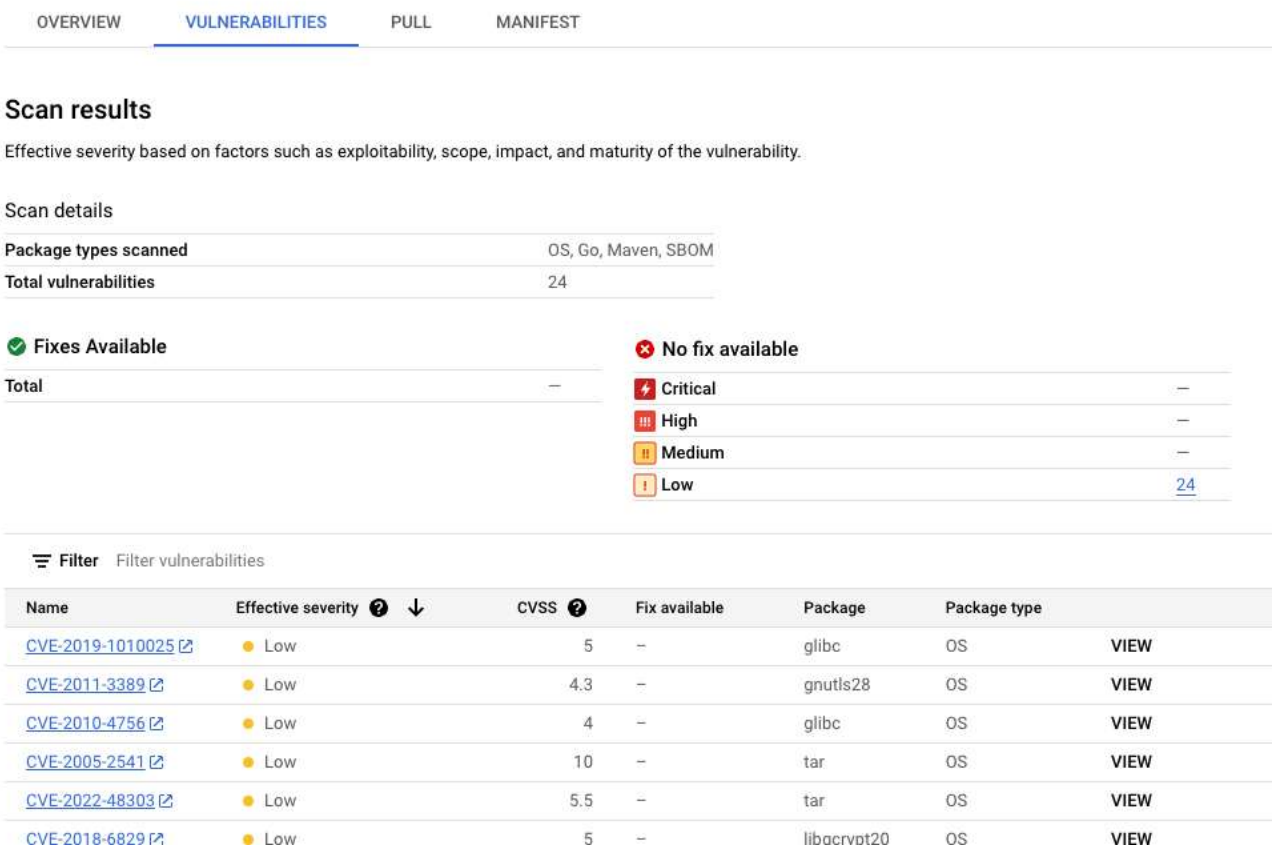
Here are some low-fidelity wireframes illustrating the user interface for the product.

- **7.1 Image Dashboard:**



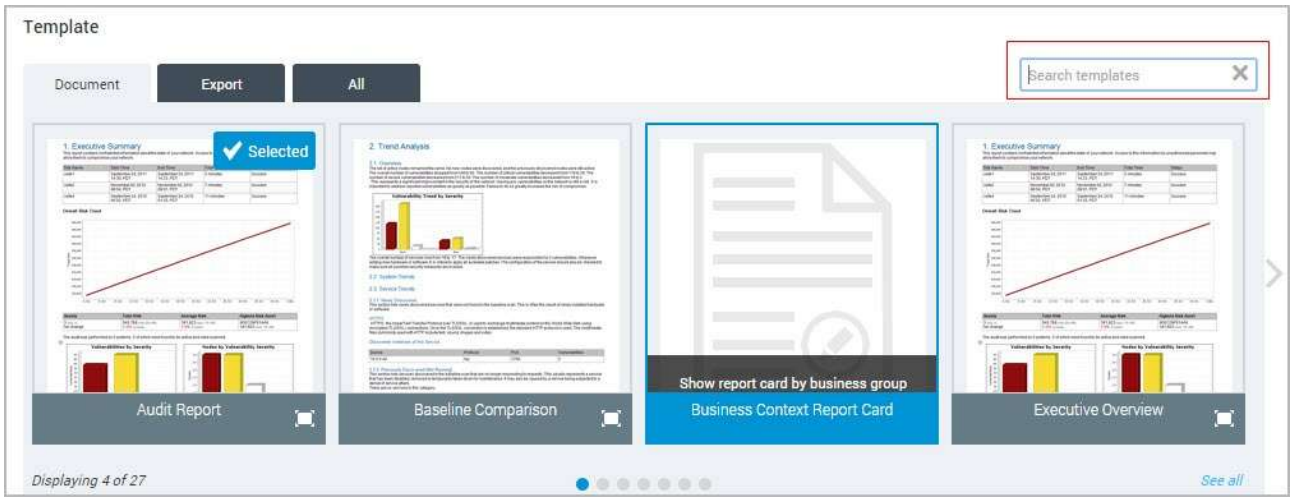
- A table listing container images. Columns include:
- Image Name/Tag
- Registry
- Scan Status
- Critical Vulnerabilities
- High Vulnerabilities
- Medium Vulnerabilities
- Low Vulnerabilities

- Last Scanned
- Filters:
- Registry (dropdown)
- Tag (dropdown)
- Scan Status (checkboxes: Scanned, Not Scanned, Error)
- Vulnerability Severity (checkboxes: Critical, High, Medium, Low)
- Search bar (for searching by image name or tag)
- Sortable columns
- Pagination for handling large numbers of images
- **7.2 Image Details Page:**

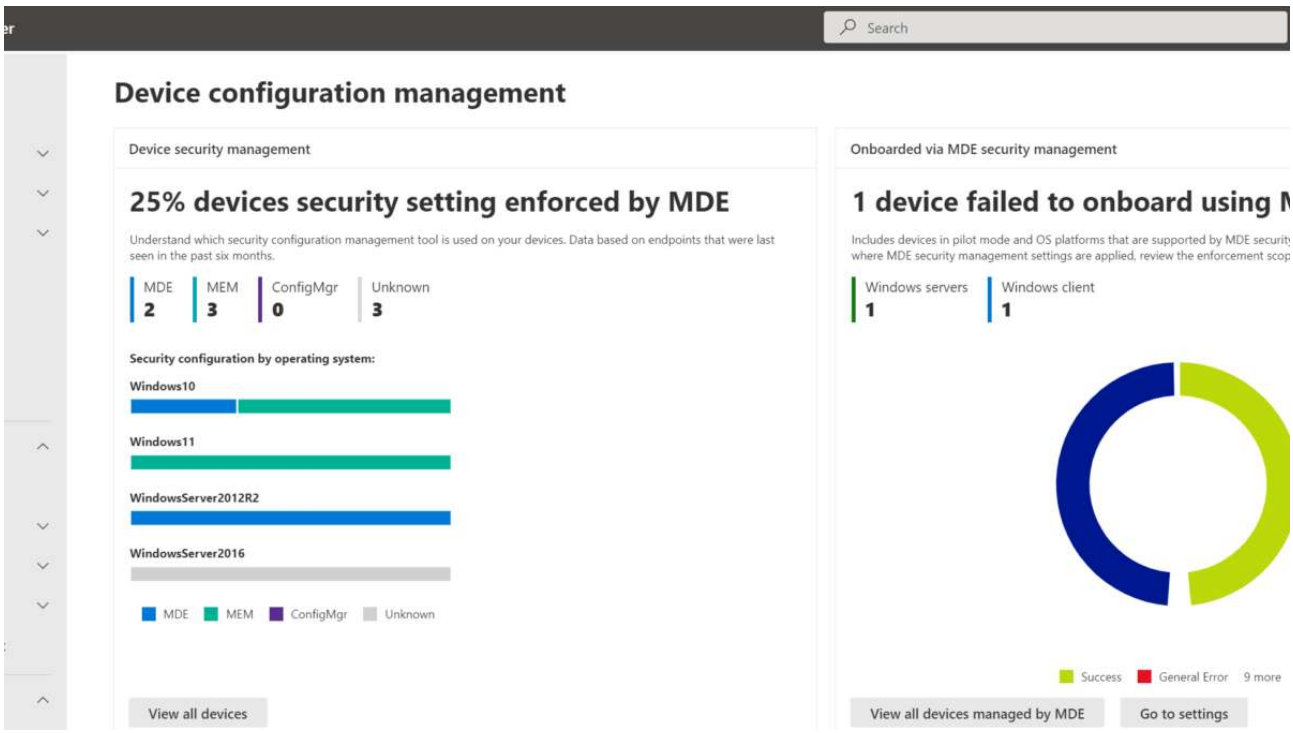


- Image Name/Tag
- Registry
- Scan Status
- Last Scanned
- Vulnerability Summary (number of vulnerabilities by severity)
- Vulnerability Details:
- CVE ID
- Severity

- Description
- Affected Package(s)
- Version(s)
- Remediation Steps
- Link to CVE Details
- **7.3 Reports Page:**



- Report generation section
- Report parameters:
- Registry (dropdown)
- Image (dropdown)
- Tag (dropdown)
- Date Range (start and end dates)
- Report Format (radio buttons: PDF, CSV, JSON)
- Generate Report button
- List of generated reports with download links
- **7.4 Settings Page**

-
- 

Sections:

- Registry Management: Add, edit, and delete container registries.
- Scan Configuration: Configure scan schedules, vulnerability database updates, and other scan settings.
- User Management: Manage user accounts, roles, and permissions.
- Notifications: Configure alerts for new vulnerabilities, scan failures, and other events.

8. Development Action Items (Bonus/Optional)

The following are some development action items that can be discussed with the development team:

- **8.1 Technology Stack:**
- Select appropriate technologies for the backend (e.g., Python, Go, Java), database (e.g., PostgreSQL, MongoDB), and frontend (e.g., React, Angular, Vue.js).
- Choose a vulnerability scanning engine (e.g., Trivy, Clair).
- **8.2 API Design:**
- Design the API endpoints for image scanning, vulnerability data retrieval, and system configuration.
- **8.3 Database Schema:**
- Define the database schema for storing image data, vulnerability data, and scan results.
- **8.4 UI Development:**
- Develop the user interface based on the wireframes and user experience best practices.
- **8.5 Integration:**
- Implement integration with container registries and CI/CD tools.
- **8.6 Testing:**
- Develop a comprehensive testing strategy, including unit, integration, and system tests.
- **8.7 Deployment:**
- Define the deployment process and environment.