Product Requirements Document: Container Image Vulnerability Scanner

1. Introduction

This document outlines the requirements for a security product designed to scan container images for known vulnerabilities and present the findings to users in a clear and actionable manner. The primary goal is to enable users with a large repository of container images to quickly identify and prioritize remediation efforts for images with critical and high-severity vulnerabilities.

2. Goals

- Provide users with a comprehensive view of vulnerabilities within their container image repository.
- Enable users to easily identify images with critical and high-severity vulnerabilities.
- Offer clear information about the identified vulnerabilities, including severity and potential impact.
- Facilitate the process of identifying which images require immediate attention and remediation.
- Support efficient navigation and filtering through a large number of images and findings.

3. Target User

- DevOps Engineers
- Security Engineers
- Platform Engineers
- Software Developers responsible for deploying containerized applications

4. Functional Requirements

- **Image Scanning:**
  - The system shall be able to scan container images from various registries (e.g., Docker Hub, private registries).
  - The system shall support scanning images based on tags and digests.
  - The system shall identify vulnerabilities in operating system packages and application dependencies within the container image.[1]
  - The system shall use a regularly updated vulnerability database (e.g., CVE database).[2]
  - The system shall provide details for each identified vulnerability, including:

        - Vulnerability ID (e.g., CVE ID)
        - Description of the vulnerability
        - Severity level (e.g., Critical, High, Medium, Low, Informational)
        - Affected component (e.g., package name, version)
        - Potential impact of the vulnerability
        - Links to relevant vulnerability databases or advisories.
        - Recommended remediation steps (if available).
- **Image Listing and Filtering:**
    - The system shall display a list of scanned container images.
    - Users shall be able to filter the list of images based on:
        - Image name/tag/digest
        - Vulnerability severity (Critical, High, Medium, Low)
        - Number of vulnerabilities
        - Scan status (e.g., scanned, not scanned, scan failed)
        - Registry
        - Date of last scan
    - Users shall be able to sort the list of images based on relevant criteria (e.g., number of critical vulnerabilities, last scanned date).
- **Vulnerability Reporting:**
    - The system shall display a summary of vulnerabilities for each scanned image, categorized by severity.
    - Users shall be able to drill down into the details of vulnerabilities for a specific image.[3]
    - Users shall be able to view a list of all vulnerabilities across all scanned images, with filtering options (e.g., by severity, affected component).
- **User Interface:**
    - The user interface shall be intuitive and easy to navigate.[4]
    - Key information (e.g., number of critical vulnerabilities) should be prominently displayed.
    - The UI should be responsive and performant, even with a large number of images.
- **Authentication and Authorization:**
    - The system shall provide secure authentication for users.
    - The system shall support role-based access control to manage user permissions.[5]

5. Non-Functional Requirements

- **Performance:** The system should be able to scan images and display results in a reasonable timeframe.

- **Scalability:** The system should be able to handle a large number of container images and scan requests.
- **Reliability:** The system should be reliable and available.
- **Security:** The system itself should be secure and protect sensitive data.
- **Maintainability:** The system should be designed for easy maintenance and updates.

6. Low-Fidelity Wireframes

Here are some basic wireframes to illustrate the user interface:

Wireframe 1: Main Dashboard / Image List View

```
+-------------------------------------------------------------------+
| **Container Image Vulnerability Scanner** |
+-------------------------------------------------------------------+
| [ Filter by: v Severity | Image Name/Tag | Scan Status | Registry | ... ] |
+-------------------------------------------------------------------+
| [ Sort by:   v Critical High Medium Low | Last Scanned ]          |
+-------------------------------------------------------------------+
|                                                                   |
| +----------------+-------+--------+--------+------+------------+-------+ |
| | **Image** | **Crit**| **High**| **Med** | **Low**| **Last Scan**| **Status**| |
| +----------------+-------+--------+--------+------+------------+-------+ |
| | my-app:latest  |  2  |   5  |   10  |  15  | 2025-04-09  | Scanned | |
| +----------------+-------+--------+--------+------+------------+-------+ |
| | db-server:v1.0 |  0  |   1  |   3   |  8   | 2025-04-08  | Scanned | |
| +----------------+-------+--------+--------+------+------------+-------+ |
| | worker-node:dev |  5  |   8  |   12  |  20  | 2025-04-07  | Scanned | |
| +----------------+-------+--------+--------+------+------------+-------+ |
| | ...           |  ... |   ... |   ... |  ... | ...        | ...   | |
| +----------------+-------+--------+--------+------+------------+-------+ |
|                                                                   |
+-------------------------------------------------------------------+
```

Key elements in Wireframe 1:

- **Filters:** Allow users to narrow down the list of images based on various criteria.
- **Sorting:** Enable users to order the list based on severity or scan date.
- **Image Table:** Displays key information for each image in a concise format,

including the number of vulnerabilities per severity level, last scan date, and scan status.
- Clicking on an image row would navigate to the detailed vulnerability view for that image.

Wireframe 2: Image Details / Vulnerability List View

```
+-------------------------------------------------------------------+
| **Container Image Vulnerability Scanner** |
+-------------------------------------------------------------------+
| <- Back to Image List | **Image: my-app:latest** |
+-------------------------------------------------------------------+
| **Vulnerability Summary:** Critical: [2] | High: [5] | Medium: [10] | Low: [15] |
+-------------------------------------------------------------------+
|                                                                   |
| +------------+---------------------+----------+-----------------+ |
| | **Severity**| **Vulnerability ID** | **Component**| **Description** | |
| +------------+---------------------+----------+-----------------+ |
| | Critical   | CVE-2023-XXXX       | libssl     | Buffer overflow in ...| |
| +------------+---------------------+----------+-----------------+ |
| | Critical   | CVE-2024-YYYY       | openssh    | Authentication bypass | |
| +------------+---------------------+----------+-----------------+ |
| | High       | CVE-2023-ZZZZ       | curl       | Information leak ...  | |
| +------------+---------------------+----------+-----------------+ |
| | ...        | ...                 | ...      | ...               | |
| +------------+---------------------+----------+-----------------+ |
|                                                                   |
+-------------------------------------------------------------------+
```

Key elements in Wireframe 2:

- **Navigation:** A clear "Back" button to return to the image list.
- **Image Identification:** Displays the specific image name and tag/digest.
- **Vulnerability Summary:** Shows a count of vulnerabilities by severity for the selected image.
- **Vulnerability Table:** Lists the individual vulnerabilities found in the image, including severity, ID, affected component, and a brief description.
- Clicking on a vulnerability row could expand to show more details (e.g., full description, remediation steps, links).

7. Development Action Items

- **Vulnerability Database Integration:** Research and integrate with a reliable and frequently updated vulnerability database (e.g., National Vulnerability Database - NVD).
- **Image Scanning Engine:** Develop or integrate with a container image scanning engine (consider open-source options like Trivy, Clair, Anchore Engine).
- **Registry Integration:** Implement mechanisms to connect and authenticate with various container registries (Docker Hub, private registries, cloud provider registries).
- **API Design:** Define the API endpoints for scanning images, retrieving results, and managing data.
- **Backend Development:** Build the backend logic to handle image scanning requests, process vulnerability data, and store results.
- **Frontend Development:** Develop the user interface based on the wireframes and functional requirements, focusing on usability and performance.
- **Data Storage:** Choose and set up a suitable database to store image metadata and vulnerability findings.
- **Authentication and Authorization Implementation:** Implement secure user authentication and role-based access control.
- **Error Handling and Logging:** Implement robust error handling and logging mechanisms for debugging and monitoring.
- **Testing Strategy:** Define a comprehensive testing strategy, including unit, integration, and end-to-end tests.
- **Deployment Strategy:** Plan the deployment process for the application.
- **Continuous Integration/Continuous Deployment (CI/CD) Pipeline:** Set up a CI/CD pipeline for automated building, testing, and deployment.[6]

This comprehensive document and the accompanying wireframes should provide a solid foundation for your team to start building the container image vulnerability scanning product. Remember that these are initial requirements and wireframes, and they may evolve as per your requirements.