

Name: Neha Nasir Seat Number: EB24210106089

Couse Name: Software Design and Architecture Course Number: SE 454

Teacher Name: Mr. Hussain Saleem

**Vulnerability Scanner Project**

## **1. Domain Overview**

The **Vulnerability Scanner System** is designed to automatically detect and report security weaknesses in systems, networks, or web applications.  
It enables **security analysts** and **administrators** to perform on-demand or scheduled scans, view detailed results through dashboards, and generate comprehensive reports for remediation.

The main goal of the system is to ensure that organizations can proactively identify potential vulnerabilities and mitigate them before exploitation. The system emphasizes **accuracy, scalability, and data security** during the entire scanning and reporting process.

## **2. Key Domain Concepts**

| **Concept** | **Description** |
| --- | --- |
| **Target System** | The network, application, or host being scanned. Contains attributes such as IP, operating system, and services. |
| **Scanner** | The scanning engine that performs vulnerability checks. Configurable for scan type and depth. |
| **Vulnerability** | A detected weakness or risk in the target system, classified by severity (Low, Medium, High, Critical). |
| **Scan Report** | Summary of scan results, including vulnerabilities, risk scores, and timestamps. |
| **User / Analyst** | Security personnel who configure scans, analyze results, and plan remediation steps. |
| **Dashboard** | Graphical UI showing vulnerabilities, trends, and remediation status. |
| **Scan Job** | The scheduled or on-demand scanning process, containing target details, scan scope, and timing. |
| **Remediation** | Recommended actions to fix detected vulnerabilities and improve security posture. |

## **3. Domain Entities and Relationships**

* **User/Analyst** initiates and manages **Scan Jobs**.
* Each **Scan Job** is executed by the **Scanner**, which inspects the **Target System**.
* The **Scanner** identifies multiple **Vulnerabilities**, each linked to a specific **Target System** and **Scan Job**.
* **Scan Reports** summarize findings and present them on the **Dashboard**.
* **Remediation steps** are linked to vulnerabilities for post-scan corrective action.

## **4. Functional Requirements**

1. Configure scan settings (target IPs, scan depth, and scope).
2. Run scans either on-demand or on a scheduled basis.
3. Detect and classify vulnerabilities using standard severity levels.
4. Generate detailed reports for each scan.
5. Display vulnerability summaries and charts on dashboards.
6. Export reports in PDF or CSV format.
7. Authenticate users and enforce role-based access (Admin, Analyst).
8. Maintain logs for auditing and historical analysis.

## **5. Non-Functional Requirements**

* **Performance:** Scans should complete efficiently without overloading target systems.
* **Accuracy:** Detection mechanisms must minimize false positives and false negatives.
* **Security:** All data and reports must be encrypted and accessible only to authorized users.
* **Scalability:** The system must handle multiple concurrent scan jobs.
* **Reliability:** System should recover gracefully from interruptions.
* **Usability:** Dashboard must provide clear visualization of vulnerabilities and trends.
* **Auditability:** Maintain complete logs for compliance and review purposes.

## **6. Domain Constraints**

* The system must comply with **legal and ethical scanning standards**.
* Only **authorized users** may initiate scans or access results.
* The system must follow **standard severity scoring models** such as **CVSS**.
* Reports must always reflect the **latest scan data**.
* Must not execute scans without explicit user consent.

## **7. Domain Glossary**

| **Term** | **Meaning** |
| --- | --- |
| **Scan Job** | A process that performs vulnerability checks on a defined scope. |
| **Target System** | The host, network, or application being analyzed. |
| **Vulnerability** | Any security weakness exploitable by attackers. |
| **Severity** | Risk rating assigned to a vulnerability (Low–Critical). |
| **Dashboard** | Visualization interface showing metrics and reports. |
| **Remediation** | Actions recommended to fix detected issues. |
| **Analyst** | Authorized user responsible for managing and analyzing scan results. |

## **8. System Architecture Design**

### 8.1 Layered Modular Design

The system follows a **layered modular architecture** to ensure flexibility, maintainability, and security.

#### **a. User Interface Layer**

* Provides dashboards and control panels for analysts.
* Allows configuration of scan jobs, monitoring of results, and report visualization.

#### **b. Scan Management Layer**

* Handles scan creation, scheduling, and execution.
* Manages scanning scope, depth, and frequency.

#### **c. Scanning Engine Layer**

* Performs automated scanning using internal scripts or external tools.
* Detects vulnerabilities and classifies them by severity.

#### **d. Data Processing & Storage Layer**

* Validates and normalizes raw scan data.
* Stores scan results, reports, and logs securely.
* Supports data retrieval and analysis queries.

#### **e. Reporting & Analytics Layer**

* Generates reports in multiple formats (PDF/CSV).
* Displays trends, charts, and summaries via dashboards.
* Supports exporting and sharing reports.

#### **f. Security & Access Control Layer**

* Enforces authentication and authorization.
* Logs every user and system event for accountability.

### 8.2 Data Flow

1. **Input:** Analyst defines scan scope, target systems, and configurations.
2. **Process:** The system schedules or triggers the scan through the scanning engine.
3. **Detection:** Vulnerabilities are identified and classified.
4. **Storage:** Processed scan results and logs are stored securely in the database.
5. **Output:** Reports and dashboards are generated and presented to the user.

### 8.3 Design Approach

* The architecture follows an **N-tier design**, separating UI, logic, and data management layers.
* Uses **API-driven communication** between components for scalability.
* Prioritizes **automation** to minimize manual input.
* Designed for **modular integration**, allowing future addition of advanced scanning features or AI-based analytics.

### 8.4 Key Design Considerations

| **Aspect** | **Description** |
| --- | --- |
| **Performance** | Multi-threaded scanning ensures efficient performance under high workloads. |
| **Security** | All user actions and scan results are authenticated, authorized, and logged. |
| **Scalability** | Architecture supports distributed scanning for large enterprise networks. |
| **Accuracy** | Leverages standardized frameworks (CVSS) for consistent vulnerability scoring. |
| **Usability** | Dashboard provides clear, color-coded summaries for quick analysis. |

### 8.5 Tools and Technologies

| **Component** | **Technology Used** |
| --- | --- |
| **Frontend** | HTML, CSS, JavaScript / React or Flutter |
| **Backend** | Python / Node.js |
| **Database** | MySQL / MongoDB |
| **Scanning Engine** | Integrated tools (Nmap, OpenVAS, Nessus) |
| **Report Generation** | Python (ReportLab, Pandas, or Matplotlib) |
| **Authentication** | Firebase Auth / OAuth 2.0 / JWT |
| **Deployment** | Docker / AWS / Azure Cloud |











