**21BAI1040 Manoj Kumar**

**21BRS1161 Nishchhal Raturi**

**Review 3 Project Report**

1. Review 1 content
2. Review 2 content
3. Algorithms and Data Structures
4. User Interface and Implementation
5. Design of Tests

**1.Review 1 Content**

**REPORT: PORT SCANNER**

****

Group:

1. Nishchhal Raturi - 21BRS1161
2. Manoj Kumar Bhargava P – 21BAI1040

**Table of Contents**

1. Customer Sentiment of Requirements………………………………………….. 3
   1. Open Ports and Associated Services……………………………………………………………… 3
   2. CVE Lookup for Identified Services………………………………………………………………. 3
   3. User Interface for Ease of Use………………………………………………………………………. 4
   4. Performance Optimization for Large-scale Scans……………………………………………. 4
2. User Stories………………………………………………………………………………. 5
   1. Network Administrator………………………………………………………………………………… 5
   2. Security Analyst…………………………………………………………………………………………. 5
   3. Penetration Tester………………………………………………………………………………………. 6
   4. System Administrator…………………………………………………………………………………. 6
   5. Developer…………………………………………………………………………………………………. 7
   6. Security Manager………………………………………………………………………………………. 7
3. Functional Requirements Specifications………………………………………. 8
   1. Scanning Functionality………………………………………………………………………………. 8
   2. Service Identification…………………………………………………………………………………. 8
   3. CVE Lookup Integration……………………………………………………………………………… 8
   4. User Interface…………………………………………………………………………………………… 8
   5. Scan Results Presentation…………………………………………………….…………………… 9
   6. Performance Optimization…………………………………………………………………………. 9
   7. Customization Options………………………………………………………………………………. 9
   8. Logging and Reporting……………………………………………………………………………….. 9
   9. Security and Permissions………………………………………………………………………….. 10
   10. Integration with Existing Tools…………………………………………………………………….10
   11. Use Case Diagram…………………………………………………………………………………… 11
   12. Traceability Matrix…………………………………………………………………………………….12
   13. Fully Dressed Use Case Descriptions ……………………………….............................14
       1. Use Case 1: Perform Port Scan………………………………………………………..14
       2. Use Case 2: Identify Services……………………………………………………………14
       3. Use Case 3: Check for CVEs…………………………………………………………….15
       4. Use Case 4: Customize Scan Parameters……………………………………………15
       5. Use Case 5: Generate Reports………………………………………………………….16
       6. Use Case 6: Integration with Existing Tools…………………………………………17

**The Customer Statement of Requirements**

The Customer Statement of Requirements outlines the essential functionalities and features desired in a network port scanner tool. This tool serves as a crucial component in the customer's arsenal for ensuring the security and integrity of their network infrastructure. The following points delve into specific requirements articulated by the customer, including the need for comprehensive port scanning capabilities, detailed identification of services running on open ports, and integration with vulnerability databases to check for associated Common Vulnerabilities and Exposures (CVEs). By addressing these requirements, the envisioned port scanner aims to provide users with robust insights into their network's security posture, enabling them to proactively mitigate potential risks and fortify their defenses against cyber threats.

**Problem 1: Identifying Open Ports and Associated Services:**

Users need a tool that can efficiently identify open ports on a given IP address and provide information about the services running on those ports.

**Solution:**

To address this, we will develop a network port scanner using Python. The scanner will utilize socket programming to scan a range of ports on the specified IP address. Upon finding an open port, the scanner will attempt to gather information about the service running on that port using protocols like HTTP, FTP, SSH, etc. This information will include service banners, protocol versions, and potentially identifiable patterns unique to each service. By leveraging Python libraries such as `socket`, `nmap`, or `scapy`, we can ensure efficient port scanning and service identification.

**Problem 2: CVE Lookup for Identified Services**

Users want the ability to check if the identified service versions have any Common Vulnerabilities and Exposures (CVE) associated with them.

**Solution:**

To meet this requirement, the scanner will integrate with the Exploit Database (exploitDB) or similar vulnerability databases. Upon identifying a service and its version, the scanner will query the database to check if any CVEs are associated with that specific service version. If a CVE is found, the scanner will provide relevant information about the vulnerability, including its severity, description, and potential exploits. This integration will enhance the security assessment capability of the port scanner, allowing users to prioritize patching or mitigating vulnerabilities on their network.

**Problem 3: User Interface for Ease of Use**

Users require a user-friendly interface to interact with the port scanner, allowing them to input target IP addresses, specify scanning options, and view scan results easily.

**Solution:**

To address this, we will develop a graphical user interface (GUI) for the port scanner using libraries such as Tkinter or PyQt. The GUI will provide intuitive input fields for specifying the target IP address, selecting scanning options (e.g., range of ports to scan), and initiating the scan. Once the scan is complete, the GUI will display the results in a structured format, highlighting open ports, associated services, and any detected CVEs. Additionally, the GUI will include features such as progress indicators, scan history, and export options for enhanced usability and convenience.

**Problem 4: Performance Optimization for Large-scale Scans:**

Users need the port scanner to perform efficiently, even when scanning large ranges of IP addresses or numerous ports.

**Solution:**

To optimize performance, the port scanner will implement multithreading or asynchronous programming techniques. By parallelizing port scans across multiple threads or processes, the scanner can effectively utilize system resources and reduce scan times. Additionally, the scanner will incorporate features such as connection timeouts and concurrency limits to prevent resource exhaustion and ensure stability during high-volume scans. This performance optimization will enable users to conduct comprehensive network assessments without sacrificing speed or reliability.

**User Stories**

1. **As a Network Administrator:**
   1. **User Story**: As a network administrator, I want to use the port scanner to identify open ports and services running on various devices within our network. I need to ensure that all devices are properly configured and that there are no unauthorized services running. Additionally, I want to leverage the CVE lookup feature to identify any potential vulnerabilities associated with the services running on our network.
   2. **Usage:** The network administrator will use the port scanner to perform regular scans on IP ranges assigned to the company's network infrastructure. By analyzing the scan results, they can identify any unauthorized or misconfigured services that may pose security risks. The CVE lookup feature allows them to prioritize remediation efforts based on the severity of identified vulnerabilities.
   3. **Relationship:** The network administrator collaborates closely with the security manager to ensure that the network remains secure and compliant with company policies. They may also work with system administrators to implement necessary changes based on the scan results.
2. **As a Security Analyst:**
   1. **User Story:** As a security analyst, I need to conduct thorough security assessments of our network infrastructure to identify potential vulnerabilities and weaknesses. I will use the port scanner to gather information about open ports and services, and leverage the CVE lookup feature to assess the severity of any identified vulnerabilities.
   2. **Usage:** The security analyst will utilize the port scanner to perform in-depth scans of specific IP ranges or individual devices within the network. They will analyze the scan results to identify security risks and recommend remediation measures to mitigate them. The CVE lookup feature provides valuable insight into the exploitability of identified vulnerabilities.
   3. **Relationship:** The security analyst works closely with the network administrator and security manager to identify and address security issues within the organization's network infrastructure. They may also collaborate with penetration testers to validate the effectiveness of security controls.
3. **As a Penetration Tester:**
   1. **User Story:** As a penetration tester, I need to simulate real-world attacks on our network to identify potential entry points and assess the effectiveness of existing security measures. I will use the port scanner to gather information about open ports and services, and explore potential avenues for exploitation.
   2. **Usage:** The penetration tester will employ the port scanner as part of their reconnaissance phase to gather initial information about the target network. They will use the scan results to identify potential attack vectors and prioritize targets for further exploitation. The CVE lookup feature assists in identifying known vulnerabilities that could be leveraged during penetration testing engagements.
   3. **Relationship:** The penetration tester collaborates closely with the security analyst and network administrator to ensure that the scope of the penetration test aligns with organizational goals and security requirements. They may also work with developers to validate the effectiveness of security controls implemented in custom applications.
4. **As a System Administrator:**
   1. **User Story:** As a system administrator, I need to monitor and maintain the security of the servers and devices under my responsibility. I will use the port scanner to periodically scan for open ports and services, ensuring that only authorized services are running and that any potential vulnerabilities are promptly addressed.
   2. **Usage:** The system administrator will utilize the port scanner to conduct regular scans on servers and devices within their domain. They will analyze the scan results to identify any unauthorized services or potential security risks that require attention. The CVE lookup feature provides valuable information about known vulnerabilities associated with the services running on their systems.
   3. **Relationship:** The system administrator collaborates with the network administrator and security analyst to ensure that security best practices are followed and that any identified vulnerabilities are promptly remediated. They may also work closely with developers to implement patches or configuration changes to mitigate security risks.
5. **As a Developer:**
   1. **User Story:** As a developer, I need to ensure that the applications and services I develop are secure and free from vulnerabilities. I will use the port scanner to scan for open ports and services during the development and testing phases, identifying any potential security issues that need to be addressed.
   2. **Usage:** The developer will incorporate the port scanner into their development workflow to assess the security posture of the applications and services they are building. They will analyze the scan results to identify any open ports or services that may introduce security risks. The CVE lookup feature assists in identifying known vulnerabilities associated with third-party libraries or dependencies used in their applications.
   3. **Relationship:** The developer works closely with the security analyst and system administrator to address any security issues identified during the development process. They may also collaborate with penetration testers to validate the security of their applications through simulated attacks.
6. **As a Security Manager:**
   1. **User Story:** As a security manager, I need to oversee the overall security posture of our organization's network infrastructure and ensure that appropriate measures are in place to protect against potential threats. I will use the port scanner to gain visibility into open ports and services across the network, and leverage the CVE lookup feature to prioritize remediation efforts based on the severity of identified vulnerabilities.
   2. **Usage:** The security manager will use the port scanner to conduct periodic scans of the organization's network infrastructure, assessing the security posture and identifying any potential risks or vulnerabilities. They will analyze the scan results to make informed decisions about security investments and allocate resources for remediation efforts. The CVE lookup feature provides valuable insight into the exploitability of identified vulnerabilities and helps prioritize security initiatives.
   3. **Relationship:** The security manager collaborates with various stakeholders, including network administrators, security analysts, and system administrators, to ensure that security measures are effectively implemented and aligned with organizational objectives. They may also liaise with executive leadership to communicate security risks and advocate for necessary investments in cybersecurity initiatives.

**Functional Requirements for Network Port Scanner**

* + - 1. **Scanning Functionality:**
         1. The port scanner should support scanning a range of TCP and UDP ports on a specified IP address or range of IP addresses.
         2. Users should have the option to specify the start and end ports for the scan, as well as the type of scan (TCP, UDP, or both).
         3. The scanner should be capable of detecting open ports within a reasonable timeframe, providing quick results to users.
      2. **Service Identification:**
         1. Upon identifying open ports, the scanner should attempt to identify the services running on those ports.
         2. The scanner should extract service banners or other identifying information to determine the type and version of the service.
         3. For common services such as HTTP, FTP, SSH, etc., the scanner should provide detailed information about the service, including protocol version and any available headers or responses.
      3. **CVE Lookup Integration:**
         1. The scanner should integrate with vulnerability databases such as Exploit Database (exploitDB) to check if the identified service versions have any associated Common Vulnerabilities and Exposures (CVE).
         2. Upon detecting a service version, the scanner should query the CVE database to retrieve relevant vulnerability information.
         3. The scanner should display CVE information to users, including CVE ID, severity rating, description, and any available exploit references
      4. **User Interface:**
         1. The port scanner should feature a user-friendly interface to facilitate easy interaction with the tool.
         2. The interface should include input fields for specifying the target IP address or range, start and end ports, and scan type.
         3. Users should have the option to initiate scans and view results directly within the interface.
         4. The interface should display scan progress indicators and provide feedback to users during the scanning process.
      5. **Scan Results Presentation:**
         1. Upon completing a scan, the scanner should present the results in a structured and easily understandable format.
         2. The results should include a list of open ports along with the corresponding services identified on those ports.
         3. For each identified service, the scanner should display relevant information such as service name, protocol version, and any available CVE information.
         4. Users should have the option to export scan results in formats such as text, CSV, or HTML for further analysis or reporting purposes.
      6. **Performance Optimization:**
         1. The scanner should be optimized for performance to ensure efficient scanning even on large networks or extensive port ranges.
         2. Techniques such as multithreading or asynchronous scanning should be employed to parallelize port scans and utilize system resources effectively.
         3. The scanner should implement timeouts and error handling mechanisms to handle unresponsive or inaccessible hosts gracefully without impacting overall scan performance.
      7. **Customization Options:**
         1. Users should have the flexibility to customize scan parameters according to their specific requirements.
         2. Advanced options such as scan speed throttling, service detection sensitivity, and timeout settings should be available for customization.
         3. The scanner should support both interactive mode, where users can adjust settings on-the-fly, and configuration files for predefined scan profiles.
      8. **Logging and Reporting:**
         1. The scanner should maintain logs of scan activities, including scan parameters, scan results, and any encountered errors or exceptions.
         2. Users should have the option to review scan logs within the interface or export them for archival or auditing purposes.
         3. The scanner should generate comprehensive reports summarizing scan findings, including open ports, identified services, and CVE information, if applicable.
      9. **Security and Permissions:**
         1. The scanner should implement appropriate security measures to ensure secure operation and protect sensitive information.
         2. Access controls should be in place to restrict usage of the scanner to authorized users and roles within the organization.
         3. Users should authenticate themselves before initiating scans or accessing sensitive features such as CVE lookup integration.
      10. **Integration with Existing Tools:**
          1. The scanner should seamlessly integrate with existing network management and security tools commonly used within the organization.
          2. Integration with network monitoring systems, SIEM (Security Information and Event Management) solutions, or vulnerability management platforms should be supported through APIs or data exchange formats such as JSON or XML.
          3. The scanner should provide compatibility with scripting languages or automation frameworks to enable integration into custom workflows or automated security processes.

By fulfilling these functional requirements, the network port scanner will provide a comprehensive solution for network administrators, security analysts, penetration testers, system administrators, developers, and security managers to assess and secure their network infrastructure effectively.

**Use Case Diagram**

A diagram of a network port scanner

Description automatically generated

**Traceability Matrix**

|  |  |  |  |
| --- | --- | --- | --- |
| **Requirement ID** | **Requirement Description** | **Use Cases** | **Test Cases** |
| R1 | Scanning Functionality | UC1: Perform port scan on specified IP address or range | TC1.1: Scan single IP address |
|  |  |  | TC1.2: Scan range of IP addresses |
|  |  |  | TC1.3: Specify start and end ports |
|  |  |  | TC1.4: Select TCP, UDP, or both scans |
| R2 | Service Identification | UC2: Identify services running on open ports | TC2.1: Identify HTTP service |
|  |  |  | TC2.2: Identify FTP service |
|  |  |  | TC2.3: Identify SSH service |
| R3 | CVE Lookup Integration | UC3: Check for CVEs associated with identified services | TC3.1: Query CVE database for identified service version |
|  |  |  | TC3.2: Display CVE information |
| R4 | User Interface | UC4: Provide user-friendly interface for interaction | TC4.1: Input target IP address |
|  |  |  | TC4.2: Specify start and end ports |
|  |  |  | TC4.3: Initiate scan |
|  |  |  | TC4.4: View scan results |
| R5 | Scan Results Presentation | UC5: Display scan results in structured format | TC5.1: List open ports and associated services |
|  |  |  | TC5.2: Display service information |
|  |  |  | TC5.3: Show CVE information |
| R6 | Performance Optimization | UC6: Optimize performance for large-scale scans | TC6.1: Multithreading for parallel scans |
|  |  |  | TC6.2: Implement timeouts for unresponsive hosts |
| R7 | Customization Options | UC7: Provide customization options for scan parameters | TC7.1: Adjust scan speed |
|  |  |  | TC7.2: Customize service detection sensitivity |
|  |  |  | TC7.3: Configure timeout settings |
| R8 | Logging and Reporting | UC8: Maintain logs and generate reports | TC8.1: Log scan activities |
|  |  |  | TC8.2: Export scan logs |
|  |  |  | TC8.3: Generate comprehensive reports |
| R9 | Security and Permissions | UC9: Implement security measures for secure operation | TC9.1: Authenticate users |
|  |  |  | TC9.2: Restrict access to authorized users |
| R10 | Integration with Existing Tools | UC10: Integrate with existing network and security tools | TC10.1: Compatibility with SIEM solutions |
|  |  |  | TC10.2: Support for scripting languages |
|  |  |  | TC10.3: Integration with vulnerability management platforms |

This traceability matrix establishes the relationship between the project requirements, their corresponding use cases, and the associated test cases. It ensures that each requirement is validated through appropriate test cases, ensuring the functionality and quality of the network port scanner.

**Fully Dressed Use Case Descriptions**

**Use Case 1: Perform Port Scan**

* **Actor:** User (Network Administrator, Security Analyst, Penetration Tester, etc.)
* **Description:** The user initiates a port scan to identify open ports and services running on a specified IP address or range of IP addresses.
* **Preconditions:** 
  + The user has access to the port scanning tool.
  + The user has the necessary permissions to perform a port scan.
* **Basic Flow:**
  + The user launches the port scanning tool.
  + The user specifies the target IP address or range of IP addresses to be scanned.
  + Optionally, the user specifies the start and end ports for the scan.
  + The user selects the type of scan (TCP, UDP, or both).
  + The user initiates the scan.
  + The tool performs the port scan on the specified IP address(es).
  + The tool identifies open ports and records them.
  + The tool presents the scan results to the user.
* **Alternate Flows:**
  + Step 5a: If the user cancels the scan, the process terminates.
* **Postconditions:** 
  + The user receives the scan results, including a list of open ports and associated services.

**Use Case 2: Identify Services**

* **Actor:** User (Network Administrator, Security Analyst, Penetration Tester, etc.)
* **Description:** The user identifies services running on open ports detected during a port scan.
* **Preconditions:**
  + The user has access to the port scanning tool.
  + The user has performed a port scan and obtained a list of open ports.
* **Basic Flow:**
  + The user selects an open port from the scan results.
  + The tool attempts to identify the service running on the selected port.
  + The tool retrieves service information such as banners or response headers.
  + The tool displays the identified service to the user.
* **Alternate Flows:**
  + Step 2a: If the service cannot be identified, the tool marks it as unknown.
* **Postconditions:**
  + The user receives information about the services running on open ports.

**Use Case 3: Check for CVEs**

* **Actor:** User (Network Administrator, Security Analyst, Penetration Tester, etc.)
* **Description:** The user checks for Common Vulnerabilities and Exposures (CVEs) associated with identified service versions.
* **Preconditions:**
  + The user has access to the port scanning tool.
  + The user has identified services running on open ports.
* **Basic Flow:**
  + The user selects a service from the scan results.
  + The tool retrieves the version information for the selected service.
  + The tool queries the CVE database for vulnerabilities associated with the service version.
  + The tool presents CVE information, if available, to the user.
* **Alternate Flows:**
  + Step 3a: If no CVEs are found for the service version, the tool notifies the user accordingly.
* **Postconditions:**
  + The user receives information about any CVEs associated with identified service versions.

**Use Case 4: Customize Scan Parameters**

* **Actor:** User (Network Administrator, Security Analyst, Penetration Tester, etc.)
* **Description:** The user customizes scan parameters to tailor the scanning process according to specific requirements.
* **Preconditions:**
  + The user has access to the port scanning tool.
* **Basic Flow:**
  + The user accesses the settings or preferences menu of the tool.
  + The user selects customization options such as scan speed, service detection sensitivity, and timeout settings.
  + The user saves the customized settings.
* **Alternate Flows:**
  + Step 2a: If the user cancels the customization process, the tool retains the default settings.
* **Postconditions:**
  + The scanning tool applies the customized settings to subsequent scans.

**Use Case 5: Generate Reports**

* **Actor:** User (Network Administrator, Security Analyst, Penetration Tester, etc.)
* **Description:** The user generates comprehensive reports summarizing scan findings for documentation and analysis purposes.
* **Preconditions:** 
  + The user has access to the port scanning tool.
  + The user has performed one or more port scans.
* **Basic Flow:**
  + The user accesses the reporting feature of the tool.
  + The user selects the scan(s) for which they want to generate a report.
  + The tool compiles scan results, including open ports, identified services, and CVE information.
  + The tool generates a report in a specified format (e.g., text, CSV, HTML).
  + The tool presents the generated report to the user or saves it to a specified location.
* **Alternate Flows:**
  + Step 2a: If the user does not select any scans, the tool generates a report for the most recent scan by default.
* **Postconditions:**
  + The user receives a comprehensive report summarizing scan findings.

**Use Case 6: Integration with Existing Tools**

* **Actor:** User (Network Administrator, Security Analyst, Penetration Tester, etc.)
* **Description:** The user integrates the port scanning tool with existing network and security management tools for enhanced functionality.
* **Preconditions:**
  + The user has access to the port scanning tool.
  + The user has access to existing network and security management tools.
* **Basic Flow:**
  + The user accesses the integration settings of the port scanning tool.
  + The user selects the desired integration option (e.g., SIEM integration, vulnerability management platform integration).
  + The user provides necessary configuration details for the integration.
  + The tool establishes a connection with the existing tool(s) and shares relevant data or receives commands as per integration specifications.
* **Alternate Flows:**
  + Step 3a: If the user encounters configuration errors, the tool prompts the user to correct them.
* **Postconditions:**
  + The port scanning tool integrates seamlessly with existing network and security management tools, enhancing overall functionality and interoperability.

These fully dressed use case descriptions provide detailed insights into the functionality and interactions of the network port scanning tool, catering to the diverse needs of users such as network administrators, security analysts, penetration testers, and system administrators.

**2. Review 2 Content:**

**1) Domain Model:**

**i) Concept Definitions:**

|  |  |  |
| --- | --- | --- |
| **Responsibility Description** | **Type** | **Concept** |
| R01: Conducts port scanning operations on target IP addresses | D | Scanner |
| R02: Stores the results of port scanning operations, including open ports and services. | K | ScanResult |
| R03: Stores scan results in a structured format for future retrieval and analysis | D | Database |
| R04: Provides a web interface for interacting with the port scanner, including triggering scans and retrieving results. | D | API |
| R05: Initiates port scanning operations and retrieves scan results through the API interface | D | User |

**ii) Association Definitions:**

|  |  |  |
| --- | --- | --- |
| **Concept Fair** | **Association Description** | **Association Name** |
| Scanner ↔ Database | Scanner stores scan results in the database | Stores |
| API ↔ Scanner | API triggers port scanning operations through the scanner | Initiates |
| User ↔ API | User interacts with the port scanner through the API interface | Uses |

**iii) Attribute Definitions:**

|  |  |  |
| --- | --- | --- |
| **Concept** | **Attribute** | **Attribute Definition** |
| Scanner | |  | | --- | | IP Address | | Port Range | | Scan Flags | | |  | | --- | | IP address of the target system to be scanned | | Range of ports to be scanned (e.g., 1-1024) | | Flags or options to be passed to the scanner for conducting the scan (e.g., "-sS" for SYN scan). | |
| ScanResult | |  | | --- | | IP Address | | Open Ports | | Services | | |  | | --- | | IP address of the target system to be scanned | | List of open ports discovered during the scan | | List of services running on the open ports. | |
| Database | |  | | --- | | Connection String | | Tables | | |  | | --- | | URL or connection parameters required to establish a connection to the database. | | Database tables used for storing scan results | |
| API | |  | | --- | | Endpoints | | Request Payload | | |  | | --- | | List of API endpoints exposed for triggering scans and retrieving results | | Data format expected by the API endpoints for triggering scans and providing input parameters | |

**iii)Traceability Matrix:**

| **Use Case** | **Project Work** | **Scanner** | **Scan Result** | **Database** | **API** | **User** |
| --- | --- | --- | --- | --- | --- | --- |
| UC1 | 9 | x |  | x | x | x |
| UC2 | 5 | x | x | x | x | x |
| UC3 | 8 | x |  | x | x | x |

**2)Interaction Diagrams**

**A diagram of a fast api

Description automatically generated**

**3.) Class diagram and interface specification**

**A diagram of a computer

Description automatically generated**

**Class Descriptions:**

**Scanner**: Represents the port scanner functionality of the system. It is responsible for scanning ports on the specified IP address using the given port range and scan flags.

Attributes:

ipAddress: The IP address to be scanned.

portRange: The range of ports to scan.

scanFlags: Flags or options used for scanning ports**.**

**ScanResult:** Represents the result of a port scan. It stores information about the open ports and the services running on them.

Attributes:

ipAddress: The IP address for which the scan result is generated.

openPorts: List of open ports found during the scan.

services: List of services running on the open ports.

**Database:** Manages the storage of scan results. It provides methods to connect to the database and store scan results.

Attributes:

connectionString: The connection string used to establish a connection with the database.

tables: Information about tables in the database where scan results are stored.

**API:** Represents the interface through which users interact with the system. It exposes endpoints for initiating port scans and retrieving scan results.

**Relationships:**

Scanner performs 1 to many scans (1..\*) resulting in ScanResult objects.

Database interacts with ScanResult to store the scan results.

API serves as an interface for users to interact with the system, including triggering port scans and retrieving scan results.

Scanner interacts with Database to store scan results.

API interacts with Database to fetch scan results and serve them to users.

**4) System Architecture and System Design:**

**Architectural Styles:**

Our design architecture is based on the client-server model, specifically a 2-tier architecture. In this model, clients directly connect with a centralized server, which acts as both a database and a service provider. All communication and data processing occur through the server, ensuring a unified infrastructure and centralized management of resources.

**Identifying Subsystems:**

Scanner: Conducts port scanning operations and interacts with the API for initiating scans.

Scan Result: Stores scan results in the database for future retrieval.

Database: Stores information related to scan results and system configuration.

API: Facilitates communication between clients and the server for triggering scans and retrieving results.

User Interface: Provides an interface for users to interact with the system, including initiating scans and viewing results.

**Mapping Subsystems to Hardware:**

The server runs on a master computer, hosting the database and serving as the central hub for data processing.

Clients, such as desktop computers or laptops, access the port scanner application through their web browsers or dedicated client software.

**Persistent Data Storage:**

The system stores scan results and configuration data in a relational database, ensuring long-term reliability and ease of data management.

Backups are scheduled to maintain data integrity and prevent data loss.

**Network Protocol:**

The system utilizes Microsoft Azure Mobile Services SQL database, offering scalability and flexibility in data management.

Data is stored in a cloud-based server, enabling seamless access and scalability based on demand.

**Global Control Flow:**

The system operates in a linear fashion, with each action dependent on the previous one.

Events, such as low inventory alerts, trigger responses in real-time to ensure efficient inventory management.

Concurrency is implemented through multi-threading, allowing multiple clients to initiate scans simultaneously and ensuring smooth system operation.

**3.****Algorithms and Data Structures**

**Algorithms:**

**Port Scanning Algorithm:**

This algorithm utilizes the Nmap library to asynchronously scan ports on the specified target IP address. It iterates through each host and protocol, identifying open ports and their corresponding services.

**Pseudo Code:**

**```**

def scan\_ports(target\_ip):

scan\_results = {}

nm = nmap.PortScanner()

nm.scan(hosts=target\_ip, arguments='-p-')

for host in nm.all\_hosts():

if nm[host].state() == 'up':

scan\_results[host] = {}

for proto in nm[host].all\_protocols():

scan\_results[host][proto] = []

lport = nm[host][proto].keys()

for port in lport:

port\_info = nm[host][proto][port]

if port\_info['state'] == 'open':

service = port\_info['name']

scan\_results[host][proto].append({"port": port, "service": service})

return json.dumps(scan\_results) **```**

**Storing Scan Results Algorithm:**

After scanning, the results are stored in a PostgreSQL database to persistently store scan data for future reference.

**Pseudo Code:**

**```**

def store\_scan\_results(scan\_results):

with conn.cursor() as cur:

cur.execute("""

INSERT INTO scans (result)

VALUES (%s)

""", (Json(scan\_results),))

conn.commit()

**```**

**Retrieving Scan Results Algorithm:**

This algorithm retrieves scan results from the database along with any associated remarks.

**Pseudo Code:**

**```**

def get\_scan\_results():

with conn.cursor() as cur:

cur.execute("""

SELECT s.id, s.result, r.remarks

FROM scans s

LEFT JOIN remarks r ON s.id = r.scan\_id

""")

rows = cur.fetchall()

results = []

for row in rows:

scan\_id = row[0]

result = row[1]

remarks = row[2] if row[2] else "None"

result\_dict = {"id": scan\_id, "result": result, "remarks": remarks}

results.append(result\_dict)

return results

**```**

**Data Structures:**

JSON: Used to store scan results in JSON format before inserting into the database.

Database Tables: Utilizes PostgreSQL tables to store scan results and remarks persistently

**4) User Interface and Implementation**

**Scanning IP in postman:  
A screenshot of a computer

Description automatically generatedViewing remarks in postman:**

**A screenshot of a computer

Description automatically generated**

**Giving remarks in postman:**

**A screenshot of a computer

Description automatically generated**

**Swagger UI for the routes:**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

**5)Design of Tests**

**Port Scanner Tests:**

**Test Case 1: Scanning Ports**

**Test-Case Identifier: TC-01**

Function Tested: scan\_ports(target\_ip: str) -> str

Pass/Fail Criteria: Test will pass if port scanning is successful and returns expected scan results.

Test Procedure:

Call scan\_ports function with a valid target IP address.

Verify that the scan results are returned as expected.

Expected Results:

Ports are successfully scanned.

Scan results are returned as a JSON string.

**Test Case 2: Storing Scan Results**

**Test-Case Identifier: TC-02**

Function Tested: store\_scan\_results(scan\_results: str) -> None

Pass/Fail Criteria: Test will pass if scan results are successfully stored in the database.

Test Procedure:

Call store\_scan\_results function with sample scan results.

Verify that the scan results are stored in the database.

Expected Results:

Scan results are successfully stored in the database.

**Test Case 3: Retrieving Scan Results**

**Test-Case Identifier: TC-03**

Function Tested: get\_scan\_results() -> List[Dict[str, Any]]

Pass/Fail Criteria: Test will pass if stored scan results are retrieved correctly from the database.

Test Procedure:

Call get\_scan\_results function.

Verify that the retrieved scan results match the stored scan results.

Expected Results:

Stored scan results are successfully retrieved from the database.

**Test Case 4: Adding Remarks to Scan**

**Test-Case Identifier: TC-04**

Function Tested: add\_remarks(scan\_id: int, remarks: str) -> None

Pass/Fail Criteria: Test will pass if remarks are successfully added to a scan entry in the database.

Test Procedure:

Call add\_remarks function with a sample scan ID and remarks.

Verify that the remarks are added to the corresponding scan entry in the database.

Expected Results:

Remarks are successfully added to the scan entry in the database.

**Test Case 5: Performing Integration Test**

**Test-Case Identifier: TC-05**

Function Tested: Integrated System Functionality

Pass/Fail Criteria: Test will pass if the system functions as expected end-to-end.

Test Procedure:

Trigger a port scan.

Store the scan results.

Retrieve the stored scan results.

Add remarks to the scan.

Verify that all operations are performed successfully without errors.

Expected Results:

All system functionalities work seamlessly without any issues