Open Port Scanner using Python
With a GUI based dashboard

A Mini Project Report

Submitted by

Abdul Musafir A	-	2403617614921001
Anusha N	-	2403617614922003
Danush S	-	2403617614921011
Ilaya Bharathi A	-	2403617614921022
Karthikeyan S	-	2403617614921025
Mohammed Yunus M	-	2403617614921030
Mugeshkannan G	-	2403617614921031
Prasanth M	-	2403617614921041
Renuka N	-	2403617614922049
Sangavi K	-	2403617614922052

In partial fulfilment from the students of degree

In

BE-CSE (CYBER SECURITY)

ADHIYAMAAN COLLEGE OF ENGINEERING (AUTONOMOUS)

D MGR Nagar, Hosur - 635109

Anna University : Chennai 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"OPEN PORT SCANNER"** is the Bonafide work of **"ABDUL MUSAFIR A (2403617614921001), ANUSHA N (2403617614922003), DANUSH S (2403617614921011), ILAYA BHARATHI A (2403617614921022), KARTHIKEYAN S (2403617614921025), MOHAMMED YUNUS M (2403617614921030), MUGESHKANNAN G (2403617614921031), PRASANTH M (2403617614921041), RENUKA N (2403617614922049), SANGAVI K (2403617614922051)"** who carried out the project under my supervision.

SIGNATURE

DR. M. LILY FLORENSE.,

HEAD OF THE DEPARTMENT

PROFESSOR,

Department of CSE (CYBER SECURITY),

Adhiyamaan College of Engineering,

(Autonomous),

Dr. M.G.R. Nagar,

Hosur – 635 130

ACKNOWLEDGEMENT

It is one of the most efficient tasks in life to choose the appropriate words to express one's gratitude to the beneficiaries. We are very much grateful to God who helped us all the way through the project's and how moulded us into what we are today.

We grateful to our beloved Principal, **Dr R. RADHAKRISHNAN, M.E., Ph.D.**, Adhiyamaan College of Engineering (Autonomous), Hosur for providing the opportunity to do this work in premises.

We grateful to our beloved **Dr. M. Lily Florence**, Professor and Head of the Department, Department of CSE (CYBER SECURITY), Adhiyamaan College of Engineering (Autonomous), Hosur, and the supervisor for her guidance and valuable suggestions and encouragement throughout this project and made us to complete this project successfully.

We also extend our thanks to the Project Coordinators for their support on completing this project successfully.

Finally, we would like to thank to our parents, without their motivation and support would not have been possible for us to complete this project successfully.

Abstract

The Open Port Scanner Web Application is an intuitive and interactive tool designed to enhance network security by identifying open ports on a local machine. Open ports can pose significant security risks, as they may provide potential entry points for unauthorized access, cyber-attacks, and exploitation by malicious entities. Attackers can leverage these open ports to execute malicious activities, including data breaches, unauthorized remote access, and network intrusions. This application is built using Dash, a Python-based web framework, and employs socket programming to conduct thorough port scans, specifically scanning ports within the range of 1 to 1024. By utilizing socket programming, the application systematically attempts connections to each port within the specified range, determining which ones are open and potentially vulnerable. This scanning process helps users gain visibility into their system's network exposure and take appropriate security measures to prevent unauthorized access. The results are then displayed in a user-friendly web interface, providing immediate and dynamic feedback to the user. The interface is designed with simplicity and usability in mind, ensuring that even users with minimal technical knowledge can easily navigate and interpret the results. Additionally, the application educates users on the significance of different ports, highlighting commonly used ports for various services, such as HTTP, HTTPS, FTP, and SSH, while also providing security recommendations to mitigate risks. By integrating both scanning functionality and security awareness, the Open Port Scanner Web Application serves as an essential tool for network administrators, cybersecurity professionals, and individuals aiming to safeguard their digital assets. By regularly scanning for open ports and addressing potential vulnerabilities, users can significantly reduce the risk of cyber threats and unauthorized access, ensuring a more secure and well-monitored network environment.

List of Tables

LIST OF FIGURES

LIST OF SYMBOLS, NOMENCALTURE AND ABBREVISTIONS

Symbols

- ⚠ – Warning Indicator (Used to highlight open ports)
- ✔ – Success Indicator (Used to indicate no open ports detected)
- ◈ – Bullet Point Indicator for Important Information
- 🛡 – Security Indicator (Used for security-related content)
- 🔧 – Tool Indicator (Used for explaining port usage)

Abbreviations

- **ACE** – Adhiyamaan College of Engineering
- **CSE(CS)** – Computer Science and Engineering(Cyber Security)
- **CSS** – Cascading Style Sheets
- **DNS** – Domain Name System
- **FTP** – File Transfer Protocol
- **GUI** – Graphical User Interface
- **HTTP** – Hypertext Transfer Protocol
- **HTTPS** – Hypertext Transfer Protocol Secure
- **IDS** – Intrusion Detection System
- **IMAP** – Internet Message Access Protocol
- **LAN** – Local Area Network
- **POP3** – Post Office Protocol v3

- **RDP** – Remote Desktop Protocol
- **SSH** – Secure Shell
- **UI** – User Interface
- **VPN** – Virtual Private Network

Nomenclature

- **Dash** – A Python framework for building analytical web applications
- **Dash Bootstrap Components (dbc)** – A library for adding Bootstrap UI elements to Dash applications
- **dcc (Dash Core Components)** – Dash module for interactive elements like buttons and input fields
- **html (Dash HTML Components)** – Dash module for creating HTML-based UI elements
- **Socket** – A Python module used for network connections
- **Port** – A communication endpoint for a specific service or protocol in a computer network
- **Port Scanner** – A tool that scans and detects open ports in a system or network
- **Firewall** – A network security device that controls incoming and outgoing traffic
- **Localhost** – Refers to the current device in use (IP: 127.0.0.1)
- **Open Port** – A network port that is currently accessible and can accept connections
- **Closed Port** – A network port that is restricted and not open to connections

Table of Content **Page No**

Chapter 1
Introduction

1.1 Overview

The Open Port Scanner Web Application is a powerful and interactive tool designed to enhance network security by identifying open ports on a local machine. Open ports can pose significant security risks as they may serve as entry points for unauthorized access, cyber-attacks, and exploitation by malicious entities. By scanning for open ports, users can gain critical insights into their network's exposure and take proactive measures to secure potential vulnerabilities.

This application is developed using Dash, a Python-based web framework known for creating interactive web interfaces. It utilizes socket programming to conduct thorough port scans, specifically targeting ports within the range of 1 to 1024. These ports are commonly associated with well-known services such as HTTP (port 80), HTTPS (port 443), FTP (port 21), and SSH (port 22). Due to their widespread usage, these ports are frequent targets for attackers, making it essential to identify and secure them.

The scanning process involves systematically attempting TCP connections to each port within the specified range. If a connection is successfully established, the port is marked as open, indicating a potential security risk. Conversely, if the connection fails, the port is considered closed or filtered. The application uses multithreading to optimize the scanning speed, enabling multiple ports to be scanned simultaneously. This significantly reduces the overall scanning time and enhances performance, particularly when scanning a large number of ports.

The results of the scan are displayed in a dynamic and user-friendly web interface, providing real-time feedback to the user. The interface is designed with simplicity and usability in mind, ensuring that users of all technical backgrounds can easily navigate and interpret the results. The application categorizes the scan results, clearly indicating which ports are open and closed. It also provides additional information about the significance of each port, including the services typically associated with them.

One of the unique features of the Open Port Scanner Web Application is its educational component. In addition to identifying open ports, the application educates users about the importance of securing them. It highlights the potential risks associated with open ports and provides security recommendations, such as configuring firewalls, disabling unused services, and using secure protocols. This educational approach empowers users with the knowledge needed to implement effective security measures.

The application is designed to be a versatile tool suitable for a wide range of users, including network administrators, cybersecurity professionals, and individuals concerned about their digital security. It can be used for network security audits, penetration testing, compliance checks, and troubleshooting connectivity issues. By regularly scanning for open ports and addressing potential vulnerabilities, users can proactively secure their networks and minimize the risk of cyber threats.

In conclusion, the Open Port Scanner Web Application provides an effective and comprehensive solution for enhancing network security. Its combination of advanced port scanning functionality, user-friendly interface, and educational features makes it a valuable tool for safeguarding digital assets. By promoting security awareness and enabling proactive risk management, the application contributes to a safer and more resilient network environment.

1.2 Objectives

The primary objective of this project is to develop an efficient and user-friendly web application that allows users to scan and identify open ports on their local system. By doing so, the application enhances network security awareness and provides actionable insights to mitigate potential cyber threats. The key objectives of the project are:

- To provide an intuitive web-based interface for scanning open ports, ensuring ease of use for both technical and non-technical users.
- To implement a reliable scanning mechanism using Python's socket programming to detect open ports within the range of 1 to 1024.
- To educate users about the risks associated with open ports and provide recommendations on securing their system from potential threats.
- To display real-time scanning results dynamically on the web interface, allowing users to interpret findings quickly and take necessary actions.
- To categorize open ports based on their functionality and provide insights into commonly used services such as web hosting, remote access, and database management.
- To encourage proactive network security practices by equipping users with knowledge on firewalls, intrusion detection systems, and best security practices for port management.
- To create an expandable and scalable system that can be enhanced with additional features, such as remote scanning and integration with security tools in future versions.

1.3 Importance of the Project in Enhancing Network Security

Cyber threats continue to evolve, with hackers constantly developing new techniques to exploit vulnerabilities. One of the most common attack vectors involves scanning for open ports and attempting to exploit services running on them. Attackers can use port scanning techniques to identify entry points into a system, often leveraging unsecured ports to deploy malware, launch denial-of-service (DoS) attacks, or gain unauthorized control over a system.

By using a port scanner, system administrators and users can stay ahead of potential threats by regularly auditing their network's open ports and taking necessary security measures. The Open Port Scanner Web Application plays a vital role in early threat detection and serves as a preventative cybersecurity measure. Some key benefits of this application include:

- Preventing unauthorized access by identifying and closing open ports that are not required for system functionality.
- Improving network visibility by helping users understand which services are running on their system and whether they are necessary.
- Enhancing compliance with security standards by ensuring that systems do not expose unnecessary ports, reducing the attack surface.
- Reducing the likelihood of cyber-attacks by equipping users with the tools to proactively secure their network environment.

With the increasing number of cyber-attacks, data breaches, and ransomware cybersecurity measures. The Open Port Scanner Web Application provides an accessible, practical, and effective solution incidents, it is essential for organizations and individuals to adopt robust to strengthen network security by identifying and mitigating vulnerabilities related to open ports.

1.4 Technologies Used

The Open Port Scanner Web Application is developed using a combination of modern technologies to provide an efficient, user-friendly, and secure solution for port scanning and network security. The selection of these technologies ensures optimal performance, scalability, and a seamless user experience. The key technologies used in building this application are as follows:

Python:

Python serves as the core programming language for the application due to its simplicity, readability, and extensive support for network programming. Python's powerful standard libraries, such as socket, are utilized for conducting port scans by systematically attempting TCP connections to each port. Python's versatility also allows for rapid development and easy integration with other frameworks and tools.

Dash:

Dash, a Python-based web framework, is used to build the interactive and user-friendly web interface. Dash is known for its simplicity and capability to create responsive web applications with real-time updates. It is built on top of Flask, Plotly, and React.js, enabling seamless integration of front-end components with back-end logic. This allows the application to display real-time port scanning results in an organized and intuitive manner.

Socket_Programming:

The application leverages Python's socket module to perform port scanning by attempting to establish TCP connections with ports in the range of 1 to 1024. This approach allows the application to determine which ports are open, closed, or filtered based on the connection response. Socket programming

provides low-level network communication, ensuring precise and efficient port scanning.

Multithreading:

To optimize scanning speed and enhance performance, the application uses multithreading. By enabling multiple threads to scan different ports simultaneously, the application significantly reduces the total scanning time. This is particularly useful when scanning a large number of ports, providing faster results without compromising accuracy.

Flask:

Dash is built on top of Flask, a lightweight web framework in Python. Flask manages the back-end routing and handles HTTP requests, ensuring smooth communication between the user interface and the port scanning logic. It also provides scalability and flexibility, allowing the application to handle multiple user requests efficiently.

Bootstrap_(via_Dash_Components):

The web interface is styled using Bootstrap components integrated with Dash. This ensures a clean, responsive, and visually appealing design that enhances the user experience. Bootstrap's grid system and pre-designed components allow for a consistent and organized layout across different devices and screen sizes.

Plotly:

Plotly, integrated within Dash, is used for data visualization. It enables dynamic and interactive visual representation of port scanning results, helping users easily interpret the findings. Graphical elements such as bar charts or pie charts provide an intuitive overview of open and closed ports.

HTML/CSS:

Custom HTML and CSS are used alongside Dash components to enhance the design and functionality of the user interface. This combination allows for a more tailored user experience, with better control over layout, styling, and responsive behavior.

JavaScript_(viaReact.js):

React.js, integrated within Dash, provides dynamic front-end interactivity, enabling real-time updates of scan results. It enhances the responsiveness of the application, ensuring a smooth and engaging user experience.

Deployment_Tools(Optional):

For deploying the application, tools such as Docker or cloud platforms like Heroku or AWS can be used. Docker ensures consistency across different environments, while cloud platforms provide scalability and remote access to the application.

The combination of these technologies allows the Open Port Scanner Web Application to deliver an efficient, secure, and user-friendly solution for network security. By leveraging the power of Python, Dash, and socket programming, the application provides real-time visibility into open ports and potential vulnerabilities. The use of multithreading and responsive web design ensures optimal performance and a seamless user experience, making it an essential tool for cybersecurity professionals and network administrators.

Chapter 2

System Analysis

2.1 Functional Requirements

The Open Port Scanner Web Application is designed to enhance network security by identifying open ports on a local machine, providing real-time insights into potential vulnerabilities. The functional requirements of this application define the specific functionalities and features that it must provide to achieve its objectives effectively. These requirements ensure that the application operates smoothly and meets the needs of its users, including network administrators, cybersecurity professionals, and individual users concerned about digital security.

1. Port Scanning Functionality:

- **Range_of_Ports:**
  The application should scan ports within the range of 1 to 1024, as these ports are commonly associated with well-known services and are frequent targets for cyber-attacks.
- **Scanning_Method:**
  The application should utilize TCP connection attempts to determine the status of each port. If a connection is successfully established, the port is marked as open. Otherwise, it is considered closed or filtered.
- **Multithreading_Support:**
  To optimize scanning speed and performance, the application should support multithreading, allowing multiple ports to be scanned

simultaneously. This ensures faster and more efficient scanning, especially when scanning a large number of ports.

- **Real_Time_Progress_Display:**
  The application should display the scanning progress in real-time, including the current port being scanned and the percentage of completion.

- **Detailed_Results_Display:**
  The application should provide detailed scan results, categorizing ports as open, closed, or filtered. It should also display the associated service names for commonly used ports.

2. User Interface and Experience:

- **Interactive_Dashboard:**
  The application should have an interactive and user-friendly dashboard built using Dash. The interface should be simple and intuitive, allowing users of all technical levels to easily navigate and interpret the scan results.

- **Dynamic_Visualization:**
  The application should utilize Plotly to present scan results in a graphical format, such as bar charts or pie charts, providing a visual overview of open and closed ports.

- **Responsive_Design:**
  The user interface should be responsive, ensuring compatibility with various devices, including desktops, tablets, and mobile phones.

- **Real_Time_Updates:**
  The dashboard should provide real-time updates of scan progress and results, enhancing user engagement and experience.

- **User_Input_for_Custom_Scans:**
  The application should allow users to specify a custom range of ports to scan, enabling targeted security checks.

3. Security and Privacy:

- **Local_Machine_Scanning:**
  The application should only scan ports on the local machine where it is hosted, ensuring user privacy and compliance with ethical security practices.
- **Data_Security:**
  No sensitive data should be stored or transmitted. The application should ensure that all scan results are processed locally and are not shared or uploaded to external servers.
- **Access_Control:**
  If deployed in a multi-user environment, the application should implement basic authentication mechanisms to restrict unauthorized access.

4. Educational Features:

- **Port_Information_and_Significance:**
  The application should provide educational insights about commonly used ports and their associated services, such as HTTP (port 80), HTTPS (port 443), FTP (port 21), and SSH (port 22).
- **Security_Recommendations:**
  Based on the scan results, the application should provide security recommendations, such as configuring firewalls, disabling unused

services, and using secure protocols. This feature aims to raise security awareness and guide users in mitigating potential risks.

5. Performance and Efficiency:

- **Optimized_Scanning_Speed:**
  The application should perform port scans quickly and efficiently using multithreading, minimizing the total scan time.
- **Resource_Management:**
  The application should be lightweight and resource-efficient, ensuring minimal CPU and memory usage during the scanning process.
- **Scalability:**
  The application should be designed to handle multiple scan requests efficiently, ensuring smooth performance even in high-demand scenarios.

6. Deployment and Maintenance:

- **Cross_Platform_Compatibility:**
  The application should be compatible with multiple operating systems, including Windows, macOS, and Linux.
- **Easy_Deployment:**
  The application should be easy to deploy locally or on cloud platforms using tools such as Docker.
- **Regular_Updates_and_Maintenance:**
  The application should be maintained regularly to ensure compatibility with the latest system environments and to address security vulnerabilities.

7. Error Handling and User Notifications:

- **Error_Handling:**
  The application should gracefully handle network errors, connection timeouts, and invalid input from users, providing appropriate error messages.
- **User_Notifications:**
  The application should notify users of scan completion, errors, or interruptions through clear and concise messages.

These functional requirements ensure that the Open Port Scanner Web Application provides an efficient, secure, and user-friendly solution for identifying open ports and enhancing network security. By fulfilling these requirements, the application will effectively cater to the needs of network administrators, cybersecurity professionals, and individual users, empowering them to safeguard their digital assets against potential cyber threats.

2.2 Non-Functional Requirements

The non-functional requirements of the Open Port Scanner Web Application ensure its efficiency, security, reliability, and overall user experience, making it a robust and effective tool for network security. These requirements define the quality attributes, performance standards, and operational constraints that the application must meet to deliver consistent and reliable functionality. They focus on various aspects, including performance, usability, security, reliability, compatibility, maintainability, deployment, and compliance, ensuring the application operates seamlessly while safeguarding user data and maintaining system integrity.

Performance requirements emphasize speed, efficiency, and scalability, enabling the application to perform port scans swiftly while efficiently utilizing system resources. It is designed to scan the range of 1 to 1024 ports in under 60 seconds, leveraging multithreading to handle multiple connections simultaneously. This approach enhances scanning speed and ensures minimal latency, maintaining real-time responsiveness in the user interface. Additionally, the application is scalable, capable of processing multiple scan requests without significant performance degradation, ensuring consistent functionality even under heavy load conditions.

Usability is a crucial aspect, as the application targets a broad user base, including network administrators and individuals with varying technical expertise. The user interface is designed to be intuitive and user-friendly, built using Dash to provide dynamic visualizations and real-time feedback. The layout is simple and easy to navigate, with clear instructions and tooltips to guide users through the scanning process. Accessibility is also a priority, adhering to web accessibility standards to ensure compatibility with screen readers and usability for users with disabilities. The interface is responsive, providing a seamless experience across devices, including desktops, tablets, and mobile phones.

Security requirements are critical, given the application's role in network security. It ensures data privacy and integrity by processing all scan results locally, preventing unauthorized data storage or transmission. Access control mechanisms are implemented to restrict unauthorized access, safeguarding the application from potential cyber threats. Secure communication protocols like HTTPS are used to encrypt data exchanges, ensuring secure interactions between the user interface and back-end components. Furthermore, the application incorporates measures to prevent misuse, such as restricting scanning to the local machine and implementing rate-limiting mechanisms to mitigate denial-of-service attacks.

Reliability and availability are maintained through robust error handling, system resilience, and high availability, ensuring continuous uptime and consistent functionality. The application is designed to handle unexpected inputs, network failures, and connection timeouts gracefully, providing clear error messages and allowing users to retry or resume scans without data loss. Data consistency is maintained throughout the scanning process, avoiding duplicate or conflicting data entries.

Compatibility requirements ensure cross-platform support, enabling the application to run on major operating systems like Windows, macOS, and Linux. It is also compatible with modern web browsers, ensuring a consistent user experience across different platforms. Maintainability and extensibility are prioritized through modular design, comprehensive documentation, and version control systems, facilitating future updates, feature enhancements, and collaborative development. The application also adheres to legal and ethical standards, ensuring compliance with cybersecurity regulations and promoting responsible usage.

By fulfilling these non-functional requirements, the Open Port Scanner Web Application provides a secure, efficient, and user-friendly solution for identifying open ports and enhancing network security, ensuring a reliable and high-quality user experience.

2.3 Existing Solutions and the Need for a Simpler Alternative

Currently, there are several established tools available for port scanning and network security, such as Nmap, Netcat, and Angry IP Scanner. These tools are widely used by cybersecurity professionals and network administrators to identify open ports and assess network vulnerabilities. Nmap, for example, is a powerful and versatile network scanner that provides detailed information about active

devices, open ports, and running services. It supports advanced features like OS detection, version detection, and scriptable interactions, making it a comprehensive tool for in-depth network analysis. Similarly, Netcat is known for its flexibility in debugging and investigating network connections, while Angry IP Scanner offers a user-friendly interface for fast and efficient scanning.

Despite their effectiveness, these existing solutions often present challenges for users with limited technical expertise. Tools like Nmap and Netcat require command-line knowledge and familiarity with complex parameters, which can be intimidating for beginners or non-technical users. Additionally, the abundance of features and detailed configurations may overwhelm users who only need basic port scanning capabilities. While Angry IP Scanner provides a simpler graphical interface, its functionality is still geared towards more experienced users who understand network concepts and security implications.

Another limitation of existing tools is their focus on detailed network analysis rather than user education. These tools provide comprehensive scan results but often lack contextual explanations about the significance of open ports and associated security risks. This gap leaves novice users struggling to interpret the data and take appropriate security measures. Moreover, many of these tools are designed as standalone applications that require installation and specific system configurations, which can be inconvenient for users seeking a quick, browser-based solution.

Given these challenges, there is a clear need for a simpler, more accessible alternative that caters to a wider audience, including non-technical users who wish to enhance their network security without extensive learning curves. The Open Port Scanner Web Application addresses this need by offering an intuitive, web-based platform built using Dash and Python, ensuring ease of use and accessibility across different devices. It focuses on fundamental port scanning

functionalities, scanning ports within the range of 1 to 1024, and presents the results in a user-friendly interface with dynamic visualizations.

Unlike existing solutions, this application emphasizes user education by providing contextual information about commonly used ports and associated security risks. It also offers security recommendations to help users understand the implications of open ports and take preventive measures. By combining simplicity, usability, and educational value, the Open Port Scanner Web Application bridges the gap between advanced cybersecurity tools and the needs of everyday users. This makes it an ideal choice for individuals, small businesses, and educational institutions seeking to improve their network security awareness and practices without the complexity of traditional port scanners.

Chapter 3
System Design

The Open Port Scanner Web Application is designed with a structured architecture that ensures efficient port scanning, real-time result updates, and an intuitive user interface. The system is divided into two primary components: the frontend (UI), responsible for user interaction and display, and the backend (port scanning logic), which handles scanning operations and data processing. The application leverages Dash, Bootstrap, and Python's socket library to create a responsive, functional, and interactive scanning tool.

3.1 Overall System Architecture

The Open Port Scanner Web Application follows a client-server model where the frontend (client side) interacts with the backend (server side) to initiate scans and retrieve results. When a user clicks the "Scan" button, the frontend triggers a Dash callback function, which executes the port scanning logic using Python's socket module. The scan results are then processed and dynamically displayed on the web interface without requiring a page reload.

The system also includes an information page, allowing users to view project details, team members, and an explanation of the code implementation. The navigation menu enables users to toggle between the main scanning dashboard and the project details page, enhancing usability. The use of Dash Bootstrap Components (DBC) ensures that the interface remains visually appealing and responsive across different devices.

3.2 Frontend Design

The frontend of the Open Port Scanner Web Application is developed using Dash, a Python-based framework that enables the creation of interactive web applications. The UI consists of structured components, ensuring a smooth and user-friendly experience. Key design elements include:

- **Header and Navigation Bar**: Displays the application title and includes a navigation button that redirects users to the project details page.
- **Scan Button and Loading Animation**: The "Scan" button triggers the port scanning process, and a loading animation is displayed while the scan is in progress.
- **Dynamic Result Display**: Once the scan is complete, the results are dynamically updated on the page without requiring a refresh. Open ports are listed alongside their associated services and security implications.
- **Educational Sections**: The UI provides additional content explaining what a port scanner is, how it works, and best practices for securing open ports.
- **Responsive Design**: The use of Dash Bootstrap Components ensures that the interface adapts to different screen sizes, making it accessible on desktops, tablets, and mobile devices.
- **User Interaction Buttons:** Additional buttons allow users to access team details, project explanations, and an in-depth breakdown of the scanning process, enhancing the application's informational value.

The frontend logic is managed using Dash callbacks, which update the displayed content based on user interactions. These callbacks handle button clicks, dynamic content rendering, and interactive UI updates, ensuring a seamless user experience.

3.3 Backend Design

The backend of the Open Port Scanner Web Application is responsible for executing the actual port scanning logic. It is implemented using Python's socket module, which allows the system to attempt connections to each port within the range of 1 to 1024 and determine whether they are open or closed.

Key components of the backend include:

- **Port Scanning Logic**: The system iterates through ports 1 to 1024, attempting to establish a socket connection with each port. If the connection is successful, the port is considered open; otherwise, it is closed or filtered.
- **Timeout Handling**: A timeout setting ensures that the scan runs efficiently without getting stuck on unresponsive ports. This prevents delays and system slowdowns during execution.
- **Real-Time Data Processing**: The scan results are processed immediately and returned to the frontend, where they are formatted and displayed dynamically.
- **Service Identification**: The application maps common ports to their respective services, providing additional insights into what each open port is used for.
- **Security Considerations**: To prevent misuse, the port scanning functionality is limited to the localhost, ensuring that users cannot scan external networks.

By structuring the backend in this way, the system ensures that port scanning is executed efficiently while maintaining security and performance.

3.4 User Interaction Flow

The user interaction flow is designed to be intuitive and efficient, allowing users to initiate scans and view results with minimal effort. The interaction follows these steps:

1. **User Accesses the Application**: The user opens the Open Port Scanner Web Application, where the main dashboard is displayed.
2. **User Initiates a Scan:** The user clicks the "Scan" button, triggering a Dash callback function that executes the port scanning logic.
3. **Scan Execution:** The backend processes the scan, iterating through ports 1 to 1024, and identifies open ports.
4. **Real-Time Results Display**: Once the scan is complete, the list of open ports is displayed dynamically on the interface, along with descriptions of common ports and their uses.
5. **User Reviews the Information:** The user can review open port details, refer to security recommendations, and navigate to the project details page for more information.
6. **User Acts:** Based on the scan results, the user can implement recommended security measures, such as closing unnecessary ports, configuring a firewall, or securing open ports with authentication and encryption.

Chapter 4
Implementation Details

The Open Port Scanner Web Application is developed using Dash, Dash Bootstrap Components, and Python's socket programming, ensuring a functional, user-friendly, and interactive system. The application consists of frontend and backend components that work together to provide real-time port scanning capabilities while maintaining a clean and responsive user interface. This chapter provides a detailed breakdown of the implementation process, key functions, and the role of each component in the system.

4.1 Setting Up the Development Environment

To implement the Open Port Scanner Web Application, the following dependencies must be installed:

pip install dash dash_bootstrap_components

These libraries are crucial for creating the interactive web interface and handling frontend interactions efficiently. Dash is used for building and structuring the UI, while Dash Bootstrap Components (DBC) enhance the styling and responsiveness of the application.

4.2 Implementing Socket Programming for Port Scanning

The core functionality of the application is implemented using Python's socket module, which is responsible for performing port scans on the local machine. The backend logic follows this process:

1. Initialize the scan process when the user clicks the "Scan" button.
2. Iterate through ports 1 to 1024, attempting a socket connection for each port.
3. Determine if a port is open by checking if the connection is successful.
4. Store open ports in a list and format them for display.
5. Send results back to the frontend, dynamically updating the interface.

The socket scanning function is implemented as follows:

```
import socket
def scan_ports():
    total_ports = 1024
    open_ports = []
    for port in range(1, total_ports + 1):
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(0.5)
        result = s.connect_ex(('localhost', port))
        if result == 0:
            open_ports.append(port)
        s.close()
    return open_ports
```

This function iterates through each port, attempts a connection, and records open ports, which are later displayed on the frontend.

4.3 Connecting User Actions to Backend Logic with Callbacks

Dash uses callback functions to connect frontend interactions with backend processes. The Scan button triggers a callback that executes the port scanning function, retrieves the scan results, and dynamically updates the results display.

The callback function is structured as follows:

```
from dash.dependencies import Input, Output
from dash import html

@app.callback(
    Output("output-content", "children"),
    [Input("scan-button", "n_clicks")]
)
def scan_ports(n_clicks):
    if n_clicks:
        open_ports = scan_ports()
        if open_ports:
            return html.Div([
                html.H4("⚠️ Open Ports Detected!"),
                html.P(f"Open Ports: {', '.join(map(str, open_ports))}"),
            ])
        else:
            return html.Div([
                html.H4("✅ No Open Ports Found"),
                html.P("Your system is secure!")
            ])
    return html.Div()
```

This callback function:

1. Waits for the “Scan” button to be clicked.
2. Calls the scan_ports() function to scan for open ports.
3. Displays the results on the interface dynamically.

The Dash callback system ensures that the application responds to user input in real time, making the scanning experience seamless and interactive.

4.4 Displaying Results and Enhancing User Experience

Once the scan is complete, the results are displayed clearly and structured for easy interpretation. The output section includes:

- Total scanned ports summary, giving users an overview of the number of ports analyzed.
- List of open ports with descriptions, showing which ports are accessible and their associated services like HTTP (80) or SSH (22).
- Security recommendations, advising users on closing unnecessary ports, using firewalls, and implementing encryption.

Formatted with Dash HTML components, the results are dynamically updated in real-time, ensuring clarity and allowing users to quickly assess and secure their system.

4.5 Implementation of the Information Page

The application also features an information page, where users can:

- View team member details and project contributors.
- Read about project objectives and purpose.
- See a detailed explanation of how the code works.

This section is implemented using Dash callbacks, allowing users to toggle between team details, project descriptions, and the code explanation dynamically.

Example callback function for toggling content:

```
@app.callback(
    Output("details-content", "style"),
    [Input("details-button", "n_clicks")],
    [State("details-content", "style")]
)
def toggle_details(details_clicks, details_style):
    if details_clicks:
        return {"display": "block"} if details_style["display"] == "none" else {"display": "none"}
    return details_style
```

This feature ensures that users can access information without cluttering the interface, keeping the UI clean and structured.

Chapter 5

Input and Output Screens

This chapter provides a detailed view of the user interface. It includes screenshots of:

- The main page with a button to initiate the port scan.
- The results displayed after the scan is complete.
- The information page with project details and team member information.

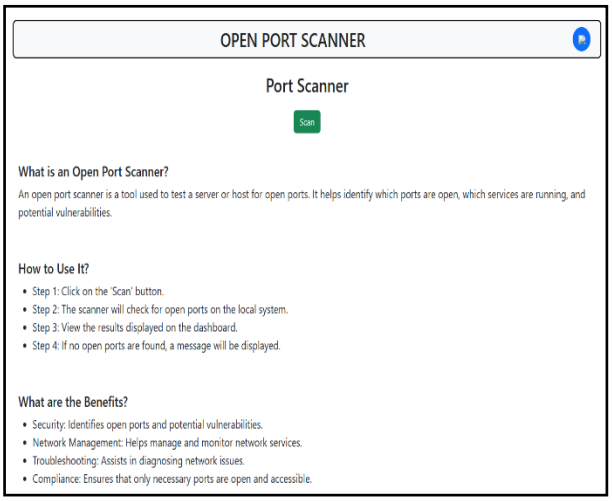

FIGURE 5.1

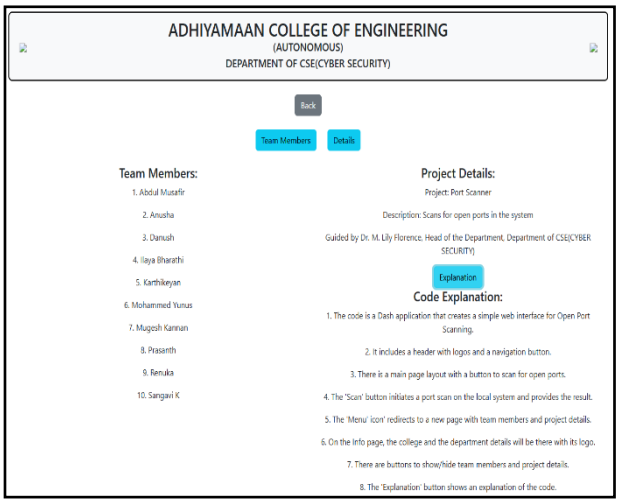

FIGURE 5.2

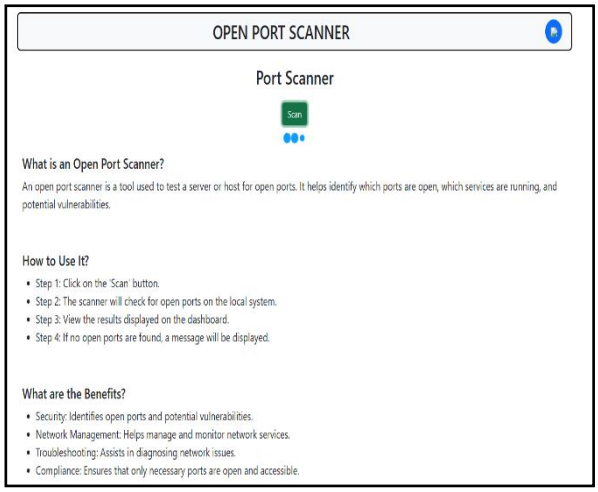

FIGURE 5.3

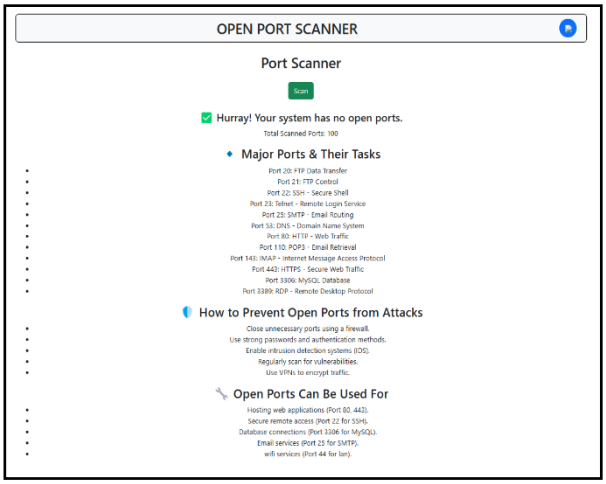

FIGURE 5.4

Chapter 6
Reports

The Open Port Scanner Web Application generates several reports that provide users with a detailed overview of the scan results, security insights, project information, and system performance metrics. These reports serve as essential tools for network administrators, cybersecurity professionals, and general users to evaluate their system's security status and take appropriate action. The application dynamically compiles these reports based on real-time scan results and presents them in a structured format for easy interpretation.

6.1 Port Scan Report

Port Scan Report provides a summary of all scanned ports and identifies which ports are open. When a user initiates a scan by clicking "Scan" button, backend processes each port using socket programming, determines its status, and dynamically updates the UI with the results. The report includes:

- Total number of scanned ports to ensure a comprehensive security analysis.
- A list of open ports, highlighting which ports are accessible and their associated services.
- Detailed descriptions of common ports, such as HTTP (80), SSH (22), FTP (21), HTTPS (443), explaining their typical usage and security risks.

The results are displayed in a structured format, using Dash HTML components to organize and present the data clearly. Open ports are emphasized, allowing users to quickly assess potential vulnerabilities.

6.2 Security Report

The Security Report offers comprehensive recommendations based on the open ports detected during the scan. Since open ports can serve as potential entry points for attackers, this report provides best practices to help users mitigate security risks and strengthen their system defenses. It includes:

- **Guidelines for closing unnecessary open ports**, ensuring that only essential services remain accessible to reduce the attack surface.
- **Firewall configuration recommendations**, advising users to block unused ports and allow traffic only through necessary services, minimizing exposure to potential threats.
- **Suggestions for enabling authentication and encryption**, particularly for ports handling sensitive data such as SSH and remote access services, preventing unauthorized access and data breaches.
- **Use of intrusion detection systems (IDS)** to monitor network traffic, detect suspicious activities, and respond to potential cyber threats in real time.
- **Regular security audits and vulnerability scans**, ensuring that open ports are continuously monitored, and any new security risks are promptly identified and addressed.
- **Implementing network segmentation**, restricting access between different systems and limiting the potential impact of a security breach.

This report helps users understand how attackers exploit open ports and provides actionable steps to enhance their system security. The information is presented in an easy-to-understand format, making it accessible to both technical and non-technical users.

6.3 Project Report

The Project Report serves as a comprehensive documentation of the Open Port Scanner Web Application, detailing its functionalities, development process, and key contributors. This report ensures transparency into the design, architecture, and implementation of the application, providing valuable insights for users, developers, and cybersecurity professionals. It is accessible through the information page, which users can navigate to via the menu button.

The **Project Report** includes:

- **Project Overview**: A high-level description of the purpose and objectives of the application, emphasizing its role in network security and the importance of identifying open ports to mitigate cyber threats.
- **System Architecture**: A detailed breakdown of the frontend and backend components of the application. The frontend, built using Dash and Bootstrap, ensures a user-friendly and responsive interface, while the backend, implemented with Python's socket programming, efficiently scans ports and processes scan results dynamically.
- **Team Members**: A list of developers and contributors involved in designing and building the system, showcasing their respective roles and contributions to the project.
- **Implementation Details**: A technical overview of how Dash, Bootstrap, and socket programming were integrated to build a seamless and interactive port scanning tool. This section provides details on how Dash components handle user interactions, callbacks trigger scan execution, and Python's socket module scans ports efficiently.

- **User Interaction Flow**: A step-by-step explanation of how users navigate the application, initiate scans, and interpret scan results. This section describes the real-time scanning process, how scan results are displayed dynamically, and how users can access security recommendations.
- **Challenges and Solutions**: An analysis of the key challenges faced during development, such as optimizing scan performance, ensuring real-time responsiveness, and improving UI usability, along with the solutions implemented to overcome them.
- **Future Enhancements**: Suggestions for expanding the application's functionality, including remote scanning capabilities, integration with external security tools, and real-time threat detection features.
- **Security Considerations:** A review of security measures implemented in the application, such as limiting scans to localhost, preventing unauthorized access, and optimizing timeout settings to avoid resource exhaustion. The ethical implications of port scanning are also discussed, ensuring that the tool is used responsibly and within legal boundaries

The Project Report provides a clear and structured overview of the application, ensuring that users and future developers understand how the system functions, how it was built, and how it can be improved. This documentation serves as a valuable reference for anyone looking to extend, modify, or integrate new security features into the tool, making it an essential part of the application's development lifecycle.

6.4 System Performance Report

The System Performance Report analyzes the efficiency of the Open Port Scanner Web Application, focusing on scan duration, system resource consumption, and

overall responsiveness. Since the port scanning process involves iterating through 1024 ports, it is crucial to ensure that the application operates efficiently without causing system slowdowns or excessive resource usage. Performance evaluation helps in identifying bottlenecks, improving scanning speed, and ensuring the application remains lightweight and effective for everyday use.

The report includes:

- **Scan Duration**: The total time taken to complete a full scan from port 1 to 1024, measuring how quickly the application can analyze open ports. Performance tests ensure that the scan time remains reasonable, even under different network conditions.
- **Resource Utilization**: The amount of CPU and memory consumption during the scan, ensuring that the application does not strain system resources. This aspect is crucial for maintaining performance on both high-end and low-end devices.
- **Optimization Strategies**: Various techniques, such as timeout settings, efficient socket handling, and limiting redundant operations, have been implemented to enhance scanning speed and reduce unnecessary processing delays. These optimizations help in reducing the total scan time without sacrificing accuracy.
- **Comparison with Other Scanning Tools**: A performance analysis of how the application compares to traditional scanning tools like Nmap, Zenmap, and Netcat, particularly in terms of speed, accuracy, and resource consumption. Unlike command-line-based tools, which may require extensive configurations, this application offers a simplified and optimized scanning process.

- **Scalability and Future Enhancements**: The current system is designed to scan only localhost ports, but future enhancements could include remote scanning capabilities, deeper network analysis, and integration with cloud-based security tools. Scalability ensures that as more features are added, system performance remains unaffected.

The Dash-based UI ensures that results are displayed dynamically without requiring page reloads, reducing processing overhead and improving response time. Additionally, the real-time loading animation prevents users from experiencing delays without feedback. By evaluating these performance metrics, users can determine the application's efficiency, reliability, and overall usability while also gaining insights into how future optimizations can enhance its effectiveness in real-world cybersecurity applications.

Chapter 7
Conclusion and Recommendations

The Open Port Scanner Web Application was successfully developed as an interactive, user-friendly, and efficient tool for identifying open ports on a local machine. The project achieved its primary objective of creating a web-based port scanner using Dash, Bootstrap, and Python's socket programming. The application allows users to initiate a scan with a single click, dynamically displays real-time results, and provides security recommendations to mitigate potential vulnerabilities. By integrating an intuitive graphical interface, the tool makes port scanning accessible to both technical and non-technical users, unlike traditional command-line scanners.

7.1 Summary of Achievements

The Open Port Scanner Web Application successfully:

- Implemented a web-based port scanner that scans ports 1 to 1024 and identifies open ports using Python's socket module.
- Developed a dynamic and interactive UI using Dash and Bootstrap, ensuring an intuitive user experience.
- Integrated real-time result updates using Dash callbacks, eliminating the need for manual page refreshes.
- Provided security insights and best practices to help users understand why open ports can be a risk and how to secure them.

- Designed a structured information page containing project details, team members, and a breakdown of the system architecture.
- Ensured smooth system performance by optimizing scanning speed, minimizing CPU/memory usage, and improving the responsiveness of the application.

Through these accomplishments, the application serves as a valuable cybersecurity tool that allows users to monitor and secure their system effectively.

7.2 Recommendations for Future Enhancements

While the current implementation of the Open Port Scanner Web Application fulfills its intended purpose, several potential improvements can be made to enhance its functionality and security capabilities:

- Remote Port Scanning: The current application is limited to localhost scanning. Adding support for remote port scanning would allow users to scan external servers and devices on a network, making the tool more versatile. However, this feature should include proper security measures and user authentication to prevent unauthorized misuse.
- Automated Security Vulnerability Detection: Enhancing the tool with a vulnerability assessment feature would allow the application to analyze open ports and compare them with known security vulnerabilities. This could be achieved by integrating a cybersecurity vulnerability database such as CVE (Common Vulnerabilities and Exposures).
- Graphical Data Visualization: The current results are displayed in a list format, but adding graphs, charts, and heatmaps could help users visualize the severity of open ports and their potential risks more effectively.
- Integration with Firewalls and Security Tools: Future versions could allow users to automatically close detected open ports by integrating the

application with firewall management systems. This would provide an additional layer of security automation.

- Multi-Platform Compatibility: Expanding the application to support mobile devices or providing a standalone desktop application could make it more accessible for users who prefer different platforms.
- User Authentication and Role-Based Access: If the application is extended to scan remote servers, adding a login system with user authentication would help prevent unauthorized scans and restrict access to specific users.
- Performance Optimization for Large-Scale Networks: If expanded to scan entire subnets or external IP ranges, the system should implement multi-threading and parallel processing to enhance scan speed and efficiency.

7.3 Final Thoughts

The Open Port Scanner Web Application is a lightweight, accessible, and efficient cybersecurity tool designed to identify open ports, provide security insights, and promote best practices for securing network services. Unlike traditional scanning tools that require technical expertise, this application delivers a simplified and interactive experience that makes port scanning easier for all users.

By implementing the recommended enhancements, the application could evolve into a more advanced network security tool, offering remote scanning, automated threat detection, and direct security integrations. Ultimately, the Open Port Scanner Web Application is a step toward improving cybersecurity awareness, equipping users with the tools they need to protect their systems from unauthorized access and potential cyber threats.

Chapter 8
Future Scope

As cybersecurity threats continue to evolve, it is crucial to enhance and expand the capabilities of the Open Port Scanner Web Application. While the current version efficiently scans ports 1 to 1024 on the local machine, there are several potential extensions that could improve its functionality, security, and usability. Future developments should focus on expanding scanning capabilities, integrating with existing security tools, and improving automation and real-time monitoring features.

8.1 Remote Port Scanning

One of the most significant enhancements would be the ability to scan remote systems. Currently, the application is limited to scanning localhost due to security constraints. Enabling remote scanning would allow network administrators and cybersecurity professionals to monitor servers, devices, and entire networks for open ports and vulnerabilities. However, to implement this feature, proper security measures such as authentication, encryption, and user access control would need to be added to prevent unauthorized scans.

8.2 Integration with Network Security Tools

To make the Open Port Scanner Web Application more powerful, future versions could integrate with existing cybersecurity and network security tools. Potential integrations include:

- **Firewall Management Systems –** Allow users to automatically block open ports based on scan results.
- **Intrusion Detection Systems (IDS) –** Combine port scanning with real-time threat detection to monitor suspicious activity**.**
- **Vulnerability Databases (CVE, NVD, or Shodan API) –** Compare open ports against known security vulnerabilities to assess risk levels.
- **SIEM (Security Information and Event Management) Systems –** Forward scan results to a centralized security platform for in-depth analysis.

These integrations would help users not only identify open ports but also take immediate action to secure their systems.

8.3 Automated Security Assessments

Currently, the application relies on manual scans, meaning users must actively trigger a scan. In future versions, the system could implement automated security assessments, where scheduled scans run at predefined intervals and generate periodic reports. Features could include:

- Daily or weekly automated port scans, alerting users to newly opened or exposed ports.
- Custom scan scheduling, allowing users to define specific time intervals for scans.
- Email or dashboard notifications, informing users of potential vulnerabilities as they arise.

By automating these security checks, the application would proactively monitor network security, reducing human effort while enhancing overall protection.

8.4 Advanced Monitoring and Visualization Features

Currently, scan results are displayed in a simple, text-based format, listing open ports and their associated services. Future improvements could enhance data visualization and reporting, making it easier for users to interpret scan results and identify security risks. Some enhancements include:

- **Graphical Charts and Heatmaps –** Display scan results using interactive charts that categorize open ports based on risk levels, frequency of detection, or service type.
- **Historical Data Tracking –** Allow users to store scan history and track network security trends over time.
- **AI-Powered Risk Analysis** – Implement machine learning algorithms to analyze scan patterns and predict potential threats before they occur.

These features would enhance user experience, making cybersecurity assessments more intuitive, efficient, and data-driven.

8.5 Mobile and Cloud-Based Port Scanning

As more organizations shift to cloud computing and mobile-driven IT infrastructure, developing a mobile version of the Open Port Scanner Web Application would increase accessibility. Possible extensions include**:**

- **Mobile Application Support –** A mobile-friendly version that allows users to initiate scans and receive alerts on the go.

- **Cloud-Based Scanning –** A web-hosted version where users can scan remote devices or cloud instances without local installation.
- **Multi-Device Compatibility –** Ensuring that the application runs smoothly on various platforms, including desktops, tablets, and smartphones.

By expanding beyond traditional local scanning, the tool could evolve into a versatile, cloud-powered cybersecurity solution.

8.6 Multi-User and Role-Based Access Control

As the Open Port Scanner Web Application grows in complexity, multi-user support and role-based access control (RBAC) could be introduced. This feature would be particularly useful for enterprise environments where multiple security analysts need access to the system. Enhancements could include:

- **User Authentication and Login System –** Restrict scanning capabilities to authorized users only.
- **Role-Based Access Control (RBAC) –** Define different user roles such as admin, auditor, and viewer, each with specific permissions.
- **Audit Logs and Activity Tracking –** Maintain a detailed record of all scans performed, ensuring compliance with security policies.

These security measures would make the application more robust and enterprise-ready, ensuring controlled access and data integrity.

8.7 Future Enhancements in Performance and Optimization

While the current application performs efficiently for localhost scans, future enhancements could focus on improving speed, reducing resource consumption, and enabling multi-threaded scanning. Some key improvements include:

- **Multi-Threading and Parallel Scanning –** Reducing scan time by scanning multiple ports simultaneously rather than sequentially.
- **Optimized Memory Management –** Ensuring that the application remains lightweight and efficient, even during high-intensity scans.
- **AI-Based Anomaly Detection –** Using machine learning models to detect unusual patterns in scan results, helping users identify potential attacks before they occur.

By continuously optimizing the core scanning engine, the application can remain efficient and scalable as security demands increase.

REFERENCES

This chapter lists the references and resources used in the development and documentation of the Open Port Scanner Web Application. These sources provided technical guidance, framework support, and best practices for implementing the frontend, backend, and cybersecurity features of the application.

1. **Plotly Dash Documentation. (2023).** ***Dash*** *– A framework for building analytical web applications.* Retrieved from https://dash.plotly.com
2. **Python Documentation:** Socket Library. (2022). *Official Python documentation on socket programming and network connections.* Retrieved from https://docs.python.org/3/library/socket.html
3. **Bootstrap Documentation. (2022).** *Bootstrap 5 framework for responsive web design and UI styling.* Retrieved from https://getbootstrap.com/docs/5.0/getting-started/introduction/
4. **Nmap: The Network Mapper. (2022).** *Official Nmap guide on port scanning techniques and network security analysis.* Retrieved from https://nmap.org/book/man.html
5. **OWASP (Open Web Application Security Project). (2023**). *Best practices for network security and port vulnerability assessments.* Retrieved from https://owasp.org/www-project-top-ten/
6. **Flask Documentation. (2022).** ***Flask*** *– Lightweight web framework used within Dash applications.* Retrieved from https://flask.palletsprojects.com
7. **Cybersecurity and Infrastructure Security Agency (CISA). (2023).** *Guidelines on securing open ports and preventing unauthorized access.* Retrieved from https://www.cisa.gov

8. **Shodan API Documentation. (2023).** ***Shodan*** *– A search engine for discovering internet-connected devices and analyzing open ports.* Retrieved from https://developer.shodan.io

These references provided essential information for implementing the Open Port Scanner Web Application, ensuring best practices in network security, web development, and performance optimization.

APPENDIX

Appendix A: Implementation of the Code

```
import dash
import dash_bootstrap_components as dbc
from dash import dcc, html
from dash.dependencies import Input, Output, State
import socket

app = dash.Dash(__name__, external_stylesheets=[dbc.themes.BOOTSTRAP])
server = app.server

# Define the layout for the main page
main_page_layout = dbc.Container([
    dbc.Row([
        dbc.Col(html.Div([
            html.H2("OPEN PORT SCANNER", style={"margin-bottom": "0"})
        ], style={"text-align": "center"})),
         dbc.Col(dbc.Button(html.Img(src="images/menu.png"), href="/info", color="primary", n_clicks=0,
                                style={"border-radius": "50%", "width": "40px", "height": "40px"}), width="auto"),
       ], align="center", justify="between", className="my-4", style={"background-color": "#f8f9fa", "padding": "10px", "border-radius": "8px", "border": "2px solid #000"}),
    html.H2("Port Scanner", className="text-center my-4"),
    dbc.Row([
        dbc.Col([
            dbc.Button("Scan", id="scan-button", color="success", n_clicks=0),
        ], width="auto"),
```

```
    ], className="text-center mb-4", justify="center"),
    dbc.Row([
        dbc.Col([
            dcc.Loading(
                id="loading-animation",
                type="dot",
                children=[html.Div(id="output-content")]
            )
        ], className="text-center")
    ]),
    # Additional content sections
    dbc.Row([
        dbc.Col([
            html.H3("What is an Open Port Scanner?", style={"font-size": "24px"}),
            html.P("An open port scanner is a tool used to test a server or host for open ports. It helps identify which ports are open, which services are running, and potential vulnerabilities.", style={"font-size": "20px"}),
        ], className="my-4"),
    ]),
    dbc.Row([
        dbc.Col([
            html.H3("How to Use It?", style={"font-size": "24px"}),
            html.Ul([
                html.Li("Step 1: Click on the 'Scan' button.", style={"font-size": "20px"}),
                html.Li("Step 2: The scanner will check for open ports on the local system.", style={"font-size": "20px"}),
                html.Li("Step 3: View the results displayed on the dashboard.", style={"font-size": "20px"}),
                html.Li("Step 4: If no open ports are found, a message will be displayed.", style={"font-size": "20px"}),
            ]),
```

```
        ], className="my-4"),
    ]),
    dbc.Row([
        dbc.Col([
            html.H3("What are the Benefits?", style={"font-size": "24px"}),
            html.Ul([
                html.Li("Security: Identifies open ports and potential vulnerabilities.", style={"font-size": "20px"}),
                html.Li("Network Management: Helps manage and monitor network services.", style={"font-size": "20px"}),
                html.Li("Troubleshooting: Assists in diagnosing network issues.", style={"font-size": "20px"}),
                html.Li("Compliance: Ensures that only necessary ports are open and accessible.", style={"font-size": "20px"}),
            ]),
        ], className="my-4"),
    ]),
])

# Define the layout for the info page
info_page_layout = dbc.Container([
    dbc.Row([
        dbc.Col(html.Img(src="images/acelogo.png", height="80px"), width="auto"),
        dbc.Col(html.Div([
            html.H2("ADHIYAMAAN COLLEGE OF ENGINEERING", style={"margin-bottom": "0"}),
            html.H5("(AUTONOMOUS)", style={"margin-top": "0", "margin-bottom": "5px"}), # Added line space here
            html.H5("DEPARTMENT OF CSE(CYBER SECURITY)", style={"margin-top": "5px"})
        ], style={"text-align": "center"})),
        dbc.Col(html.Img(src="images/cyberlogo.png", height="80px"), width="auto")
```

```
    ], align="center", justify="center", className="my-4", style={"background-color": "#f8f9fa", "padding": "10px", "border-radius": "8px", "border": "2px solid #000"}),  # Added border
    dbc.Row([
        dbc.Col(dbc.Button("Back", href="/", color="secondary", n_clicks=0), width="auto")
    ], align="center", justify="center", className="my-4"),
    dbc.Row([
        dbc.Col([
            dbc.Button("Team Members", id="team-button", color="info", className="mr-2", n_clicks=0)
        ], width="auto"),
        dbc.Col([
            dbc.Button("Details", id="details-button", color="info", className="mr-2", n_clicks=0)
        ], width="auto")
    ], className="text-center mb-4", justify="center"),
    dbc.Row([
        dbc.Col([
            html.Div(id="team-members-content", style={"display": "none", "text-align": "center"})
        ]),
        dbc.Col([
            html.Div(id="details-content", style={"display": "none", "text-align": "center"})
        ])
    ]),
])

@app.callback(
    Output("team-members-content", "style"),
    [Input("team-button", "n_clicks")],
    [State("team-members-content", "style")]
```

```
)
def toggle_team_members(team_clicks, team_style):
    if team_clicks:
        if team_style["display"] == "none":
            return {"display": "block", "text-align": "center"}
        else:
            return {"display": "none", "text-align": "center"}
    return team_style

@app.callback(
    Output("details-content", "style"),
    [Input("details-button", "n_clicks")],
    [State("details-content", "style")]
)
def toggle_details(details_clicks, details_style):
    if details_clicks:
        if details_style["display"] == "none":
            return {"display": "block", "text-align": "center"}
        else:
            return {"display": "none", "text-align": "center"}
    return details_style

@app.callback(
    Output("team-members-content", "children"),
    [Input("team-button", "n_clicks")]
)
def display_team_members(n_clicks):
    if n_clicks:
        return html.Div([
```

```
            html.H4("Team Members:"),
            html.P("1. Abdul Musafir"),
            html.P("2. Anusha"),
            html.P("3. Danush"),
            html.P("4. Ilaya Bharathi"),
            html.P("5. Karthikeyan"),
            html.P("6. Mohammed Yunus"),
            html.P("7. Mugesh Kannan"),
            html.P("8. Prasanth"),
            html.P("9. Renuka"),
            html.P("10. Sangavi K")
        ])
    return html.Div()

@app.callback(
    Output("details-content", "children"),
    [Input("details-button", "n_clicks")]
)
def display_project_details(n_clicks):
    if n_clicks:
        return html.Div([
            html.H4("Project Details:"),
            html.P("Project: Port Scanner"),
            html.P("Description: Scans for open ports in the system"),
              html.P("Guided by Dr. M. Lily Florence, Head of the Department, Department of CSE(CYBER SECURITY)"),
            dbc.Button("Explanation", id="explanation-button", color="info", n_clicks=0),
            html.Div(id="explanation-content", style={"display": "none", "text-align": "center"})
        ])
```

```
    return html.Div()


@app.callback(
    Output("explanation-content", "style"),
    [Input("explanation-button", "n_clicks")],
    [State("explanation-content", "style")]
)
def toggle_explanation(explanation_clicks, explanation_style):
    if explanation_clicks:
        if explanation_style["display"] == "none":
            return {"display": "block", "text-align": "center"}
        else:
            return {"display": "none", "text-align": "center"}
    return explanation_style


@app.callback(
    Output("explanation-content", "children"),
    [Input("explanation-button", "n_clicks")]
)
def display_explanation(n_clicks):
    if n_clicks:
        return html.Div([
            html.H4("Code Explanation:"),
            html.P("1. The code is a Dash application that creates a simple web interface for Open Port Scanning."),
            html.P("2. It includes a header with logos and a navigation button."),
            html.P("3. There is a main page layout with a button to scan for open ports."),
            html.P("4. The 'Scan' button initiates a port scan on the local system and provides the result."),
```

```
            html.P("5. The 'Menu' icon' redirects to a new page with team members and project details."),
            html.P("6. On the Info page, the college and the department details will be there with its logo."),
            html.P("7. There are buttons to show/hide team members and project details."),
            html.P("8. The 'Explanation' button shows an explanation of the code.")
        ])
    return html.Div()


# Port Scanning Callback
@app.callback(
    Output("output-content", "children"),
    [Input("scan-button", "n_clicks")]
)
def scan_ports(n_clicks):
    if n_clicks:
        total_ports = 100 # Total ports scanned
        open_ports = []
        common_ports = {
            20: "FTP Data Transfer",
            21: "FTP Control",
            22: "SSH - Secure Shell",
            23: "Telnet - Remote Login Service",
            25: "SMTP - Email Routing",
            53: "DNS - Domain Name System",
            80: "HTTP - Web Traffic",
            110: "POP3 - Email Retrieval",
            143: "IMAP - Internet Message Access Protocol",
            443: "HTTPS - Secure Web Traffic",
            3306: "MySQL Database",
```

```
    3389: "RDP - Remote Desktop Protocol"
}

for port in range(1, total_ports + 1):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(0.5)
    result = s.connect_ex(('localhost', port))
    if result == 0:
        open_ports.append(port)
    s.close()

if open_ports:
    open_ports_info = html.Div([
        html.H4("⚠ Open Ports Detected!"),
        html.P(f"Total Scanned Ports: {total_ports}"),
        html.P(f"Open Ports: {', '.join(map(str, open_ports))}"),
    ])
else:
    open_ports_info = html.Div([
        html.H4("✅ Hurray! Your system has no open ports."),
        html.P(f"Total Scanned Ports: {total_ports}")
    ])

major_ports_section = html.Div([
    html.H3("🔹 Major Ports & Their Tasks"),
    html.Ul([html.Li(f"Port {port}: {desc}") for port, desc in common_ports.items()])
])

security_tips = html.Div([
```

```
        html.H3("🛡 How to Prevent Open Ports from Attacks"),
        html.Ul([
            html.Li("Close unnecessary ports using a firewall."),
            html.Li("Use strong passwords and authentication methods."),
            html.Li("Enable intrusion detection systems (IDS)."),
            html.Li("Regularly scan for vulnerabilities."),
            html.Li("Use VPNs to encrypt traffic.")
        ])
    ])

    open_port_uses = html.Div([
        html.H3("🔧 Open Ports Can Be Used For"),
        html.Ul([
            html.Li("Hosting web applications (Port 80, 443)."),
            html.Li("Secure remote access (Port 22 for SSH)."),
            html.Li("Database connections (Port 3306 for MySQL)."),
            html.Li("Email services (Port 25 for SMTP)."),
            html.Li("wifi services (Port 44 for lan).")
        ])
    ])

    return html.Div([
        open_ports_info,
        major_ports_section,
        security_tips,
        open_port_uses
    ])

return html.Div()
```

```
@app.callback(Output("page-content", "children"),
              [Input("url", "pathname")])
def display_page(pathname):
    if pathname == "/info":
        return info_page_layout
    else:
        return main_page_layout


app.layout = html.Div([
    dcc.Location(id="url", refresh=False),
    html.Div(id="page-content")
])


if __name__ == "__main__":
    app.run_server(debug=True)
```

B.1 Launching the Application

To use the Open Port Scanner Web Application, the following steps must be followed to ensure proper setup and execution:

Prerequisites

Before launching the application, ensure that the following dependencies are installed:

pip install dash dash_bootstrap_components

The application requires Python 3.x, the Dash framework, and Bootstrap components for UI design.

Running the Application Locally

1. Download the application files or clone the project repository from GitHub.
2. Navigate to the project directory using the terminal or command prompt.
3. Run the following command to start the application:

python app.py

4. Wait for the application to launch, then open a web browser and enter the following URL**:**

http://127.0.0.1:8050/

5. The application dashboard will load, allowing users to begin port scanning.

Deploying the Application on a Server

To make the application accessible over a network or the internet, users can deploy it using:

- Flask and Gunicorn for server hosting.
- Cloud platforms such as Heroku, AWS, or Google Cloud.
- Docker containers for an isolated deployment environment**.**

Appendix B: User Guide for Operating the Web Application

This user guide provides step-by-step instructions on how to install, launch, and operate the Open Port Scanner Web Application. It includes details on how to run the application, perform port scans, and navigate the information page for project-related details.

B.2 Using the Application

Initiating a Port Scan

1. Open the Open Port Scanner Web Application in a browser.
2. Click the "Scan" button on the main dashboard to initiate a scan.
3. The system will scan ports 1 to 1024 and attempt connections to determine which ports are open.
4. A loading animation will appear while the scan is in progress.

Viewing Scan Results

- Once the scan is complete, the results will be displayed dynamically on the dashboard.
- If open ports are found, they will be listed along with their port numbers and associated services (e.g., Port 22 – SSH, Port 80 – HTTP).
- If no open ports are found, a message will indicate that the system is secure.

Understanding Security Recommendations

- Below the scan results, the application provides security recommendations to help users secure their network.

- Suggestions include closing unnecessary ports, enabling firewalls, and using authentication and encryption to protect sensitive services.

B.3 Accessing the Information Page

Viewing Project and Team Details

1. Click on the menu button (navigation icon) on the dashboard.
2. Select "Project Information" to access team member details and system documentation.
3. **The page includes:**
    - Project Overview – A brief summary of the application's purpose.
    - Team Members – Names of contributors to the project.
    - Code Explanation – A breakdown of how the port scanning system is implemented.

Navigating Between Pages

- Users can return to the main scanning dashboard by clicking the "Back" button on the Project Information page.
- Additional sections include team details and a guide to understanding the scanning process.

B.4 Troubleshooting

Common Issues and Fixes

Issue	Possible Cause	Solution
Application does not launch	Dependencies not installed	Run pip install dash dash_bootstrap_components
Scan results do not appear	Firewall blocking scan	Disable local firewall temporarily for testing
High CPU usage	Too many ports being scanned	Reduce the number of scanned ports or optimize timeout settings
"Address already in use" error	Port conflict with another process	Restart the application or use a different port

This user guide provides all necessary steps to run, navigate, and troubleshoot the Open Port Scanner Web Application, ensuring a smooth user experience.