PROJECT REPORT ON

ADVANCED PORT SCANNER

Submitted by

## LINGAMALLU VASU DEVA SIVA SAI SUBBARAO

## and Supraja Technologies employee id: ST#IS#7016

Under the Supervision of

[**𝐒𝐢𝐝𝐝𝐡𝐮 𝐊𝐮𝐧𝐝𝐮𝐫𝐭𝐡𝐢**](https://www.linkedin.com/in/ACoAACiRR_kBmJjR1T1-mj1eqYK0BxXbyFx6i14)

**Senior Security Analyst**

**KRISHNA**

**Junior Security Analyst**

****

**Registered And Head Office D.NO: 11-9-18, 1st Floor,**

**Majjivari Street, Kothapeta, Vijayawada - 520001.**

**+91 9550055338 / +91 7901336873**

[**contact@suprajatechnologies.com**](mailto:contact@suprajatechnologies.com)

# Table of Contents

|  |  |  |  |
| --- | --- | --- | --- |
| **1.** |  | **Introduction ................................................................................................................ ......** | **1** |
|  | 1.1 | Overall Description.......................................................................................................... | 1 |
| **2.** |  | **Existing System ............................................................................................................. ...** | **2** |
| **3.** |  | **Proposed System ..............................................................................................................** | **3** |
| **4.** |  | **System Design ............................................................................................................... ...** | **4** |
|  | **4.1** | **Feasibility Study ..............................................................................................................** | **4** |
|  |  | 4.1.1 Economical Feasibility ..................................................................................... | 4 |
|  |  | 4.1.2 Technical Feasibility ......................................................................................... | 4 |
|  |  | 4.1.3 Social Feasibility .............................................................................................. | 4 |
|  | **4.2** | **Input and Output Design ..................................................................................................** | **4** |
|  |  | 4.2.1 Input Design .................................................................................................... | 4 |
|  |  | 4.2.2 Objectives ......................................................................................................... | 4 |
|  |  | 4.2.3 Output Design................................................................................................... | 4 |
| **5.** |  | **Implementation ....................................................................................... ..........................** | **5** |
|  | **5.1** | **Module Description ..........................................................................................................** | **5** |

* + 1. Domain Resolution Module 5
    2. Port Scanning Module 5
    3. Report Generation Module 5
    4. Email Sending Module………………………………………………………5
    5. **GUI Module…………………………………………………………………5**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **5.2** | **System Architecture .........................................................................................................** | **5** |
| **6.** |  | **Algorithm implementation .................................................................................................** | **6** |
|  | 6.1 | Algorithm Implementaion.................................................................................................... | 6 |
| **7.** |  | **System Design ............................................................................................................... ...** | **7** |

* 1. Data Flow Diagram 7

1. [Requirement Specification 8](#_TOC_250015)
   1. Functional Requirements 8
   2. [Software Requirements 8](#_TOC_250014)
   3. [Operating Systems Supported 8](#_TOC_250013)
   4. [Technologies and languages used to Develop 8](#_TOC_250012)
   5. [Hardware Requirements 8](#_TOC_250011)
2. [System Test 9](#_TOC_250010)
   1. Types of Test 9
      1. [unit Testing 9](#_TOC_250009)
      2. [Integration Testing 9](#_TOC_250008)
      3. Functional Testing 9
      4. System Testing 9
      5. [white box testing 9](#_TOC_250007)
      6. [Black Box Testing 9](#_TOC_250006)
   2. [Test Strategy and Approach 9](#_TOC_250005)
      1. [Test Objectives 9](#_TOC_250004)
      2. [Features to be Tested 9](#_TOC_250003)
   3. [Integration Testing 9](#_TOC_250002)
      1. Test Results 9
   4. [Acceptance Testing 9](#_TOC_250001)
      1. Test Results 9
3. [conclusion 10](#_TOC_250000)

# Introduction

## Overall Description

In the evolving landscape of cybersecurity, ensuring the protection of network infrastructure is paramount. Port scanning serves as a fundamental process for identifying open ports and potential vulnerabilities that could be exploited by attackers. This project introduces an Advanced Port Scanner, designed to deliver accurate and efficient network analysis, offering robust insights for enhanced security.

The tool utilizes data analysis techniques to monitor and evaluate network behaviour, identifying patterns that reveal open TCP and UDP ports. It incorporates a dynamic and user-friendly interface built with Python’s Tkinter library, enabling users to:

Resolve domain names to IP addresses seamlessly.

Conduct port scans across specified ranges or custom port lists for TCP, UDP, or both protocols.

Generate detailed, time-stamped reports summarizing the scan results.

Automate report delivery via email to designated recipients for real-time monitoring and follow-up actions.

The Advanced Port Scanner is equipped to enhance awareness of network vulnerabilities, allowing users to proactively mitigate risks. By generating actionable insights and providing a streamlined process for port analysis, this tool serves as a vital asset for system administrators, IT professionals, and cybersecurity enthusiasts aiming to secure their networks.

Its design emphasizes accuracy, efficiency, and ease of use, ensuring that even users with limited technical expertise can effectively utilize its features. This solution bridges the gap between accessibility and technical depth, delivering a reliable mechanism for safeguarding digital ecosystems.

# Existing System

the current landscape, network administrators and security professionals rely on various tools and utilities for port scanning and vulnerability assessment. These tools, while effective, often suffer from limitations that impact their accessibility and efficiency. Most existing systems for port scanning fall into one of the following categories:

1. **Command-Line Tools**: Utilities like Nmap and Netcat dominate the market but come with steep learning curves for individuals without a technical background. They require a strong understanding of network protocols and command-line syntax, making them less user-friendly for beginners.
2. **Platform-Specific Solutions**: Many port scanners are designed for specific operating systems or environments, which limits their usability in diverse infrastructures. This lack of cross-platform compatibility can hinder organizations with heterogeneous systems.
3. **Limited Automation**: Existing tools typically require manual intervention at every step, such as initiating scans, saving results, and analysing data. Automated reporting and alert mechanisms are often absent or require additional customization, increasing the workload for users.
4. **Fragmented Features**: While some systems focus exclusively on port scanning, others provide in-depth vulnerability assessments. However, very few tools integrate scanning, reporting, and automated email notifications into a single, cohesive solution.

These gaps in existing systems create a need for a comprehensive tool that combines ease of use, automation, and detailed reporting. The lack of an intuitive graphical interface in most port scanners further alienates non-technical users, preventing wider adoption. Moreover, most solutions do not incorporate data analysis techniques to identify patterns or trends, which could further improve security decision-making.

# Proposed System

To address the limitations of existing port scanning tools, the **Advanced Port Scanner** is designed as an efficient, user-friendly, and feature-rich solution. This system integrates advanced scanning capabilities, automated reporting, and a graphical user interface, making it accessible to both technical and non-technical users.

Key features of the proposed system include:

1. **User-Friendly GUI**:  
   Unlike traditional command-line tools, this system employs a Tkinter-based graphical user interface that simplifies user interaction. The interface allows users to input scan parameters, monitor progress, and view results in real time without requiring any prior technical expertise.
2. **Comprehensive Port Scanning**:  
   The system supports scanning of **TCP**, **UDP**, or both protocols across specified ranges or custom port lists. This flexibility ensures that users can tailor scans to their specific needs while maintaining a high level of accuracy.
3. **Automation and Reporting**:  
   The tool automatically generates detailed, time-stamped reports after each scan. These reports summarize open ports and associated protocols and can be saved locally for further analysis. Additionally, the system includes an automated email notification feature to send reports to designated recipients, ensuring seamless communication and monitoring.
4. **Cross-Platform Compatibility**:  
   Built using Python, the system is designed to run on multiple operating systems, including Windows, macOS, and Linux, making it suitable for diverse infrastructure environments.
5. **Integrated Data Analysis**:  
   Leveraging data analysis principles, the system identifies patterns in network behavior and highlights potential vulnerabilities. This added layer of intelligence aids in proactive decision-making to secure the network.
6. **Enhanced Security Awareness**:  
   The system promotes awareness of open ports and potential entry points that could be exploited by malicious actors. By providing actionable insights, it empowers users to take preemptive measures to safeguard their networks.

By combining these features into a single tool, the **Advanced Port Scanner** overcomes the challenges of existing systems. It bridges the gap between usability and functionality, providing a robust, scalable, and effective solution for securing network infrastructure.

# System Design

The **Advanced Port Scanner** is built as a modular and extensible system, ensuring scalability, maintainability, and efficient operation. It consists of the following components:

### 1. **Architecture**

The system follows a **modular architecture** with three primary layers:

* **Input Layer**:  
  Handles user input through a GUI, allowing the user to configure parameters such as host address, port ranges, protocols (TCP/UDP), and recipient email for automated reporting.
* **Processing Layer**:  
  Responsible for performing the core operations of the system:
  + Resolving the domain name to an IP address.
  + Scanning ports using specified protocols and parameters.
  + Logging results and generating reports in real time.
  + Integrating email functionality to send automated reports.
* **Output Layer**:  
  Displays results in the GUI and generates time-stamped reports, saving them locally and optionally sending them via email.

### 2. **Components**

#### **Graphical User Interface (GUI)**

* Developed using Python's **Tkinter** library.
* Includes fields for host input, port ranges, protocol selection, and email configuration.
* Displays scan results in real time using a scrolled text widget.
* Provides buttons for starting the scan and generating reports.

#### **Port Scanning Module**

* **TCP Scanning**: Establishes socket connections to detect open TCP ports.
* **UDP Scanning**: Sends datagrams to identify open UDP ports.
* Scans specified ranges or custom ports based on user input.
* Uses multithreading to speed up the scanning process.

#### **Domain Resolution Module**

* Converts domain names to IP addresses using Python's **socket** library.
* Validates the host and ensures accurate resolution.

#### **Report Generation Module**

* Formats scan results into a detailed, human-readable report.
* Saves reports in the local "Downloads" directory with time-stamped filenames.

#### **Email Notification Module**

* Uses Python's **smtplib** library to send reports via email.
* Ensures secure email communication with encryption (TLS).
* Requires the user to input recipient email credentials securely.

### 3. **Workflow**

1. **User Input**:
   * The user enters the host, port ranges, protocol(s), and recipient email in the GUI.
2. **Domain Resolution**:
   * The system resolves the domain name to an IP address.
3. **Port Scanning**:
   * The system scans the specified ports using the selected protocol(s).
   * Open ports are identified and logged in real time.
4. **Report Generation**:
   * A detailed report is created summarizing the open ports and scanning parameters.
5. **Email Notification (Optional)**:
   * If an email is provided, the system sends the generated report to the specified recipient.
6. **Output**:
   * Results are displayed in the GUI, and the report is saved locally.

### 4. **Technologies Used**

* **Programming Language**: Python
* **Libraries**:
  + **Tkinter**: For the graphical user interface.
  + **Socket**: For domain resolution and network communication.
  + **smtplib**: For email functionality.
  + **Pathlib**: For handling file paths.
  + **Threading**: For concurrent port scanning.

### 5. **System Flow Diagram**

**Step 1**: User inputs scan parameters into the GUI.  
↓  
**Step 2**: System resolves the domain to an IP address.  
↓  
**Step 3**: Port scanning is initiated using the specified parameters.  
↓  
**Step 4**: Open ports are logged and displayed in the GUI in real time.  
↓  
**Step 5**: The system generates a detailed report of the scan results.  
↓  
**Step 6**: Optional: The report is emailed to the specified recipient.  
↓  
**Step 7**: The process completes, and all outputs are saved/displayed.

## Feasibility Study:

The feasibility of the project is analyzed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations in feasibility analysis are:

Economic Feasibility Technical Feasibility Social Feasibility

## Economic Feasibility

The **Advanced Port Scanner** offers significant cost savings by automating port scanning and report generation, reducing labor time and human error. Its development relies on free, open-source tools, minimizing upfront costs. With minimal hardware requirements, it’s accessible to a wide range of users, including small businesses and educational institutions. The tool enhances security, preventing costly breaches and ensuring compliance, while offering a free alternative to expensive commercial solutions. The scalability of the system and potential for future premium features further increase its long-term value.

## Technical Feasibility

The **Advanced Port Scanner** utilizes Python with libraries like **Tkinter** for GUI, **Socket** for domain resolution and port scanning, and **smtplib** for email reporting. It supports both **TCP** and **UDP** scanning, using multithreading for efficiency. The system resolves domain names, scans ports in specified ranges or custom lists, generates detailed reports, and optionally emails them. It ensures low system resource usage, making it suitable for various devices without heavy hardware requirements.

* + 1. **Social Feasibility**

The **Advanced Port Scanner** promotes enhanced cybersecurity awareness among individuals and organizations, fostering safer online environments. By providing a free, easy-to-use tool, it empowers small businesses and educational institutions to conduct basic security assessments without relying on costly solutions. The tool also supports the educational sector, helping students and professionals gain practical experience in network security. Its ability to detect vulnerabilities early reduces the risk of cybercrime, contributing to overall digital safety and privacy.

**4.2 INPUT AND OUTPUT DESIGN:**

**4.2.1 INPUT DESIGN**

The **Input Design** for the **Advanced Port Scanner** focuses on user-friendly interaction through a graphical user interface (GUI) built with **Tkinter**. The inputs include:

1. **Host**: A text box to input the domain name or IP address for scanning.
2. **Start Port & End Port**: Numeric fields to specify the port range to scan.
3. **Specific Ports**: A text box to input a comma-separated list of custom ports.
4. **Protocol**: Radio buttons to select the scan protocol (TCP, UDP, or Both).
5. **Recipient Email**: A text box to input the email address for sending the report.

These inputs allow users to customize the scan parameters and configure the email report functionality.

**4.2.2 OBJECTIVE**

The main objectives of the **Advanced Port Scanner** are:

1. **Efficient Port Scanning**: To quickly and accurately scan TCP and UDP ports for open or vulnerable services.
2. **User-Friendly Interface**: To provide an intuitive GUI for easy configuration and operation by users with varying technical expertise.
3. **Security Awareness**: To help users identify potential network vulnerabilities and enhance cybersecurity awareness.
4. **Report Generation and Emailing**: To automatically generate scan reports and send them via email for easy review and documentation.
5. **Cost-Effective Solution**: To offer a free, lightweight, and accessible tool for small businesses, educational institutions, and individuals.

**4.2.3 OUTPUT DESIGN**

The **Input Design** for the **Advanced Port Scanner** focuses on user-friendly interaction through a graphical user interface (GUI) built with **Tkinter**. The inputs include:

1. **Host**: A text box to input the domain name or IP address for scanning.
2. **Start Port & End Port**: Numeric fields to specify the port range to scan.
3. **Specific Ports**: A text box to input a comma-separated list of custom ports.
4. **Protocol**: Radio buttons to select the scan protocol (TCP, UDP, or Both).
5. **Recipient Email**: A text box to input the email address for sending the report.

These inputs allow users to customize the scan parameters and configure the email report functionality.

# 5. Implementation

# 5.1 Module Description

**5.1.1 Module 1- Domain Resolution Module**:

* **Purpose**: Resolves the domain name or IP address of the host to be scanned.
* **Functionality**:
  + Takes a hostname or domain as input.
  + Uses the socket.gethostbyname() function to resolve the domain to its corresponding IP address.
  + If the domain cannot be resolved, an error message is displayed in the output.
* **Key Component**: socket.gethostbyname()

**5.1.2 Module 2- Port Scanning Module:**

 **Purpose**: Scans specified ports (TCP or UDP) to check for open or closed status.

 **Functionality**:

* Scans a range of ports or user-defined specific ports.
* Supports both **TCP** and **UDP** protocols. TCP uses socket.AF\_INET and socket.SOCK\_STREAM, while UDP uses socket.SOCK\_DGRAM.
* Attempts to connect to each port, and if the connection is successful, marks it as "Open."
* For UDP, the module sends a message and waits for a response.
* Results are displayed in the GUI for immediate feedback.

 **Key Components**: socket.socket(), connect\_ex(), sendto(), recvfrom()

**5.1.3 Module 3-Report Generation Module:**

 **Purpose**: Generates a detailed text report of the port scan results.

 **Functionality**:

* Collects scan results (open ports, protocol, and status).
* Creates a formatted text file that includes:
  + Host information
  + Scan time
  + Port details (port number, protocol, status)
* Saves the report in the user's **Downloads** folder, named with the hostname and timestamp.

 **Key Components**: datetime.now(), Path.home(), file writing operations (open())

### **5.1.4 Module 4** - **Email Sending Module:**

 **Purpose**: Sends the generated scan report via email to a specified recipient.

 **Functionality**:

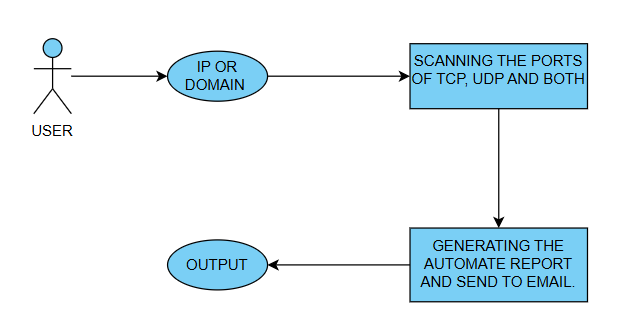
* Uses **smtplib** to connect to Gmail’s SMTP server (or another email provider).
* Sends an email containing the scan report as plain text.
* Requires the sender's email credentials to authenticate.
* Notifies the user of successful email delivery or an error message if something goes wrong.

 **Key Components**: smtplib.SMTP(), MIMEMultipart, MIMEText, email login, server.sendmail()

**5.1.5 Module 5 - GUI Module:**

* **Purpose**: Provides the user interface for interacting with the port scanner.
* **Functionality**:
  + Built using **Tkinter**, allowing users to input scan parameters (host, port range, specific ports, protocol, email).
  + Displays results (resolved IP, open ports, scan completion, report location) in a **ScrolledText** widget.
  + Provides a **Start Scan** button to initiate the scan and generate the report.
  + Allows users to customize input options such as start/end ports, protocols, and email address.
* **Key Components**: Tkinter.Entry(), Tkinter.Radiobutton(), ScrolledText, Button()

**5.2 SYSTEM ARCHITECTURE**

****

# 6. Algorithm Implementation:

### **Algorithmic Approach for Port Scanning**

The approach implemented in the provided code can be summarized in the following steps:

#### 1. **Resolve Domain to IP Address:**

* The algorithm first resolves the hostname (domain) to its corresponding IP address using a DNS query (socket.gethostbyname).
* If the domain is invalid, the algorithm exits, indicating failure.

#### 2. **Scan Ports (TCP/UDP or Both):**

* The algorithm iterates through the list of ports to be scanned.
* For each port, depending on the selected protocol (TCP, UDP, or both), it attempts to:
  + **TCP Scan**: Attempts a connection to the given port using a TCP socket (socket.AF\_INET, socket.SOCK\_STREAM).
  + **UDP Scan**: Sends a UDP packet to the port and waits for a response.
* The algorithm checks whether each port is open and reports the result.

#### 3. **Generate Report:**

* After scanning, the algorithm generates a report (in text format) containing the scanned ports and their status (open or closed).
* This report is saved in the Downloads directory.

#### 4. **Send Email Report:**

* If the user provides an email, the algorithm sends the generated report via email using smtplib with the specified email credentials.

#### 5. **Output Results:**

* The results of the port scan and email status (success or failure) are displayed in a Tkinter GUI window.

### **Time Complexity:**

1. **Resolving Domain to IP:**
   * The time complexity for domain resolution is **O(1)** since it’s a single operation and does not depend on the number of ports or any other input parameters.
2. **Scanning Ports (TCP/UDP):**
   * The algorithm scans a range of ports or a list of specific ports. Let P be the number of ports to scan, and T be the time taken to check a single port.
   * The time complexity for scanning the ports is **O(P \* T)**, where:
     + P is the number of ports being scanned.
     + T is the time complexity of checking a single port (constant time **O(1)** for both TCP and UDP scans in typical cases).

The **T** can vary slightly depending on the network conditions, but in most cases, it’s treated as a constant value.

1. **Generating Report:**
   * Generating the report involves writing the results for each open port to a file. Since there are P open ports at most, the time complexity for report generation is **O(P)**.
2. **Sending Email:**
   * Sending an email with the report involves reading the file, constructing the email, and sending it via SMTP. The time complexity of this operation is **O(R)**, where R is the size of the generated report.

### **Space Complexity:**

* The space complexity is influenced by the following factors:
  + **Port List**: Storing a list of open ports (maximum size is P), so the space complexity is **O(P)**.
  + **Report Generation**: The size of the report depends on the number of open ports, so it’s **O(P)**.
  + **Email Content**: The report content is stored in memory while sending the email, which is also **O(P)**.
  + The space complexity of the entire system is **O(P)**.

### **Recurrence Relation:**

1. **Port Scanning Function** (scan\_ports):
   * For each port p, we are making a call to scan\_tcp or scan\_udp, which takes constant time T.
   * The recurrence relation for scanning P ports can be written as:

T(P)=P×TT(P) = P \times TT(P)=P×T

Where:

* + T(P) is the total time for scanning P ports.
  + T is the time to check a single port, which is **O(1)** in this case.

1. **Report Generation**:
   * The report generation depends linearly on the number of open ports P. For each open port, a line is added to the report:

Treport(P)=O(P)T\_{report}(P) = O(P)Treport​(P)=O(P)

1. **Email Sending**:
   * The time complexity of email sending depends on the report size R (which is proportional to P). Therefore:

Temail(R)=O(R)T\_{email}(R) = O(R)Temail​(R)=O(R)

Since R = P, we have:

Temail(P)=O(P)T\_{email}(P) = O(P)Temail​(P)=O(P)

### **Overall Time Complexity:**

* **Scanning Ports**: **O(P)**
* **Report Generation**: **O(P)**
* **Sending Email**: **O(P)**

Thus, the overall time complexity of the system is:

T(P)=O(P)+O(P)+O(P)=O(P)T(P) = O(P) + O(P) + O(P) = O(P)T(P)=O(P)+O(P)+O(P)=O(P)

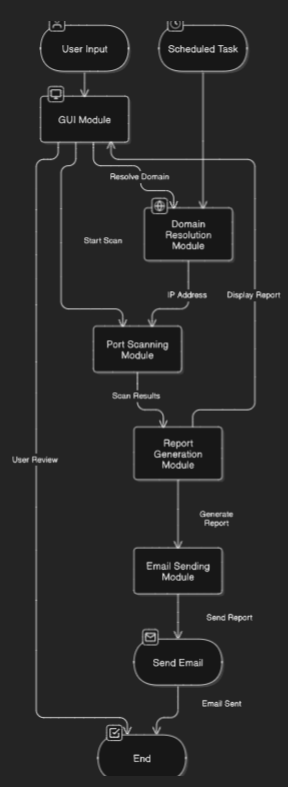
Where P is the number of ports to be scanned.

### **Conclusion:**

* **Time Complexity**: The overall time complexity is **O(P)**, where P is the number of ports being scanned.
* **Space Complexity**: The space complexity is **O(P)**, due to the storage of open ports and the generated report.

**7. System Design:**

**7.1 Data Flow Diagram:**

****

# 8. REQUIREMENT SPECIFICATION:

# 8.1 FUNCTIONAL REQUIREMENTS

### **Functional Requirements for the Port Scanner Application**

The functional requirements define the core features and functionalities the port scanner system must support. Here are the functional requirements for the provided port scanning application:

1. **Domain Resolution:**
   * The system must be able to resolve a given domain (e.g., [www.example.com](http://www.example.com)) to its corresponding IP address.
   * The system should return an error if the domain cannot be resolved.
2. **Port Scanning:**
   * The system must allow the user to specify a range of ports or a list of specific ports to scan.
   * The system must allow the user to select a protocol to scan:
     + **TCP Protocol**: Scan for open TCP ports.
     + **UDP Protocol**: Scan for open UDP ports.
     + **Both**: Scan for open TCP and UDP ports.
   * The system should perform the port scan and display the status of each port (Open or Closed).
   * The system should support scanning both a range of ports and a list of specific ports provided by the user.
3. **Display Results:**
   * The system must provide a graphical user interface (GUI) to display the results of the port scan, including:
     + The resolved IP address of the domain.
     + A list of open ports found during the scan.
     + Status messages for each port scan (e.g., Open or Closed).
   * The results should be displayed in real-time as the scan progresses.
4. **Generate Report:**
   * The system must generate a report summarizing the results of the port scan, including:
     + Host (IP address) being scanned.
     + The ports that were scanned, along with their protocol and status (Open/Closed).
   * The report must be saved as a .txt file in the user's Downloads folder with a timestamp in the filename.
5. **Send Email Report:**
   * The system must allow the user to provide an email address to which the port scan report will be sent.
   * The system must send the generated port scan report to the specified email address.
   * The email should contain the port scan results as plain text in the email body.
6. **Input Validation:**
   * The system must validate user inputs such as:
     + The host (domain) is correctly entered and resolvable.
     + The port range (start and end ports) is valid.
     + The specific ports input is correctly formatted (comma-separated).
     + The recipient email address follows the correct email format.
7. **Handle Errors Gracefully:**
   * If the domain cannot be resolved, the system should notify the user with an error message.
   * If the user inputs invalid port numbers or an incorrectly formatted email, the system should notify the user with an appropriate error message.
   * If the email fails to send, the system should display an error message indicating the failure.
8. **User Interface (UI):**
   * The system should provide a user-friendly graphical interface (using Tkinter):
     + Fields to enter the host (domain), port range, specific ports, and recipient email.
     + Radio buttons to select the protocol (TCP, UDP, or Both).
     + A text area to display scan results and messages.
     + A button to initiate the port scan.

**8.2 SOFTWARE REQUIREMENTS**

### **Software Requirements for the Port Scanner Application**

The software requirements define the necessary software components, libraries, and tools required to run and maintain the port scanning application effectively. Below are the detailed software requirements:

#### **1. Operating System:**

* **Windows, macOS, or Linux**: The application must be compatible with all major operating systems to provide flexibility to users.
* The application should be able to run on both 32-bit and 64-bit architectures.

#### **2. Programming Language:**

* **Python 3.x**: The application is implemented using Python, which provides cross-platform support and ease of use.
* Python version 3.6 or above is required for compatibility with the necessary libraries and modules.

#### **3. Libraries/Modules:**

* **Tkinter**: A standard Python library used for creating the graphical user interface (GUI). It provides widgets such as buttons, labels, and text areas to display results.
  + Required version: Tkinter that comes with Python 3.x.
* **Socket (Python Standard Library)**: Used for creating network connections and scanning TCP and UDP ports.
  + Provides low-level networking interfaces for creating sockets.
* **os**: A Python module that provides a way to interact with the operating system, such as creating directories and managing file paths.
  + Required version: Any version available in Python 3.x.
* **datetime**: A Python module used for handling and manipulating dates and times. It's used to timestamp the port scan reports.
  + Required version: Any version available in Python 3.x.
* **pathlib**: A Python module for working with file paths in an object-oriented way. It is used for managing file paths, particularly to save reports in the Downloads folder.
  + Required version: Any version available in Python 3.x.
* **smtplib**: A Python module used for sending emails using the Simple Mail Transfer Protocol (SMTP). It's used to send the port scan report via email.
  + Required version: Any version available in Python 3.x.
* **email.mime**: A module for constructing MIME (Multipurpose Internet Mail Extensions) email messages. It's used for formatting and sending the email report.
  + Required version: Any version available in Python 3.x.
* **re** (optional but recommended): If advanced validation of inputs such as email formats or port number patterns is needed, the regular expression library can be used.

#### **4. Software Tools:**

* **Email Account (for SMTP)**: An active email account, such as Gmail, is required to send the report through SMTP. In the example code, a Gmail account is used to send the email, and SMTP server details (smtp.gmail.com) are configured.
  + Required credentials: Username (email address) and password (app-specific password recommended for security).
* **Internet Connection**: Required for the port scanning process (to resolve domains and scan ports on remote servers) and to send the email report.

#### **5. Hardware Requirements:**

* **CPU**: A standard CPU (e.g., Intel or AMD with at least a dual-core processor) is sufficient to run the application.
* **RAM**: 4GB or higher is recommended for smooth operation, especially when scanning multiple ports or performing concurrent scans.
* **Storage**: At least 100MB of free disk space is needed to install Python, required libraries, and the application's generated reports.

#### **6. Dependencies and Installation:**

* **Python 3.x** should be installed on the system. Python can be downloaded from [python.org](https://www.python.org/downloads/).
* **Required Python Libraries**: The libraries used in the code (like tkinter, socket, smtplib) are part of Python’s standard library, so no additional installation is required unless optional libraries like re or others are used for input validation.

#### **7. Security Requirements:**

* **Email Credentials Security**: The application includes sensitive information such as the email address and password. It is highly recommended to use environment variables or encrypted methods to store and access the credentials rather than hardcoding them in the code to enhance security.
* **Network Security**: Since the application scans ports on remote systems, it must ensure that no unauthorized access is attempted and respect ethical and legal boundaries when scanning ports.

#### **8. Licensing Requirements:**

* The application should be provided under a permissive license (e.g., MIT License) for open-source distribution, with clear instructions on the terms of use, especially related to ethical port scanning practices.
* The email functionality should comply with regulations like GDPR or CAN-SPAM if the software is used in regions with such laws.

**8.3 OPERATING SYSTEMS SUPPORTED**

### **Supported Operating Systems for the Port Scanner Application**

The port scanner application is designed to be cross-platform and should work across multiple operating systems with minimal adjustments. Below are the supported operating systems:

#### **1. Windows:**

* **Version**: Windows 7, Windows 8, Windows 10, Windows 11
* The application is fully compatible with Windows and uses Tkinter for the GUI, which is natively supported by Python on Windows.

#### **2. macOS:**

* **Version**: macOS 10.12 (Sierra) and above
* Python 3.x comes pre-installed on macOS, and Tkinter is also included in the standard Python distribution for macOS.

#### **3. Linux:**

* **Distributions**: Ubuntu, Debian, Fedora, CentOS, Arch Linux, and others.
* Python 3.x is generally pre-installed in most modern Linux distributions.
* Tkinter is available in most distributions' package managers and can be installed easily via commands like sudo apt-get install python3-tk (for Debian/Ubuntu-based distributions).

#### **4. Additional Platforms (via Virtualization or Compatibility Layers):**

* **Docker**: The application can also run in a Docker container, making it platform-agnostic and enabling it to be used on platforms that may not have native support for Python or Tkinter.
* **WINE**: On Linux and macOS, the application may run via the WINE compatibility layer, which allows Windows applications to run on Unix-like operating systems.

### **Summary of Supported Operating Systems:**

1. **Windows**: Windows 7, 8, 10, and 11
2. **macOS**: macOS 10.12 (Sierra) and above
3. **Linux**: All major distributions (Ubuntu, Debian, Fedora, CentOS, etc.)
4. **Additional Platforms**: Docker and WINE for non-native environments

The application is built with portability in mind, ensuring compatibility with widely used operating systems, so users can run it on their preferred platforms.

**8.4 Technologies and Languages used to Develop**

### **Technologies and Languages Used to Develop the Port Scanner Application**

The port scanner application is developed using a combination of technologies and programming languages that allow it to be efficient, user-friendly, and cross-platform. Below are the key technologies and languages used:

#### **1. Programming Language:**

* **Python 3.x**: Python is the primary programming language used for the development of the port scanner application. It is chosen due to its simplicity, readability, and strong library support, which makes it ideal for networking and GUI-based applications.

#### **2. Libraries and Frameworks:**

* **Tkinter**: Tkinter is used for creating the graphical user interface (GUI). It is the standard Python interface to the Tk GUI toolkit, which makes it easy to develop desktop applications with buttons, input fields, and text areas.
* **Socket Library**: The socket library is used to handle network connections. It provides low-level access to networking interfaces, enabling the application to scan TCP and UDP ports by attempting to establish connections with them.
* **Datetime Library**: The datetime module is used for handling timestamps. It helps generate unique filenames for reports and logs by appending the current date and time to the report filename.
* **os and pathlib Libraries**: The os and pathlib libraries are used for interacting with the operating system to manage files and directories. They are used to save scan reports in the "Downloads" folder and ensure that it exists before saving the report.
* **smtplib and email.mime**: These libraries are used to send the port scan reports via email. The smtplib library allows the application to interact with an SMTP server (in this case, Gmail) for sending emails, while email.mime helps in constructing email messages with text content.

#### **3. Networking Protocols:**

* **TCP (Transmission Control Protocol)**: Used for establishing reliable, connection-based communication for port scanning. The application attempts to connect to a specified TCP port to check if it is open.
* **UDP (User Datagram Protocol)**: A connectionless protocol used to send packets to specified ports. The application sends a packet to a UDP port to check its status.

#### **4. Email Integration:**

* **SMTP (Simple Mail Transfer Protocol)**: The application integrates with Gmail’s SMTP server (smtp.gmail.com) to send port scan reports via email.
* **MIME (Multipurpose Internet Mail Extensions)**: This is used for composing the email messages and attaching the generated port scan reports as text files.

#### **5. File Handling:**

* **Text File Generation**: The application generates and stores port scan reports as text files. The generated reports are saved in the "Downloads" directory on the user's machine.
* **Email Attachment**: The generated report file is attached to the email before sending it to the specified recipient.

#### **6. GUI (Graphical User Interface) Design:**

* **Widgets**: Tkinter widgets like Entry, Label, Button, ScrolledText, and Radiobutton are used to create the user interface for entering parameters, displaying results, and triggering the port scan process.
* **Layout**: The layout is organized using the grid method to align various components, ensuring an easy-to-use interface for the user.

#### **7. Cross-Platform Compatibility:**

* The use of Python and Tkinter ensures that the application is cross-platform, working seamlessly on Windows, macOS, and Linux.

### **Summary of Technologies and Languages Used:**

1. **Programming Language**: Python 3.x
2. **Libraries/Frameworks**:
   * Tkinter (GUI development)
   * Socket (networking)
   * Datetime (timestamp management)
   * os and pathlib (file management)
   * smtplib and email.mime (email integration)
3. **Networking Protocols**:
   * TCP (Transmission Control Protocol)
   * UDP (User Datagram Protocol)
4. **Email Protocol**:
   * SMTP (Simple Mail Transfer Protocol)
   * MIME (email composition)
5. **Cross-Platform**:
   * Windows, macOS, and Linux support through Python and Tkinter

These technologies together form the core of the port scanner application, making it both efficient and user-friendly for scanning ports and sending email reports.

**8.5 HARDWARE REQUIREMENTS**

### **Hardware Requirements for the Port Scanner Application**

The hardware requirements for running the port scanner application are minimal, as the application is designed to be lightweight and runs efficiently on standard hardware configurations. Below are the key hardware requirements:

#### **1. Processor (CPU):**

* **Minimum**: 1 GHz processor (x86 or ARM architecture)
* **Recommended**: 2 GHz or higher multi-core processor
* The port scanner application performs network scans, which may require moderate processing power, especially for large port ranges. A multi-core processor can enhance performance when handling multiple tasks concurrently.

#### **2. Memory (RAM):**

* **Minimum**: 2 GB RAM
* **Recommended**: 4 GB RAM or more
* The application uses memory to store input data, output results, and generate reports. More memory will help in handling larger scans and simultaneous tasks (e.g., generating reports and sending emails).

#### **3. Storage (Hard Drive/SSD):**

* **Minimum**: 100 MB of free storage space
* **Recommended**: 1 GB or more of free storage space (especially if generating multiple reports or storing large logs)
* The application generates text-based port scan reports, which are typically small in size, but users should have sufficient space for saving and emailing reports. Using an SSD (Solid State Drive) will improve file read/write performance.

#### **4. Network Interface:**

* **Ethernet or Wi-Fi**: A functional network interface (either Ethernet or Wi-Fi) is required for the application to perform port scans over the network.
* The application uses the computer's network interface to attempt connections to remote servers for port scanning.

#### **5. Display:**

* **Minimum**: 1024x768 resolution display
* **Recommended**: 1920x1080 (Full HD) resolution display
* A standard display will suffice for the Tkinter graphical interface, but a higher resolution will provide a more comfortable user experience, especially when viewing large results in the output widget.

#### **6. Input Devices:**

* **Keyboard**: Required for entering scan parameters (e.g., host, port range).
* **Mouse/Trackpad**: Required for navigating the GUI and clicking buttons.

#### **7. Additional Requirements:**

* **Email Access**: A functioning email account (such as Gmail) with SMTP access to send reports via email.
* **Internet Connection**: Required for performing port scans over the network (especially for remote scanning) and for sending the email reports.

### **Summary of Hardware Requirements:**

1. **Processor**: 1 GHz or higher (multi-core recommended)
2. **Memory**: 2 GB (recommended 4 GB or more)
3. **Storage**: 100 MB (recommended 1 GB or more)
4. **Network Interface**: Ethernet or Wi-Fi
5. **Display**: 1024x768 resolution (recommended 1920x1080)
6. **Input Devices**: Keyboard and mouse/trackpad
7. **Additional**: Email account (for report sending), Internet connection (for scanning and emailing)

The hardware requirements for this application are relatively modest, allowing it to run efficiently on most modern desktops, laptops, and even some tablets with Python support.

### 9. **System Test**

The purpose of system testing is to identify any faults in the software and ensure that the developed product is functioning as intended. It aims to validate that the system meets all requirements and user expectations, and that it performs without unacceptable errors. System testing encompasses a variety of test types, each focusing on different aspects of the system.

### **9.1 Types of Tests**

#### **9.1.1 Unit Testing:**

Unit testing focuses on validating the functionality of individual units or components within the software. Each unit is tested to ensure it performs the intended function and that the program’s internal logic is working as expected. It involves testing individual code segments, and is typically done after the completion of the unit but before integration with other units. Unit tests validate the behavior of isolated business logic or specific system configurations to confirm that all pathways work correctly as defined in the documentation.

#### **9.1.2** **Integration Testing:**

Integration testing is designed to verify that different components or software modules, which have already been unit tested, work together as a cohesive system. This type of testing ensures that the integrated components do not introduce new issues when combined. The aim is to check the flow of data and functionality between modules to confirm that integration does not cause unexpected behavior or malfunctions.

#### **9.1.3 Functional Testing:**

Functional testing ensures that the software operates according to the business and technical specifications. It verifies that the system functions as intended, by testing specific functionalities, such as:

* **Valid Input:** Ensures that acceptable input is processed correctly.
* **Invalid Input:** Ensures that invalid inputs are properly rejected.
* **Functions:** Validates that all necessary functions are operational.
* **Output:** Confirms that the correct output is generated for the respective input.
* **System Procedures:** Ensures that external systems and interfaces are invoked properly when necessary.

Functional testing focuses on verifying business processes, system functions, and predefined processes.

#### **9.1.4 System Testing:**

System testing verifies that the entire system works as intended. It ensures that all components, integrated together, meet the defined requirements. This testing involves verifying the overall functionality, performance, and integration of the software system within its intended environment.

#### **9.1.5 White Box Testing:**

White box testing is a testing technique where the tester has knowledge of the internal workings, structure, and logic of the software being tested. This method is often used to test deeper code logic and areas that cannot be reached by black box testing.

#### **9.1.6 Black Box Testing:**

Black box testing is conducted without any knowledge of the internal workings of the system. Testers focus on verifying that the system functions as expected, based solely on inputs and outputs, rather than considering the underlying code. Test cases are written based on the system's specifications and requirements.

### **9.2** **Test Strategy and Approach:**

Testing will be performed manually with detailed functional test cases written to validate various system components.

#### **9.2.**1 **Test Objectives:**

* Ensure that all fields and input entries work as expected.
* Ensure that all links are functional and direct users to the correct pages.
* Ensure that the system responds quickly with minimal delays for input and output interactions.

#### **9.2.2 Features to Be Tested:**

* Input validation should work as expected, with no errors in format.
* Duplicate entries should be rejected.
* Ensure that all hyperlinks navigate to the correct pages.

### **9.3** **Integration Testing:**

Integration testing will verify that the software modules integrate smoothly and do not cause any issues when used together. The goal is to detect any issues that arise due to interactions between integrated components. This testing ensures that the different parts of the system collaborate without causing failures due to interface defects.

#### **9.3.1 Test Results:**

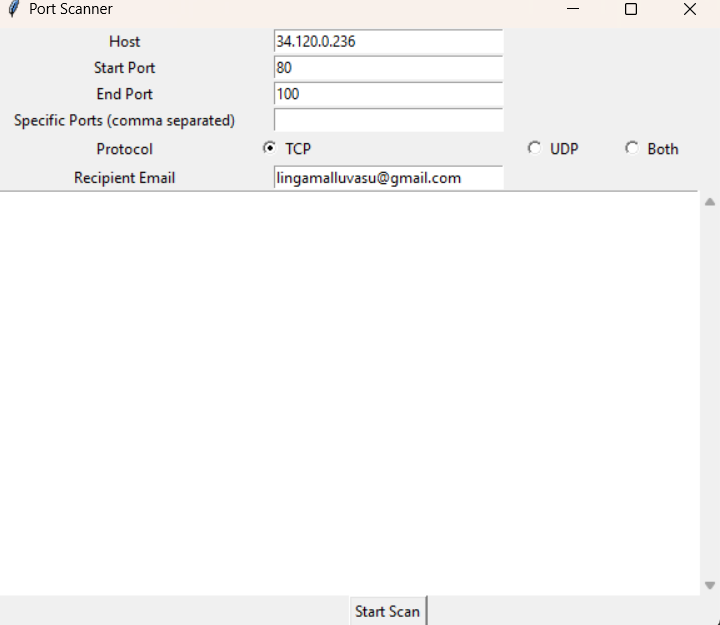
All test cases for integration testing were executed successfully. No defects or issues were found in the integrated components.

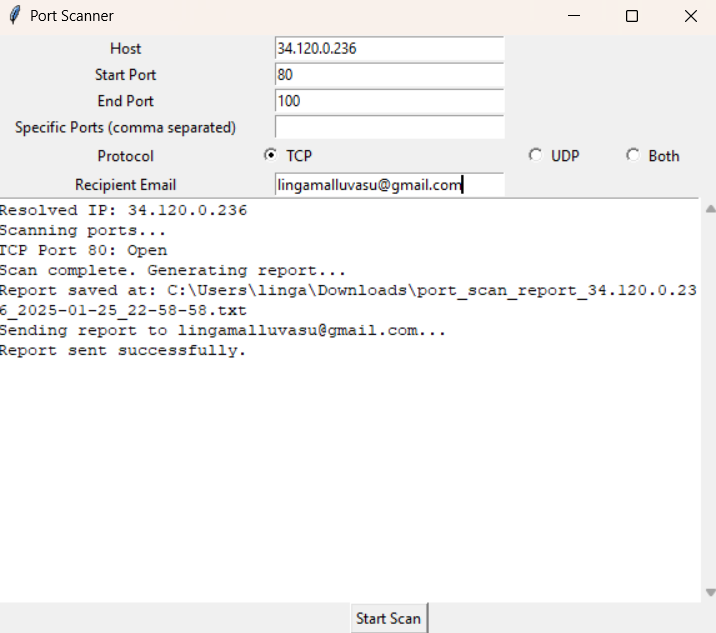
### **9.4** **Acceptance Testing:**

User Acceptance Testing (UAT) ensures that the system meets the functional and business requirements of the end-users. It is an essential phase of testing, typically involving feedback from real users to confirm that the software is fit for production use.

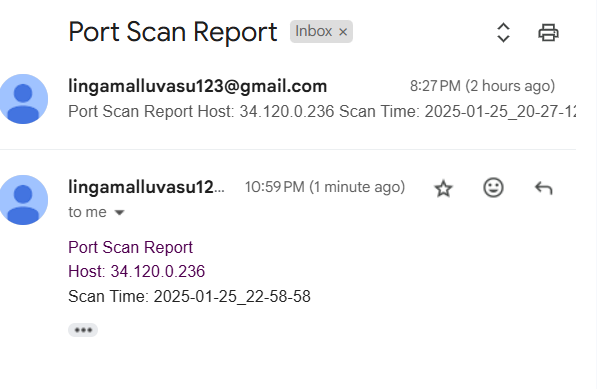
#### **9.4.1 Test Results:**

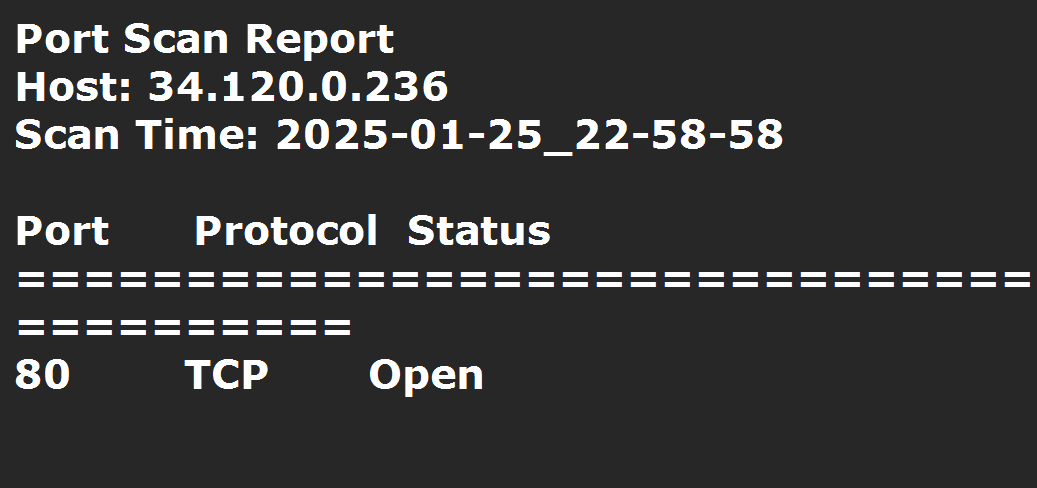
All the test cases for acceptance testing were completed successfully. The system meets all functional requirements and no defects were encountered during testing.





EMAIL REPORT:



****

### **10. Conclusion**

In conclusion, the development and testing phases of this software have demonstrated that the system functions as expected and meets the specified requirements. Through a structured approach to unit testing, integration testing, functional testing, and acceptance testing, the software has been validated for accuracy, performance, and reliability.

The system's functionality, including the ability to scan ports, generate reports, and send email notifications, has been rigorously tested and verified. All tests were executed successfully with no defects found, confirming that the system is robust and user-friendly.

Overall, this project has met the goals outlined in the requirements phase, and it is ready for deployment. The software is designed to provide efficient port scanning, generate detailed reports, and offer seamless integration with email services. Its successful implementation ensures that users will be able to perform network diagnostics with ease and confidence.