FOCUS - Firewall Optimization and Control Utility System

Project Documentation

1 Introduction

The **Firewall Optimization and Control Utility System** is a cybersecurity tool designed to monitor, control, and secure network ports on a system. It actively scans open ports, enforces a whitelist, and utilizes **face recognition-based authentication** to prevent unauthorized modifications. The system ensures that only pre-approved network connections are maintained, enhancing both security and efficiency.

The project is currently implemented as a functional prototype running on a laptop, demonstrating its potential for deployment in enterprise servers.

2 Project Structure

The project is structured into multiple scripts and directories, each playing a specific role in security enforcement and process monitoring.

2.1 Folder Structure

- **Main Directory**
  - **'FOCUS-Main/'** → Contains all necessary scripts and modules for the system.
    - `main.py` → Core script that executes the port monitoring and security functions.
    - `functions.py` → Contains reusable functions for the system.
    - `PortScanner.py` → Scans for all open ports on the system.
    - `PortClosingTest.py` → Tests the system's ability to close unauthorized ports.
    - `PortOpener.py` → Opens ports for testing unauthorized access scenarios.
    - `Whitelist.py` → Manages the whitelist, secured via face recognition.
    - `Whitelisted_Ports.txt` → Stores the list of allowed ports.
  - **Requirements.txt** → Lists dependencies required to run the project.
  - **Photo/** → Stores images related to authentication.
    - `auth/` → Stores a **single** clear image of the authorized user for face recognition.
    - `unauth/` → Stores images of individuals who attempt unauthorized access.

3 Features and Functionalities

3.1 Port Scanner (`PortScanner.py`)

- Scans the system for all currently open ports.
- Displays active connections.
- Provides a clear overview of the system's network activity.

3.2 Port Master (`1st Option - main.py`)

- **Monitors** active ports and cross-checks them against the whitelist.
- **Automatically closes** any unauthorized ports that are not in the whitelist.
- Runs **continuously in the background**, ensuring that unauthorized ports remain closed.
- **Sends alerts** to the system administrator if an unauthorized user attempts to modify the whitelist.

3.3 Whitelist Management (`Whitelist.py`)

- Provides an interface to manage the list of approved ports.
- Secured via **face recognition authentication**, preventing unauthorized modifications.
- Logs unauthorized access attempts, stores their photos in the `unauth/` directory and also sends an email of the face to the admin.

3.4 Port Closer (`PortClosingTest.py`)

- Manually closes a specified port when run.
- Used for debugging and manual security enforcement.

3.5 Port Opener (`PortOpener.py`)

- Used for testing unauthorized access scenarios.
- Opens a specified port to simulate an external program attempting to access the system. Also can be used by the server admin to open needed ports.

4 Testing Scenario for Mini Hackathon

For judges or testers who want to evaluate the project's functionality without prior knowledge, follow these steps:

4.1 Step 1: Install Dependencies

Open a terminal and install all necessary Python modules mentioned in Requirements.txt.

4.2 Step 2: Setup Authentication

- Place **one clear image** of the test user inside `photo/auth/`.
- Any unauthorized individual attempting access will have their image logged in `photo/unauth/` when run Whitelist.py .
- A photo will also be captured and sent to the server admin's email upon any authentication attempt.

4.3 Step 3: Scan Open Ports

Run the port scanner to see all currently open ports:

```
python PortScanner.py
```

The output will list active ports.

4.4 Step 4: Whitelist Ports

Run the whitelist manager:

```
python Whitelist.py
```

- The system will use **face recognition** for authentication.

- If access is granted, it opens Whitelist$_{P}$*orts.txtandaddtherequiredportsto*`Whitelisted_Ports.txt`.**It is**

4.5 Step 5: Test Unauthorized Port Opening

Run `PortOpener.py` to open a random unauthorized port:

```
python PortOpener.py
```

This simulates an unauthorized process trying to establish a connection.

4.6 Step 6: Activate Security System

Start `main.py` to enforce security:

```
python main.py
```

- Choose '1. Port Master' to begin functionality.
- The system will detect and close unauthorized ports automatically.
- Unauthorized attempts to modify the whitelist will be logged and **sent to the admin via email**.

5 Conclusion

This project provides an **automated and secure port management system** with **face recognition authentication**. By enforcing a **strict whitelist**, the system ensures that only approved ports remain open, preventing unauthorized background tasks. The implementation is currently a **working prototype**, with features that demonstrate its potential for real-world deployment in enterprise settings.

For more details and source code, visit the GitHub Repository.

5.1 Presentation Link

For a detailed overview of the system, visit the Prezi presentation: `https://prezi.com/view/H9Voap45IgFbxAH62ifW/`