Suphapon Oi Mahawong
Farah Jaber
Andy Halter

NetScan

Problem Description

The goal of this project is to develop an application that is capable of scanning all devices in a given network, identifying open ports, and gathering service details for each active device. The application will be equipped with two primary modes of operation: a command-line interface (CLI) for terminal-based scanning and a graphical user interface (GUI) for a more user-friendly experience. The program will enable users to quickly gain a comprehensive view of the network structure, including active IP addresses, MAC addresses, vendor details, open ports, and running services.

Requirements To Run Program:

scapy 2.6.1 (Npcap is required to use scapy running program on Windows system. Scapy will run natively on Linux).
mac_vendor_lookup 0.1.12

Program Design

The network discovery program is designed to identify active devices within a specified IP range using the Scapy library in Python. The application primarily employs the ARP (Address Resolution Protocol) to send requests and receive responses from devices within the network, determining their availability and capturing details like IP addresses and MAC addresses.

The program is structured as follows:

- **Input Handling**: The program accepts an IP range (e.g., 192.168.0.0/24) as input. This range determines the scope of the network scan, allowing the user to specify which part of the network to examine.
- **ARP Request Creation**:

    - For each IP within the specified range, the program generates an ARP request packet using the Scapy library.
    - An Ethernet frame with a broadcast destination address (`ff:ff:ff:ff:ff:ff`) is created to ensure the ARP request reaches all devices on the network.
- **Sending and Receiving Packets**:
    - The ARP request packet is broadcasted to the network, and Scapy's `srp` function is used to send and receive packets at the data link layer.
    - The program waits for responses, and any device that responds is assumed to be active. Scapy's `srp` function is configured with a timeout of 2 seconds to allow sufficient time for devices to respond.
- **Concurrent Port Scanning with concurrent.futures**:
    - Concurrent.futures is utilized for scanning all ports and returning which ones are open.
    - For each active IP address, a set of port scan tasks is dispatched using `ThreadPoolExecutor`, where each task checks a specific port. The program records which ports are open and available on each active device.

System Implementation

The networking discovery system is implemented with a focus on efficient IP scanning, port identification, and vendor lookup, all presented in a user-friendly GUI. Key libraries and components integrated into the project include scapy, concurrent.futures, mac_vendor_lookup, and Tkinter, each serving specific roles in the system's functionality.

1. **Scapy:**
    - The Scapy library is responsible for generating and sending ARP requests across the specified IP range to discover active devices.
    - Scapy's capability for handling network packets at the data link layer enables the retrieval of IP and MAC addresses of devices within the network.
    - The ARP responses received from devices provide foundational data for the network scan, ensuring that each active device within the IP range is identified and logged.
2. **Concurrent.futures:**
    - The `concurrent.futures` library enables efficient, concurrent scanning of ports across discovered devices**.**

- By utilizing thread pools, the system performs simultaneous port scans, drastically reducing the time required to identify which ports are open on each active device.
- This parallelization enhances the performance of the scan, making the system capable of quickly handling large networks or multiple devices.

3. **mac_vendor_lookup:**
   - To provide additional information about discovered devices, the `mac_vendor_lookup` library is used to determine the manufacturer of each MAC address.
   - This feature allows users to recognize the type or brand of each device, providing context for each entry in the scan results.
4. **Tkinter:**
   - The Tkinter library is used to create a simple graphical user interface (GUI), enabling users to interact with the application without command-line operations.
   - The GUI displays scan results in a clear and organized format, presenting each device's IP, MAC address, vendor, and open ports in a user-friendly manner.
5. **Logging and Performance Metrics:**
   - The application logs each scan run's duration and performance metrics to ensure consistent monitoring and troubleshooting capabilities.
   - Execution times are recorded and averaged, allowing users to track the speed and efficiency of network scans.
6. **System Requirement:**

   The following dependencies and libraries are required to run the system:
   1. scapy: For ARP scanning and package manipulation.
   2. socket: For scanning port.
   3. mac_vendor_lookup: For looking up MAC address vendors.
   4. tkinter: For graphical user interface.
   5. psutil: For managing processes and measuring execution time (use in test_netscan.py)

   The required packages are specified in the `requirements.txt` file:

   psutil==6.1.0

Development Process

1. **Initial Requirements and Design**

The application began with the need to scan a network for devices, identify them, and retrieve information about open ports and vendor details. A network discovery tool was designed to handle multiple devices and ports concurrently. The decision was made to use ARP for network discovery and a socket-based approach for port scanning. Vendor information was obtained via MAC address prefixes.

2. **Development of Core Components:**
    - **Network Discovery:** Using the ARP protocol, devices in the specified CIDR block are discovered by broadcasting ARP requests.
    - **Port Scanning:** A function was written to scan ports 1- 1024 for each discovered device. The function checks if the ports are open and identifies the services running on them based on well-known port numbers.
    - **Vendor Lookup:** A vendor lookup system was implemented to identify manufacturers using MAC address prefixes from a vendor list stored in a text file. The mac_vendor_lookup library was employed to enhance the accuracy and ease of lookup.
    - **Result Reporting:** The results were initially printed to the console. Later the system was enhanced to generate an HTML file for storing and sharing scan results, and a Tkinter-based GUI was added for a more user-friendly experience.
3. **Testing and Validation:** The network scan was tested on Andy's home network through the virtual machine. This allowed cross-referencing with devices known to be connected to the network.
4. **Performance Measurement:** The network scan performance was measured by running the system several times and recording the scan duration for each run. The average time for scanning the 192.168.0.0/24 network was calculated to be **0.531813 seconds** based on the five runs.

Result Log for Networking: 192.168.0.0/24:

Run 1 took 0.532986 seconds
Run 2 took 0.530529 seconds
Run 3 took 0.525223 seconds
Run 4 took 0.532764 seconds
Run 5 took 0.537565 seconds

Measure | Average Time (s)

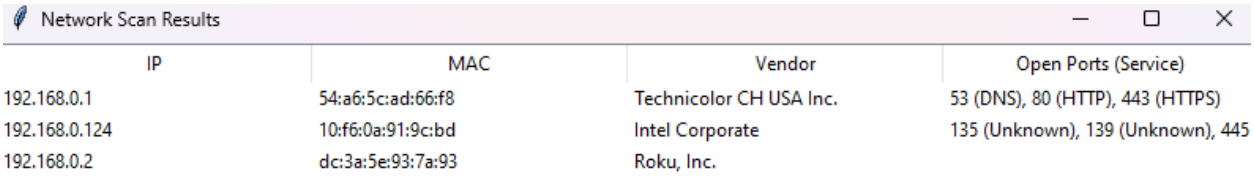
---

netscan | 0.531813

5. **Challenge Faced:**
    - **Vendor Lookup Accuracy and Display:** One of the primary challenges encountered during development of this network discovery application was ensuring accurate vendor information for each discovered MAC address. Initially, the program struggled to retrieve vendor data, which impacted the quality of information displayed to users.
    - **Solution and Resolution:** To overcome this challenge, the mac_vendor_lookup was integrated into the program. This library uses a reliable database of MAC addresses mapped to vendor names, allowing the program to cross-reference MAC addresses quickly and retrieve accurate vendor information. By installing and configuring mac_vendor_lookup via pip, we were able to streamline the vendor lookup process, ensuring that each MAC address returned an accurate vendor name.

Testing:

NetScan Result:



Results in HTML:

Network Scan Results

IP	MAC	Vendor	Open Ports (Service)
192.168.0.1	54:a6:5c:ad:66:f8	Technicolor CH USA Inc.	53 (DNS), 80 (HTTP), 443 (HTTPS)
192.168.0.2	dc:3a:5e:93:7a:93	Roku, Inc.	
192.168.0.124	10:f6:0a:91:9c:bd	Intel Corporate	135 (Unknown), 139 (Unknown), 445 (SMB)

NetScan Speed Performance:

```
 C:\Users\abhjc\OneDrive\Documents\GitHub\netScan\main.py

1 92.168.0.0/24


--------------------------------------------------------------------------------
 Run 1 took 0.532986 seconds
 Run 2 took 0.530529 seconds
 Run 3 took 0.525223 seconds
 Run 4 took 0.532764 seconds
 Run 5 took 0.537565 seconds
--------------------------------------------------------------------------------
 Measure              | Average Time (s)
--------------------------------------------------------------------------------
 netscan              | 0.531813
```

Resources Used:

Concurrent.futures documentation:
https://docs.python.org/3/library/concurrent.futures.html
Scapy documentation:
https://scapy.readthedocs.io/en/latest/
Tkinter documentation:
https://docs.python.org/3/library/tkinter.html
Mac-Vendor-Documentation lookup: https://pypi.org/project/mac-vendor-lookup/
W3 Schools: https://www.w3schools.com/
Stack Overflow: https://stackoverflow.blog/
GeeksforGeeks: https://www.geeksforgeeks.org/