Network Scanning Using Python and Threading

This project focuses on building an efficient network scanner with Python to discover active devices on a Local Area Network (LAN). It uses multithreading to send Address Resolution Protocol (ARP) requests concurrently, allowing it to map the local network and identify connected devices (IP, MAC, and hostname) very quickly.

Objective

To develop a Python script that automatically detects active devices on a local network by retrieving their IP addresses, MAC addresses, and hostnames, using multithreading to significantly accelerate the scanning process.

Introduction

Network scanning is the process of identifying active hosts, ports, and services on a network. It is a fundamental step in network reconnaissance, allowing administrators and security professionals to map a network's topology and identify connected devices. In this project, Python's scapy, threading, queue, ipaddress, and socket modules are used to send concurrent ARP requests and report the results in a clear, tabular format.

Working Process

1. The script prompts the user to enter a network range in CIDR (Classless Inter-Domain Routing) notation (e.g., 192.168.1.0/24).
2. The ipaddress library parses this input and generates a list of all scannable host IP addresses within that range.
3. All IP addresses are added to a thread-safe Queue to be processed.
4. A pool of worker threads is created (e.g., 100 threads).
5. Each thread continuously pulls an IP address from the queue and executes the scan() function.
6. The scan() function uses scapy to craft and broadcast an ARP request for its assigned IP address.
7. If a device on the network replies to the ARP request, its IP address and unique MAC address are captured.
8. The script then attempts to resolve the device's hostname using the socket library. If no hostname is found, it defaults to "Unknown".
9. The complete result (IP, MAC, Hostname) is added to a separate, thread-safe result_queue.
10. After all threads have finished and the scan_queue is empty, the main thread collects all results from the result_queue and displays them in a formatted table.

Tools and Libraries

Tool	Purpose
Python 3	The core programming language.
scapy	A library to craft, send, and receive network packets (used for ARP requests).
threading	A standard library module to manage concurrent scan threads.
queue	A standard library module to safely share IP addresses and results between threads.
ipaddress	A standard library module to parse CIDR notation and generate IP ranges.
socket	A standard library module used to resolve hostnames from IP addresses.
time	A standard library module to measure the total scan execution time.

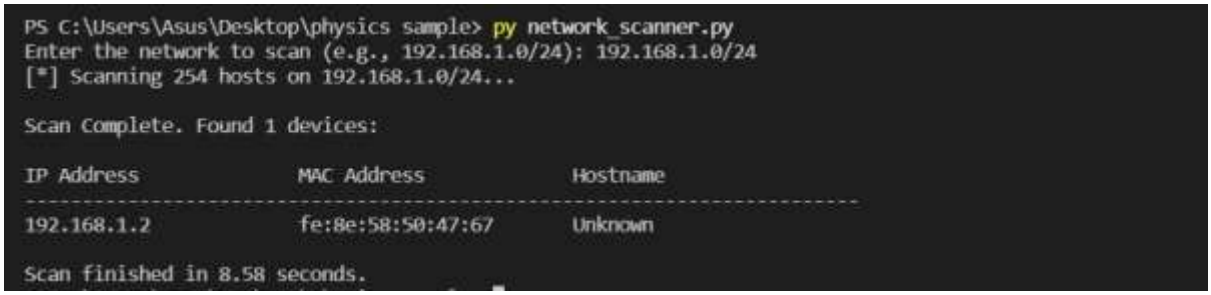
Code Explanation

- **scan(ip)**: Sends a single ARP request to the given IP. If a reply is received, it captures the MAC address, attempts to resolve the hostname, and places the complete result into the result_queue.
- **print_results(results)**: Takes the final list of discovered devices and prints them in a clean, formatted table.
- **worker(q)**: This is the target function for each thread. It continuously takes an IP from the scan_queue (named q) and calls scan() on it until the queue is empty.
- **main()**: Prompts the user for the CIDR input, uses ipaddress to create the list of hosts, populates the scan_queue, creates and starts all the worker threads, waits for them to complete (t.join()), and finally calls print_results() to show the findings

Sample Input & Output

Input (Command Line):

Output (Success):



Results

The script successfully parallelized the network scan. By using 100 threads, it was able to scan all 254 IPs in a /24 network in just a few seconds. It reliably identified all connected devices, including the

router, other computers, and mobile phones, retrieving their IP addresses, MAC addresses, and (where available) their network hostnames.

Advantages

- **Extremely Fast:** Multithreading reduces the scan time from several minutes (for a sequential scan) to mere seconds.
- **Rich Data:** Provides Layer 2 (MAC) and Layer 3 (IP) addresses, along with hostname information, giving a comprehensive view of each device.
- **Reliable on LAN:** ARP is a very reliable method for discovering hosts on a local network, as most devices cannot block ARP replies without losing network connectivity.
- **Low Resource:** The script is lightweight and does not consume significant CPU or memory.

Limitations

- **Requires Admin/Root Privileges:** scapy needs elevated permissions (Administrator on Windows, sudo on Linux/macOS) to send raw network packets.
- **Local Network Only:** ARP packets are not routable, meaning this scanner can *only* discover devices on the same local subnet as the computer running the script.
- **Hostname Resolution:** Hostname lookups can sometimes fail or be slow if the network's reverse DNS is not properly configured.
- **Network Stability:** Running with too many threads (e.g., 1000+) could potentially add strain to low-power network hardware like a home router.

Conclusion

This project successfully demonstrates the creation of a powerful and practical network scanner using Python. It is a perfect example of how multithreading can be applied to network-bound tasks to achieve a dramatic increase in performance. The tool serves as a valuable utility for network mapping, inventory, and reconnaissance within an educational cybersecurity context.