Distributed Network and TCP Port Scanner

CSE 509 Final Project - Report

Spring '19

Github Link:

https://github.com/pavan-peela/Distributed-Network-and-TCP-Port-Scanner.git

Pavan Kumar Peela (112045988)

Introduction

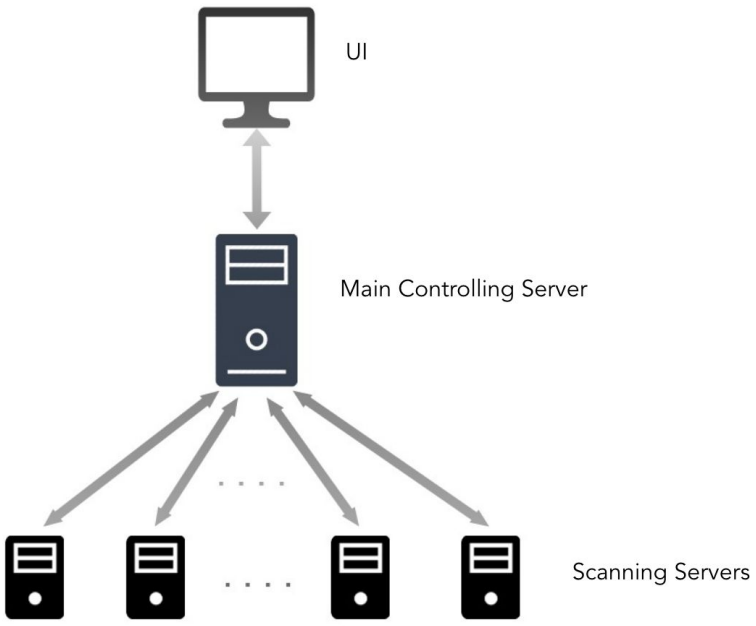
The primary goal of the project is to develop a Distributed Network and TCP Port Scanner, that takes the scan jobs as input from the user and carries out the operation in a distributed fashion with the help of scanning nodes.

Network port scanning is a procedure for identifying active hosts in a network, either for network security assessment, system maintenance or for the purpose of performing attacks by hackers.

The following are advantages of using a distributed network scanning;

- Speed: With multiple scanning nodes, the jobs can be executed more efficiently as compared to a single server.
- No single point of failure: As long as even a single scanning node is alive, the scanning operation can still be carried out.
- Scalability: Scanning nodes can be added/removed as per load on the server.

Architecture



The distributed network scanner consists of three components:

Main Controlling Server

- The primary task of the server is to receive jobs from the web UI which is then distributed and sent to all the registered scanning nodes.
- Once the server receives the results from the scanning nodes it then consolidates the results and sends it back to the front end (UI).
- The server uses a timeout of a certain period for the scanning node to respond with the result. If this timeout is reached, the scan is considered to have failed.
- The server is also responsible to maintain logs of the scans requested, so when it receives all the results and consolidates it, before sending to the front-end it adds the records to past_scans.csv - log files which stores the fifteen past scan requests and their results.

Scanning Node

- The task of a scanning node is to perform a scan job provided to it by the main server.
- On initialization, the scanning node registers itself with the main controlling server to receive scanning jobs using the IP and the port number on which the main controlling server accepts connections.
- Once registered, the scanning node waits for a job and if received, performs it and sends the response back to the main controlling server.
- The scanning nodes use Scapy python module for performing the different types of port scans.

- The following are the different operations the scanning nodes perform;
    - IP-Alive :

      We use 'The Internet Control Message Protocol' (ICMP) to ping the target IP and verify if it is 'Alive' or not.
    - IP Subnet-Alive :

      Uses ICMP to ping each of the target IP in the IP subnet.
    - Port Scanning :

      There are three modes of port scans performed by scanning nodes;

    ➢ Normal Port Scan:

        - In this type of port scan, the scanning node sends a TCP packet to the target port.
        - If nothing is received and timeout is reached, it concludes that the port is closed but if an SYN-ACK packet is received, then the banner is grabbed.

    ➢ Stealth Scan:

        - In this stealth scan, the scanning node sends a TCP packet to the target port.
        - If nothing is received and timeout is reached it concludes that the port is closed.
        - If an SYN-ACK packet is received, the port is considered open and an RST is sent to tear down the connection. Else the port is concluded to be closed.

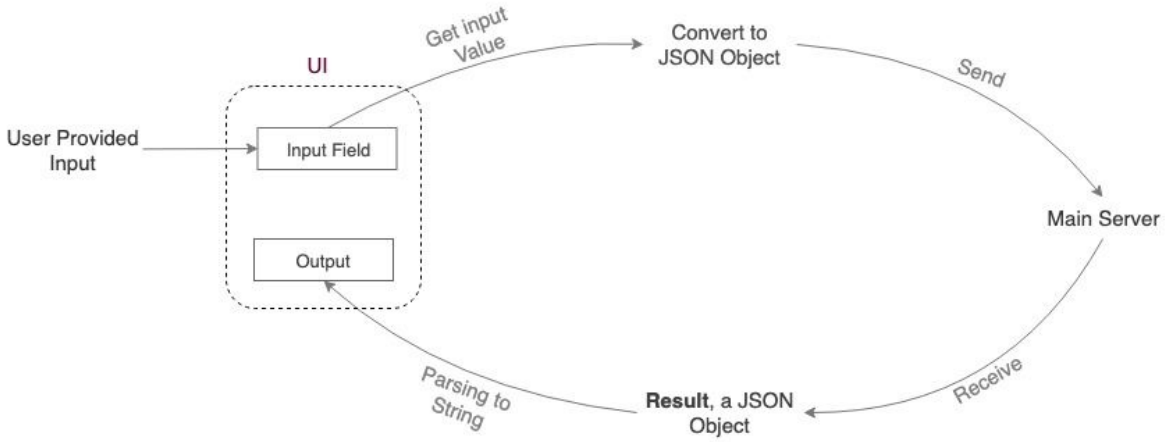
    ➢ Fin-Scan :

        - In this stealth scan, the scanning node sends a TCP packet to the target port.

- If no response is received and timeout is reached, it concludes that the port is open. Otherwise, the target responds with an RST packet.
- If an RST packet is received, the port is concluded to be closed.

Web UI

- A minimalistic UI design is adopted and is designed using bootstrap.
- The interface takes the required input and displays the desired results.
- Integration to the main controlling server (Flask App) is done using ajax call (written specific function) and this will route to the respective function for performing the task.



The following are the various implemented design use cases :

- Dynamic input fields :
  - UI changes and updates according to the operation selected.
- Randomize option :
  - A checkbox provided in the UI allows the user to specify if the port scanning should be performed in a sequential/random order.

- Input validation :
    - The input text boxes for IP/Block and Port range cannot be empty and when the submit button is clicked with empty input, the UI blocks the scan job and prompts the user to provide valid input.

- Blocking multiple scan requests :
    - The server only accepts one scanning job at a point of time and so the UI blocks the user from submitting a new scan job when a scan is already in progress and displays a prompt “scan in progress”.

Implementation details

- We have implemented the main server and scanning nodes in Python and hence use scapy python module to perform the port scanning operations. Scapy framework provides an easy interface and is very efficient. It also supports most common port scanning operations provided by popular network scanning tools like nmap.
- In case a scanning node fails, the scan result for the corresponding ports is updated as ‘Not Scanned’ but for those ports that were scanned successfully, the results are still updated in the UI.
- In case of Normal TCP Scanning, for grabbing the banner, the scanning node reads 1024 bytes of the response and sends it to the main server.

Setting Up and Testing

- Required Packages:

Component	Required Packages

Server	• flask_socketio • pandas • flask • gevent • scapy • html
Scanning Node	• Scapy • Python-socketio

- Testing Instructions:
  - To run the main server:
    - Switch to the main_server/ directory in the project repo
    - Run the app using the following command:

      ```
      $ python app.py
      ```

      This will host the app in localhost and runs on port 8899
  - To run scanning_node:
    - Switch to the scanning_node/ directory in the project repo
    - Run the scanning node using the following command when the main server is hosted on localhost:

      ```
      # sudo python scan_node.py 127.0.0.1 8899
      ```

    - The same command can be used to instantiate multiple instances of scanning nodes in different terminal instances.
    - The client gets registered if the main server is running and awaits scanning jobs from the server.
  - Visit **http://localhost:8899** in the browser to perform a scan.

References

I. Scapy documentation:
https://github.com/secdev/scapy

II. Port scanning using scapy
https://resources.infosecinstitute.com/port-scanning-using-scapy/#gref

III. Flask socketio documentation
https://flask-socketio.readthedocs.io/en/latest/

IV. Python socketio documentation
https://python-socketio.readthedocs.io/en/latest/

V. Socket-io connection example
https://blog.miguelgrinberg.com/post/easy-websockets-with-flask-and-gevent/page/12

VI. Bootstrap template
https://startbootstrap.com/templates/small-business/