CSE 509 System Security

Project Report - Spring 2019

Distributed Network and TCP Port Scanner with Web UI

Priyanka Sangtani 112026558
Akanksha Mahajan 112074564
Gourav Dhama 112046525

1 Overview

Network and Port Scanning is an integral part of Threat and Vulnerability Analysis to perform vulnerability scanning, penetration testing etc. to reduce threats, risk and exposure. This usually needs to be performed across tens of thousands of devices with thousands of ports. Every penetration test starts out with a network scan to see what devices are on the network. Initial scanning is usually done against thousands of devices and can take several days, so the scanner must scale.

In order to address this, we built a distributed network and port scanner. The Network scanner checks whether the IP address is alive or not. The Port scanner then scans the reachable IP address for the open ports (all or a custom list). This scanner is built on a distributed architecture.

- The Master node consists of a web front-end that accepts the request from user through a form and then schedules those scans and stores the scan results in a database. The part is scan agents that send request for any scan jobs available from Master and performs the port/network scanning.
- The scan agents subscribe to the Master using REST API when they are ready to accept scan jobs. The scan agents then return the results to the Master, where they are displayed back to the user. Scan agents perform following tasks:
  - Check whether an IP address is alive
  - Find which IP addresses are alive in a block of IP addresses
  - Port scan an IP address for a given set of ports using any of the following scanning modes:
    - Normal Port Scanning (full TCP connect to the remote IP address)
    - TCP SYN Scanning (only send the initial SYN Packet and then send RST when client responds with SYN—ACK)
    - TCP FIN Scanning
  - Any of the above host/port scanning methods must also be able to be done sequentially or in random order (e.g. instead of first scanning port 1, then port 2, then port 3, etc., randomize the order of ports, or order of IP addresses)It checks whether an IP address is alive or not.

Github link for this project:
https://github.com/PriyankaSangtani/System-Security-Distributed-Network-and-Port-Scanner

2 WorkFlow

The Web UI running on the Master node is built using **Django Framework** which follows a Model-View-Template pattern. The User Interface is built using HTML with embedded python code. The styling is done using **Bootstrap library**. The database being used is SQLite. It is used as it is closely integrated with the Django Web Framework.
The communication between the Master node and the Scan Agents is achieved using REST API.

- The form submitted by the user through the Web UI via POST request.
- The Scan Agent always initiates the request through GET with the Master node. The Master does not initiate the request. This simplified our mechanism of checking which scan agents are alive. Initially, the scan agent sends a GET request to the Master node with a token and it's IP address. This indicates to the Master that this particular IP address is available to accept scan jobs. So the Master sends back a response to this IP address with the Destination IP address (to be scanned), ports to be scanned, and the type of scan (Full TCP Connect Scan, TCP SYN Scan, TCP FIN Scan) to be performed.

- The Scan Agents performs the scan and returns the results back to the Master again via GET request. This concludes one session.

3 Architecture

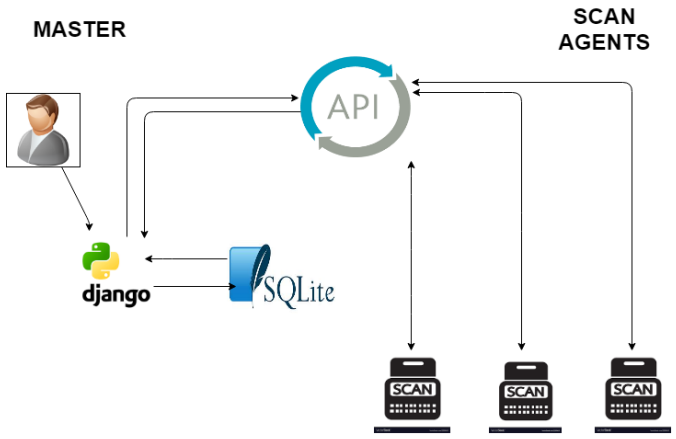

Figure 1: Architecture Diagram

3.1 Master Node flow

Broadly, the flow for the Master node is displayed in **Figure 2**.

All the incoming jobs are stored in *Table JobEntry*. These are ordered by the time of creation (or the time specified by the user). When a new GET request comes from a Scan Agent, the following steps takes place:

- See if the Agent is contacting the Master for the first time. If yes, a new entry is created, else, the "token" is validated.
  - Each Agent has a unique token, and it must include it in the request so that master can verify the authenticity. If someone steals the token, and tries to ping from a different IP address, the request will be rejected. A single IP address can have multiple tokens but a token cannot be associated with multiple IP addresses. It's a very simple security check but it guarantees basic level of security.
  - If the Agent is already busy, the request is dropped and the Agent is informed.
  - If the Agent is free, the next available job is assigned.
- How to assign job?
  - Each scan agent can be given only certain number of ports to scan at a time (let's say K) which is configurable. This allows the scan of large number of ports to run parallel in a distributed fashion.
  - K or remaining ports in a job (which one is lower), are assigned to a scan agent randomly and passed to the agent as a comma-separated string between clients.
  - All the hosts within a given subnet (entered by the user via web form) are generated on the Master node and considered as individual scan requests (one request per IP address) and assigned to the Scan Agents as separate jobs. This also ensures that we scan across all the Scan Agents in parallel.
- Now Master node puts the job status as In Progress and waits for the scan agents to deliver the results.
  - If all agents (related to particular job) deliver results, master node consolidates the responses at runtime, puts the status as DONE and also displays the results to the user through the UI.
  - If any one of the agent results with an Error, master treats the whole job as failed, puts the status as ERROR and puts the error in user UI.

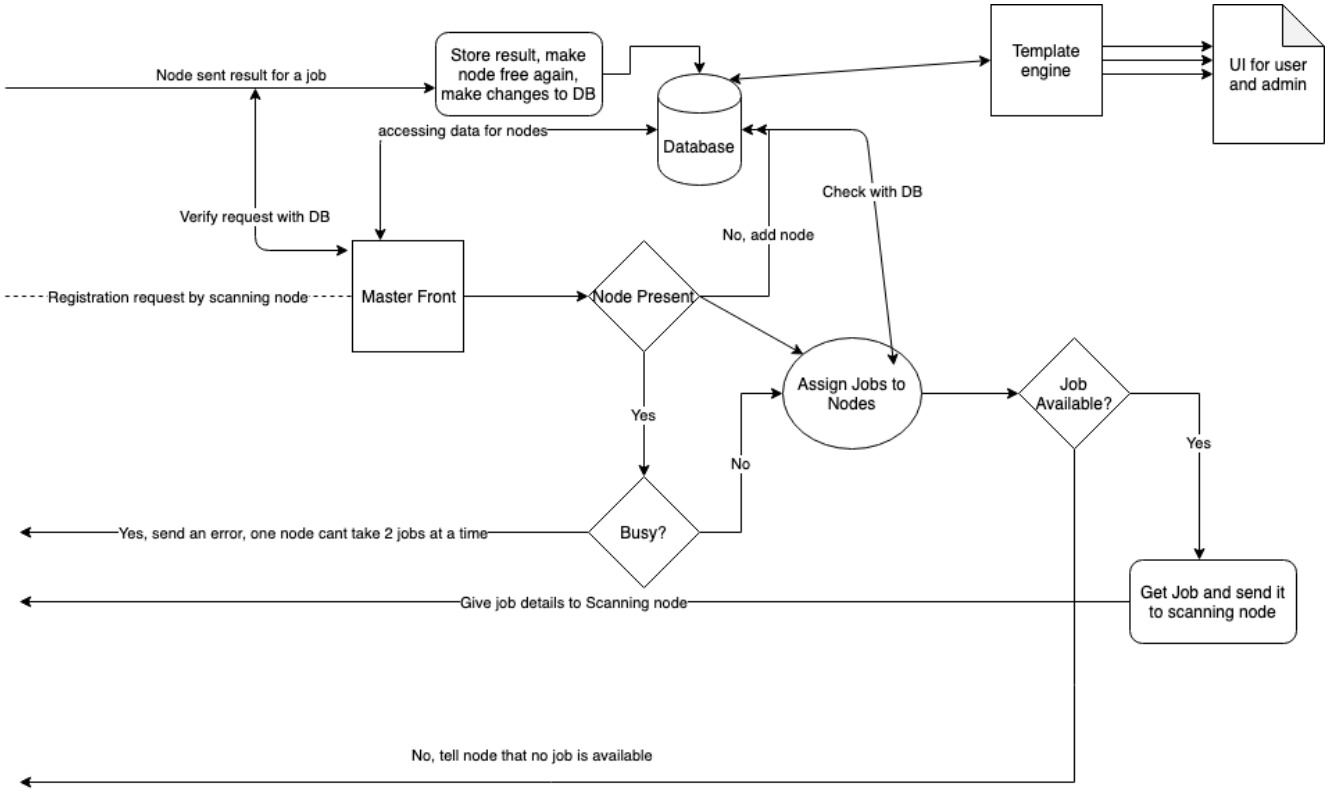

Figure 2: Master Internal Architecture

3.2 Scan Agents

3.2.1 Network Scanning

Scan Agents perform network scanning by using ping command (*implemented via the Python subprocess module*)to check if the IP address passed to it through Master is alive or not. It then sends the result to Master Node through GET request.

3.2.2 Port Scanning

Based on the type of TCP-scan passed by user, Scan Agents perform the scanning of the ports using **Scapy** Python module.

Types of TCP-scan supported are:

- **Full TCP Connect**: This scan is based on the full TCP three-way handshake between the client and the server. A client trying to connect to a server initializes the connection by sending a TCP packet with the SYN flag set and the port to which it wants to connect. If the port is open on the server and the port is accepting connections, it responds with a TCP packet with the SYN and ACK flags set. The connection is established by the client by sending an acknowledgement ACK and RST flag in the final handshake. RST flag is sent to close the connection. If the server responds with a RST instead of a SYN-ACK, then it indicates that the particular port is closed on the server.

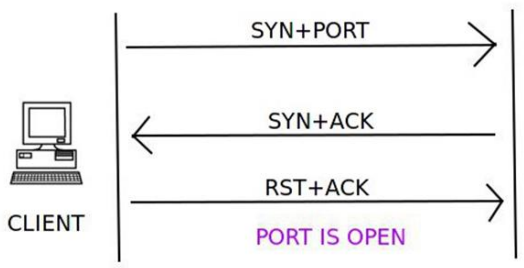

Figure 3: Full TCP connect

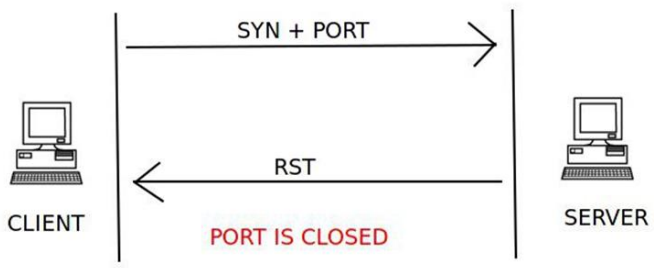

Figure 4: Full TCP Connect

- **TCP SYN**: This technique is similar to the TCP connect scan in terms of the handshake process. The client sends a TCP packet with the SYN flag set and the port number to connect to. If the port is open, the server responds with the SYN and ACK flags inside a TCP packet. However, the key difference between this type of scan and Full TCP connect scan is that the client sends a RST flag in a TCP packet and not RST+ACK. This technique is used to avoid port scanning detection by firewalls. The server responds with an RST flag set inside a TCP packet to indicate that the port is closed on the server.

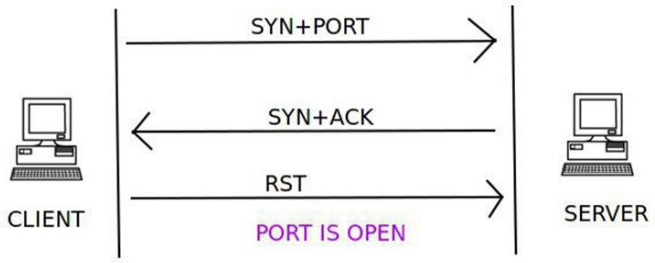

Figure 5: TCP SYN Scan

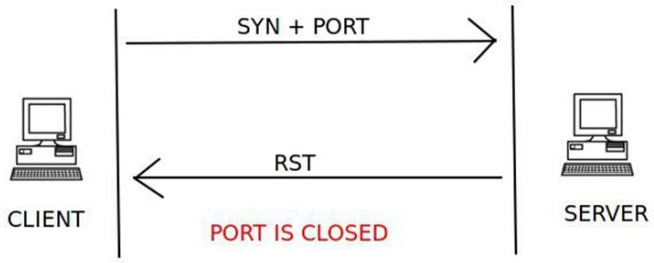

Figure 6: TCP SYN Scan

- **TCP FIN**: The TCP FIN scan utilizes the FIN flag inside the TCP packet, along with the port number to connect to on the server. If there is no response from the server, then the port is open as shown in **Figure 7**. If the server responds with an RST flag set in the TCP packet for the FIN scan request packet, then the port is closed on the server. An ICMP packet with ICMP type 3 and code 1, 2, 3, 9, 10, or 13 in response to the FIN scan packet from the client means that the port is filtered and the port state cannot be found.

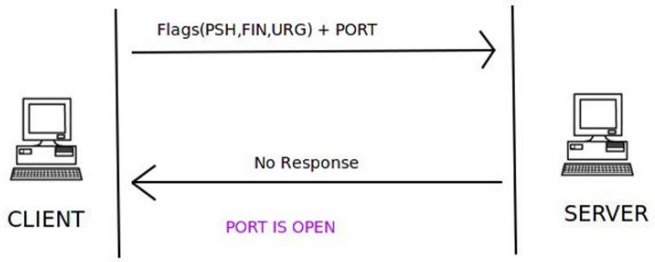

Figure 7: TCP FIN Scan

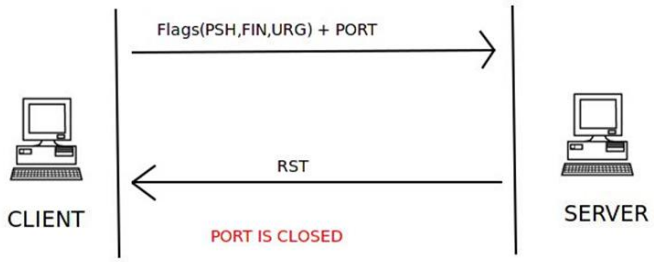

Figure 8: TCP FIN Scan

4 Implementation Details and Testing

4.1 Installation process

Master Node Setup

- Project runs on Python2.7
- Install Django v1.7.11
  $*pip install django*
- $*cd myproject*
- $*python manage.py migrate*
- If there is an error, delete sqlite3.db file and run $*python manage.py migrate* again
- $*python manage.py runserver*
  The server runs on port 8000 by default, can be changed if required.
- Install python ipaddress module $*pip install ipaddress*

Scan Agent Setup

- Install scapy $*pip install scapy*
- Install Requests module $*pip install python-requests*

4.2 Brief Explanation of How Django project works (snippets in the bottom):

- Project is divided into multiple apps (scanapi and formsetapp in our case). Each of them are reachable with the help of URL patterns which are stored in main project and respective apps urls.py folder.
- Each URL pattern is associated with a function (or handler) defined in the views.py of respective app.
- This handler does all the processing work and passes on the result to the scanning node or to a HTML page, or both depending upon the circumstances.
- All database tables associated with a particular app are defined in models.py file. (Note: Different apps can access each others models by default).

4.3 Why REST API?

- REST is easier to implement and test than other methods of communication like SSH, SOAP, etc.
- Though SSH supports encryption, but we are sending REST over HTTPS which takes care of the security. Still, if we see bigger picture, its unsafe to send parameters over URL as URLs can be stored in server log,etc.
- Python and django have very good REST implementation with database, this is a trade-off but we preferred REST over others for the project.

4.4 Why Django ?

- Python has vast support for network related libraries.
- As Django is based upon python, it was easy to just put the code with django project and let it run naturally.
- Django is a full stack framework, unlike MEAN, LAMP,etc. which has several level of implementations. (Ruby Rails was also a good option but python still won on support)

4.5 User Interface

Brief Explanation of User Interface:

- Home page has buttons to all basic functionality like Add a job, Retrieve data related to job entry (admin usage), or current state of job for a user. It also has a link to see data of all scanning nodes (admin usage).
- To add an entry user is presented with a form, which has many fields but not limited to subnet, IP and port range.

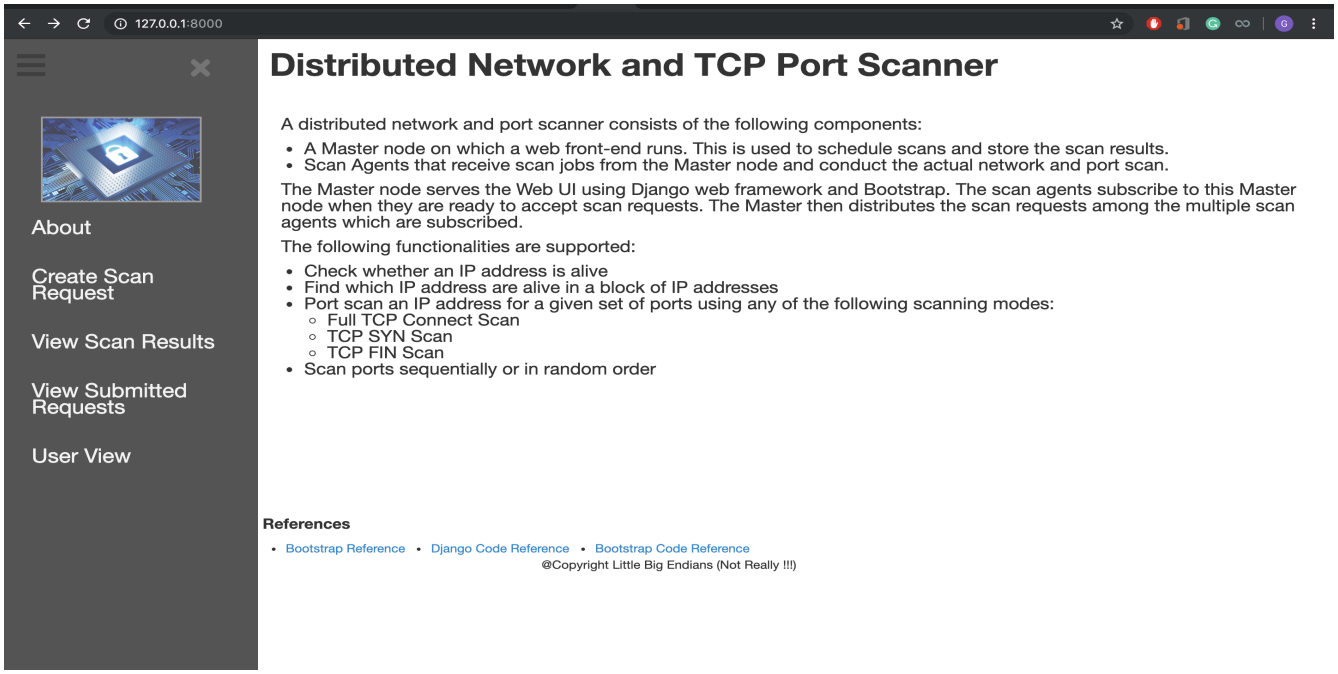

Figure 9: Home Page

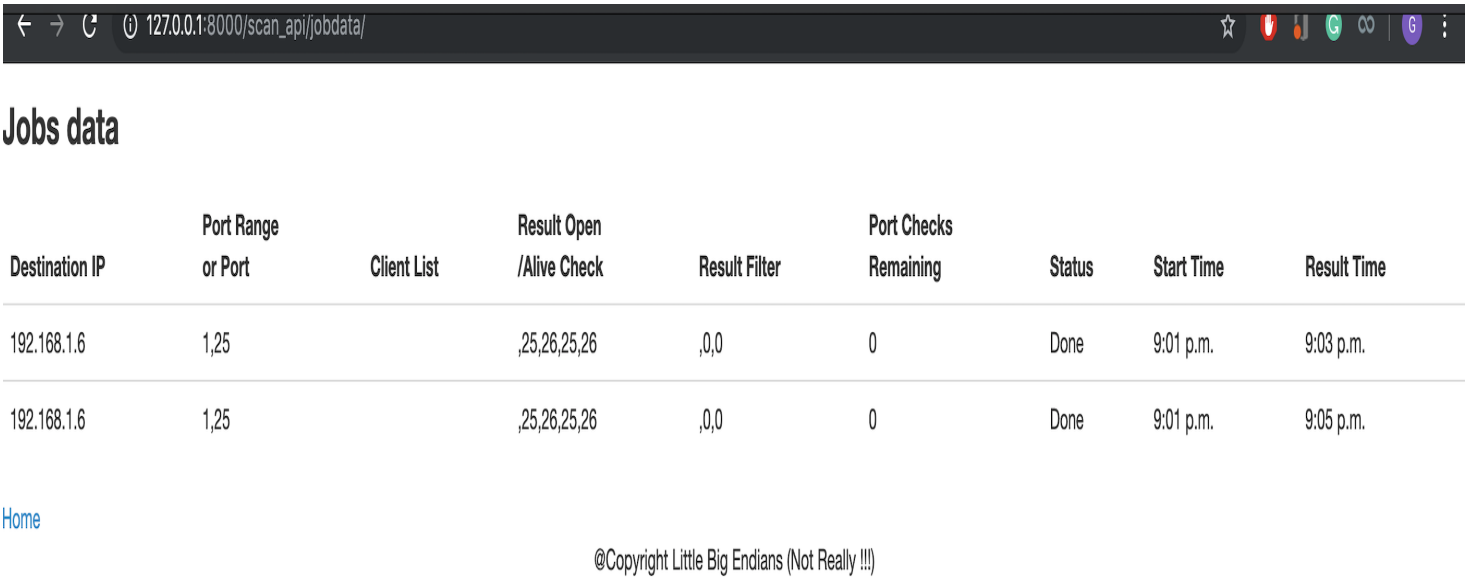

Figure 10: Job Data/ Status

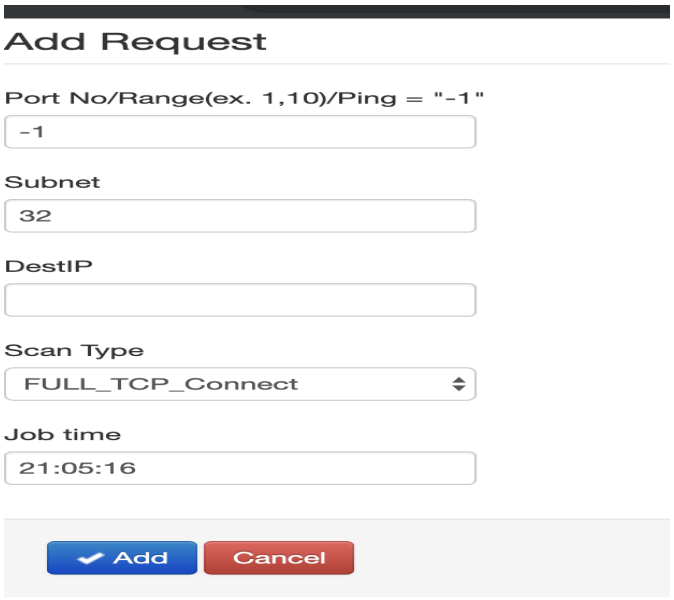

Figure 11: Request ADD form

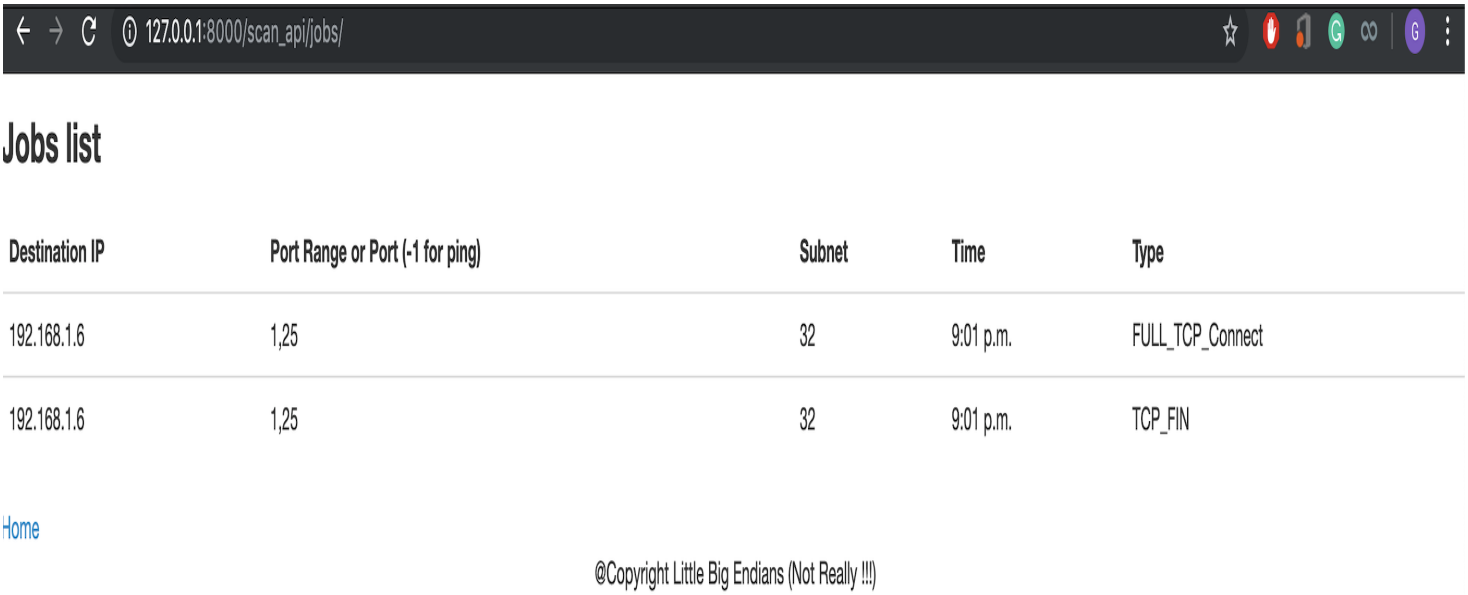

Figure 12: Registered Jobs

4.6 Testing Framework

- Postman Desktop client was used to test the REST API
- Multiple Virtual Machines were used to simulate the Master Node and Scan Agents.
- The setup consists of 1 Master node (running Ubuntu OS with Django), 3 Scan Agents (having IP addresses across different networks), 3 test nodes with IP addresses across different networks having custom ports open and 1 Ubuntu VM simulated as a router.
- The VMs were provisioned in an automated manner through Vagrant.
- The custom ports were opened by using sockets.

5 Distribution of Project Work

The project was done through very close collaboration between all the team members. Specifically, Priyanka and Akanksha handled the client-side functionality of performing the network and port scans. Priyanka implemented the complete test setup and web UI design. Gourav worked on setting up the Web framework and implementing the server-side functionality. Finally, all of us sat together and integrated our functionality to make it work end-to-end.

References

[1] https://resources.infosecinstitute.com/port-scanning-using-scapy/gref

[2] https://getbootstrap.com/2.3.2/

[3] https://docs.djangoproject.com

[4] https://github.com/morelab/django-bootstrap

[5] https://deparkes.co.uk/2017/10/27/provision-desktop-environment-vagrant/

[6] https://pynet.twb-tech.com/blog/python/ipaddress.html

[7] https://scapy.readthedocs.io/en/latest/usage.html

[8] http://www.compjour.org/tutorials/intro-to-python-requests-and-json/

[9] https://www.geeksforgeeks.org/get-post-requests-using-python/

[10] https://www.django-rest-framework.org/tutorial/2-requests-and-responses/

[11] https://djangobook.com/request-response-objects/

[12] https://www.virtualbox.org/manual/ch06.htmlnatforward

[13] https://docs.python.org/2/library/socket.html