# LINUX FIREWALL CONFIGURATION TOOLKIT FOR DA-1 & DA-2

## COURSE – OPERATING SYSTEMS

## FACULTY – DR. VALLIDEVI K.

## PROJECT REPORT BY -

VEDANT S KUDALKAR (22BAI1321)

VIVEK TRIPATHI (22BAI1388)

KIRAN KUMAR BHATRA (22BAI1405)

Date – 7th Nov, 2023.

## Introduction:

### What is a Linux firewall?

A firewall is a network security device or software that acts as a protective barrier between a trusted network, such as a local area network (LAN) or a computer, and the untrusted external network, which is often the internet. Its primary purpose is to monitor and control incoming and outgoing network traffic, allowing or blocking data packets based on a set of predefined security rules.

## Key points about firewalls:

1. **Security Barrier:** Firewalls are like digital gatekeepers. They examine network traffic and data packets to decide whether they are safe or malicious.

2. **Access Control:** Firewalls use a set of rules to determine which network connections are permitted and which are denied. These rules can be based on factors like source and destination IP addresses, port numbers, and the type of network protocol (e.g., TCP, UDP).

3. **Protection Against Threats:** Firewalls protect networks and devices from a range of cybersecurity threats, including unauthorized access, malware, viruses, denial of service attacks, and more.

4. **Network Segmentation:** Firewalls can be used to segment a network into different security zones, ensuring that sensitive data is kept separate from less secure parts of the network.

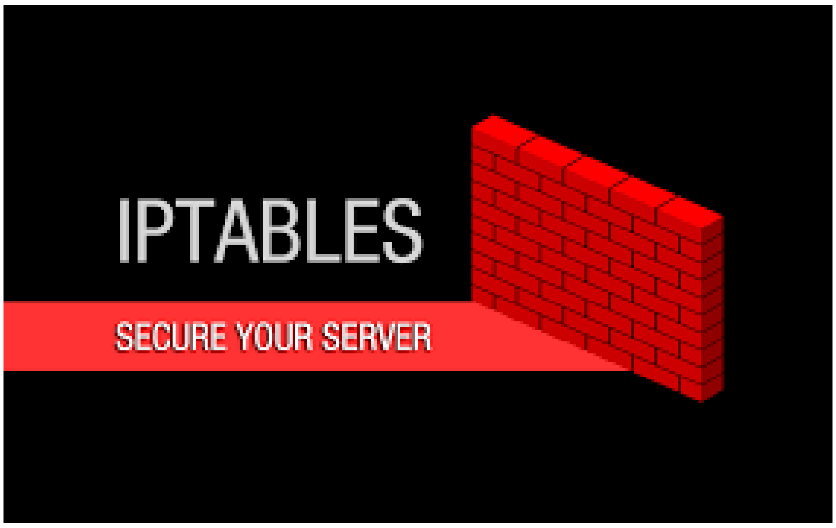
5. **Logging and Monitoring:** Firewalls often provide logging and reporting features, allowing administrators to track and analyse network activity for security purposes.

6. **Types of Firewalls:** There are several types of firewalls, including hardware firewalls (physical devices), software firewalls (installed on computers or network devices), and cloud-based firewalls that protect cloud-hosted applications and services.

7. **Stateful Inspection:** Many modern firewalls use stateful inspection to keep track of the state of active connections. This allows them to make more informed decisions about whether to allow or block traffic.

## Here are some popular Linux firewall examples:

**1. iptables**: iptables is a widely used and highly flexible command-line firewall tool for Linux. It allows administrators to define rules for packet filtering, network address translation (NAT), and packet mangling. It's a fundamental part of many Linux distributions and is often used in conjunction with other tools like iptables-restore and iptables-save for rule management.



**2. ufw (Uncomplicated Firewall**): ufw is a user-friendly front-end for iptables, designed to simplify firewall management. It provides a simplified interface for adding and managing firewall rules. It's often used on Ubuntu and other Debian-based systems.

**3. nftables:** nftables is a newer framework for packet filtering in Linux that aims to replace iptables and other older tools. It offers enhanced performance, flexibility, and extensibility. nftables is becoming increasingly popular and is included in many modern Linux distributions.

**4. Shorewall**: Shorewall is a high-level configuration tool for iptables. It simplifies the process of configuring complex firewall rules by using a configuration file format that is easier to understand and manage. It's well-documented and widely used for setting up firewalls on Linux systems.

**5. iptables-restore and iptables-save**: These are not standalone firewalls but rather tools used in conjunction with iptables. They allow you to save and restore firewall rules from files, making it easier to manage and maintain rulesets.

**6. firewalld:** firewalld is a dynamic firewall management tool for Linux systems. It allows users to define and modify firewall rules without disrupting established network connections. firewalld is commonly used on Red Hat-based distributions such as CentOS and Fedora.

**7. Fail2ban:** Fail2ban is a specialized intrusion prevention system that uses iptables to block IP addresses that repeatedly fail authentication or exhibit suspicious behavior. It's often used to protect services like SSH and web servers.

**8. IPFire:** IPFire is a Linux-based firewall distribution designed for easy setup and configuration. It includes a web-based management interface and offers features like intrusion detection, VPN support, and proxy services.

# ABOUT OUR PROJECT

The project is about creating a tool that makes it easier to control and manage the rules that determine which data can come into and go out of your computer. It's like a virtual gatekeeper that protects your computer from bad things on the internet while allowing the good things in.

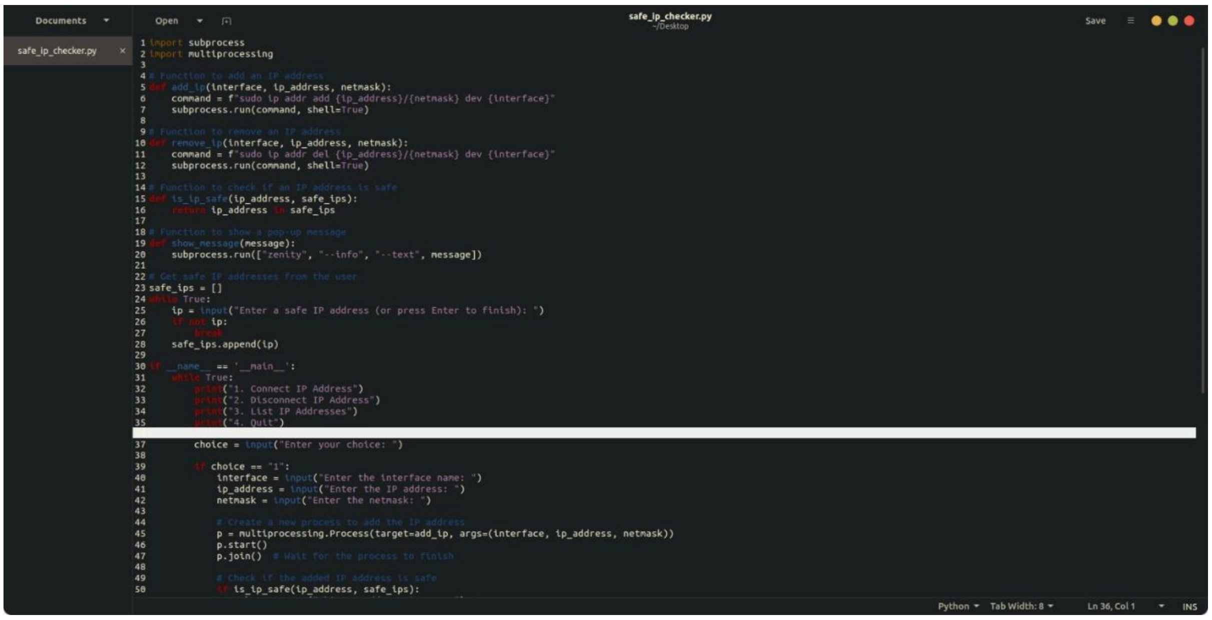
Imagine you have a door with a security guard. The door has rules like "Only let in people with an invitation." This project is like creating a control panel to change those rules. You can say, "Let in everyone for a party" or "Block everyone because I want privacy."

The project uses computer code to do this, and it has a simple menu. You can add new rules to allow or block certain types of data (like people with or without invitations), or you can check the existing rules. It's like a tool that helps you control who can come to your party (your computer) and who can't.

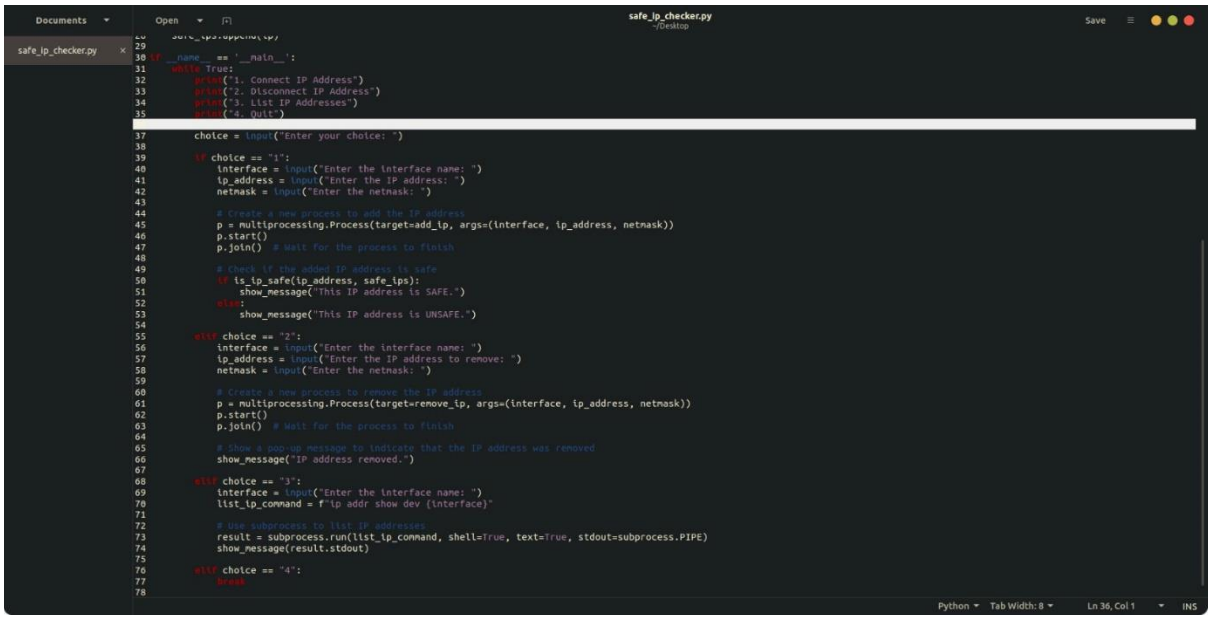
So, in simple words, the project is like a computer bouncer that helps you decide who gets in and who doesn't, making your computer more secure.

**There are various firewalls command which can be used on a Linux machine for securing it, which have already been mentioned in our report above. Out of all those methods "iptable" is one of the most effective method therefore we implemented it on our personal Linux machine and its code and output is given below.**

# PYTHON FILE



```python
import subprocess
import multiprocessing

# Function to add an IP address
def add_ip(interface, ip_address, netmask):
    command = f"sudo ip addr add {ip_address}/{netmask} dev {interface}"
    subprocess.run(command, shell=True)

# Function to remove an IP address
def remove_ip(interface, ip_address, netmask):
    command = f"sudo ip addr del {ip_address}/{netmask} dev {interface}"
    subprocess.run(command, shell=True)

# Function to check if an IP address is safe
def is_ip_safe(ip_address, safe_ips):
    return ip_address in safe_ips

# Function to show a pop-up message
def show_message(message):
    subprocess.run(["zenity", "--info", "--text", message])

# Get safe IP addresses from the user
safe_ips = []
while True:
    ip = input("Enter a safe IP address (or press Enter to finish): ")
    if not ip:
        break
    safe_ips.append(ip)

if __name__ == '__main__':
    while True:
        print("1. Connect IP Address")
        print("2. Disconnect IP Address")
        print("3. List IP Addresses")
        print("4. Quit")

        choice = input("Enter your choice: ")

        if choice == "1":
            interface = input("Enter the interface name: ")
            ip_address = input("Enter the IP address: ")
            netmask = input("Enter the netmask: ")

            # Create a new process to add the IP address
            p = multiprocessing.Process(target=add_ip, args=(interface, ip_address, netmask))
            p.start()
            p.join()  # Wait for the process to finish

            # Check if the added IP address is safe
            if is_ip_safe(ip_address, safe_ips):
```



```python

if __name__ == '__main__':
    while True:
        print("1. Connect IP Address")
        print("2. Disconnect IP Address")
        print("3. List IP Addresses")
        print("4. Quit")

        choice = input("Enter your choice: ")

        if choice == "1":
            interface = input("Enter the interface name: ")
            ip_address = input("Enter the IP address: ")
            netmask = input("Enter the netmask: ")

            # Create a new process to add the IP address
            p = multiprocessing.Process(target=add_ip, args=(interface, ip_address, netmask))
            p.start()
            p.join()  # Wait for the process to finish

            # Check if the added IP address is safe
            if is_ip_safe(ip_address, safe_ips):
                show_message("This IP address is SAFE.")
            else:
                show_message("This IP address is UNSAFE.")

        elif choice == "2":
            interface = input("Enter the interface name: ")
            ip_address = input("Enter the IP address to remove: ")
            netmask = input("Enter the netmask: ")

            # Create a new process to remove the IP address
            p = multiprocessing.Process(target=remove_ip, args=(interface, ip_address, netmask))
            p.start()
            p.join()  # Wait for the process to finish

            # Show a pop-up message to indicate that the IP address was removed
            show_message("IP address removed.")

        elif choice == "3":
            interface = input("Enter the interface name: ")
            list_ip_command = f"ip addr show dev {interface}"

            # Use subprocess to list IP addresses
            result = subprocess.run(list_ip_command, shell=True, text=True, stdout=subprocess.PIPE)
            show_message(result.stdout)

        elif choice == "4":
            break

```

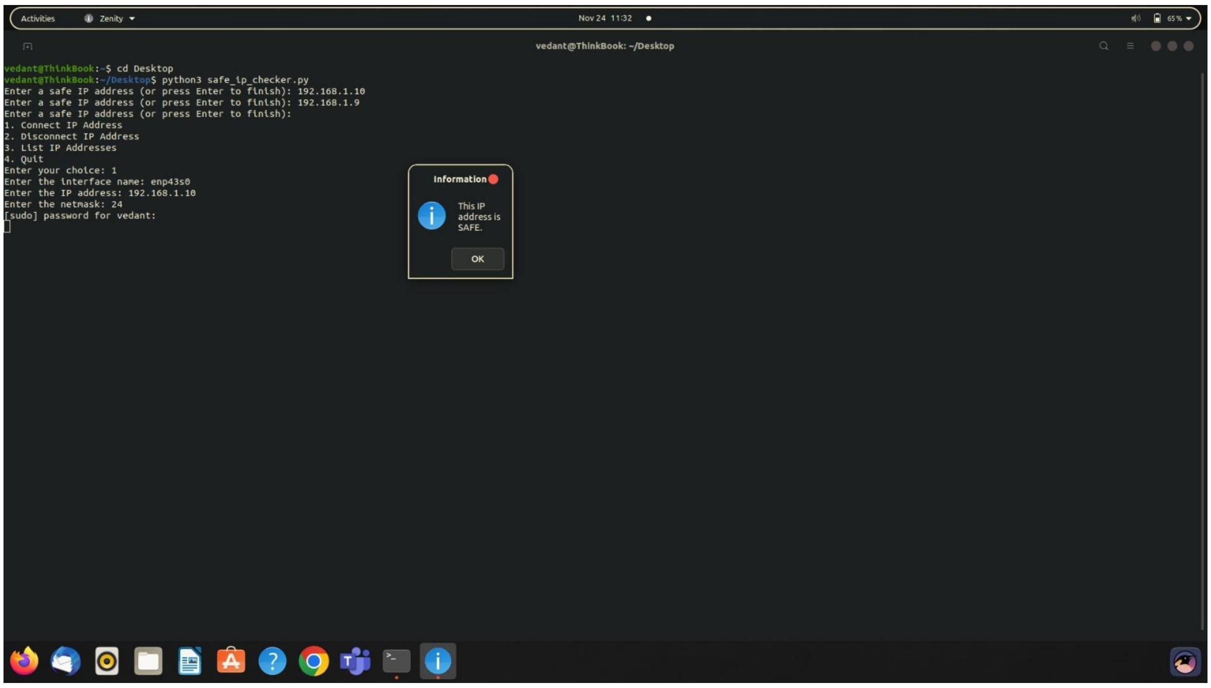
## ABOUT THE CODE

This Python script appears to be a basic network management tool that allows users to add and remove IP addresses from a specified network interface. It provides a simple command-line interface for users to interact with different functions. The script uses the `subprocess` module to execute shell commands and the `multiprocessing` module to run certain tasks in parallel processes.

The first part of the script defines functions to add and remove IP addresses, check if an IP address is considered safe, and display pop-up messages. The user is prompted to input a list of safe IP addresses, creating a whitelist. The main part of the script then enters a loop where the user is presented with a menu of options (connect IP address, disconnect IP address, list IP addresses, and quit). Depending on the user's choice, the script either adds or removes an IP address from the specified network interface, lists the current IP addresses on that interface, or exits the loop.

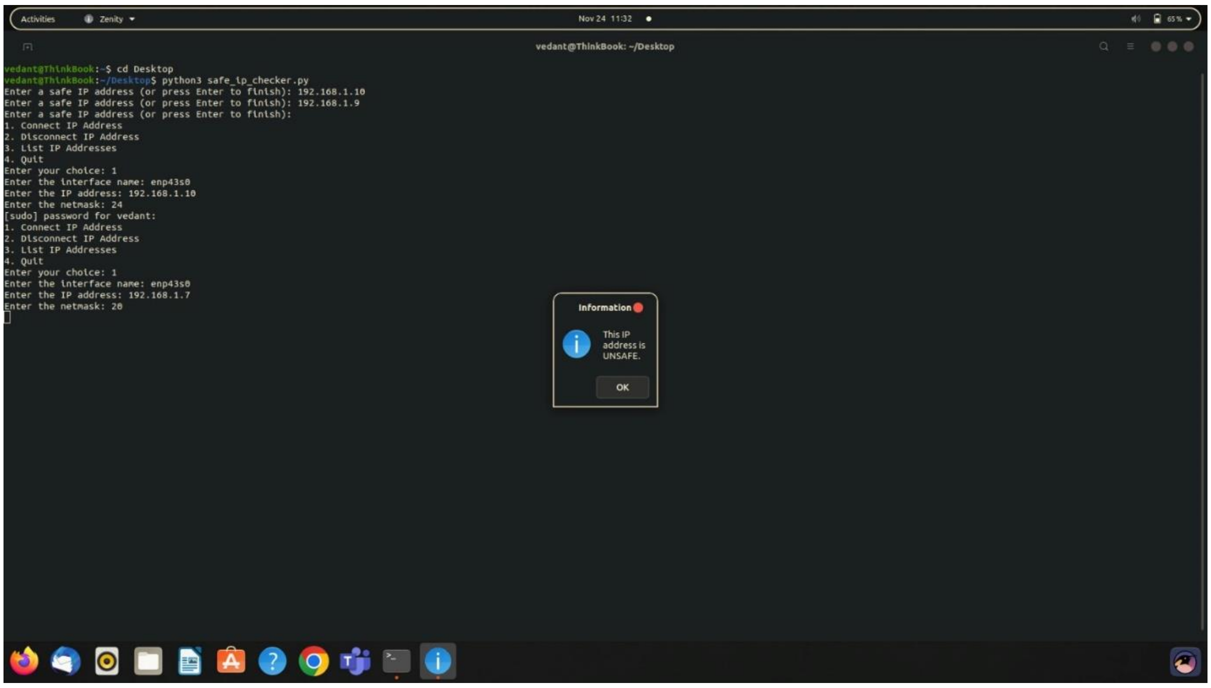
The second part of the script contains the main execution block where the user interacts with the tool. It uses the `multiprocessing` module to create a new process for each task, such as adding or removing an IP address. After each process is started, the script waits for it to finish using the `join()` method. Additionally, the script displays pop-up messages indicating whether the IP address is safe or unsafe. The user can repeatedly choose from the menu options until they decide to quit the program.

## IMPLEMENTATION IN TERMINAL



```
vedant@ThinkBook:~$ cd Desktop
vedant@ThinkBook:~/Desktop$ python3 safe_ip_checker.py
Enter a safe IP address (or press Enter to finish): 192.168.1.10
Enter a safe IP address (or press Enter to finish): 192.168.1.9
Enter a safe IP address (or press Enter to finish):
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.10
Enter the netmask: 24
[sudo] password for vedant:
```

This IP address is SAFE.

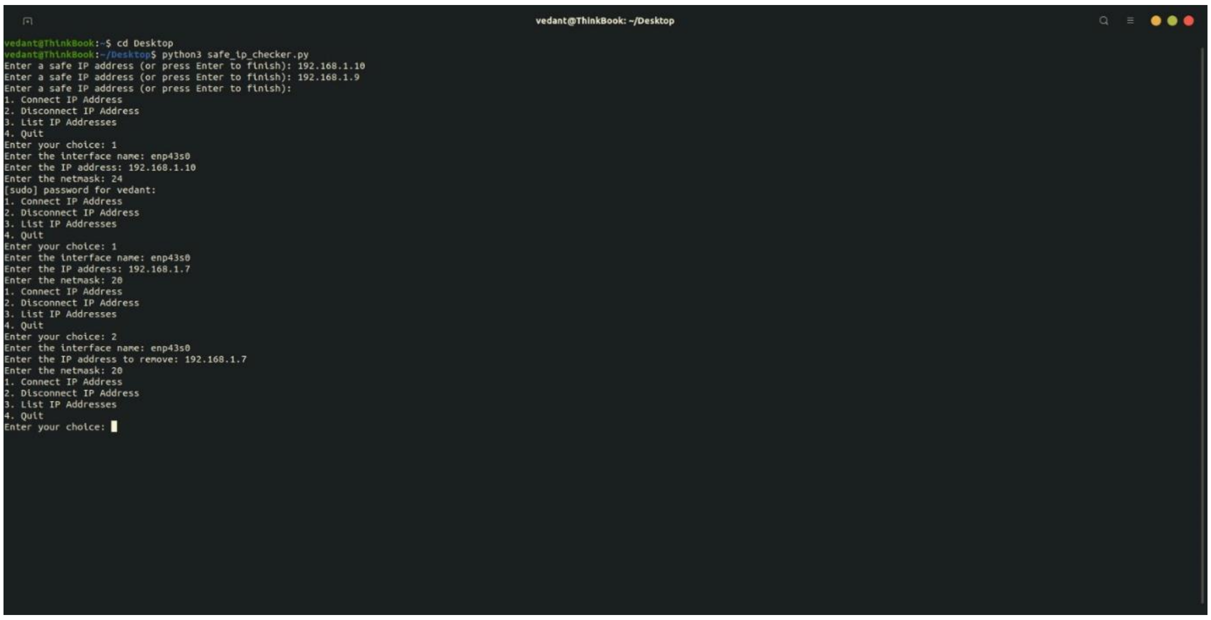


```
vedant@ThinkBook:~$ cd Desktop
vedant@ThinkBook:~/Desktop$ python3 safe_ip_checker.py
Enter a safe IP address (or press Enter to finish): 192.168.1.10
Enter a safe IP address (or press Enter to finish): 192.168.1.9
Enter a safe IP address (or press Enter to finish):
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.10
Enter the netmask: 24
[sudo] password for vedant:
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.7
Enter the netmask: 20
```

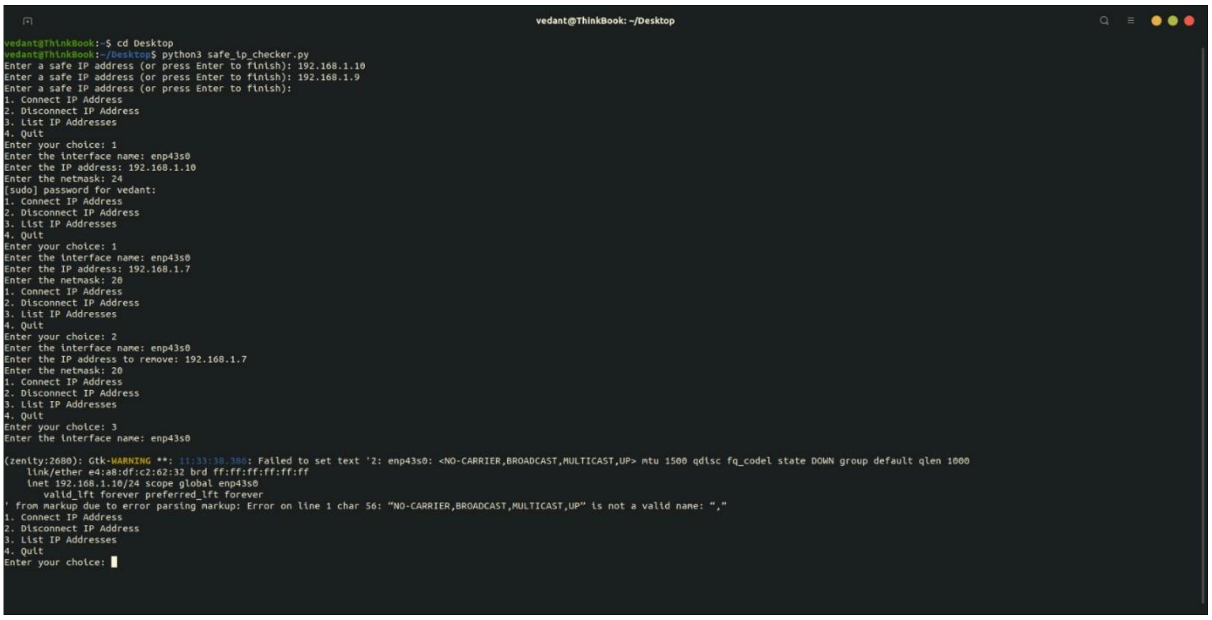
Information

This IP address is UNSAFE.

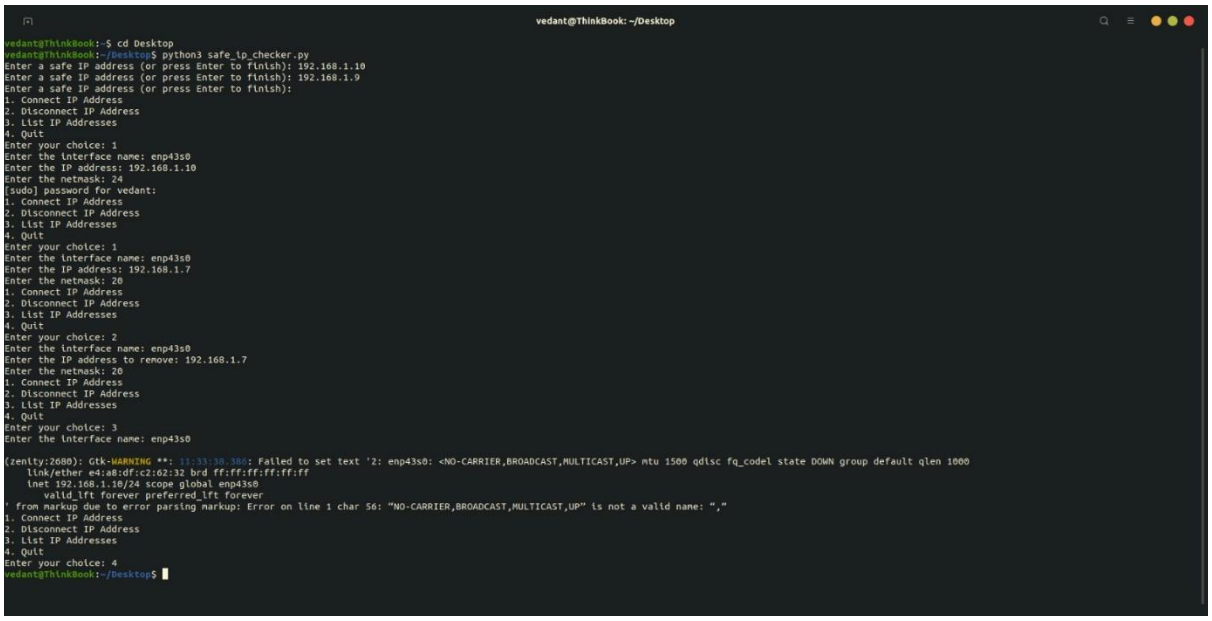
OK



```
vedant@ThinkBook:~$ cd Desktop
vedant@ThinkBook:~/Desktop$ python3 safe_ip_checker.py
Enter a safe IP address (or press Enter to finish): 192.168.1.10
Enter a safe IP address (or press Enter to finish): 192.168.1.9
Enter a safe IP address (or press Enter to finish):
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.10
Enter the netmask: 24
[sudo] password for vedant:
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.7
Enter the netmask: 20
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 2
Enter the interface name: enp43s0
Enter the IP address to remove: 192.168.1.7
Enter the netmask: 20
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice:
```



```
vedant@ThinkBook:~$ cd Desktop
vedant@ThinkBook:~/Desktop$ python3 safe_ip_checker.py
Enter a safe IP address (or press Enter to finish): 192.168.1.10
Enter a safe IP address (or press Enter to finish): 192.168.1.9
Enter a safe IP address (or press Enter to finish):
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.10
Enter the netmask: 24
[sudo] password for vedant:
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.7
Enter the netmask: 20
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 2
Enter the interface name: enp43s0
Enter the IP address to remove: 192.168.1.7
Enter the netmask: 20
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 3
Enter the interface name: enp43s0

(zenity:2680): Gtk-WARNING **: 11:33:38.386: Failed to set text '2: enp43s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether e4:a8:df:c2:62:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.10/24 scope global enp43s0
       valid_lft forever preferred_lft forever
' from markup due to error parsing markup: Error on line 1 char 56: "NO-CARRIER,BROADCAST,MULTICAST,UP" is not a valid name: ","
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 
```



```
vedant@ThinkBook:~$ cd Desktop
vedant@ThinkBook:~/Desktop$ python3 safe_ip_checker.py
Enter a safe IP address (or press Enter to finish): 192.168.1.10
Enter a safe IP address (or press Enter to finish): 192.168.1.9
Enter a safe IP address (or press Enter to finish):
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.10
Enter the netmask: 24
[sudo] password for vedant:
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 1
Enter the interface name: enp43s0
Enter the IP address: 192.168.1.7
Enter the netmask: 20
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 2
Enter the interface name: enp43s0
Enter the IP address to remove: 192.168.1.7
Enter the netmask: 20
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 3
Enter the interface name: enp43s0

(zenity:2680): Gtk-WARNING **: 11:33:38.386: Failed to set text '2: enp43s0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc fq_codel state DOWN group default qlen 1000
    link/ether e4:a8:df:c2:62:32 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.10/24 scope global enp43s0
       valid_lft forever preferred_lft forever
' from markup due to error parsing markup: Error on line 1 char 56: "NO-CARRIER,BROADCAST,MULTICAST,UP" is not a valid name: ","
1. Connect IP Address
2. Disconnect IP Address
3. List IP Addresses
4. Quit
Enter your choice: 4
vedant@ThinkBook:~/Desktop$ 
```

## Connection with O.S

We have used ‘python’ language to connect our project with the operating system.

Python is a widely used programming language which enables us to use various libraries out of which we have specifically used:

1. **OS Library:** The OS module in Python provides functions for interacting with the operating system. OS comes under Python's standard utility modules. This module provides a portable way of using operating system-dependent functionality.

2. **Subprocess Library:** Subprocess in Python is a module used to run new codes and applications by creating new processes. It lets you start new applications right from the Python program you are currently writing.

## Conclusion

The Linux Firewall Configuration Tool project has provided a practical and simplified solution for managing firewall rules on Linux systems. Through this project, we have created a command-line tool that allows users to add and list firewall rules using the `iptables` command. While this tool is basic in its current form, it demonstrates the fundamental concepts of firewall rule management and serves as a starting point for understanding how to interact with firewall configurations on Linux.

The project emphasizes the importance of securing network connections and controlling the flow of data in and out of a system. It highlights the role of firewalls as essential security components, safeguarding computers and networks from potential cyber threats. Furthermore, the project introduces users to the concept of firewall rules, illustrating how specific rules can permit or block network traffic based on various criteria, such as port numbers and protocols.

To build a more comprehensive and user-friendly Linux Firewall Configuration Tool with a graphical user interface (GUI), further development work is required. Such a tool would provide an intuitive and accessible way for users to define and manage firewall rules, enhancing the security of their Linux systems while reducing the complexity associated with command-line configuration.

In conclusion, the Linux Firewall Configuration Tool project lays the foundation for understanding and managing firewall rules on Linux. It serves as a valuable educational resource and an initial step towards the development of a more advanced tool for effective firewall configuration and security management on Linux systems.