Personal Firewall using Python – Project Report

1. Objective

The main goal of this project is to build a lightweight personal firewall using Python that can:

- Monitor incoming and outgoing network traffic in real time.

- Block or allow traffic based on customizable rule sets.

- Log suspicious activity for auditing purposes.

- Optionally integrate with iptables for system-level blocking.

- Provide a GUI interface for live packet monitoring.

This project was developed and tested on Kali Linux, using the Scapy library for packet sniffing and Tkinter for GUI (optional).

2. Setting Up the Project Directory

Open Terminal and Create a Project Folder

Command: mkdir firewall

Open Terminal and Navigate to Your Firewall Directory

Command: cd firewall

3. Tools & Technologies Used

- Python: Main programming language used for scripting.

- Scapy: A powerful Python library used for packet sniffing and manipulation.

- Tkinter: Built-in Python library for creating GUI interfaces (optional).

- iptables: Linux-based packet filtering tool for enforcing system-level blocking.

- logging module: Used to record logs of suspicious packets.

- JSON: Used to store user-defined firewall rules (IP, port, protocol).

4. File Description

• firewall.py

- Main firewall engine.

- Sniffs packets using Scapy.

- Applies rule checks from rules.json.

- Calls the logger if the packet is suspicious.

• packet_logger.py

- Handles structured logging of flagged packets.

- Writes detailed logs into suspicious_packets.log.

• gui_monitor.py

- Optional real-time monitoring interface built with Tkinter.

- Displays packet count and flagged packets in a visual format.

• rules.json

- JSON file containing user-defined rules.

- Structure: lists of blocked IPs, blocked ports, and allowed protocols.

• suspicious_packets.log

- Log file containing all suspicious packets that triggered rules.

- Includes timestamps, IP addresses, ports, protocols, and the reason for flagging.

• __pycache__/

- Auto-generated folder for Python bytecode.

5. How It Works

STEP 1: Packet Sniffing

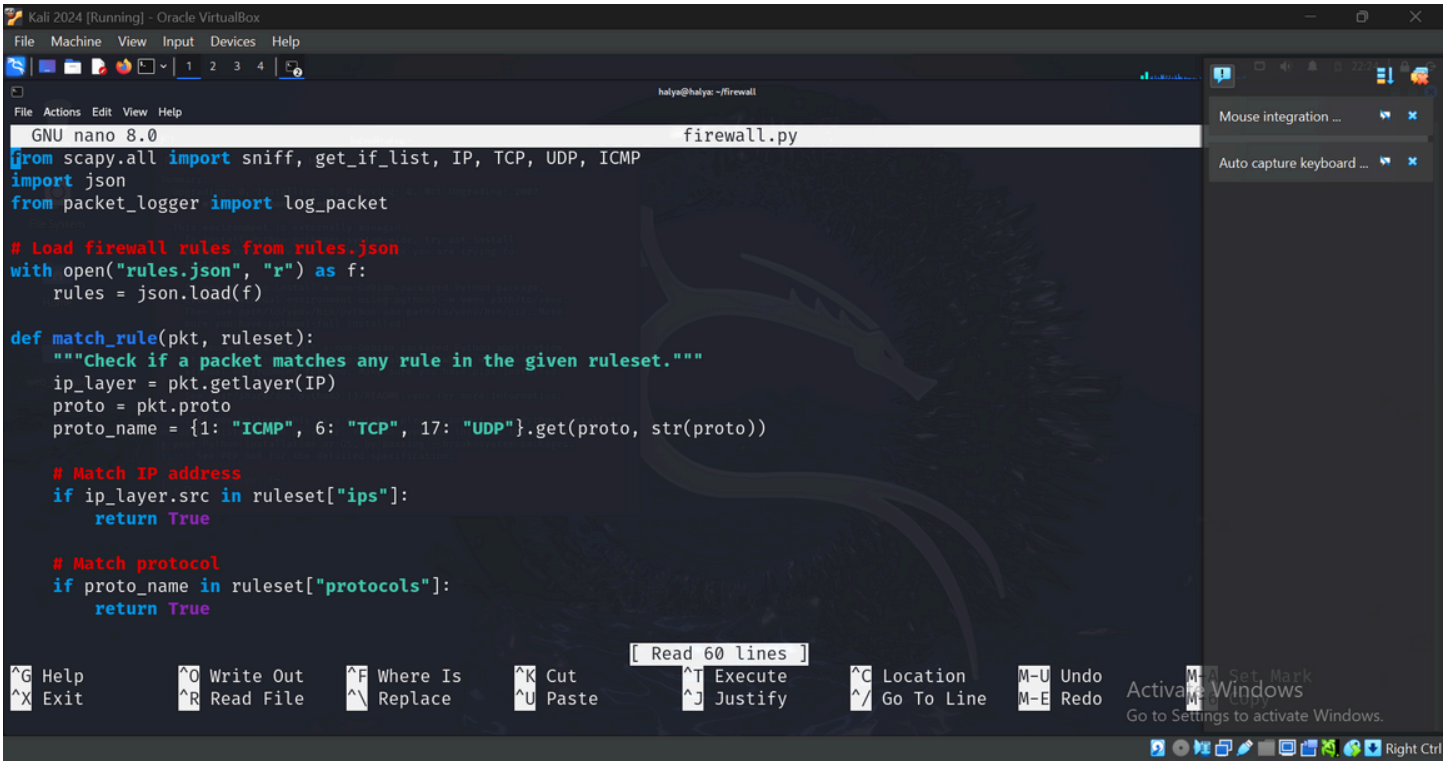
- The firewall.py script uses Scapy's `sniff()` function to capture live packets from the selected interface.

- To use this, open the directory that we created before, 'cd firewall'.
- **Command:** nano firewall.py.
- Use the above command and add the script to it.
- (Use **Right Click > Paste** in terminal or Shift + Ctrl + V)

```
from scapy.all import sniff, get_if_list, IP, TCP, UDP, ICMP
import json
```

```
from packet_logger import log_packet

# Load firewall rules from rules.json
with open("rules.json", "r") as f:
    rules = json.load(f)

def match_rule(pkt, ruleset):
    """Check if a packet matches any rule in the given ruleset."""
    ip_layer = pkt.getlayer(IP)
    proto = pkt.proto
    proto_name = {1: "ICMP", 6: "TCP", 17: "UDP"}.get(proto, str(proto))

    # Match IP address
    if ip_layer.src in ruleset["ips"]:
        return True

    # Match protocol
    if proto_name in ruleset["protocols"]:
        return True

    # Match ports for TCP/UDP
    if proto_name == "TCP" and pkt.haslayer(TCP):
        if pkt[TCP].sport in ruleset["ports"] or pkt[TCP].dport in ruleset["ports"]:
            return True
    elif proto_name == "UDP" and pkt.haslayer(UDP):
        if pkt[UDP].sport in ruleset["ports"] or pkt[UDP].dport in ruleset["ports"]:
            return True

    return False

def packet_callback(pkt):
    """Callback function for sniffed packets"""
    if not pkt.haslayer(IP):
        return

    if match_rule(pkt, rules["block"]):
        log_packet(pkt, status="BLOCKED")
        print(f"[BLOCKED] {pkt[IP].src} -> {pkt[IP].dst}")
    elif match_rule(pkt, rules["allow"]):
        print(f"[ALLOWED] {pkt[IP].src} -> {pkt[IP].dst}")
      .
```

```
    else:
        log_packet(pkt, status="UNKNOWN")
        print(f"[UNKNOWN] {pkt[IP].src} -> {pkt[IP].dst}")

def start_firewall():
    """Start packet sniffing on selected interface."""
    print("[*] Available interfaces:")
    interfaces = get_if_list()
    for i, iface in enumerate(interfaces):
        print(f"{i}: {iface}")

    iface = input("[?] Enter interface name to sniff (e.g., Ethernet, Wi-Fi): ")
    print(f"[*] Starting packet sniffing on interface: {iface}")

    sniff(iface=iface, filter="ip", prn=packet_callback, store=False)

if __name__ == "__main__":
    start_firewall()
```



- **Press:**
  CTRL + O → press Enter to save
  CTRL + X → to exit nano

STEP 2: Rule Checking

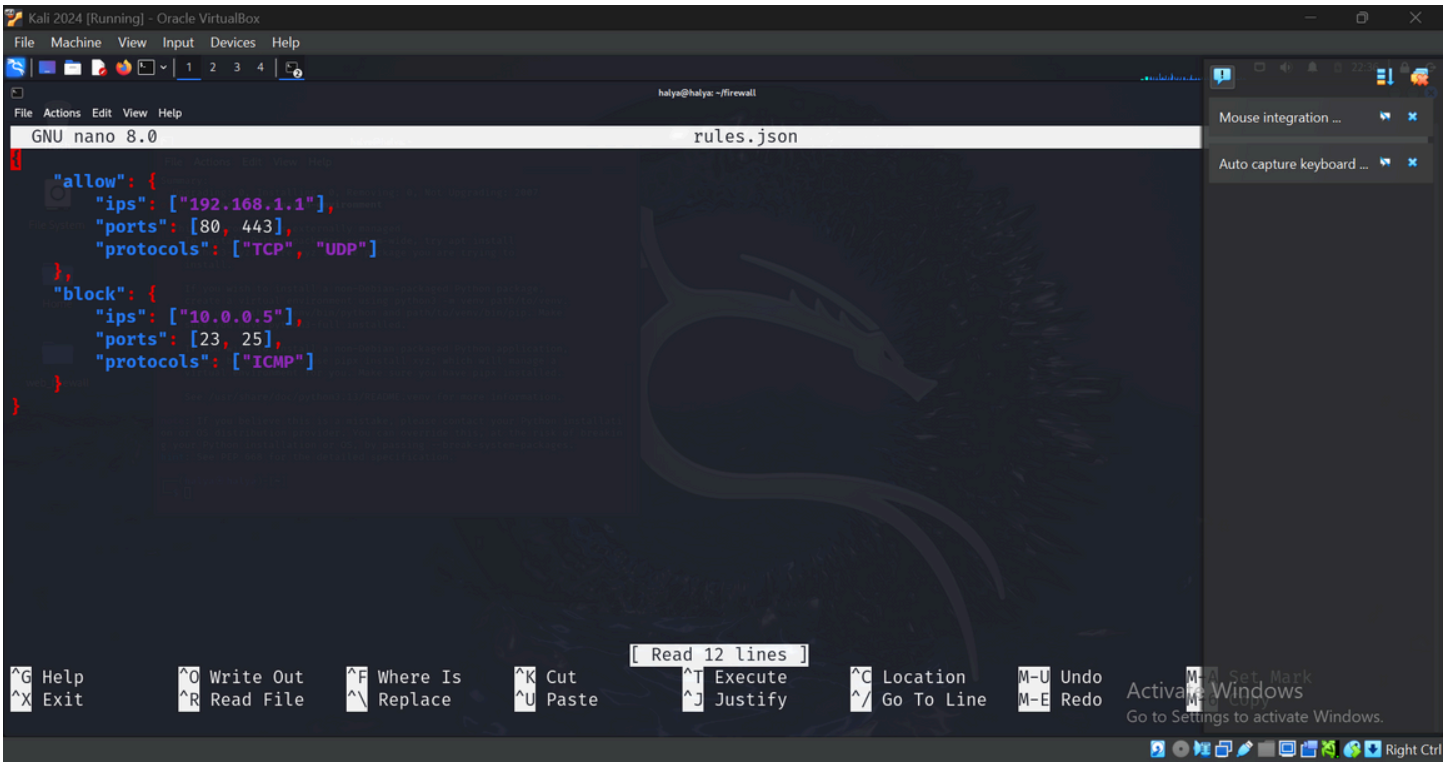
- For each packet, it extracts protocol, IP, and port information.

- These details are compared against the rules set in rules.json.

- If a packet violates a rule (e.g., blocked IP or port), it is flagged as suspicious.

- In the same **firewall** folder, create
- **Command:** nano rules.json.
- Use the above command and add the script to it.
- (Use **Right Click > Paste** in terminal or Shift + Ctrl + V)

```
{
  "allow": {
    "ips": ["192.168.1.1"],
    "ports": [80, 443],
    "protocols": ["TCP", "UDP"]
  },
  "block": {
    "ips": ["10.0.0.5"],
    "ports": [23, 25],
    "protocols": ["ICMP"]
  }
}
```



- **Press:**
  CTRL + O → press Enter to save
  CTRL + X → to exit nano

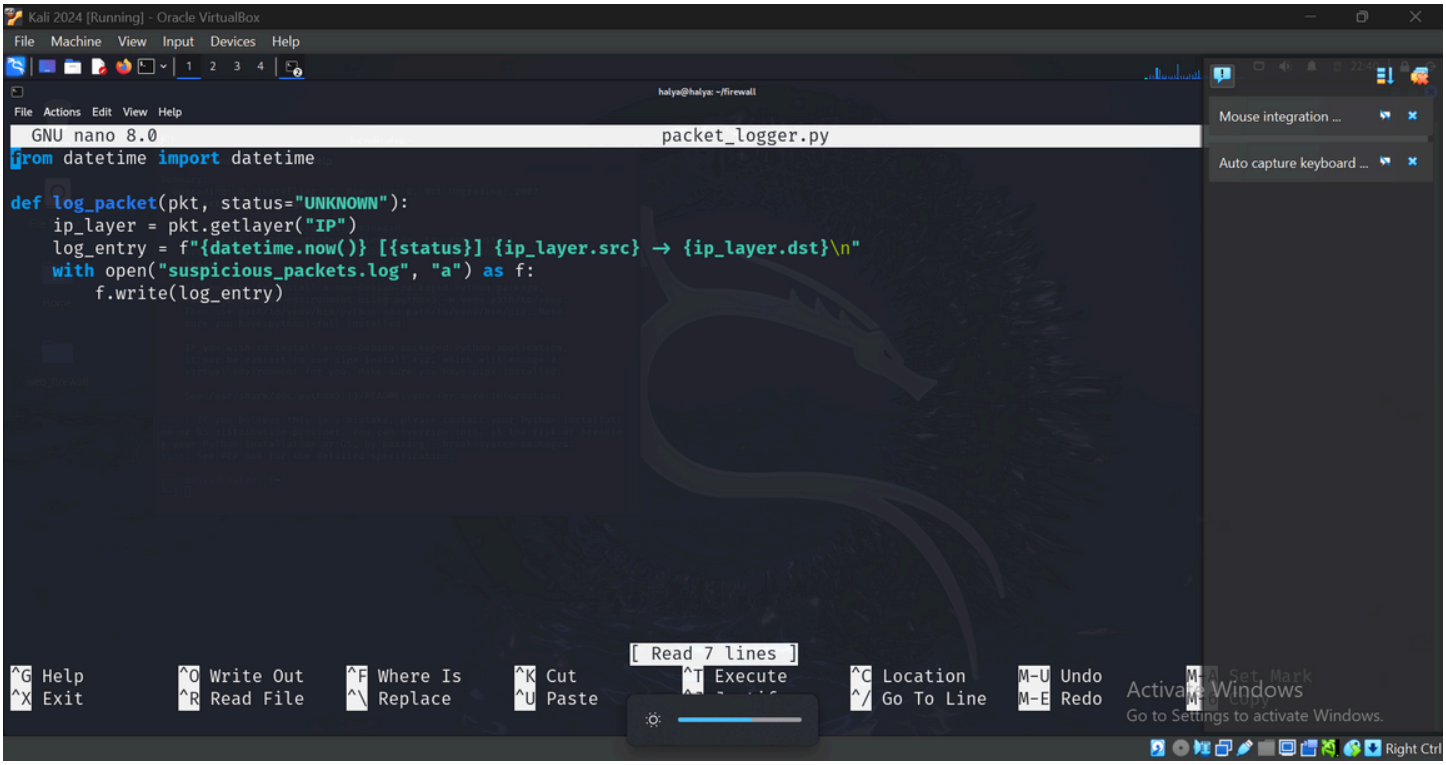
STEP 3: Logging

- Flagged packets are passed to packet_logger.py for logging.

- Each log includes details like timestamp, IPs, port, protocol, and reason for being blocked.

- In the same **firewall** folder, create
- **Command:** nano packet_logger.py.
- Use the above command and add the script to it.
- (Use **Right Click > Paste** in terminal or Shift + Ctrl + V)

```
from datetime import datetime

def log_packet(pkt, status="UNKNOWN"):
    ip_layer = pkt.getlayer("IP")
    log_entry = f"{datetime.now()} [{status}] {ip_layer.src} -> {ip_layer.dst}\n"
    with open("suspicious_packets.log", "a") as f:
        f.write(log_entry)
```



- **Press:**
  CTRL + O → press Enter to save
  CTRL + X → to exit nano

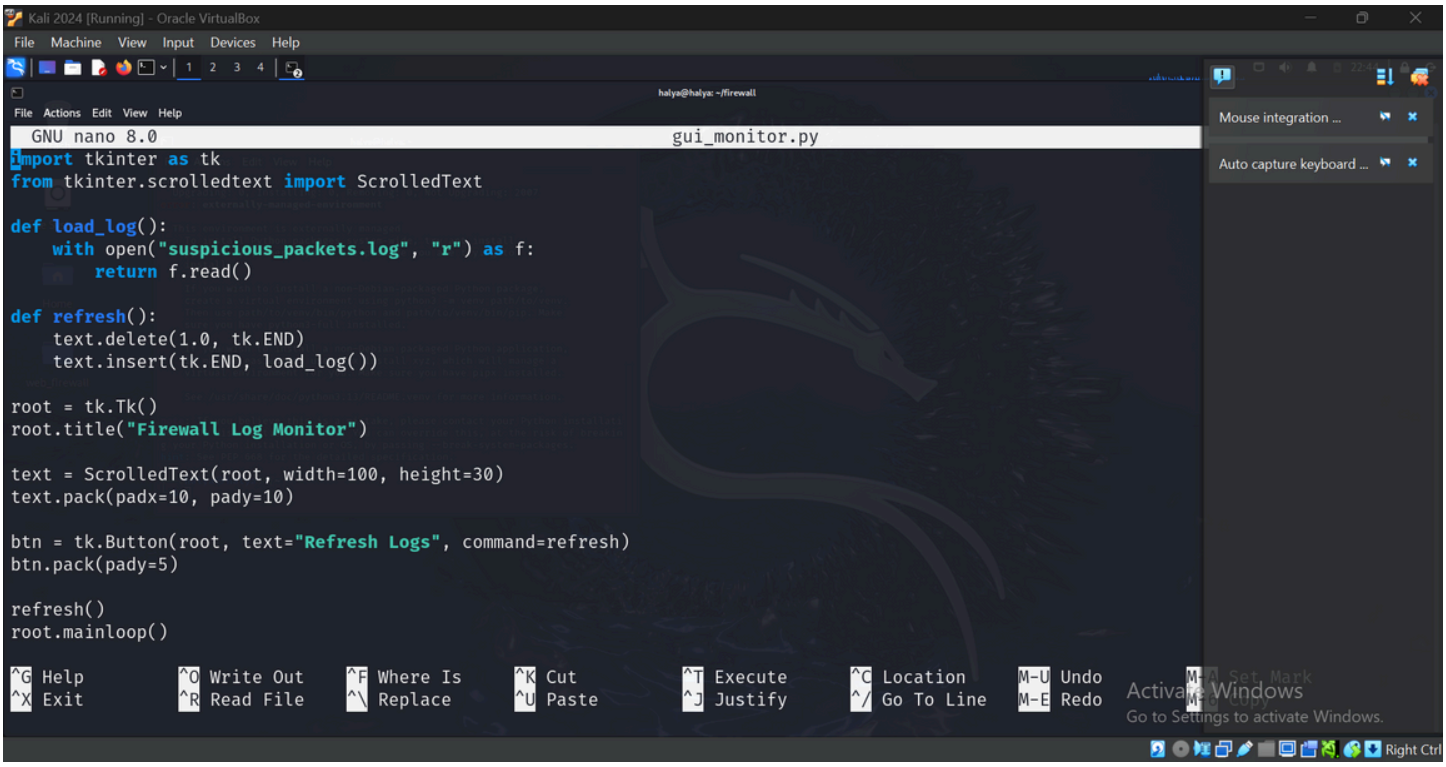
STEP 4: GUI Monitoring (Optional)

- gui_monitor.py provides a live dashboard that displays packet statistics and blocked activity in real-time.

- In the same **firewall** folder, create
- **Command:** nano gui_monitor.py.

- Use the above command and add the script to it.
- (Use **Right Click > Paste** in terminal or Shift + Ctrl + V)

```
import tkinter as tk
from tkinter.scrolledtext import ScrolledText

def load_log():
    with open("suspicious_packets.log", "r") as f:
        return f.read()

def refresh():
    text.delete(1.0, tk.END)
    text.insert(tk.END, load_log())

root = tk.Tk()
root.title("Firewall Log Monitor")

text = ScrolledText(root, width=100, height=30)
text.pack(padx=10, pady=10)

btn = tk.Button(root, text="Refresh Logs", command=refresh)
btn.pack(pady=5)

refresh()
root.mainloop()
```



- **Press:**
  CTRL + O → press Enter to save
  CTRL + X → to exit nano

Step 5: Verify It Was Saved

Command: Use the **'cat'** command to view the files you have saved

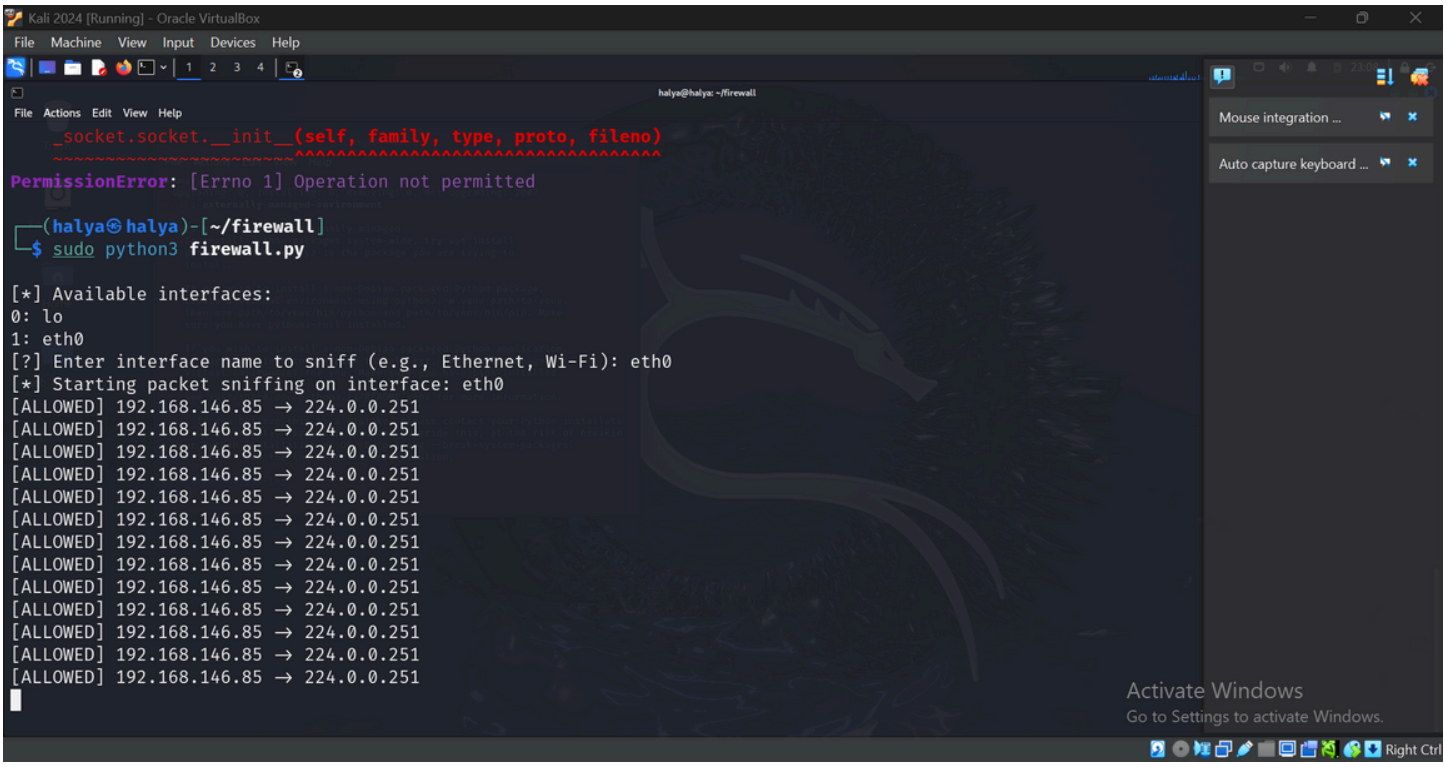
- **For example:** cat rules.json.
- like the same use for other files.

Step 6: Run the Application

Run the Packet Sniffing

Command: python3 firewall.py

- Check how it is working
- Select your network interface name to sniff.



- To stop this, use **CTRL + C**

7. Key Features

- Real-time packet sniffing
- Rule-based traffic filtering
- Customizable rules via rules.json
- Suspicious packet logging with timestamp
- GUI-based live monitoring (optional)
- Modular Python design
- Lightweight and fast

8. Deliverables

- firewall.py – main script
- gui_monitor.py – optional GUI dashboard
- packet_logger.py – packet logging utility
- rules.json – configuration file for rules
- suspicious_packets.log – log file for flagged packets

9. Limitations & Future Enhancement

1. **Manual Rule Configuration**

- **Limitation:** Currently, the firewall rules must be manually edited in the rules.json file using a text editor.

- **Improvement:** Implement a **GUI-based or web-based rule editor** where users can add, edit, or delete rules interactively without touching the code.

No Alerting System

- **Limitation:** The firewall logs suspicious packets but does **not notify the user in real time** when a threat is detected.
- **Improvement:** Add a **notification system**, such as desktop pop-ups or **email alerts** for high-severity traffic, so users can respond immediately.

Basic Packet Analysis

- **Limitation:** The current version checks only **IP, port, and protocol**, and doesn't inspect packet payloads (contents).
- **Improvement:** Introduce **deep packet inspection (DPI)** to detect threats hidden in the payload, such as suspicious commands, malware signatures, or attack patterns.

No iptables Integration

- **Limitation:** While rules are applied in Python for detection and logging, they are not enforced at the system level using **iptables**.
- **Improvement:** Automatically execute **iptables commands** to drop or block packets in real time when a rule is matched, adding a strong layer of system-level security.

Single-threaded Execution

- **Limitation:** The packet sniffing and processing are done in a single thread, which may slow down performance under heavy traffic.
- **Improvement:** Use **multithreading or asynchronous programming** to handle packet capture and rule evaluation in parallel, improving speed and responsiveness.

10. Conclusion

The **Personal Firewall using Python** is a practical and educational project designed to enhance understanding of real-time network traffic analysis, rule-based filtering, and packet inspection on Linux systems. By leveraging the power of **Scapy** for packet sniffing, **JSON** for dynamic rule management, and an optional **Tkinter GUI** for live monitoring, this firewall provides a customizable and modular security solution.

It successfully demonstrates how a user can:

- Monitor incoming and outgoing traffic,
- Define rules for blocking suspicious IPs, ports, and protocols.
- Log security events for auditing purposes,
- And (optionally) present activity visually through a simple GUI.

Though lightweight in its current form, the firewall lays a strong foundation for more advanced features such as:

- **Deep packet inspection (DPI),**
- **Automated iptables enforcement,**
- **Real-time alerting,**
- And **multithreaded performance.**

This project not only reinforces core concepts in networking and cybersecurity but also encourages further development in host-based intrusion detection and Linux firewall engineering. It’s an ideal tool for students, security researchers, or anyone looking to understand and control traffic flow on their machine.