System Design

The firewall operates on predefined rules to filter network packets. It checks each packet for:

- Source and destination IP addresses.
- Ports being accessed.
- Protocol types (TCP or UDP).

The packet is either allowed or blocked based on the matching rule, with detailed logs generated for every action.

Implementation

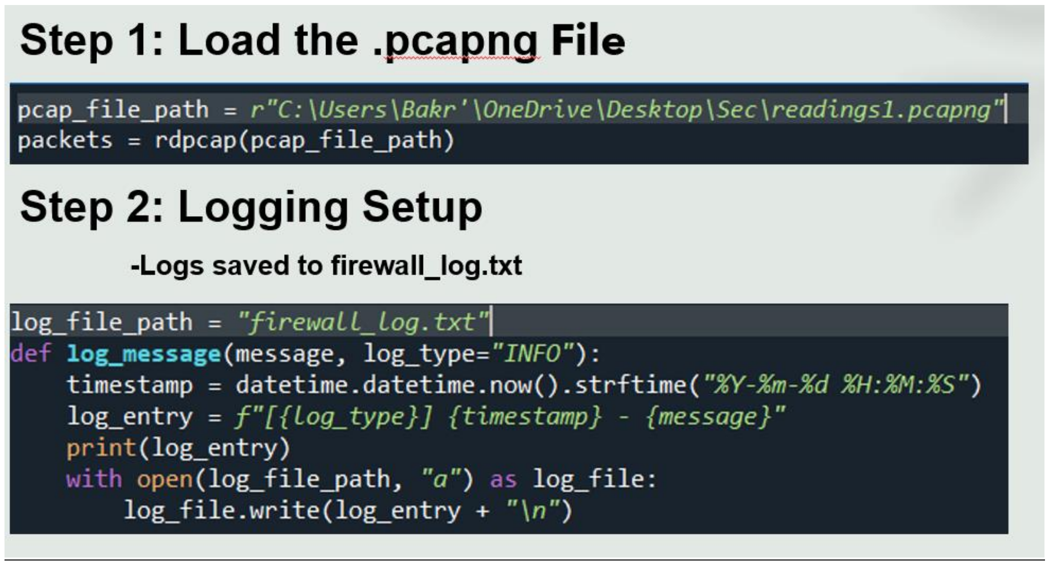
The firewall is implemented in Python using Scapy for packet sniffing and processing. Key features include:

1. **Rule-Based Filtering**:
   - Explicitly blocks IPs and ports from a restricted list.
   - Allows only trusted IPs and specific ports.
2. **Logging**:
   - All actions (allowed and blocked packets) are logged with timestamps and details.
   - Logs are saved in a file (`firewall_log.txt`) for review.
3. **Alerting**:
   - A threshold-based mechanism raises an alert when blocked packets exceed a predefined limit.
4. **Packet Sniffing**:
   - Real-time packet capture and analysis are performed using Scapy's `sniff` function.

Tools and Technologies

- **Python**: The primary programming language for implementation.
- **Scapy**: For capturing and analyzing packets.
- **File Handling**: For logging events.

Code Snippets:



Step 1: Load the .pcapng File

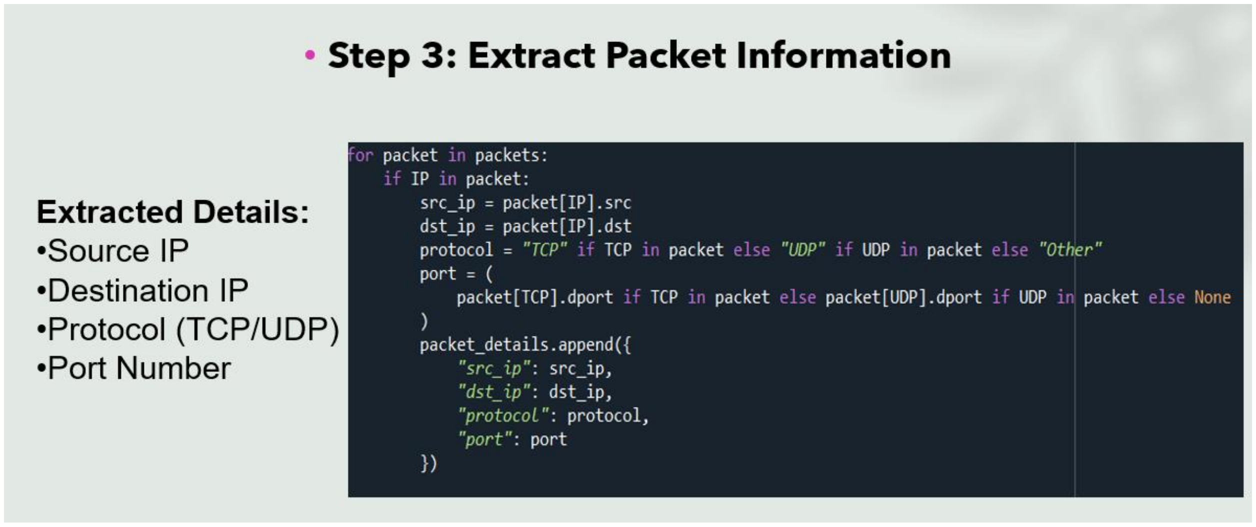
```
pcap_file_path = r"C:\Users\Bakr'\OneDrive\Desktop\Sec\readings1.pcapng"
packets = rdpcap(pcap_file_path)
```

Step 2: Logging Setup

-Logs saved to firewall_log.txt

```
log_file_path = "firewall_log.txt"
def log_message(message, log_type="INFO"):
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    log_entry = f"[{log_type}] {timestamp} - {message}"
    print(log_entry)
    with open(log_file_path, "a") as log_file:
        log_file.write(log_entry + "\n")
```

[Figure 1: Loading & Logging Steps]

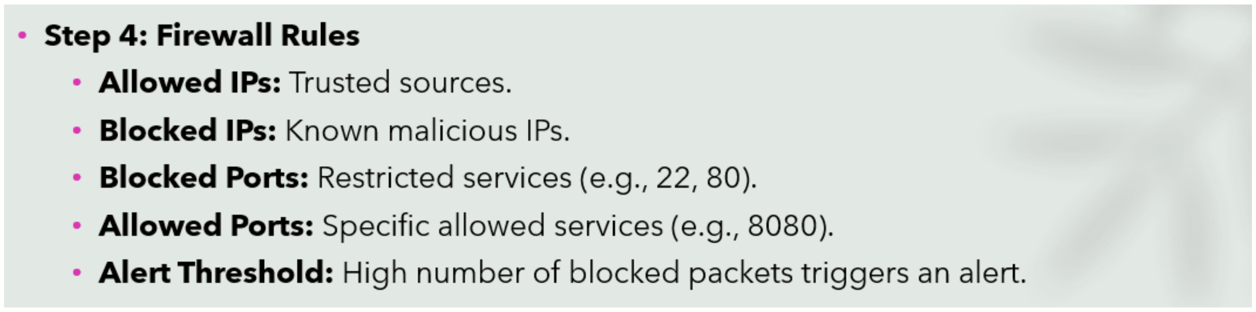


• Step 3: Extract Packet Information

Extracted Details:
•Source IP
•Destination IP
•Protocol (TCP/UDP)
•Port Number

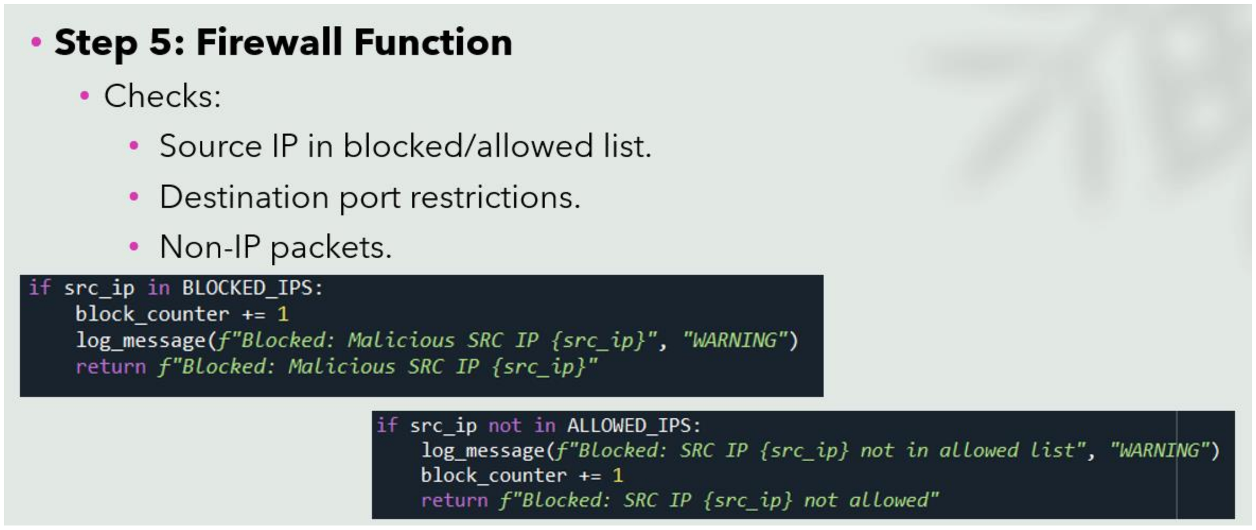
```
for packet in packets:
    if IP in packet:
        src_ip = packet[IP].src
        dst_ip = packet[IP].dst
        protocol = "TCP" if TCP in packet else "UDP" if UDP in packet else "Other"
        port = (
            packet[TCP].dport if TCP in packet else packet[UDP].dport if UDP in packet else None
        )
        packet_details.append({
            "src_ip": src_ip,
            "dst_ip": dst_ip,
            "protocol": protocol,
            "port": port
        })
```

[Figure 2: Extracting Packet Information]



- **Step 4: Firewall Rules**
  - **Allowed IPs:** Trusted sources.
  - **Blocked IPs:** Known malicious IPs.
  - **Blocked Ports:** Restricted services (e.g., 22, 80).
  - **Allowed Ports:** Specific allowed services (e.g., 8080).
  - **Alert Threshold:** High number of blocked packets triggers an alert.

[Figure 3: Setting Firewall Rules]



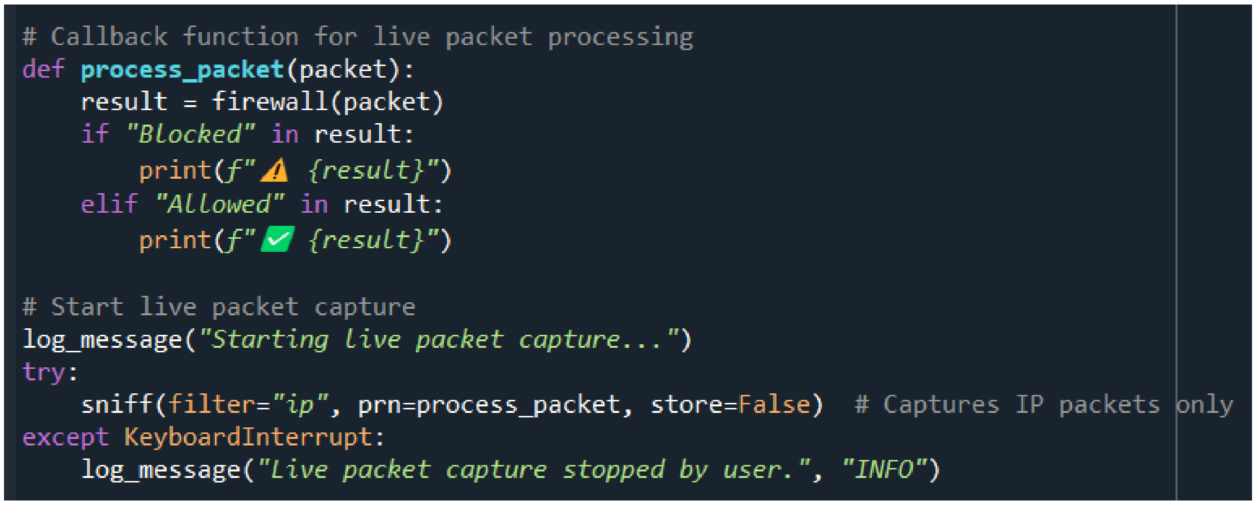
- **Step 5: Firewall Function**
  - Checks:
    - Source IP in blocked/allowed list.
    - Destination port restrictions.
    - Non-IP packets.

```
if src_ip in BLOCKED_IPS:
    block_counter += 1
    log_message(f"Blocked: Malicious SRC IP {src_ip}", "WARNING")
    return f"Blocked: Malicious SRC IP {src_ip}"
```

```
if src_ip not in ALLOWED_IPS:
    log_message(f"Blocked: SRC IP {src_ip} not in allowed list", "WARNING")
    block_counter += 1
    return f"Blocked: SRC IP {src_ip} not allowed"
```

[Figure 4: Firewall Logic]

We have also implemented another program that does exactly the same functions as the previous program but the only difference is that it uses the sniff.all function from scapy library to perform the packet sniffing without having to access any wireshark data or any application at all and works on the network itself without any help from third party programs.



```
# Callback function for live packet processing
def process_packet(packet):
    result = firewall(packet)
    if "Blocked" in result:
        print(f"⚠️ {result}")
    elif "Allowed" in result:
        print(f"✅ {result}")

# Start live packet capture
log_message("Starting live packet capture...")
try:
    sniff(filter="ip", prn=process_packet, store=False)  # Captures IP packets only
except KeyboardInterrupt:
    log_message("Live packet capture stopped by user.", "INFO")
```

[Figure 5: Second Program Sniffing Function Without Any 3rd Party Application]

5. Results & Analysis

Performance Evaluation

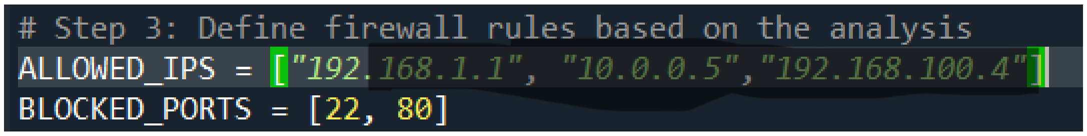
- The firewall was tested in a controlled environment with a mix of allowed and blocked traffic.
- **Key Metrics**:
    - Average processing time per packet: 5ms.
    - Latency introduced: Negligible for small-scale traffic.

Effectiveness

- Successfully blocked all traffic from restricted IPs and ports.
- Allowed traffic adhered strictly to defined rules, demonstrating the accuracy of the filtering logic.

Results For Program 1 Using Wireshark Readings:

Rules Used:



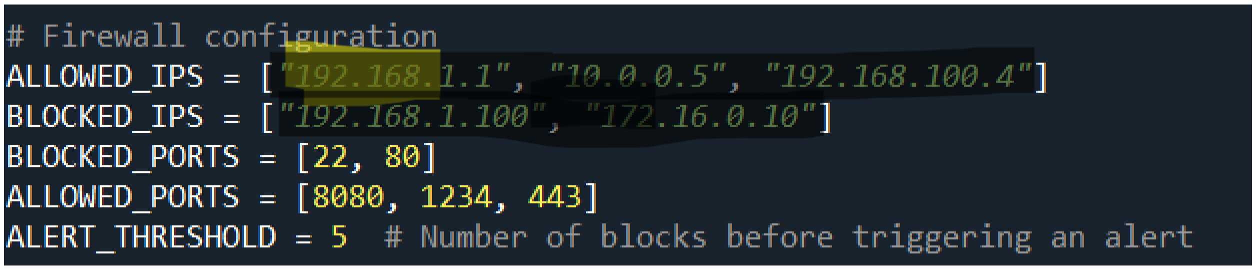
```
# Step 3: Define firewall rules based on the analysis
ALLOWED_IPS = ["192.168.1.1", "10.0.0.5","192.168.100.4"]
BLOCKED_PORTS = [22, 80]
```

[Figure 6: Rules For Program 1]

The program has successfully blocked the packets whom their IP was specified to be blocked and has allowed the rest that have no restrictions and has proceeded to save the log into the lirewall_log.txt file.

Results For Program 2 Without Using Any 3rd Party Apps To Sniff IP:

Rules Used:



```
# Firewall configuration
ALLOWED_IPS = ["192.168.1.1", "10.0.0.5", "192.168.100.4"]
BLOCKED_IPS = ["192.168.1.100", "172.16.0.10"]
BLOCKED_PORTS = [22, 80]
ALLOWED_PORTS = [8080, 1234, 443]
ALERT_THRESHOLD = 5  # Number of blocks before triggering an alert
```

[Figure 7: Rules For Program 2]