Lightweight Python Personal Firewall

This project implements a customizable personal firewall in Python, leveraging **Scapy** for packet sniffing and rule-based filtering. It provides the foundation for building a lightweight firewall with the ability to capture network packets, apply filtering rules, log suspicious activity, and enforce system-level security policies using **iptables**. Optionally, a simple **Tkinter** GUI can be used for real-time monitoring and management of the firewall.

Key Features:

- **Packet Sniffing**: Capture and analyze real-time network traffic using Scapy.
- **Rule-Based Filtering**: Define custom rules to allow/block specific IPs, ports, and protocols.
- **Logging**: Log suspicious or denied packets for audit purposes.
- **iptables Integration**: Optionally block malicious IPs at the OS level using iptables.
- **GUI Monitoring**: Optional Tkinter-based graphical interface for monitoring and managing the firewall in real-time.

Why this Project?

This firewall offers a lightweight, educational tool to understand networking, packet manipulation, and basic security concepts. It is highly customizable and can be adapted for personal use or further development.

Tech Stack:

- **Python**
- **Scapy**
- **iptables (optional)**
- **Tkinter (optional)**

Perfect for learning network security and Python-based firewall development!

Core Components:

- **Packet Sniffing**: Capturing and inspecting real-time packets.
- **Rule-Based Filtering**: Defining rules to allow or block traffic.
- **Logging Suspicious Packets**: Keeping track of denied packets for audit purposes.
- **System-Level Rule Enforcement**: Using iptables for OS-level packet blocking (optional).
- **Optional GUI Monitoring**: Using Tkinter for a basic graphical interface.

Step 1: Packet Sniffing

Let’s make the packet sniffing section a bit clearer.

---

1. Packet Sniffing (Scapy)

Scapy is a powerful Python library for network packet manipulation. In this step, we'll use Scapy to capture both incoming and outgoing packets in real-time. The captured packets will be processed using a custom handler.

Example:

```
from scapy.all import sniff


def packet_handler(packet):
    # Process or filter packet as needed
    pass


# Start sniffing
sniff(prn=packet_handler, store=0)
```

`sniff`: This function captures packets and passes them to the handler (`packet_handler`).

Step 2: Rule-Based Filtering

Clarifying how rules can be structured and applied in Python.

---

2. Rule-Based Filtering

In this step, we will define the firewall's rules using Python data structures like lists or dictionaries. These rules will specify which IPs, ports, or protocols are allowed or blocked.

Example:

```
allowed_ips = ['192.168.1.1', '127.0.0.1']
blocked_ports = [22, 80]


def is_allowed(packet):
    if packet.haslayer("IP"):
        src_ip = packet["IP"].src
        # Check if source IP is in the allowed list
        if src_ip in allowed_ips:
            return True
```

 # Add additional port/protocol checks as needed

return False

`allowed_ips`: List of IP addresses that are allowed.

- `blocked_ports`: List of ports that should be blocked (such as SSH or HTTP).
- The `is_allowed` function checks if a packet meets the rule conditions and returns `True` or `False` accordingly.

The packet handler can then use these rules to decide whether to allow or block each packet.

Step 3: Logging Suspicious Packets

Improving the explanation for logging.

---

3. Logging Suspicious Packets

It's essential to log all denied or suspicious packets for auditing purposes. This will allow us to track potential attacks or violations.

Example:

import logging

 # Set up logging

logging.basicConfig(filename='firewall.log', level=logging.INFO)

def packet_handler(packet):

if not is_allowed(packet):

logging.info(f"Blocked: {packet.summary()}")

This code logs the summary of any blocked packet to a file called `firewall.log`.

- `logging.info`: Logs each blocked packet with details for auditing.

We can extend the logging functionality to include alerts or notifications if needed.

Step 5: System-Level Rule Enforcement (iptables, Optional)

Clarifying the optional use of iptables.

---

4. System-Level Rule Enforcement (iptables)

If we need actual packet blocking at the OS level, we can invoke `iptables` rules from Python. This can be done using the `subprocess` library to run commands directly in the system shell.

Example (Blocking an IP):

```
import subprocess

# Block an IP using iptables
subprocess.run(['sudo', 'iptables', '-A', 'INPUT', '-s', 'BLOCKED_IP', '-j', 'DROP'])
```

This code adds an iptables rule to block an IP address.

- Make sure to replace `BLOCKED_IP` with the actual IP we want to block.

This allows for automated defense actions based on the rules we set up in Python.

Step 6: Optional GUI Monitoring (Tkinter)

A more concise explanation of Tkinter's role.

---

5. Optional GUI Monitoring (Tkinter)

While optional, Tkinter can be used to create a simple graphical interface for real-time monitoring of the firewall's activity. We can display logs, manage rules, or interact with the firewall using buttons and other UI components.

Example:

```
import tkinter as tk

# Create a window
window = tk.Tk()

# Add widgets (labels, buttons, etc.) for monitoring/logs
window.mainloop()
```

`Tkinter` provides the tools to build a basic GUI for your firewall, but it’s not necessary for core functionality.

$$