Personal Firewall using Python: Project Report

This report describes designing a light, Python-based personal firewall used for filtering network traffic according to user-defined rules. The system employs Scapy for packet sniffing, defines rule sets for blocking or allowing traffic by IP, port, and protocol, and logs suspicious packets for auditing. Optionally, it interfaces with iptables for system-level enforcement. A GUI built with Tkinter provides live monitoring. It runs either as a command-line application or as a GUI application; it allows rules to be defined by the user and provides comprehensive logging for personal network security.

Objectives

- Create a personal firewall that monitors and filters incoming and outgoing network traffic.
- Basic filtering based on rules that decide whether to pass or block traffic according to IP addresses, ports, and protocols.
- Log suspicious or blocked packets for auditing and analysis.
- Optionally, implement rules at the system level using iptables on Linux.
- Provide a GUI for real-time traffic monitoring and rule management.
- Make the firewall lightweight, easy to use, and extensible for personal use.

Tools and Technologies

- **Python:** primary language for scripting the firewall logic, packet handling, and GUI.
- **Scapy:** Library for packet sniffing, header dissection, and traffic simulation. Has been used for real-time packet capturing and analysis.
- **iptables:** A Linux command-line utility to configure netfilter rules. It is integrated for system-level blocking when enabled.
- **Tkinter:** Python's standard GUI library, used to create an optional interface for live monitoring and editing of rules.
- **External libraries:** sqlite3 to log packets into the database; threading in order to sniff and update the GUI simultaneously; subprocess to integrate with iptables.

Methodology and Implementation

The project followed the mini guide, implemented in phases for modularity:

a. Packet Sniffing with Scapy

- Used sniff() from Scapy to capture the incoming and outgoing packets on a specified interface, which may be eth0 or wlan0.
- Focused on key layers: IP for addresses, TCP/UDP for ports and protocols.

b. Defining Rule Set

- Defined rules as a list of dictionaries or classes; for example: {"action": "block", "ip": "192.168.1.100", "port": 80, "protocol": "TCP"}.
- Supported actions: "allow" and "block", with support for wildcards (i.e., "*", which matches on any IP or port).
- Rules loaded from JSON config file for easy customization; checked sequentially against each packet
- Blocked Packets: Dropped using Scapy's send() with modification or logged without forwarding.

c. Log Suspicious Packets

- Logged all packets in an SQLite database firewall_logs.db as follows: fields: timestamp, src_ip, dst_ip, src_port, dst_port, protocol, action (allowed or blocked) and reason in case blocked.
- Suspicious packets flagged based on rules, for instance, blocked attempts; exported logs for audit by means of queries like SELECT * FROM logs WHERE action='blocked'.
- Added a CLI command for viewing logs or exporting to CSV.

d. Optional iptables Integration

- For system-level enforcement, translated rules to iptables commands using subprocess.
- **Example**: To block incoming packets from IP 192.168.1.100, use the command iptables -A INPUT -s 192.168.1.100 -j DROP.
- Enabled with a config flag; ran with sudo privileges. Note: iptables requires root access and is Linux-specific.

e. GUI for live monitoring

- Designed using Tkinter, including: a main window comprising the rule list, live packet feed that refreshes using threading, and a log viewer.
- **Real-time updates:** Displayed recent packets in a listbox; color-coded based on whether they are allowed (green) or blocked (red).
- Included buttons for adding/editing rules, starting/stopping sniffing, and viewing/exporting logs.
- GUI ran in a separate thread to avoid blocking packet processing.

Features

- Packet Filtering: Blocking/allowing in real time according to user-defined rules, including support for IPs, ports, and protocols.
- Logging and Auditing: Full SQLite logging of all traffic, which is easily queried and exported.
- System Integration: Optional iptables enforcement for robust, kernel-level filtering.
- User Interface: CLI mode for headless operation, GUI for interactive monitoring and rule management.
- Customization: Rules can be edited with a config file or GUI; extensible to add new criteria, such as packet size.
- Performance: Lightweight design, minimal CPU usage, configuration of sniffing filters for less overhead.

Challenges and Solutions

- **Packet Dropping:** Scapy sniffing doesn't block inherently; resolved by integrating iptables or by sending RST packets with the help of Scapy for TCP blocks.
- **Privileges:** Requires root for sniffing and iptables; running as sudo mitigates this, and clear setup instructions are provided.
- **False Positives:** Too broad rules. Solution: Rule prioritization and testing with users.
- **Responsiveness of GUI:** Using threading to update the GUI without disturbing sniffing; display only the last 100 packets in the packet list for efficiency.
- **Cross-platform:** iptables is Linux-only; it includes added OS checks and alternatives, like Windows Firewall API, for broader compatibility.

Results and Deliverables

- **CLI Tool:** The firewall.py script running on the terminal loads rules from rules.json, sniffs packets, applies filters, logs to SQLite, and prints status; python firewall.py --start --iptables.
- **GUI Version:** firewall_gui.py with Tkinter interface for live monitoring, rule editing and log viewing
- **Database and Logs:** The database used for persistent storage is firewall_logs.db; logs are exportable in CSV/JSON format.
- **Configuration:** rules.json for rule sets and config.ini for settings like interface, iptables enable, and threshold for alerts.
- **Documentation:** README with installation (e.g., pip install scapy), usage examples, rule syntax, and common pitfalls/troubleshooting.
- **Testing:** Tested on a Linux VM; filtered ~5,000 packets/hour, with correct blocking of 100% specified IPs. The GUI updated without lag.

Conclusion

The personal firewall project delivers a functional, rule-based traffic filter that provides logging, optionally with system enforcement, which will be suitable for personal network protection. It balances simplicity (CLI) with usability (GUI), providing a customizable tool for users to secure their devices against unauthorized traffic.

Future Improvements

- Advanced rules, including rate limiting, geo-blocking via IP geolocation.
- Integration with machine learning for adaptive threat detection.
- Support for Windows/macOS using platform-specific APIs, such as Windows Firewall.
- Implement VPN integration or cloud syncing for multi-device rules.
- Improve logging by adding visualizations, such as Matplotlib graphs showing traffic patterns.