Below is a step-by-step guide—in plain, human language—that breaks your Personal Firewall project into 12 easy, manageable steps. This plan is designed to help you implement a lightweight firewall using Python, Scapy, optional iptables integration, and even an optional Tkinter GUI for live monitoring. Each step has clear details for writing your code, testing, and documenting your work.

Step 1: Understand the Project and Define Your Goals

- What to Do: Write down what your firewall must achieve. For example, decide exactly which traffic (by IP, port, or protocol) will be allowed or blocked.

- Goals:

- Sniff incoming/outgoing packets using Scapy.

- Define and enforce rule sets.

- Log any suspicious packets for later review.

- (Optional) Use iptables to enforce system-level rules.

- (Optional) Create a GUI for real-time monitoring.

- Why: You need a clear game plan before writing a single line of code. This ensures your project is focused and helps when you explain it in interviews.

Step 2: Set Up Your Development Environment

- What to Do:

- Create a new folder or GitHub repository for the project.

- Set up a Python virtual environment (python -m venv env).

- Install necessary libraries:

pip install scapy

- (Tkinter is usually pre-installed on most systems, but verify it on your setup.)

- Why: Keeping your project organized and setting up a clean environment helps avoid version conflicts and makes later debugging easier.

Step 3: Write a Basic Packet Sniffer with Scapy

- What to Do:

- Create a Python script (e.g., sniffer.py) that uses Scapy's sniff() function.

- Print basic details—source IP, destination IP, ports, protocol—for every packet.

- Sample Snippet:

```
from scapy.all import sniff


def packet_callback(pkt):
```

```
    print(f"Packet: {pkt.summary()}")


print("Starting packet capture...")

sniff(prn=packet_callback, store=0)
```

- Why: This step verifies that you can successfully monitor network traffic, which is the core of your personal firewall.

Step 4: Define Your Rule Sets

- What to Do:

- Decide what constitutes allowed or blocked traffic.

- Create a simple data structure (like a dictionary or list) to maintain rules.

- Example:

```
# Define rules in a simple dictionary format

rules = {

    "blocked_ips": ["192.168.1.100"],

    "blocked_ports": [23],  # Telnet

    "allowed_protocols": ["TCP", "UDP"]

}
```

- Why: Having a clear and easy-to-maintain rule set makes it straightforward to check each captured packet against your security policies.

Step 5: Implement Rule Checking in Your Sniffer

- What to Do:

- Write a function that examines each packet's details and compares them with your defined rules.

- Decide actions: for example, flag a packet if it comes from a blocked IP or uses a blocked port.

- Ideas:

```
def check_packet(pkt):

    ip = pkt.getlayer("IP")

    if ip:

        if ip.src in rules["blocked_ips"]:

            return "BLOCKED: Source IP blocked."

        if pkt.haslayer("TCP") or pkt.haslayer("UDP"):

            sport = pkt.sport

            if sport in rules["blocked_ports"]:
```

```
        return "BLOCKED: Port blocked."
    return "ALLOWED"
```

☐☐- Why: Integrating rule checking ensures that every packet is analyzed in real time, forming the core logic of your firewall.

Step 6: Add Logging for Suspicious Packets

- What to Do:

- Use Python's logging module to record details of any packet that does not meet your rule criteria.

- Example:

```
import logging

logging.basicConfig(filename='firewall.log', level=logging.INFO, format='%(asctime)s - %(message)s')

def log_packet(pkt, reason):
    logging.info(f"Packet: {pkt.summary()} - Reason: {reason}")
```

☐☐- Why: Logging is critical for later auditing and helps you verify that your firewall is filtering packets correctly.

Step 7: (Optional) Integrate iptables for System-Level Enforcement

- What to Do:

- Write functions that use Python's subprocess module to run iptables commands.

- Test these functions carefully, ideally on a VM or in a safe lab environment.

- Sample Code:

```
import subprocess

def enforce_iptables(rule_command):
    subprocess.run(rule_command, shell=True)

# Example to block an IP:
enforce_iptables("sudo iptables -A INPUT -s 192.168.1.100 -j DROP")
```

☐☐- Why: This adds an extra layer of security by not just monitoring traffic but actively enforcing rules at the OS level.

Step 8: Create a Command-Line Interface (CLI)

- What to Do:

- Develop a simple CLI to allow yourself (or users) to add, remove, or view rules.

- Use Python’s built-in input() function or libraries like argparse for more options.

- Simple Approach:

```
def manage_rules():
    print("1. View rules\n2. Add rule\n3. Remove rule")
    choice = input("Enter your choice: ")
    # Process the user's choice accordingly...
```

□□- Why: A CLI provides flexibility to modify the firewall behavior without changing the code.

Step 9: Build a Basic GUI with Tkinter (Optional)

- What to Do:

- Create a new Python file (e.g., firewall_gui.py) that opens a window displaying live logs and current rules.

- Design simple buttons for adding/removing rules and a text area for live updates.

- Basic Tkinter Structure:

```
import tkinter as tk

def refresh_logs():
    # Code to update log display
    pass

root = tk.Tk()
root.title("Personal Firewall Monitor")
log_text = tk.Text(root, height=20, width=80)
log_text.pack()
refresh_button = tk.Button(root, text="Refresh Logs", command=refresh_logs)
refresh_button.pack()

root.mainloop()
```

□□□□- Why: A GUI can make your firewall friendlier to use, especially during presentations or demo sessions.

Step 10: Integrate All Components and Run Thorough Testing

- What to Do:

- Combine your packet sniffing, rule checking, logging, CLI/GUI, and (if implemented) iptables modules.

- Conduct tests in different environments and simulate various network scenarios to ensure your firewall responds as expected.

- Testing Tips:

- Use your local network to simulate attacks.

- Check if blocked traffic doesn't appear in the log while allowed traffic does.

- Why: Testing is key to verifying that your firewall is robust and reliable.

Step 11: Prepare Your Project Documentation and Report

- What to Do:

- Write your 1–2 page final report in PDF format. Keep it concise and focused.

- Sections to include:

- Introduction: What is a personal firewall and why it's important.

- Abstract: A short summary of your project.

- Tools Used: Python, Scapy, iptables (optionally), Tkinter.

- Steps Involved: Outline your 12-step process briefly.

- Conclusion: Discuss what you learned and potential enhancements.

- Why: A clear report is essential for interviews and evaluations, showing that you not only built a working solution but also understand every part of it.

Step 12: Finalize and Submit Your Project

- What to Do:

- Do a last round of testing and code review. Ensure your GitHub repository is updated with all final code and documentation.

- Create the final report PDF and ensure it does not exceed 2 pages.

- Prepare the submission package as specified (GitHub link plus the report).

- Why: A thorough final check ensures everything works as intended and meets all project guidelines.

This 12-step plan breaks down the project into clear, simple tasks that any developer can follow. Each step builds on the previous one, ensuring steady progress toward a complete, well-documented, and functional personal firewall. Happy coding, and enjoy refining your project as you move through these steps!