```
import tkinter as tk
from tkinter import filedialog, messagebox
import platform
import json
import logging
import os
import shutil
import threading
import time
from datetime import datetime
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas

# --- Logging Setup ---
log_path = os.path.join(os.getcwd(), 'firewall_compliance_tool.log')
for handler in logging.root.handlers[:]:
    logging.root.removeHandler(handler)

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

def log_event(message):
    logging.info(message)

last_pdf_report = None
monitoring_enabled = False

# --- Rule Parsers ---
def parse_windows_rules(file_path):
    try:
        with open(file_path, "r") as f:
            rules = json.load(f)
        if isinstance(rules, dict):
            rules = [rules]
    except Exception as e:
        raise ValueError("Invalid JSON file: " + str(e))

    compliance = {
        "RDP_Block": True,
        "SSH_Block": True,
        "Default_Inbound_Deny": True,
        "No_ANY_ANY": True,
    }

    suggestions = {
        "RDP_Block": "Block inbound connections on Remote Desktop port 3389.",
        "SSH_Block": "Block inbound connections on SSH port 22.",
        "Default_Inbound_Deny": "Set default inbound policy to deny.",
        "No_ANY_ANY": "Avoid rules allowing all traffic from any source to any
destination.",
    }

    for rule in rules:
        if "RemotePort" in rule and str(rule["RemotePort"]) == "3389":
            compliance["RDP_Block"] = False
        if "RemotePort" in rule and str(rule["RemotePort"]) == "22":
```

```
            compliance["SSH_Block"] = False
        if rule.get("Direction") == "Inbound" and rule.get("Action") == "Allow":
            compliance["Default_Inbound_Deny"] = False
        if rule.get("Direction") == "Inbound" and rule.get("RemoteAddress") ==
"Any" and rule.get("LocalAddress") == "Any":
            compliance["No_ANY_ANY"] = False

    log_event("Parsed Windows rules.")
    return compliance, suggestions

def parse_linux_rules(file_path):
    try:
        with open(file_path, "r") as f:
            lines = f.readlines()
    except Exception as e:
        raise ValueError("Invalid text file: " + str(e))

    compliance = {
        "RDP_Block": True,
        "SSH_Block": True,
        "Default_Inbound_Deny": True,
        "No_ANY_ANY": True,
    }

    suggestions = {
        "RDP_Block": "Block inbound connections on Remote Desktop port 3389.",
        "SSH_Block": "Block inbound connections on SSH port 22.",
        "Default_Inbound_Deny": "Set default INPUT policy to DROP.",
        "No_ANY_ANY": "Avoid rules like '-A INPUT -s 0.0.0.0/0 -j ACCEPT'.",
    }

    for line in lines:
        line = line.lower()
        if "3389" in line and "accept" in line:
            compliance["RDP_Block"] = False
        if "22" in line and "accept" in line:
            compliance["SSH_Block"] = False
        if "-s 0.0.0.0/0" in line and "-j accept" in line:
            compliance["No_ANY_ANY"] = False
        if "input" in line and "-j accept" in line:
            compliance["Default_Inbound_Deny"] = False

    log_event("Parsed Linux rules.")
    return compliance, suggestions

# --- PDF Report Generator ---
def generate_pdf_report(compliance_data, suggestions, os_type, source_file):
    global last_pdf_report
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    file_name = f"firewall_compliance_report_{os_type}_{timestamp}.pdf"
    c = canvas.Canvas(file_name, pagesize=A4)
    width, height = A4

    c.setFont("Helvetica-Bold", 16)
    c.drawString(50, height - 50, f"Firewall Compliance Report ({os_type})")
    c.setFont("Helvetica", 12)
    c.drawString(50, height - 80, f"Generated: {datetime.now().strftime('%Y-%m-%d
%H:%M:%S')}")
    c.drawString(50, height - 100, f"Source File: {source_file}")
```

```
    y = height - 140
    all_passed = True

    for rule, status in compliance_data.items():
        status_str = "PASS" if status else "FAIL"
        c.drawString(50, y, f"{rule.replace('_', ' ')}: {status_str}")
        if not status and rule in suggestions:
            c.setFont("Helvetica-Oblique", 10)
            c.drawString(60, y - 15, f"Suggestion: {suggestions[rule]}")
            c.setFont("Helvetica", 12)
            y -= 15
            all_passed = False
        y -= 20

    if all_passed:
        c.setFont("Helvetica-Bold", 12)
        c.drawString(50, y, "System Status: COMPLIANT")
        y -= 20
        c.drawString(50, y, "All checks passed. No remediation required.")

    c.save()
    last_pdf_report = file_name
    log_event(f"Generated PDF report: {file_name}")
    return file_name, all_passed

# --- GUI Functions ---
def browse_file():
    file_path = filedialog.askopenfilename(title="Select Firewall Rules File")
    if file_path:
        entry_file.delete(0, tk.END)
        entry_file.insert(0, file_path)

def run_compliance_check():
    file_path = entry_file.get()
    selected_os = os_var.get().lower()

    if not file_path or not selected_os:
        messagebox.showerror("Error", "Please select both OS and file.")
        return

    try:
        if selected_os == "windows":
            compliance, suggestions = parse_windows_rules(file_path)
        else:
            compliance, suggestions = parse_linux_rules(file_path)

        report_file, all_passed = generate_pdf_report(compliance, suggestions,
selected_os, file_path)

        status = "COMPLIANT" if all_passed else "NON-COMPLIANT"
        messagebox.showinfo("Compliance Result", f"Status: {status}\nReport:
{report_file}")

    except Exception as e:
        log_event(f"Compliance Check Failed: {str(e)}")
        messagebox.showerror("Error", f"Failed: {str(e)}")

def export_pdf():
```

```
    if not last_pdf_report or not os.path.exists(last_pdf_report):
        messagebox.showerror("Error", "No report to export.")
        return
    dest = filedialog.asksaveasfilename(defaultextension=".pdf")
    if dest:
        shutil.copy(last_pdf_report, dest)
        messagebox.showinfo("Success", f"Report saved to:\n{dest}")

def export_log():
    if not os.path.exists(log_path):
        messagebox.showerror("Error", "Log file missing.")
        return
    dest = filedialog.asksaveasfilename(defaultextension=".log")
    if dest:
        shutil.copy(log_path, dest)
        messagebox.showinfo("Success", f"Log saved to:\n{dest}")

def monitor_background():
    while monitoring_enabled:
        run_compliance_check()
        log_event("Real-time monitoring check executed.")
        time.sleep(60)  # 1 minute delay

def toggle_monitoring():
    global monitoring_enabled
    if not monitoring_enabled:
        monitoring_enabled = True
        threading.Thread(target=monitor_background, daemon=True).start()
        messagebox.showinfo("Monitoring", "Real-time monitoring started (every 60
seconds).")
    else:
        monitoring_enabled = False
        messagebox.showinfo("Monitoring", "Monitoring stopped.")

# --- GUI Setup ---
root = tk.Tk()
root.title("Firewall Compliance Self-Assessment Tool")

tk.Label(root, text="Select OS:").pack()
os_var = tk.StringVar()
tk.Radiobutton(root, text="Windows", variable=os_var, value="Windows").pack()
tk.Radiobutton(root, text="Linux", variable=os_var, value="Linux").pack()

tk.Label(root, text="Firewall Rules File:").pack()
entry_file = tk.Entry(root, width=50)
entry_file.pack()
tk.Button(root, text="Browse", command=browse_file).pack(pady=5)

tk.Button(root, text="Run Compliance Check",
command=run_compliance_check).pack(pady=10)
tk.Button(root, text="Export PDF Report", command=export_pdf).pack(pady=2)
tk.Button(root, text="Export Log File", command=export_log).pack(pady=2)
tk.Button(root, text="Toggle Real-Time Monitoring",
command=toggle_monitoring).pack(pady=10)

root.mainloop()
```