1. Introduction

In an era of escalating cyber threats and a shortage of skilled penetration testers, organizations need automated, AI-powered solutions that accelerate vulnerability discovery reporting. sentraX is a modular, end-to-end network pen testing workflow that leverages established scanning tools and GPT-4 to guide decision-making, generate remediation steps, and produce concise, actionable reports. This document details the architecture, setup, and usage of sentraX, aligned to the hackathon's challenge: automating the pen testing lifecycle with LLM integration, improving efficiency, accuracy, and usability.

2. Objectives and Scope

The sentraX solution fulfils the following objectives:

- **Automation**
  Automate all six pen testing phases—reconnaissance, scanning, exploitation, maintaining access, covering tracks, and reporting—via a single command or individual phase triggers.
- **LLM Integration**
  Use GPT-4 to analyse raw data from tools (e.g., Nmap, Nessus), recommend exploits or remediation, and craft human-readable summaries and reports.
- **Efficiency**
  Minimize manual intervention, reduce time-to-insight from days to minutes, and support batch processing of multiple targets.
- **User-Friendly Interface**
  Provide a React/Tailwind single-page application with intuitive tabs, real-time logs, progress indicators, and one-click report generation.
- **Modularity**
  Architect each phase as a standalone Python module. New tools or custom workflows can be added by dropping scripts into the backend/ pen test folder.

3. System Architecture

Layer	Technology	Responsibility
Frontend	React 18, TypeScript, Tailwind CSS	User interface for workflow selection, real-time logs, AI anal and report export
Workflow Engine	n8n	Orchestration of tasks and sequential execution of backend calls
Backend API	FastAPI, Python 3.10+	Exposes endpoints for each pen test phase; handles LLM cal returns JSON logs
Reconnaissance Module	Shodan API, Nessus, GPT-4	Collects host data, vulnerability scan results; generates AI summaries

Layer	Technology	Responsibility
Scanning Module	python-nmap, GPT-4	Runs port/service discovery and OS fingerprinting; produces driven insights
Exploitation Module	pymetasploit3, Metasploit RPC	Automates exploit selection and execution based on GPT recommendations
Persistence & Cleanup	GPT-4, session commands	Suggests and executes post-exploit persistence and track-covering actions
Reporting Module	GPT-4, PDF/HTML generator	Generates executive and technical reports, risk matrices, remediation roadmaps

Add overall system topology diagram here

4. Detailed Module Descriptions

4.1. Reconnaissance (backend/pentest/recon.py)

Function do_recon(target_ip, logger) → dict

1. **Automated Data Collection**
    - Query Shodan for open ports, services, and metadata
    - (Optional) Run Nessus scan and parse plugin results
2. **AI Summarization**
    - Construct prompt:
      "Provide a reconnaissance summary for IP {target_ip} with the following data: {Shodan/Nessus raw output}."
    - Call ask_gpt(prompt) for a concise, prioritized narrative of potential vulnerabilities
3. **Output**
   Returns { shodan: {…}, nessus: {…}, gpt_summary: "…" }

Add screenshot of Recon UI panel here

4.2. Scanning (backend/pentest/scan.py)

Function do_scan(target_ip, recon_info, logger) → dict

1. **Nmap Execution**
    - Arguments: -sV -O -p- for comprehensive port, service, and OS detection
2. **AI-Driven Analysis**

    - Prompt:
      “Analyse this Nmap output for IP {target_ip}: {raw_scan_data}. Recommend focused scan options or next steps.”
    - GPT-4 returns prioritized targets (e.g., high-risk ports) and parameter suggestions

3. **Output**
   Returns { raw_scan: <PortScanner data>, analysis: "…" }

4.3. Exploitation (backend/pentest/exploit.py)

Function do_exploit(target_ip, scan_data, logger) → dict

1. **Exploit Recommendation**
    - Prompt:
      “Based on scan analysis: {scan_data.analysis}, suggest Metasploit module and parameters.”
2. **Automated Execution**
    - Parse GPT module name and RPORT
    - MsfRpcClient uses module, sets RHOSTS/RPORT, and executes payload
3. **Session Management**
    - Collect active session IDs and metadata
4. **Output**
   Returns { module: "exploit/...", params: {…}, sessions: […], gpt_details: "…" }

4.4. Persistence & Cleanup

Functions do_persist(...), do_cleanup(...)

1. **Persistence**
    - Prompt: “Recommend persistence mechanisms for sessions {session_ids}.”
    - Execute commands (e.g., cron jobs, registry tweaks) via active sessions
2. **Cleanup**
    - Prompt: “Recommend commands to cover tracks on {target_ip}.”
    - Execute file deletion, log modification, timestamp resets
3. **Outputs**
   Returns action logs and GPT-generated rationale

4.5. Reporting (backend/pentest/report.py)

Function do_report(ip, recon, scan, exploit, persist, cleanup, logger) → str

1. **Report Composition**
    - Aggregate all phase outputs

    - Prompt:
      "Write an executive and technical pentest report for {ip} including findings, risk assessment, and remediation steps."

2. **Export**
    - Generate PDF and HTML versions with formatted tables and risk matrices
3. **Output**
   Returns raw report content; UI triggers file download

Add sample report excerpt screenshot here

5. Workflow Automation with n8n

- **n8n** orchestrates frontend-triggered API calls:
    1. POST /api/recon → wait for completion
    2. POST /api/scan → …
    3. Sequentially invoke exploit, persist, cleanup, report
- **Error Handling & Retries**
    - n8n workflows include conditional checks on API responses
    - Alert via Slack or email on failures

6. Frontend Overview

Tab	Functionality
Dashboard	Input targets, select "Full Workflow" or individual phases, view terminal-style real-time logs
AI Workflow	Choose pre-built templates (Web App, Stealth, Comprehensive), customize GPT prompts, set priority
Phases	Trigger and monitor individual pentest phases; view GPT analysis per phase
Risk Engine	Interactive risk matrix chart; CVSS scores with CIA impact; AI-suggested remediation strategies
Report	Render GPT-4–generated report; buttons to export PDF or HTML

Add UI component diagram here

7. Setup and Deployment

7.1. Prerequisites

- Node.js 18+
- Python 3.10+

- OpenAI API key, Shodan API key, Nessus credentials, Metasploit RPC password

7.2. Configuration

1. Copy backend/.env.example → backend/.env; populate keys
2. Ensure vite.config.ts proxies /api to http://localhost:8000
3. Enable CORS in FastAPI for http://localhost:5173

7.3. Installation

bash

```
# Backend
cd backend
pip install -r requirements.txt

# Frontend
cd ..
npm install
```

7.4. Launch

bash

```
# Start backend API
uvicorn app:app --reload --host 0.0.0.0 --port 8000

# Start frontend
npm run dev
# Access at http://localhost:5173
```

8. Usage Guide

1. **Full Workflow**: Dashboard → Enter IPs → Click "Start Full Workflow" → Review streaming logs
2. **Phase-by-Phase**: Navigate to Phases → Select phase → Provide target → Execute → View GPT analysis
3. **AI Workflow**: AI Workflow → Choose template → Edit prompts/payloads → Apply
4. **Reporting**: Report tab → Preview executive summary & risk matrix → Download PDF/HTML

9. Real-World Use Cases

1. **Enterprise Security Assessments**: Automated nightly scans across corporate ranges, with board-ready reports delivered each morning.

2. **DevSecOps Integration**: Pre-deployment scans in CI/CD pipelines; block risky builds.
3. **Managed Service Providers**: On-demand pentest offerings, branded reports, and AI-driven recommendations for SMB clients.
4. **Continuous Monitoring**: Scheduled scans of critical assets, instant remediation alerts via email/Slack.

10. Extensibility and Customization

- **Adding New Tools**: Place module in backend/pentest/, import in app.py, update n8n workflow
- **Custom GPT Prompts**: Override default prompts in utils/gpt_client.py or per-phase functions
- **UI Theming**: Modify Tailwind configuration in tailwind.config.ts for corporate branding

11. Security Considerations

- Protect API keys; do not commit .env
- Use TLS for Metasploit RPC in production
- Implement rate-limiting and error-handling in n8n workflows to manage API quotas

12. Testing and Validation

- **Unit Tests**: Write pytest tests for each module's logic
- **Integration Tests**: Deploy in isolated lab network, validate end-to-end functionality
- **UI E2E Tests**: Cypress scripts to simulate user flows and verify report export

13. Appendix

Sample .env

text

```
OPENAI_API_KEY=sk-…

SHODAN_API_KEY=…

NESSUS_USER=…

NESSUS_PASS=…

MSF_RPC_PASSWORD=…
```

Example cURL

bash

```
curl -X POST http://localhost:8000/api/scan \

  -H "Content-Type: application/json" \

  -d '{"targets":"192.168.1.10,10.0.0.5"}'
```