Project Title: LLM Agent-based Autonomous Penetration Testing Framework

---

Detailed Description:

An autonomous penetration testing framework driven by Large Language Models (LLMs). This framework automates reconnaissance, vulnerability hypothesis generation, exploit planning, and reporting, enabling scalable and intelligent security assessments.

Component	Role
Target Reconnaissance Agent	Uses LLM to reason about scanning strategies
Attack Surface Mapper	Summarizes exposed ports, services, APIs
Vulnerability Hypothesis Generator	Generates likely vulnerabilities using prior knowledge
Exploit Plan Designer	Plans step-by-step exploits automatically
Attack Executor	Safely launches attacks (sandboxed or simulated)
Reporting Engine	Writes a full structured pentest report

Component Details:

1. **Target Reconnaissance Agent**:
   - LLM reasons on which tools/techniques are optimal.
   - Example: "Use Nmap aggressive scan followed by dirbuster on port 80."
2. **Attack Surface Mapper**:
   - Processes outputs into structured graphs (IP → Port → Service → OS).
3. **Vulnerability Hypothesis Generator**:
   - LLM predicts which CVEs or logical flaws could exist.
4. **Exploit Plan Designer**:
   - Builds **exploit chains** (multi-stage attacks).
5. **Attack Executor**:
   - Executes the generated plans under supervision.
6. **Reporting Engine**:
   - Auto-generates a human-readable pentest report.

Overall System Flow:

- Input: IP range / domain / URL
- Output: Full pentest report based on LLM-driven actions
- Focus: **LLM agents driving autonomous pentesting**.

Internal Functioning of Each Module:

1. Target Reconnaissance Agent

- **Purpose**:
  - Plan **scanning strategy** dynamically using LLM reasoning.
- **How it works**:
  - Inputs:
    - IP ranges
    - Target info (domain, technologies detected)
  - LLM analyzes and outputs:
    - `"Start with full TCP scan -sS -p-"`
    - `"If ports 80/443 open, do content discovery"`
- **Technologies**:
  - GPT/Claude/Custom fine-tuned models for pentest thinking, etc.

---

2. Attack Surface Mapper

- **Purpose**:
  - Structure raw scan output into usable **attack graphs**.
- **How it works**:
  - Parses Nmap/Masscan results.
  - Builds a **service graph**
- **Technologies:**
  - Parsers, graph libraries (e.g., NetworkX), etc.

3. Vulnerability Hypothesis Generator

- **Purpose**:
  - Predict **likely vulnerabilities**.
- **How it works**:
  - Given detected services (e.g., "Apache 2.4.29"):
    - LLM suggests:
      - `"CVE-2019-0211: Local privilege escalation"`
      - `"Try default credentials if it's Tomcat."`
- **Advanced**:
  - Use CVE datasets fine-tuned into LLM embeddings for higher accuracy.

---

4. Exploit Plan Designer

- **Purpose**:
  - Chain multiple exploits intelligently.
- **How it works**:
  - LLM reasons:

        - `"Exploit weak FTP first → upload PHP shell → pivot into internal network"`
- **Format**:
    - Plans as step-by-step structured actions (like a tactical pentest playbook).

---

5. Attack Executor

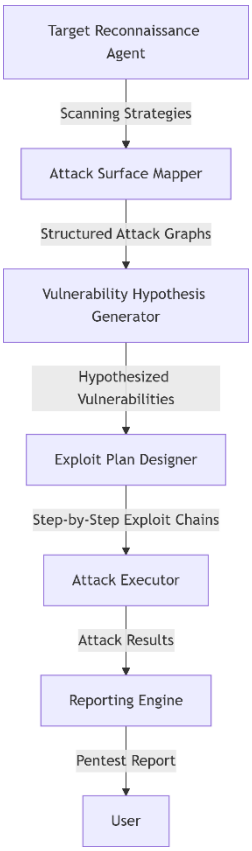
- **Purpose**:
    - Perform actions safely (sandbox or real test).
- **How it works**:
    - Validates plan safety.
    - Launches actions:
        - Nmap NSE scripts
        - Metasploit modules
        - Custom scripts

---

6. Reporting Engine

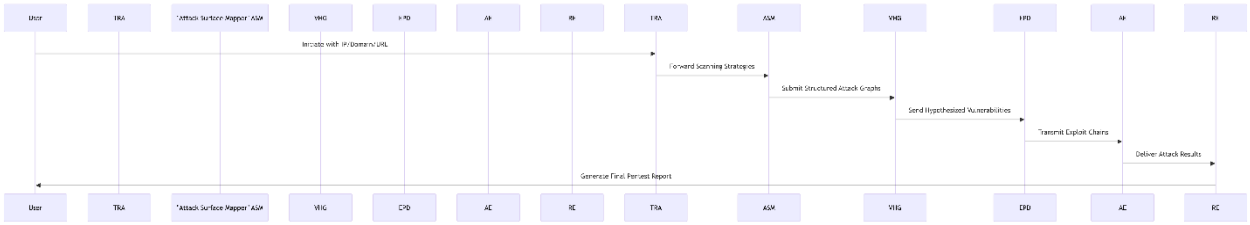
- **Purpose**:
    - Compile everything into human-readable reports.
- **How it works**:
    - Sections:
        - Findings
        - Exploitation paths
        - Screenshots/logs
        - Remediation suggestions

Component Diagram



- **Target Reconnaissance Agent**: Uses LLM to dynamically plan scanning strategies (e.g., Nmap commands, etc).
- **Attack Surface Mapper**: Structures scan results into attack graphs (IP → Port → Service).
- **Vulnerability Hypothesis Generator**: Predicts vulnerabilities (e.g., CVEs, misconfigurations, etc) using LLM reasoning.
- **Exploit Plan Designer**: Designs multi-stage attack chains (e.g., FTP exploit → pivot to internal network etc).
- **Attack Executor**: Safely executes exploits (e.g., Metasploit modules, custom scripts, etc).
- **Reporting Engine**: Generates structured reports with findings, logs, and remediation steps.

Sequence Diagram



1. **User** provides input (IP/domain/URL) to the **Target Reconnaissance Agent**.
2. **Reconnaissance Agent** determines scanning strategies and forwards them to the **Attack Surface Mapper**.
3. **Mapper** structures scan data into attack graphs and sends them to the **Vulnerability Hypothesis Generator**.
4. **Hypothesis Generator** predicts vulnerabilities and passes them to the **Exploit Plan Designer**.
5. **Plan Designer** creates exploit chains and triggers the **Attack Executor** to run them.
6. **Attack Executor** submits results to the **Reporting Engine**, which finalizes the report for the **User**.

Detailed Project Description: LLM Agent-based Autonomous Penetration Testing Framework

An autonomous penetration testing system driven by Large Language Models (LLMs). This system automates reconnaissance, vulnerability hypothesis generation, exploit planning, and reporting, enabling scalable and intelligent security assessments.

---

1. System Components and Roles

1.1 Target Reconnaissance Agent

Purpose: Dynamically plan scanning strategies using LLM reasoning.
Implementation Details (e.g.):

- **Tools**:
  - **LLM Integration**: Use OpenAI GPT-4, Claude, or a fine-tuned model (e.g., Mistral-7B) via API, etc.
  - **Scanning Tools**: Nmap, Masscan, dirbuster, etc.
- **Workflow**:
  1. **Input**: User provides target (IP, domain, or URL).
  2. **LLM Prompt**:
     ```
     prompt = f"Given target {target}, suggest optimal scanning commands. Prioritize non-intrusive methods."
     response = llm.generate(prompt)
     # Example output: "Run 'nmap -sS -p- -T4 {target}', then 'gobuster dir -u {target}:80 -w common.txt'"
     ```
  3. **Execute Scans**: Automate tool execution using Python subprocess or frameworks like `python-nmap`.

1.2 Attack Surface Mapper

Purpose: Structure raw scan data into actionable attack graphs.
Implementation Details (e.g.):

- **Tools**:
  - **Nmap Parser**: Use `libnmap` to parse XML results.

    - **Graph Representation**: NetworkX for building service graphs (IP → Port → Service).
- **Example**:

```
from libnmap.parser import NmapParser
report = NmapParser.parse_fromfile("scan.xml")
graph = nx.DiGraph()
for host in report.hosts:
    graph.add_node(host.address, type="IP")
    for service in host.services:
        graph.add_node(service.port, type="Port")
        graph.add_edge(host.address, service.port, service=service.service
)
```

1.3 Vulnerability Hypothesis Generator

Purpose: Predict vulnerabilities using LLM reasoning and CVE databases.

Implementation Details (e.g.):

- **Tools**:
    - **CVE Integration**: Query NVD API or local databases (e.g., `cve-search`).
    - **LLM Prompt**:

```
prompt = f"Service: Apache 2.4.29 on port 80. List CVEs and misconfi
gurations."
response = llm.generate(prompt)
# Output: "CVE-2019-0211 (Local Privilege Escalation). Check for defa
ult credentials on /manager/html."
```

- **Accuracy Enhancement**: Fine-tune LLM on CVE descriptions and exploit-db entries.

1.4 Exploit Plan Designer

Purpose: Generate multi-stage exploit chains.

Implementation Details (e.g.):

- **Tools**:
    - **LLM Reasoning**:

```
prompt = "Vulnerabilities: FTP weak credentials, Apache CVE-2019-021
1. Design exploit chain."
response = llm.generate(prompt)
# Output: "1. Brute-force FTP. 2. Upload reverse shell. 3. Escalate p
rivileges via CVE-2019-0211."
```

    - **Structured Plans**: Convert LLM output into JSON playbooks:

```
{
  "steps": [
    {"tool": "hydra", "args": "-l admin -P rockyou.txt ftp://{target
}"},
    {"tool": "metasploit", "module": "exploit/unix/ftp/proftpd_modco
py_exec"}
  ]
}
```

1.5 Attack Executor

Purpose: Safely execute exploits in controlled environments.

Implementation Details (e.g.):

- **Tools**:
  - **Sandboxing**: Docker containers or virtual machines (e.g., VirtualBox, etc).
  - **Automation**: Metasploit RPC, `pwntools`, or `subprocess`.
- **Safety Checks**:

```
if "rm -rf" in exploit_command:
    raise BlockedCommandError("Dangerous command detected.")
```

1.6 Reporting Engine

Purpose: Generate structured penetration test reports.

Implementation Details (e.g.):

- **Tools**:
  - **LLM Writing**:

```
prompt = "Summarize findings: FTP breach, Apache CVE. Include remedi
ation."
report = llm.generate(prompt)
```

  - **Template Engine**: Use Jinja2 for PDF/HTML reports:

---

2. System Integration and Component Interaction

1. **Reconnaissance**:
   - **Recon Agent → Attack Surface Mapper**: Scan commands → structured graph.

2. **Hypothesis Generation**:
    - **Mapper → Vulnerability Generator**: Service data → CVE list.
3. **Exploit Design**:
    - **Vulnerability Generator → Plan Designer**: CVEs → JSON playbook.
4. **Execution**:
    - **Plan Designer → Attack Executor**: Playbook → sandboxed exploits.
5. **Reporting**:
    - **Executor → Reporting Engine**: Results → auto-generated report.

---

3. Evaluation Criteria

1. **Scan Accuracy**: Percentage of open ports/services correctly identified.
2. **CVE Precision**: Relevance of LLM-suggested vulnerabilities to detected services.
3. **Exploit Success Rate**: Percentage of executed plans achieving their goal.
4. **Report Quality**: Clarity and actionability of remediation steps.

---

4. Ethical Considerations

- **Authorization**: Only test systems with explicit permission.
- **Safety**: Use sandboxed environments for exploit execution.
- **Compliance**: Follow responsible disclosure for discovered vulnerabilities.

---

5. Tools and Resources (e.g.)

- **LLMs**: GPT-4, Claude, Mistral-7B, etc.
- **Scanning**: Nmap, Masscan, Gobuster, etc.
- **Exploitation**: Metasploit, Hydra, etc.
- **Reporting**: Jinja2, LaTeX, etc.