AskBanu(cybersecurity Assitant-0):

**Core Functions:**

1. Network Monitoring
2. Vulnerbility Scanning and malware analysis
3. Analyzing logs for anomalies
4. Security advice
5. Incident reporting
6. Auditing checks

**Programming Language:**

1. Python

**Libraries and Tools:**

1. Scapy
2. Nmap
3. Osquery
4. Psutil
5. Loguru

**Set up AI for Natural Language Understanding (NLU)**

Generated API key

Installed dependies related to openapi

Created Virtual Environment within the project file.

**Vulnerability Scanning:**

* **Nmap Scripts (NSE)**: Nmap has a built-in script engine that can run security scripts to detect vulnerabilities (e.g., default credentials, known CVEs, open ports exposing vulnerable services).
  + Add the following option to your Nmap scan: -sC --script vuln.
  + Example Script: Detect known vulnerabilities like Heartbleed.
* **OpenVAS or Nessus Integration**: These tools perform deeper vulnerability scanning. You can consider integrating them for more detailed scans.

**2. Malware Signatures:**

* **YARA Rules**: Integrate YARA to scan files or network traffic for malware signatures.
* **ClamAV Integration**: ClamAV is an open-source antivirus engine that can scan files and directories for malware.
  + You can use pyclamd for Python integration to scan a target system for known malware signatures.

**3. Log Analysis:**

* **ELK Stack (Elasticsearch, Logstash, Kibana)**: For advanced log analysis, consider setting up an ELK stack where logs are ingested, analyzed, and visualized.
* **Python Log Analysis**: Write a Python function to analyze system logs (e.g., /var/log/auth.log for Linux) for suspicious activity such as failed login attempts or privilege escalations.

**4. Security Advice Using LLMs:**

* **LLM Integration (like GPT)**: You can integrate an LLM (via OpenAI API) to analyze scan results and logs and provide security advice or suggestions.
  + Example: After a vulnerability scan, the LLM could suggest remediation steps based on identified issues.
  + You can fine-tune an LLM to your needs or use a pre-built security assistant model.

**5. Incident Reporting:**

* **Automated Incident Reports**: Generate detailed incident reports in formats like PDF, which summarize detected vulnerabilities, malware threats, and log anomalies.
* You can use ReportLab in Python to generate PDFs with a summary of security events, including timestamps, severity, and suggested actions.

**6. Auditing Checks:**

* **CIS Benchmarks**: Implement auditing based on CIS benchmarks, which include security hardening standards for operating systems, databases, and applications.
* **Auditd**: For Linux, you can integrate auditd for auditing kernel-level events such as file access, privilege escalation, or unauthorized actions

Step 1: Building an Vulnerability Scanner

 **nmap.PortScanner()**: Initializes the Nmap scanner.

 **scanner.scan()**: Runs the vulnerability scan using -sV --script vuln. The -sV flag checks for service versions, and the --script vuln flag runs vulnerability scripts available in Nmap’s NSE library.

 **scanner[target\_ip]**: Extracts the scan result for the specific target IP.

 **if target\_ip in scanner.all\_hosts()**: Checks if the target is up, otherwise, it returns a message saying the host is unreachable.

**Components of the Vulnerability Scanner with AI**

* **Vulnerability Scanning Engine**: Use traditional scanning tools like Nmap and OpenVAS for detecting vulnerabilities.
* **AI for Analyzing Results**: Use an LLM (such as GPT) to analyze the scan output, prioritize vulnerabilities, and suggest remediations.
* **Database of Vulnerabilities**: Integrate a database (like NVD or CVE) for known vulnerabilities and include machine learning for anomaly detection.
* **Reporting**: Automatically generate reports with the scan results, analysis, and recommendations

To complete this project by tomorrow, here's a fast-paced plan:

**1. Finalize Chatbot Framework and Architecture**

* **Architecture**: Utilize microservices architecture to make each tool interaction a service.
* **Framework**: Use a Python framework like Flask or FastAPI to handle chatbot requests and integrate security tools (e.g., Nmap, OpenVAS, Metasploit).
* **Mimicking Tools**: For each security tool, create modules that accept parameters (IP address, port, etc.) and return actionable results (vulnerabilities, exploits).

**2. Tool Integration**

* **Nmap Integration**: Ensure existing Nmap module is fully functional for scanning and parsing results.
* **Other Tools**:
  + **Metasploit**: Integrate using msfrpc for automated exploit suggestions.
  + **OpenVAS**: Implement OpenVAS integration for deeper vulnerability analysis.
* **Script Malware Scanning**: Use ClamAV or custom signature-based scanning via Python.

**3. AI/LLM Integration for Exploit Suggestions**

* Fine-tune GPT-3.5 or GPT-4 on cybersecurity-related content to suggest exploits based on tool output.
* Train the chatbot to map vulnerabilities to exploits, offering options like remote code execution, SQL injection, etc.

**4. Natural Language Processing (NLP)**

* Implement NLP capabilities to understand expert input, convert it into tool commands, and provide explanations.

**5. Testing & Finalization**

* Test on multiple network scenarios, and adjust AI responses and tool interaction speed.
* Set up a basic UI or CLI for users to interact with the bot and select tools dynamically.

By focusing on these key areas, you should be able to finish the project quickly.