**SE 335 01 Cyber Security Term Project**

**1. Introduction**

In today's interconnected digital landscape, safeguarding networked systems against cyber threats is paramount. The "Detection of security vulnerabilities on remote computer(s) by manual software execution" project addresses this challenge by developing a solution to detect and analyze vulnerabilities within networked environments. Leveraging Mitre Attacks' knowledge base, the project focuses on executing predefined attacks, such as T1124 (System Time Discovery) and T1566.001 (Phishing), by manually uploading and executing software on target computers within the network.

Results from these executions are stored to a defined database and displayed on a web-based dashboard, empowering organizations to proactively assess and fortify their cyber defenses. Through this project, we aim to provide organizations with the tools and insights needed to protect their digital assets from evolving cyber threats.

**2. Requirements Analysis**

**2.1 Functional Requirements:**

**Attack Selection:** The system should provide the capability to select and execute two specific attacks from the MITRE ATT&CK framework: Technique T1124 - System Network Configuration Discovery and Technique T1566.001 - Phishing.

Preparation of Malicious Attachment:

File Creation: Use Python to create a malicious DOCX or PDF file that will execute a payload when opened.

Payload: The payload can be a script or a macro that performs specific actions, such as opening a backdoor or exfiltrating data.

**Attack Execution:** The system should facilitate the manual execution of selected attacks on remote computers within the network.

*T1124 (System Time Discovery):* The system will execute attacks by manually uploading and triggering malicious files (e.g., PDF, DOCX) on the victim's computer. For T1124, a flash drive containing the malicious file will be manually plugged into the victim computer. The attack will be executed when the file is opened.

Deploy the script:

Manually deploy the script to the target machine. This can be done via a USB drive or by sending the script over the network.

Run the script:

Execute the Python script on the target machine. Ensure that Python is installed on the machine and that the script has the necessary permissions to run.

Collect results:

The script will save the system time and network configuration data to a file (system\_info.txt) on the target machine. Manually retrieve this file for analysis.

*T1566.001 (Phishing):* The system will execute the T1566.001 attack by sending spearphishing emails with malicious attachments to target users. These attachments, when opened, will execute the attack payload on the victim's computer. The attachments can be various file types like PDF, DOCX, or EXE files. The objective is to exploit user trust and get the user to open the attachment, thereby triggering the malicious code.

Email Crafting:

Spearphishing Email: Craft a spearphishing email tailored to the target, making it appear to come from a trusted source and include a compelling reason for the recipient to open the attachment.

Email Content: The email should be persuasive and relevant to the recipient, such as a fake invoice, an urgent request from a colleague, or a notice from a known service provider.

Email Sending: Manually send the spearphishing email to the target using a standard email client. Attach the malicious file created in the previous step.

Execution of agent:

User Interaction: The attack relies on the recipient opening the attachment. Upon opening, the embedded exploit or macro will execute, compromising the target system.

Payload Execution: The payload can be designed to perform various actions, such as establishing a reverse shell, exfiltrating data, or downloading additional malware.

**Data Collection:** During the execution of each attack, relevant metadata and results will be collected. This data, including details of the techniques used, responses from victim computers, and outcomes of the attacks, will be stored in a centralized database managed by our attack system. This allows for comprehensive analysis, monitoring, and reporting of attack activities.

During the execution of these attacks, capture relevant data such as system time, network configuration, and user interactions.

Use the data logging functions which Python libraries provide to store this information in the local database.

Ensure that all collected data is timestamped and categorized appropriately for later analysis.

**Database Storage:** Attack results should be stored securely in a centralized database for future analysis and reference.

Execution Table:

id (Primary Key): Unique identifier for each record.

Timestamp: The time when the command was executed.

command: The command that was executed.

success: Boolean indicating if the command execution was successful.

error: Any error messages captured during execution.

System Time Table:

id (Primary Key): Unique identifier for each record.

timestamp: The time when the data was collected.

system\_time: The current system time of the target machine.

timezone: The time zone settings of the target machine.

**Web-Based Dashboard:** The system should include a web-based dashboard for administrators to view and monitor the results of executed attacks in real-time.

Connect the Flask application to the SQLite database that stores the attack results and system information.

Ensure that the application can query the database to retrieve data.

Define routes for different pages of the dashboard (e.g., home page, detailed view of attack results).

Use HTML, CSS, and JavaScript to design an intuitive and user-friendly interface.

**2.2 Non-Functional Requirements:**

**Security:** The system should incorporate security measures to protect against unauthorized access, data tampering, and other potential threats.

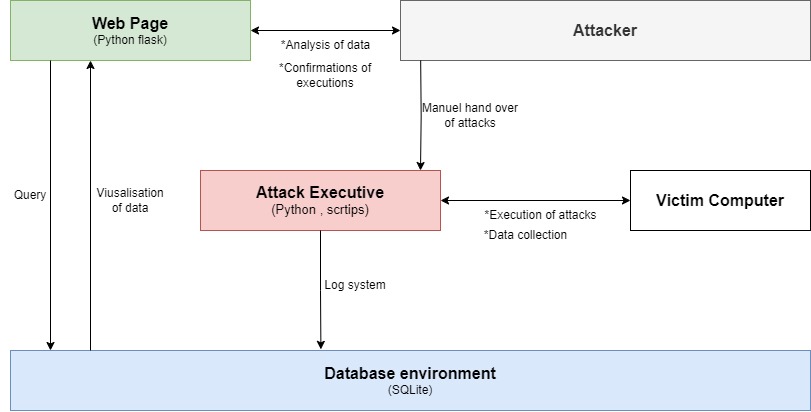
**Performance:** The system should be capable of handling multiple concurrent users and large volumes of data without significant degradation in performance.

**Scalability:** The system should be scalable to accommodate future growth and expansion of the network environment.

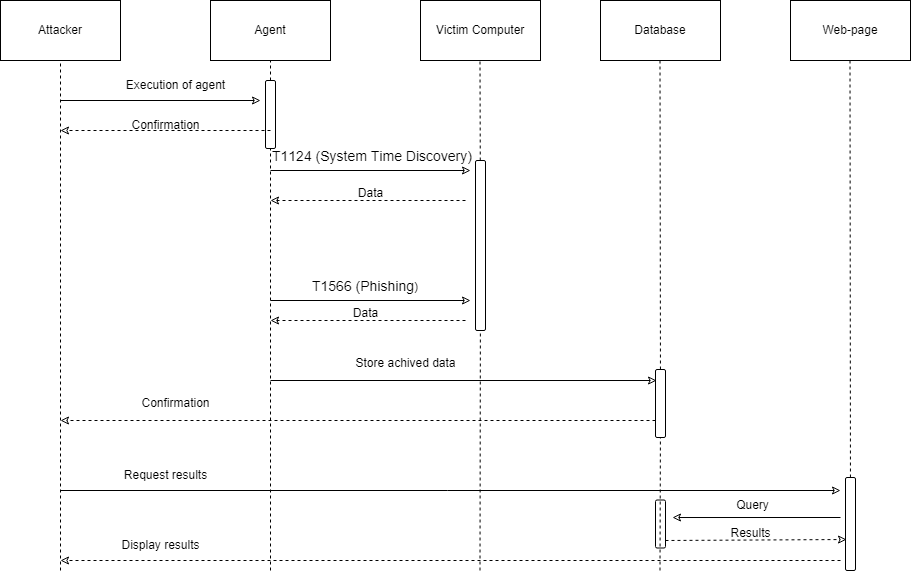
**Usability:** The user interface of the web-based dashboard should be intuitive and user-friendly, requiring minimal training for administrators to use effectively.

**3. Diagrams**

**High Level Diagram:**



**Sequence Diagram:**



**4. Technological Level**

**Technologies, Languages, and Frameworks:**

**Python**: Python will be the primary language used for developing the software components of this project. Its simplicity, readability, and extensive libraries make it well-suited for rapid development and prototyping. Additionally, Python's versatility allows for seamless integration with various operating systems and environments, facilitating the manual upload and execution of software on target computers.

**Flask:** Flask, a lightweight web framework for Python, will be utilized for developing the web-based dashboard. Its minimalistic design and easy-to-use nature make it ideal for building simple yet powerful web applications. Flask's flexibility allows for customization to meet specific project requirements while ensuring scalability and maintainability.

**SQLite:** SQLite will serve as the database management system for storing attack results and other relevant data. As a self-contained, serverless, and lightweight database engine, SQLite offers simplicity and ease of deployment without sacrificing functionality. It provides efficient data storage and retrieval capabilities, making it suitable for managing the project's data requirements.

**Justification:**

**Python:** Chosen for its simplicity, readability, and extensive libraries, Python enables efficient development and deployment of the project's software components. Its compatibility with various operating systems ensures seamless execution of attacks on target computers.

**Flask:** Selected for its lightweight nature and ease of use, Flask allows for the rapid development of the web-based dashboard. Its flexibility enables customization to meet specific project requirements while ensuring scalability and maintainability as the project evolves.

**SQLite:** Utilized for its simplicity, efficiency, and serverless architecture, SQLite provides a lightweight yet powerful solution for storing and managing attack results. Its ease of deployment and compatibility with Python further enhance the project's overall efficiency and effectiveness.

**5. Attack Execution Techniques:**

**System Time Discovery (T1124):**

Attackers may collect system time and time zone settings from local or remote systems, which are typically managed by services such as the Windows Time Service or systemsetup on macOS. This data is crucial for synchronizing time settings across a network.

System time information can be obtained using various methods, such as executing commands like net time \hostname on Windows or CLI commands like show clock detail on network devices.

This gathered information can serve multiple purposes, including aiding in the execution of scheduled tasks, determining victim locality based on time zone, or orchestrating attacks to trigger at specific dates or times.

**Phishing (T1566.001):**

Adversaries deploy spearphishing emails with malicious attachments to infiltrate victim systems. Spearphishing attachments are designed to exploit user trust and rely on recipients' actions to execute the attached malware.

These emails often employ social engineering tactics to persuade recipients to open the attachments, posing as trusted sources or presenting plausible reasons for accessing the files.

Attachments may take various forms, including Microsoft Office documents, executables, PDFs, or archived files. Once opened, the payload within the attachment exploits vulnerabilities or directly executes on the victim's system.

Adversaries may manipulate file extensions and icons to deceive users into believing that the attachments are harmless documents or files associated with familiar applications.

*Ali Fuat Tulumcu 220704314*

*Ahmet Hakan Kapusızoğlu 210704008*