SpreadScope:

Malicious Object Dissemination Assessment Tool

**Project Description:**

The project is a program that scans devices to detect objects previously identified as malicious, searches the specified paths to allocate the search currency, and measures their spread in the network. This scan is performed by the cybersecurity administrator for multiple devices and for multiple types of inputs that the program is searching for. The project utilizes the YARA library, which assists in writing search scenarios, and is integrated into our project developed in Python.

**Project Goals:**

* **Determine the Extent of Spread**: Assess the extent to which the malicious object has spread across the network. Identify and list all devices where the object has been found.
* **Reduce Administrative Time**: Minimize the time required by cybersecurity administrators to conduct searches by consolidating all results in a single, accessible location.
* **Simplify the Search Process**: Streamline the process by allowing administrators to input simple search parameters, such as the name of the malicious object (file) and the specific location to be searched.

**Benefits:**

* **Manage Threats Efficiently**: By providing a comprehensive overview of the spread of malicious objects, cybersecurity administrators can quickly respond to and mitigate threats.
* **Time Saving:** The centralized display of search results dramatically reduces the time cybersecurity administrators spend detecting and analyzing threats.
* **Simplified Search Process:** The user-friendly interface allows administrators to easily start searches with simple inputs, making the process easier and less error-prone.
* **Comprehensive Reports:** Detailed reports on detected harmful objects, their locations, and quick action.
* **Resource Optimization:** The software efficiently allocates search currency, ensuring efficient use of network resources during scans.
* **Cleverness**: The ability to scan multiple devices and handle different types of input makes the software adaptable to different network environments and threat scenarios.
* **Flexibility:** The software's support for multiple types of input enhances its flexibility, allowing it to adapt to different types of malicious objects and different network configurations.

By implementing this program, organizations can strengthen their cyber defenses, simplify administrative tasks, and ensure a more secure and efficient operating environment.

**Mechanism of Action:**

The software works in a coordinated manner between the devices being scanned and the administrator device. Here's how it works:

* Administrator setup: The cybersecurity administrator configures the search method, enters the object to search, and specifies search locations.
* Data Transfer: This research data is then transferred to devices maintained by the Cyber ​​Security Officer.
* Local device search: Each device receives the search parameters and performs a local search based on the instructions provided.
* Collect results: Search results from each device are sent back to the administrator's device.
* Data storage and analysis: Results are stored in a central database. The management interface displays these results, including a graphical representation showing the percentage and number of devices where the object was found.
* This process ensures an efficient and comprehensive scan of devices scanned by the administrator, allowing for rapid identification of potential threats and mitigation of their effects.

**Search by Dynamic Link Library (DLL):**

This method offers two ways: specify the exact file path to search or use the DLL name.

Searching by file path ensures greater accuracy, as it looks for an exact match in terms of file name, size, hash value, and content.

Instead, searching by DLL name focuses only on the libraries containing the files, omitting comparisons based on size and fragmentation, which is a good approach when all you have is the name.

A specific YARA rule is used here to carry out this process.

**Search by Hex Sequence:**

In this method, the administrator inputs a specific hex sequence to be searched within the files. Each hex within the files is read and compared to the provided sequence. This approach is particularly useful for locating a specific instruction sequence within the files.

**Center Function Descriptions:**

**1. send\_data\_to\_endpoint\_and\_write\_to\_db(url, data):**

- **Purpose**: Sends collected data to a given endpoint URL via a POST request and processes the response.

- **Steps**:

- Sets the request headers to indicate JSON content.

- Sends a POST request with the data to the specified URL.

- Checks if the request is successful (HTTP status code 200).

- Extracts the IP address from the URL using `extract\_ip\_from\_url`.

- Writes the response data to a MySQL database using `write\_response\_to\_database`.

**2. extract\_ip\_from\_url(url):**

- **Purpose**: Extracts and returns the hostname (IP address) from the given URL.

- **Steps**:

- Parses the URL using `urlparse`.

- Returns the hostname component of the parsed URL.

**3. write\_response\_to\_database(response\_data, ip\_address):**

- **Purpose**: Inserts the response data and the IP address into a MySQL database.

- **Steps**:

- Connects to a MySQL database with specified credentials.

- Prepares an SQL query to insert the response data (`HashResult`, `DLLResult`, `HexResult`, `StringResult`) along with the IP address.

- Executes the query and commits the transaction.

- Closes the database connection.

**4. start\_dll\_detect():**

- **Purpose**: Collects user input for DLL detection.

- **Steps**:

- Asks the user whether to input a DLL name or a DLL path.

- Depending on the choice, it asks for the DLL path or name and the directory to search in.

- Stores the collected inputs in the `start\_arg` dictionary.

**5. start\_hex\_detect():**

- **Purpose**: Collects user input for hex sequence detection.

- **Steps**:

- Prompts the user to enter a hex sequence and a directory to search in.

- Stores the inputs in the `hex\_arg` dictionary.

**6. hash\_input\_user():**

- **Purpose**: Collects user input for hash value detection.

- **Steps**:

- Prompts the user for a directory path and a hash value.

- Converts the hash value to lowercase and stores the inputs in the `hash\_arg` dictionary.

**7. string\_input\_user():**

- **Purpose**: Collects user input for suspicious string detection.

- **Steps**:

- Prompts the user for a directory path and a suspicious string.

- Stores the inputs in the `string\_arg` dictionary.

**Main Function:**

- main():

- **Purpose**: Coordinates the entire process by prompting the user for input and sending the data to specified endpoints.

- **Steps**:

- Prompts the user to decide whether to send data related to DLLs, hex sequences, hash values, and strings.

- Based on user responses, it calls the appropriate input functions to collect data or assigns default values.

- Prepares a data dictionary containing all collected inputs.

- Iterates over a list of endpoint URLs and sends the data to each using `send\_data\_to\_endpoint\_and\_write\_to\_db`.

**Usage**:

- The script starts by calling the `main()` function.

- Users are prompted to provide input for DLL detection, hex sequence detection, hash value detection, and suspicious string detection.

- The collected data is sent to a specified endpoint, and the response is written to a MySQL database.

**Assumptions and Requirements:**

- The MySQL database (`endpoints`) and table (`endpoints`) are already created with columns matching the data being inserted (`ip\_address`, `hash`, `dll`, `hex`, `string`).

- The endpoint URLs are assumed to be properly configured to handle the POST requests and return JSON responses.- The necessary Python libraries (`requests`, `mysql.connector`, `os`, `json`, `urllib.parse`) are installed.

**Endpoint Function Descriptions:**

**1-string\_scan(suspicious\_string, string\_Directory):**

* **Purpose**: Scans a directory for files containing a specific suspicious string using YARA rules.
* **Steps**:

- Creates a YARA rule for the suspicious string using string\_rule.

- Compiles the YARA rule.

- Iterates through all files in the specified directory.

- For each file, checks if it matches the YARA rule.

- Returns "match" if any file matches the rule, otherwise returns "no-match".

**2- match\_hash(hash\_value, hash\_Directory):**

* **Purpose:** Scans a directory for files with a specific hash value using YARA rules.
* **Steps**:

-Creates a YARA rule for the hash value using hash\_rule.

- Compiles the YARA rule.

- Iterates through all files in the specified directory.

- For each file, checks if it matches the YARA rule.

- Returns "match" if any file matches the rule, otherwise returns "no-match".

**3-detect\_hex\_sequence(hex\_sequence, directory\_path):**

* **Purpose:** Detects files containing a specific hexadecimal sequence using YARA rules.
* **Steps:**

- Creates a YARA rule for the hexadecimal sequence using hex\_sequence\_rule.

- Compiles the YARA rule.

- Iterates through all files in the specified directory.

- For each file, checks if it matches the YARA rule.

- Appends matching files to a list.

Returns the list of matched files**.**

**4-detect\_dll(dll\_name, directory\_path, f\_path, DLL\_choice):**

* **Purpose:** Detects DLL files by name and optionally by their properties (file size and MD5 hash) using YARA rules.
* **Steps:**

-If DLL\_choice is 'p' (properties):

- Creates a YARA rule using dll\_rule with the DLL name, file size, and MD5 hash.

- If DLL\_choice is 'n' (name only):

- Creates a YARA rule using dll\_import\_rule with the DLL name.

- Compiles the YARA rule.

- Iterates through all files in the specified directory.

- For each file, checks if it matches the YARA rule.

- Appends matching files to a list.

- Returns the list of matched files.

**Main Function:**

**main():**

* **Purpose:** Coordinates the entire process by running the script and starting the web server.
* **Steps:**

- Runs the Flask app on a separate thread using Thread to listen for connections.

- Calls the main() function to start the main process.

**Usage:**

* The script starts by calling the main() function.
* The Flask web server starts, allowing the script to listen for connections and process requests.

**Assumptions and Requirements:**

* The necessary YARA rules are defined in the functions string\_rule, hash\_rule, hex\_sequence\_rule, dll\_rule, and dll\_import\_rule.
* The directory paths and suspicious strings, hash values, hex sequences, or DLL names are correctly provided.
* The YARA library and required Python modules are installed and properly configured.
* **Dashboard**
* This dashboard offers a straightforward and easy-to-grasp visualization of the distribution between matching and non-matching hashes. The pie chart provides a clear breakdown, with the red slice representing the devices that have matching hashes, and the blue slice indicating the proportion of devices with non-matching hashes. This allows users to quickly identify any potential issues or areas that require further examination. The dashboard is designed to work seamlessly with our Apache web server and MySQL database setup.