Software Design Report

As part of our project, we developed a Python-based TCP port scanner. The goal of this project is to identify open ports on local and remote systems within our infrastructure.

We begin by importing essential Python libraries:

- datetime is used to track how long the scan takes.
- logging allows us to record events to a file for auditing.
- re helps us validate IP addresses using regular expressions.
- socket provides low-level networking operations, including connecting to ports.
- time is used to add delays between scans for rate limiting.

We define our configuration variables at the top of the script:

- HOST_NAME uses socket.gethostname() to get the machine's name.
- TARGET_HOST resolves the local IP address using socket.gethostbyname().
- PORT_RANGE is set to scan TCP ports from 1 to 65535
- SCAN_DELAY is set to 0.1 seconds to slow down the scan slightly and avoid triggering security defenses.

We configure the logging module to write to a file called **port_scan.log**, using INFO level and timestamped formatting. This provides us with a historical record of all scanning activity, useful for compliance and forensics.

The function validate(ip) ensures that any remote IP address input by the user is in valid IPv4 format. It does this by:

- Splitting the address into 4 parts.
- Checking that each part is a number between 0 and 255.
- Confirming the overall structure with a regex pattern.

This helps prevent incorrect or malformed inputs that could cause the script to fail or behave unpredictably.

The function is_port_open(host, port, timeout=1) attempts to create a TCP connection to a given port. If the connection is successful, the port is open; otherwise, it's closed or filtered. We use a try-except block to handle errors like timeouts or refused connections.

Then, the function scan_ports(host, port_range) iterates through the port range and:

- Calls is_port_open() for each port.
- Logs and prints open ports.
- Waits SCAN_DELAY seconds between each scan.

This function also tracks the duration of the scan using timestamps from the datetime module.

In the main script block (if __name__ == "__main__":), we prompt the user to choose between scanning the local machine or a remote host. If they choose remote (R), we ask them to input an IP address and validate it.

Once the host is selected, we initiate the scan and display the results. If no ports are open, we print a simple message. If ports are open, we list them.

We also handle several types of errors:

- If the user presses Ctrl+C (KeyboardInterrupt), we exit gracefully.
- If the host cannot be resolved (socket.gaierror), we log the error.
- Any unexpected error is caught by a general except block and logged for investigation.

Through this project, we developed a lightweight but functional TCP port scanner that aligns with our internal cybersecurity practices. It gives us visibility into which ports are open across our network, aiding vulnerability management and threat detection.

The script is modular, simple to maintain, and easily extensible. We plan to evolve it further by adding support for UDP scans, multithreading for speed, and possibly integrating it with other monitoring tools or dashboards.