Detailed Steps to Solve the Machine

Machine Information

- **Macro**: CRPT
- **Type**: Known Plaintext
- **Description**: The machine uses the same key for all encryptions. The flag is the encryption key used in an XOR-based encryption scheme with padding. The challenge involves network discovery, interacting with a web service, and decrypting data using a known plaintext attack.
- **Objective**: Retrieve the flag, which is the XOR encryption key, located at /root/flag.

Step-by-Step Process

Step 1: Network Discovery with Nmap

- **Command**: nmap -sn 192.168.4.0/24
- **Description**:
    - **Purpose**: Perform a ping scan to identify live hosts on the 192.168.4.0/24 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 192.168.4.0/24 range (256 addresses) are active.
        - Identifies the target machine's IP address within the network.
    - **Assumption**: The scan reveals 192.168.4.1 as a live host, which we target in subsequent steps.
    - **Output**: A list of active IPs, including 192.168.4.1.

Step 2: Service Scanning with Nmap

- **Command**: nmap -sV 192.168.4.1
- **Description**:

- **Purpose**: Identify open ports and services on the target machine (192.168.4.1).
- **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 8080 (HTTP) is open, running a web service.
- **Output**: A report listing open ports, with port 8080 (HTTP) confirmed as the entry point.

Step 3: Access Web Service

- **Command**: curl http://192.168.4.1:8080
- **Description**:
  - **Purpose**: Interact with the web service on port 8080 to explore its functionality.
  - **Details**:
    - Executed from 192.168.0.5.
    - Sends an HTTP GET request to the root endpoint.
  - **Assumption**: The response provides information about the web application, possibly indicating endpoints like /source, /encrypt_form, or /show_encrypted_notes.
  - **Output**: HTML or text describing the web service.

Step 4: Retrieve Source Code

- **Command**: curl http://192.168.4.1:8080/source
- **Description**:
  - **Purpose**: Download the source code of the web application to understand its logic.

- **Details**:
  - Sends a GET request to the /source endpoint.
  - Likely reveals a Python script (e.g., Flask app) that handles encryption with XOR and padding.
- **Assumption**: The source code shows the encryption logic, using XOR with a fixed key (the flag) and PKCS#7 padding (implied by pad and unpad usage) to a block size of 64 bytes.
- **Output**: Source code revealing XOR encryption with a static key and padding.

Step 5: Access Encryption Form

- **Command**: curl http://192.168.4.1:8080/encrypt_form
- **Description**:
  - **Purpose**: Retrieve the encryption form to understand how to submit data for encryption.
  - **Details**:
    - Sends a GET request to /encrypt_form.
    - Likely returns an HTML form for submitting a note parameter to /encrypt.
  - **Assumption**: The form accepts a note field, which is padded and encrypted via a POST request to /encrypt.
  - **Output**: HTML form for encryption.

Step 6: Encrypt a Known Plaintext

- **Command**: curl -X POST -d "note=1234567890123456" http://192.168.4.1:8080/encrypt
- **Description**:
  - **Purpose**: Submit a known plaintext (1234567890123456) to obtain its ciphertext.
  - **Details**:
    - Sends a POST request to /encrypt with the note parameter.

        - The plaintext is 16 bytes long, which is padded to 64 bytes (as indicated by later commands).
        - The server applies PKCS#7 padding, then XORs the padded plaintext with the key (flag) to produce the ciphertext.
    - **Assumption**: The server returns the hexadecimal ciphertext (e.g., dae0ec429ff51f30252095bca2cc7dcdef5eeb25ca583b91cef51e373abc982babf9213ac8431ac91bc3c3ae788084623897bf2b5ecc592b2ca92fb2b973e015).
    - **Output**: Ciphertext in hexadecimal format.

Step 7: View Encrypted Notes

- **Command**: curl http://192.168.4.1:8080/show_encrypted_notes
- **Description**:
    - **Purpose**: Retrieve a list of encrypted notes to confirm the submitted ciphertext and identify additional ciphertexts.
    - **Details**:
        - Sends a GET request to /show_encrypted_notes.
        - Displays previously encrypted notes, including the one from Step 6 and possibly another ciphertext (e.g., for the flag).
    - **Assumption**: Confirms the ciphertext for the known plaintext and reveals another ciphertext (dc91974098a61b4d444ad6ecfab17cc9ef5eeb25ca583b91cef51e373abc982babf9213ac8431ac91bc3c3ae788084623897bf2b5ecc592b2ca92fb2b973e015) containing the encrypted flag.
    - **Output**: List of encrypted notes, including the known ciphertext and the flag's ciphertext.

Step 8: Set Up Python Environment

- **Command**: python3
- **Description**:
    - **Purpose**: Start a Python 3 interactive shell to perform decryption.
    - **Details**:

        - Executed on 192.168.0.5.
        - Prepares for cryptographic operations.
    - **Output**: Python 3 shell prompt.

Step 9: Import Required Modules

- **Command**: from Crypto.Util.strxor import strxor
- **Command**: from Crypto.Util.Padding import pad, unpad
- **Description**:
    - **Purpose**: Import the strxor function for XOR operations and pad/unpad for handling PKCS#7 padding.
    - **Details**:
        - strxor performs byte-wise XOR between two byte strings.
        - pad and unpad handle PKCS#7 padding, used to pad the plaintext to 64 bytes and unpad the decrypted result.
    - **Output**: Modules imported successfully.

Step 10: Recover the Key

- **Command**: key = strxor(bytes.fromhex("dae0ec429ff51f30252095bca2cc7dcdef5eeb25ca583b91cef51e373abc982babf9213ac8431ac91bc3c3ae788084623897bf2b5ecc592b2ca92fb2b973e015"), pad(b"1234567890123456", 64))
- **Description**:
    - **Purpose**: Perform a known plaintext attack to recover the encryption key.
    - **Details**:
        - The ciphertext dae0ec429ff51f30252095bca2cc7dcdef5eeb25ca583b91cef51e373abc982babf9213ac8431ac91bc3c3ae788084623897bf2b5ecc592b2ca92fb2b973e015 corresponds to the known plaintext 1234567890123456.
        - The plaintext is padded to 64 bytes using PKCS#7 padding (pad(b"1234567890123456", 64)).

        - In XOR encryption, ciphertext = padded_plaintext XOR key. Thus, key = ciphertext XOR padded_plaintext.
        - bytes.fromhex converts the hexadecimal ciphertext to bytes.
        - strxor computes the XOR, yielding the key.
    - **Output**: The key as a 64-byte string.

Step 11: Load the Flag Ciphertext

- **Command**: ct = bytes.fromhex("dc91974098a61b4d444ad6ecfab17cc9ef5eeb25ca583b91cef51e373abc982babf9213ac8431ac91bc3c3ae788084623897bf2b5ecc592b2ca92fb2b973e015")
- **Description**:
    - **Purpose**: Load the ciphertext containing the encrypted flag.
    - **Details**:
        - The ciphertext dc91974098a61b4d444ad6ecfab17cc9ef5eeb25ca583b91cef51e373abc982babf9213ac8431ac91bc3c3ae788084623897bf2b5ecc592b2ca92fb2b973e015 is converted to bytes.
        - This ciphertext was likely obtained from /show_encrypted_notes and represents the encrypted flag.
    - **Output**: The ciphertext as a byte string.

Step 12: Decrypt the Flag

- **Command**: unpad(strxor(key, ct), 64)
- **Description**:
    - **Purpose**: Decrypt the flag ciphertext to recover the flag.
    - **Details**:
        - The key (from Step 10) is XORed with the flag ciphertext (ct) to obtain the padded plaintext: padded_plaintext = key XOR ciphertext.
        - strxor(key, ct) performs the XOR operation.

        - unpad(..., 64) removes the PKCS#7 padding, assuming a 64-byte block size, to reveal the original plaintext (the flag).
        - The result is decoded to a string, yielding the flag.
    - **Output**: The flag: 7CH62e3EXZrbkI42.

Final Answer

- **Flag**: 7CH62e3EXZrbkI42