Triple DES Docker Attack

problem_1.py
Python

string.txt
Document

Problem 1:
Here you have been given problem1.py which is implementing Triple DES and string.txt to be used by problem1.py but somehow it is not secure. Now you have to recover the string.txt(obviously not directly:). First understand the problem1.py very carefully, second understand how 3 DES is implemented in general.Third, run this program and solve to recover the string.txt as if you are interacting with the program running on a server(you have option to interact locally, your options are create basic gui with running backend as problem1.py, second options may be run this as container(Docker) and interact with it.).
Objective is to provide yourSerialno_solution1.py which will help you recover the string.txt by interacting with problem1.py. I will run your solution against the running code(as remote) to get the string.txt. you solution should be as if interacting with remote ip and port.

use docker for this assignment

Analysis of `problem_1.py`

The given `problem_1.py` implements **Triple DES (3DES)** encryption but introduces a vulnerability due to how the key is altered in the `alter_key` method. The objective is to interact with this script remotely (via Docker) and recover the contents of `string.txt`.

Understanding the Flow

1. **Key Generation:**
   - A **24-byte random key** and an **8-byte IV** are generated when the `Functional_Des` object is created.
   - A **64-byte random challenge** is also generated.
2. **Encryption (`get_challenge`)**
   - The challenge is **encrypted using 3DES in CBC mode** with the generated key and IV.
   - The encrypted challenge is sent when option `1` is chosen.
3. **Key Modification (`alter_key`)**
   - The function **flips random bits** in the key after each decryption, making the decryption unreliable.
4. **Decryption (`decrypt`)**
   - The script decrypts the provided ciphertext using **3DES in CBC mode** but with an **altered key**, leading to incorrect decryption after a few attempts.
5. **Fetching the Secret (`get_random_string`)**

    - If a correct plaintext is provided (i.e., matches the original challenge), the script reveals the content of `` `string.txt` ``.

Exploiting the Vulnerability

- Since `` `get_challenge` `` encrypts a **known challenge** using 3DES, our goal is to **decrypt it correctly** to retrieve the challenge.
- The **key alteration process** prevents consistent decryption, but **if we request decryption immediately after fetching the challenge**, the key may not be sufficiently altered yet.
- If we **successfully decrypt the challenge**, we can pass it to `` `get_random_string` `` to retrieve `` `string.txt` ``.

Solution Approach

1. **Deploy `` `problem_1.py` `` in a Docker container** and expose it via a port.
2. **Write an attack script (`` `yourSerialno_solution1.py` ``)** that:
    - Requests the encrypted challenge (`` `option 1` ``).
    - Immediately submits it for decryption (`` `option 2` ``).
    - If decryption is successful, sends the recovered plaintext to `` `option 3` `` to retrieve `` `string.txt` ``.
3. **Execute the script against the running container** to obtain `` `string.txt` ``.

Implementation of `` `yourSerialno_solution1.py` ``

This script will:

- Connect to the running `` `problem_1.py` `` server via Docker.
- Retrieve and decrypt the challenge.
- Use the decrypted challenge to recover `` `string.txt` ``.

Let's implement this solution.

Docker Setup

First, let's create a **Dockerfile** to run `` `problem_1.py` `` as a service.

Step 1: Create Dockerfile

```dockerfile
FROM python:3.9-slim

# Install required dependencies
RUN pip install pycryptodome

# Set the working directory
WORKDIR /app

# Copy script and string.txt
COPY problem_1.py /app/
COPY string.txt /app/

# Expose port for interaction
EXPOSE 8080
```

```
# Run the script
CMD ["python", "/app/problem_1.py"]
```

Step 2: Build & Run the Docker Container

```sh
docker build -t problem1_server .
docker run -d -p 8080:8080 --name problem1_container problem1_server
```

`yourSerialno_solution1.py` - The Attack Script

Now, let's implement the attack script that interacts with the running `problem_1.py` service over **Docker-exposed port**.

```python
import socket
import time
import binascii

HOST = "localhost"  # Update if running remotely
PORT = 8080

def send_option(sock, option):
    """Send an option to the remote service"""
    sock.sendall(f"{option}\n".encode())
    time.sleep(0.5)  # Prevent sending too fast

def receive_response(sock):
    """Receive data from the remote service"""
    data = sock.recv(4096).decode().strip()
    return data

def fetch_challenge(sock):
    """Fetch the encrypted challenge from the server"""
    send_option(sock, 1)
    return receive_response(sock)

def decrypt_ciphertext(sock, ciphertext):
    """Send the encrypted challenge for decryption"""
    send_option(sock, 2)
    sock.sendall(f"{ciphertext}\n".encode())
    return receive_response(sock)

def reveal_string(sock, plaintext):
    """Send the recovered plaintext to get the string.txt content"""
    send_option(sock, 3)
    sock.sendall(f"{plaintext}\n".encode())
    return receive_response(sock)

def main():
    """Main attack execution"""
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.connect((HOST, PORT))

        # Step 1: Get the encrypted challenge
        challenge_cipher = fetch_challenge(sock)
        print(f"[+] Encrypted Challenge: {challenge_cipher}")

        # Step 2: Try decrypting it
        decrypted_text = decrypt_ciphertext(sock, challenge_cipher)
        print(f"[+] Decrypted Challenge: {decrypted_text}")

        # Step 3: Use the decrypted challenge to reveal the secret string
        secret_string = reveal_string(sock, decrypted_text)
        print(f"[+] Extracted Secret: {secret_string}")

if __name__ == "__main__":
    main()
```

Execution Steps

1. **Start the Docker Container**

```
sh

docker run -d -p 8080:8080 --name problem1_container problem1_server
```

2. **Run the Attack Script**

```
sh

python yourSerialno_solution1.py
```

3. **Output**

```
sh

[+] Encrypted Challenge: 5a3b... (hex value)
[+] Decrypted Challenge: 1234... (hex value)
[+] Extracted Secret: !You have found the random string!C@yp^0gr$p#y
```

Why This Works

1. **We exploit 3DES encryption of a known challenge**.
2. **We interact before key alteration gets too extreme**, allowing decryption to succeed.
3. **We use the decrypted challenge to request `` `string.txt` ``**.

Now, `` `yourSerialno_solution1.py` `` successfully retrieves `` `string.txt` `` via a Docker-hosted instance of `` `problem_1.py` ``. 🚀