Reverse Engineering Challenge Write-Up

Step 1: Perform Initial Reconnaissance

- **Tools Used**: `strings`, `file`, `ltrace`, `strace`

Command:
file challenge_binary
strings challenge_binary

- **Purpose**:
  - Identify the type of binary (e.g., ELF, PE).
  - Extract any useful strings from the binary to identify key components, such as the `salt` or function names.

Step 2: Static Analysis

- **Tool Used**: `Ghidra`, `IDA Free`, or `radare2`
- **Procedure**:
  1. Load the binary into the tool.
  2. Identify the `main` function and trace the flow of the program.
  3. Locate the `derive_key` function and analyze the hashing mechanism.

Key Observations:

- The `salt` is a hardcoded string (`d1ff1cuLt_5aLt`).

A simple hash computation is performed:
hash = hash * 31 + salt[i];

-
- The `hash` is used to seed the random number generator (`srand(hash)`), which determines the derived key.

Step 3: Dynamic Analysis

- **Tool Used**: `gdb` or `ltrace`
- **Procedure**:
  1. Run the binary under a debugger.
  2. Break at the `strcmp` function to capture the derived key in memory.
  3. Monitor the function calls to understand the decryption process.

Key Commands:

```
gdb ./challenge_binary
break strcmp
run
info registers
x/s <memory_address>
```

Step 4: Derive the Key Manually

If dynamic analysis is not preferred, replicate the `derive_key` logic:

Python Script:

```
salt = "d1ff1cuLt_5aLt"
hash = 0
for char in salt:
    hash = hash * 31 + ord(char)

# Seed random and generate key
import random
random.seed(hash)
key = ''.join(chr(random.randint(97, 122)) for _ in range(20))
print("Derived Key:", key)
```

Step 5: Decrypt the Flag

Once the key is obtained, decrypt the flag using the Caesar cipher:

Python Script:

```
encrypted_flag = "*******"
key_length = len(key)
flag = ""
for char in encrypted_flag:
    if 'a' <= char <= 'z':
        flag += chr((ord(char) - ord('a') - key_length + 26) % 26 + ord('a'))
    elif 'A' <= char <= 'Z':
        flag += chr((ord(char) - ord('A') - key_length + 26) % 26 + ord('A'))
    else:
        flag += char
print("Flag:", flag)
```

Final Flag

After running the decryption script, the flag is revealed.