

Initiale Analyse

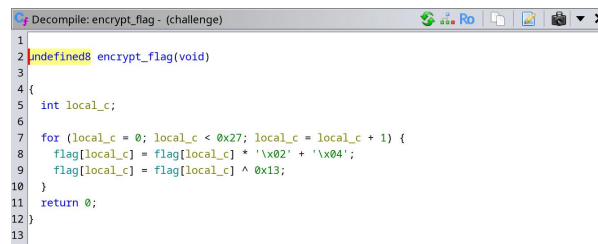
Die Challenge Beschreibung lässt uns wissen, dass die Flagge mit dem Programm verschlüsselt wurde. Die verschlüsselte Flagge erhalten wir als Datei `flag.enc`.

Das Ziel wird es wohl sein, die Flagge zu entschlüsseln. Um das zu tun, müssen wir aber erstmal herausfinden, **wie** sie verschlüsselt wurde.

Der Challenge Hint weist uns darauf hin, dass wir versuchen sollten, das Programm zu dekompilieren, was mit Ghidra getan werden kann.

Decompilation

Wenn wir das Programm in Ghidra laden, sehen wir in der Auflistung aller Funktionen, eine interessante Funktion namens `encrypt_flag`. Wählen wir diese Funktion aus, sehen wir folgendes:



```
1
2 undefined8 encrypt_flag(void)
3
4 {
5     int local_c;
6
7     for (local_c = 0; local_c < 0x27; local_c = local_c + 1) {
8         flag[local_c] = flag[local_c] * '\x02' + '\x04';
9         flag[local_c] = flag[local_c] ^ 0x13;
10    }
11    return 0;
12 }
13
```

Figure 1: Decompilation der Verschlüsselungsfunktion

Hier sehen wir, dass in einer Schleife insgesamt 0x28 mal dieselben Operationen auf ein Array namens `flag` ausgeführt werden.

Am Ende jeder Schleifeniteration, wird das Offset in das Array `flag` um eins erhöht. Es scheint also so, als ob jedes Byte der Flagge folgendermaßen verändert wird:

```
flag[i] = (flag[i] * 2 + 4) ^ 0x13
```

Wobei, \wedge die XOR-Operation ist. Jedes Flaggenbyte wird also mit 2 multipliziert, eine 4 raufaddiert und anschließend mit 0x13 “XOR genommen”.

Lösung

Um nun den ursprünglichen Wert zu erhalten, müssen wir die inversen Operationen in umgekehrter Reihenfolge anwenden. Um die XOR-Operation rückgängig zu machen, können wir es erneut mit 0x13 “XOR nehmen”. Um die Addition mit 4 rückgängig zu machen, subtrahieren wir 4. Und um anschließend die Multiplikation mit 2 rückgängig zu machen, dividieren wir durch 2.

Mit folgendem Python Skript, können wir die Aufgabe lösen:

```
# Einlesen der verschlüsselten Flagge
encrypted_flag = open("../dist/flag.enc", "rb").read()

# Anlegen der Liste für die entschlüsselten Bytes
decrypted_flag = []

# Anwenden der inversen Operation in umgekehrter Reihenfolge
for i in range(len(encrypted_flag)):
    result = ((encrypted_flag[i] ^ 0x13) - 4) // 2
    decrypted_flag.append(result)
```

```
# Umwandeln der entschlüsselten Zahlen in Buchstaben und Ausgabe der Flagge
print("".join(chr(byte) for byte in decrypted_flag))
```

Ausführen des Programms liefert die Flagge:

```
$ python solve.py
SSH{r3v3rs3d_m4lw4r3_f0r_fUn_n_pr0f17}
```