Angy Tux Solution

Aleknight

November 22, 2019

Step 1

The executable is truncate. I use ida to open it. See figure 1.

Step 2

We see first that we have a function to print on stdout at the offset 0x4000F0

Then we observe that at the offset 0x4000CE, we refer to the string Congrats, also the function called before is to check the flag.

Step 3

Then we see a function to read on stdin at 0x400105 and a cipher function 0x400133. This function made first a xor with the value 0x38, then a substraction with the entrypoint value, then a xor with the value 0x7F then an addition with the keyword 'ELF'

Step 4

We have to understand the last function that we call five times. This function add two consecutive values of the ciphered entry.(input[i] = input[i] + input[i+1]) It's a compression function let's see the evolution between the input and the end of this function

We compare 8 bytes.

So we obtain a system of equation. $\begin{cases} \sum_{k=0}^{5} \binom{5}{k} x_k \equiv f_1[256] \\ \vdots \\ \sum_{k=0}^{5} \binom{5}{k} x_{k+8} \equiv f_8[256] \end{cases}$

Step 5 However we know that $x_0 ... x_4$ is 'GH19{', so we can decompress the cipher flag. Then we can just reverse the ciphering function to obtain the clear flag. See the file exploit.py to get the reverse algorithm

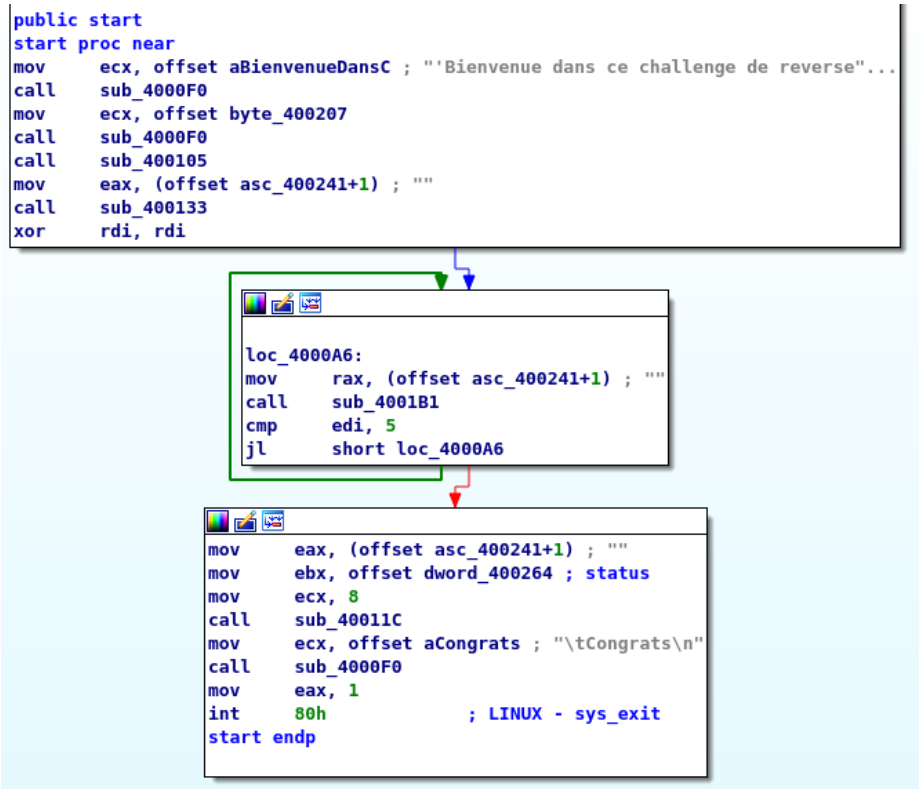

Figure 1: IDA view at start

```
mov     ecx, offset aCongrats ; "\tCongrats\n"
call    sub_4000F0
```

Figure 2: The call to write congrats on stdout

```
20 47 48 31 39 7B 6D  79 70 61 73 73 7D 0A 00  .·GH19{mypass}..
```

Figure 3: the clear input

```
C5 5B FC 43 88 B0  03 83 AC F9 80 C0 12
```

Figure 4: the ciphered input

```
5A A8 0D 24 89 F2  44 6B 00 00 00 00 00 00
```

Figure 5: the compressed input

Figure 6: the compare call