Assembly and CTF

Amit Yadav

June 1,2017

1 CTF problems

- s3cr3T1
- find_M3
- R0X
- Th1$_$h1T

2 Solutions

Started with R0X problem,I tried to read the dissassembled code but couldn't get any clue what the code the was doing.This link helped a lot in getting started.So I picked 's3cr3T1' problem.

2.1 s3cr3T1

2.1.1 Steps

dissassemble s3cr3T1
First disassemble the code.
break *main+2
Add a break point at main+2 line.
Step through the code and try to understand what the code was doing and how it was processing the input.It asks for a password ,and no matter what ,it will always print *'You are still a n00b.'* So we look at the strings stored in the memory. And ,we find a string **'mY_f1R5t_h4cK'**. This happens to be our flag.

2.2 find_M3

When executed,always prints:
'Can't you find the flag?? Use GDB kid ;)'

2.2.1 Steps

Open the file with gdb and disassemble the main function.
disas main
We find out that it is copying some characters in 'rbp-0x..' . When we examine those memory adresses ,
x/100s $rbp-0xa0
we find a string **'Ea5Y_fl4g'**. This is our flag. :)

2.3 R0X

This requires a knowledge of working of XOR nmeonic. XOR takes bitwise XOR of binary representation of arguments. Let's say we need to find XOR of 5 and 4.First convert them in binary:101 and 100.

$$\begin{array}{ccc} 1 & 0 & 1 \\ 1 & 0 & 0 \\ \hline 0 & 0 & 1 \end{array}$$

Thus the result is 1.

2.3.1 Steps

When executed,it prints *'What do I do?????'* and asks for an input which is in %lld format ('edi' tells us the format.We can find it by *x/s $edi*).Then we step through the code and we find that it is taking XOR of our input with *'0xabbebabe'* and then comparing the result with *'0xdeadbeef'*.
If the result is same as binary representation of *'0xdeadbeef'* then it prints *'Congrats the flag is the key you entered.'* And if the input is not as desired then it prints *'Git gud n00b'*. So in brief,we need to find a decimal input whose XOR with *'0xabbebabe'*,gives *'0xdeadbeef'*.For online Hex to Binary converter,click here.
Thus ,we get our flag : **1964180561**

2.4 Th1$_$h1T

2.4.1 Steps

Disassemble the code,and we see that it is copying the contents of memory at 0x400900 + ... to rbp-0x... .When we examine that memory
x/100s 0x400900
we find a string *'7_l1k3_y0u2b3'* .Our input is compared with this string and if it is incorrect it print *'Lol....!!! Password is in the link ;)'* and *'Congrats!! You got the flag'* if it is correct.But this not the flag.When we look over the next steps,which are executed only if our input is correct,we find a string at 0x400880 which reads **'m4Nd1R_w4h1_b4n4Y3ng3'**.This is our flag.