REV\_deobfuscation Challenge Writeup

Description:  
This reverse engineering (REV) challenge gives you a binary, written in assembly, that you need to analyze. The goal is to extract the flag.

1. Binary Analysis:  
   After downloading the binary, open it with Ghidra. Inside the program, look for the function that checks the user's password. You’ll see that the binary does not directly compare your input to a readable password. Instead, it uses obfuscation.
2. Data Arrays:  
   Before the password check, you’ll find two arrays in memory. At first glance, they don’t seem to be used. But they’re actually important. One is the XOR key and the other is the encoded password.
3. XOR Obfuscation:  
   The binary doesn’t store the correct password plainly. Instead, it XORs the correct password with a key to create a scrambled version. When you input a password, the binary XORs your input with the same key and compares the result to the stored scrambled version.
4. Extracted Data:  
   From the binary, we extract two arrays (0x40203 and 0x40209c)
5. Decoding Process:  
   By XORing each byte of the encoded password with the corresponding byte in the key, we recover the original password.
6. At first, you might recover a partially readable string like UMDCTF{r3v3R$E-i$\_Th3\_#B3ST#\_4nT!-M@lW@r3\_t3chN!WVd#, but some characters may appear corrupted or missing. This is because the password data must be changed to bytes or to byte array for view it completely . Once you change how the data is viewed, you will see the complete flag data. The following script performs the XOR operation between the encoded password and the key to reveal the full flag.