**Reverse engineering challenge**

**Elnaz Koopahi**

**Ek2833@nyu.edu**

I wrote a C++ code. The code takes one integer and a string.

```
#include <iostream>
using namespace std;

bool checkPass(string myString) {
    if (myString.length() != 32) {
        return false;
    }
    unsigned char buffer[32];
    std::copy(myString.begin(), myString.end(), buffer);

    int x[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };

    for (int i = 0; i < 8; i++) {
        x[i] = (buffer[i * 4] << 24) | (buffer[i * 4 + 1] << 16)| (buffer[i * 4 + 2] <<
8)| (buffer[i * 4 + 3]);
    }


    for (int i = 0; i < 8; i++) {
        cout << x[i] << "\n";

    }
    cout << "\n";
    bool y[8] = { (x[0] == 1751478885),(x[1] == 1598649439),(x[2] == 1769154085),(x[3] ==
1601777712) ,(x[4] == 1970429803) ,(x[5] == 1702453108), (x[6] == 1868527153) , (x[7] ==
859189802) };
    return y[0] & y[1] & y[2] & y[3] & y[4] & y[5] & y[6] & y[7];

    cout << "HeyHey\n";



    return false;
}

int main() {

    int room_number;
    cout << "Hello My Friend\n";
    cout << "What's Your Room Number:\n";
    cin >> room_number;

    if  ((room_number/84 +5 >22) || (room_number/84+ 5<20)) {
        cout << "See you later.\n";
        return 0;
    }
    string phrase;
    cout << "Say the phrase:\n ";
    cin >> phrase;

    if (checkPass(phrase)) {
        cout << "Welcome TO Your Vault\n";
    }
    else {
        cout << "Goodbye\n";
    }


    return 0;
}
```

---

The integer should be in a given range. The string is scrambled, and each 4 character of the string is converted to an integer and compared with a given value.

I build the project.

The binary file can be opened in Ghidra. It is not difficult to know the input types and the range for the integer. For the string, a python script can show the output.

---

```
x =[0] * 8


x[0] = 1751478885

x[1] = 1598649439

x[2] = 1769154085

x[3] = 1601777712

x[4] = 1970429803

x[5] = 1702453108

x[6] = 1868527153

x[7] = 859189802


ch = [None] * 4

buffer = ""


for i in range(0,8):

       tmp = str(bin(x[i])[2:].zfill(32))

       ch[0] = chr(int(hex(int(tmp[:8], 2)), 16))

       ch[1] = chr(int(hex(int(tmp[8:16], 2)), 16))
```

```
        ch[2] = chr(int(hex(int(tmp[16:24], 2)), 16))
        ch[3] = chr(int(hex(int(tmp[24:32], 2)), 16))
        buffer += ch[0] + ch[1] + ch[2] + ch[3]

        print(buffer)
```

The flag is here_It_is&%_y00ur_key_to_v1362*.

For making more complicated, the code for scrambling can be stored within the binary, obfuscated.  There should be a reference to some data label in whatever decompiler you are using.  In other words, there should be a reference to a long block of data that gets put into memory (using a cousin of alloc), de-obfucated, and then ran as code using the pointer to where the data was put into memory.