# WRITEUP – Into to Reversing Final Assignment

Submitted by:

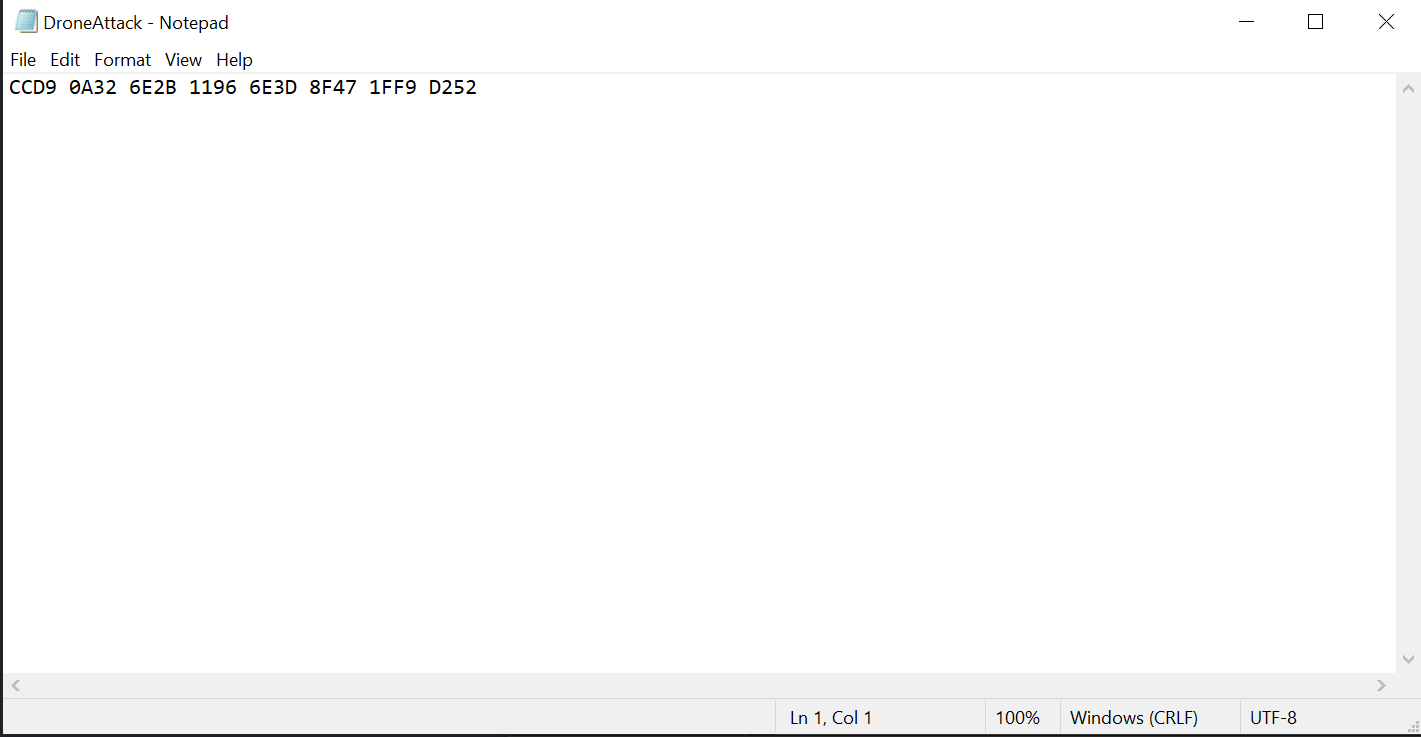
* Daniel Attali 328780879

## Stage 1:

The first part was just to run the .exe we were given and we saw printed an image and we were presented with a scenario, the second thing we did was to open the file using IDA and try to find anti debug behaviors in the tls\_callback since we didn’t find anything (yet) we went and found the \_\_main function and try to find and rename basic functions, like the function who takes a string and print it to the screen using a typewriter effect.

After Identify those basic functions we started working on the functions we found in the main, the first function seems to be taking a string has input and for each byte XORing it with 0x2A and then again XORing it with 0x2A again meaning this function didn’t modify the string in any way, so we looked into the next call in the main function and found another function and with in it a call to what we called stage\_1\_main, in the stage\_1\_main we found the following, first the function create a string “C:\ReversingCTF” and another string “DroneAttack.txt” and from our understanding of this function it tried to open the file and if not found printed an error message, so we created the file and when we ran the program again we saw the program had done 2 things:

1. Written into the .txt file
2. Added a AttackIRGC.dll file into the directory



After this it is had printed to the screen the success message and we are done with stage 1.

Stage 1: You are a special operations expert.

Your mission is to protect our pilots. Disable the anti aircraft system

Oh, intelligence report says the enemy spread decoys, find the real target, fast!

Anti aircraft system located

Intiating disable sequence

Great job. Anti aircraft system is disabled

Stage 2: You are a jet fighter pilot. The sky is clear. Your mission: release bombs on IRGC headquarters.

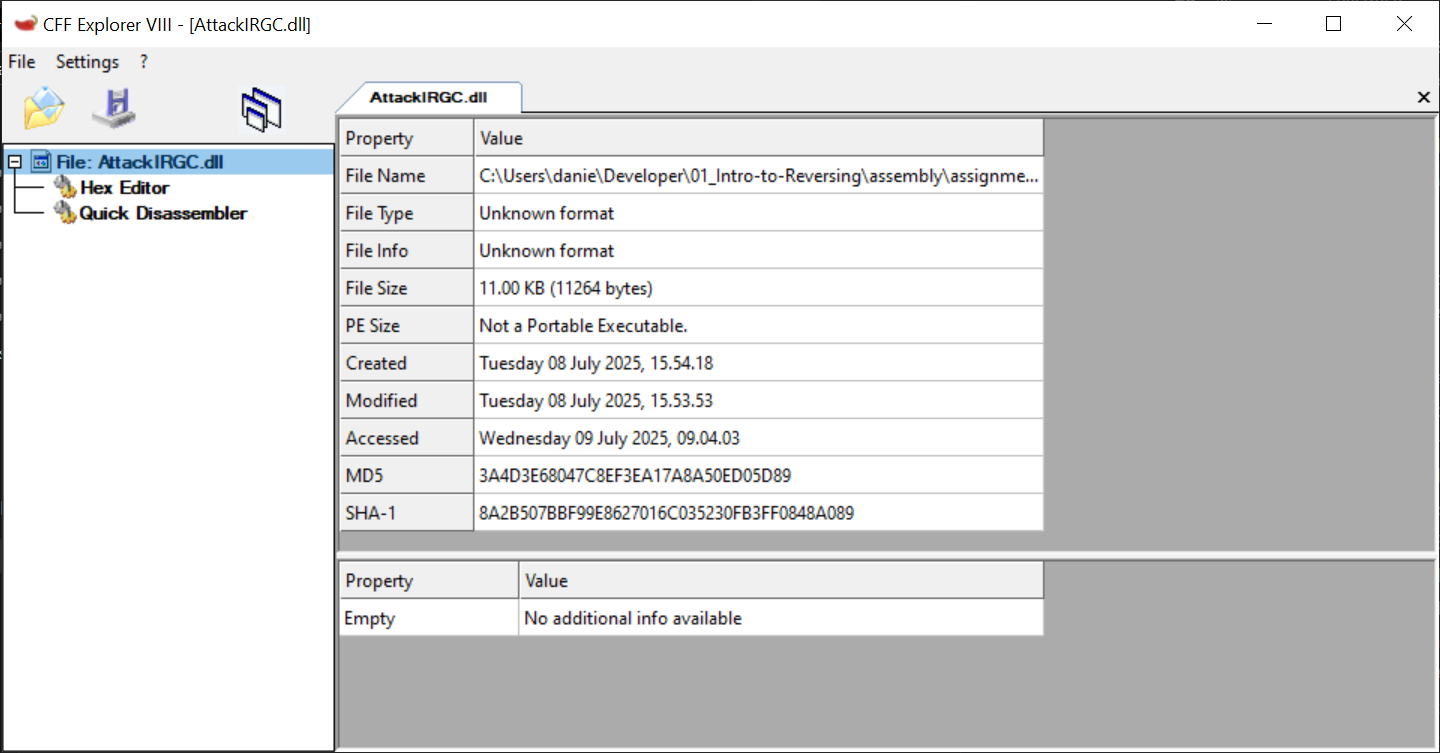
To find them, use the cyber intelligence unit

## Stage 2:

In Stage 2 of the CTF, the challenge centered on reversing a corrupted DLL file, AttackIRGC.dll, in the context of a fictional cyber operation. The goal was to demonstrate an understanding of Windows PE file structures, binary analysis, and custom obfuscation techniques.

**Initial Analysis:**

Upon progressing to Stage 2, the executable (`OperationLion.exe`) generated a suspicious DLL file, `AttackIRGC.dll`. Opening this file in CFF Explorer resulted in an error, indicating that the file was not a valid PE format.



I then compared the corrupted `AttackIRGC.dll` with a known-good DLL file using a hex editor.

A screenshot of a computer

AI-generated content may be incorrect.

* The valid DLL started with the standard PE header (`MZ` or `0x4D 0x5A`).
* The corrupted DLL began with seemingly random bytes and regular occurrences of ASCII characters spelling "BOMB".

**Hypothesizing the Corruption Pattern**:

At first, I attempted to repair the file by simply removing all ASCII occurrences of "BOMB" or by substituting bytes to match a valid PE header. However, these attempts failed—the file remained invalid.

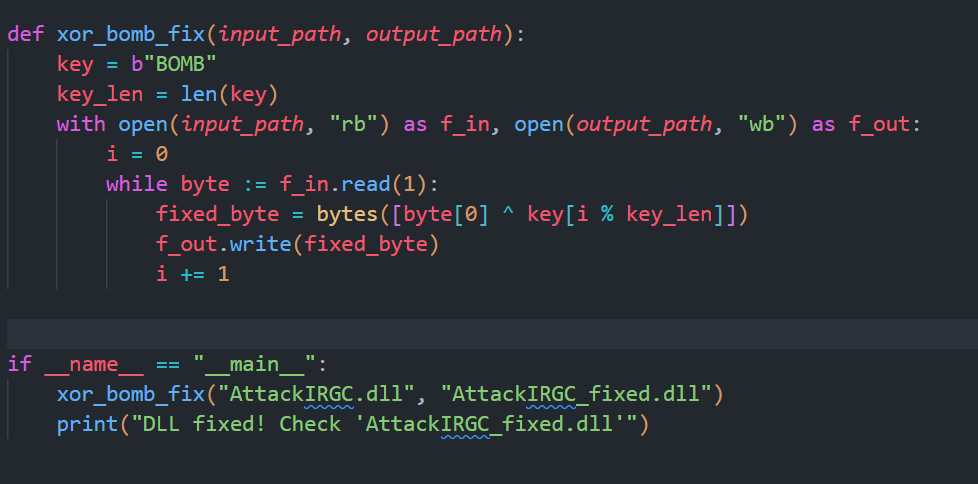
This led me to a byte-by-byte comparison. I noticed that every group of four bytes, starting from the beginning, aligned with the ASCII codes for "B", "O", "M", and "B", repeatedly.

* Example:
  + Good DLL: `4D 5A 90 00 03 00 00 00` (MZ....)
  + Corrupted: `0F 15 DD 42 4F 4D 42 46`

I hypothesized the file was XOR-encrypted with the repeating pattern "BOMB".

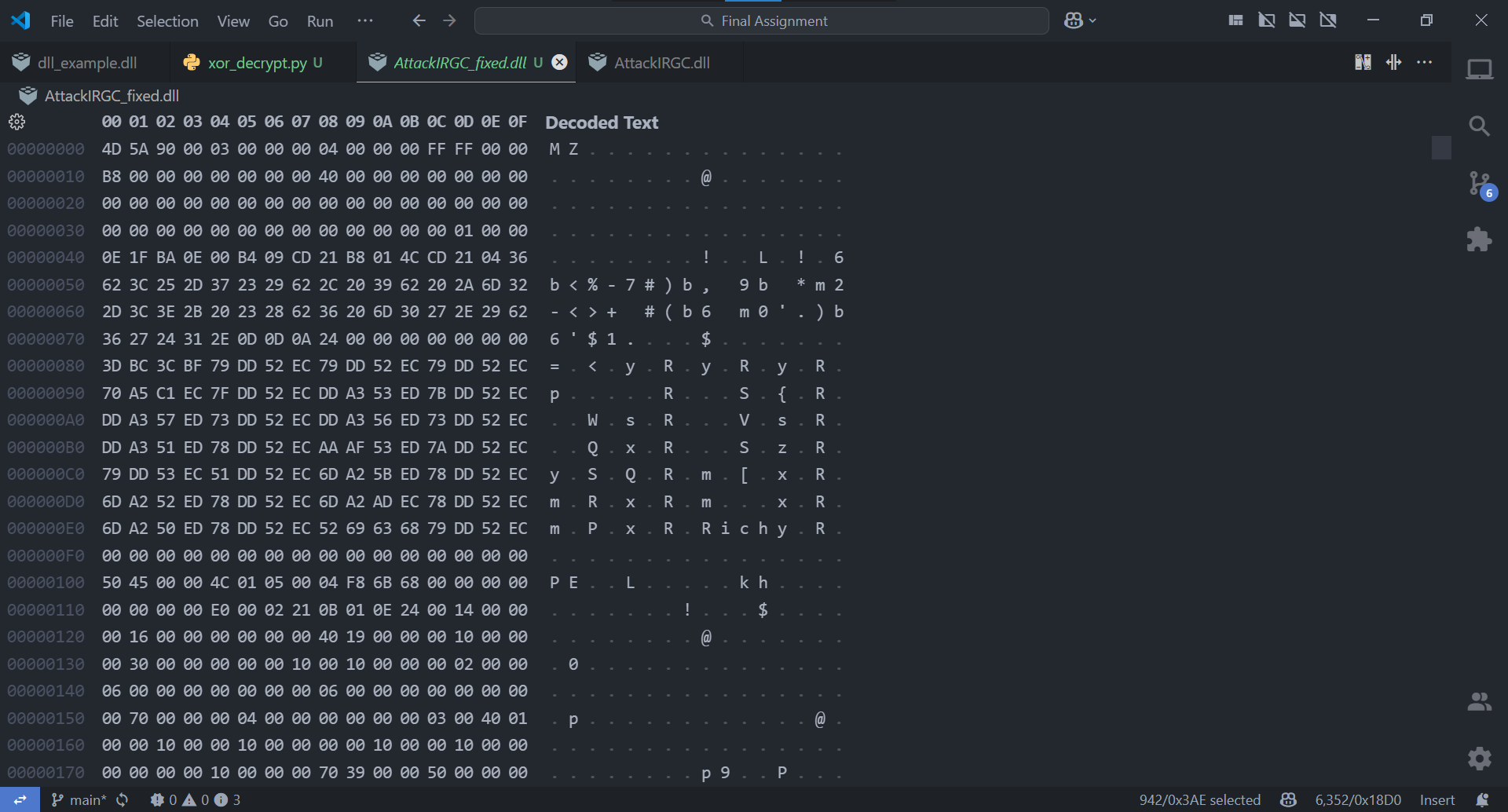
**Writing the Repair Script:**

I wrote a Python script to XOR each byte of the corrupted DLL with the repeating "BOMB" key:



**Verifying the Solution:**

After running the script, the output file (`AttackIRGC\_fixed.dll`) was a valid PE file. Opening it in CFF Explorer and a hex editor confirmed the restoration of the correct PE header.



Additionally, inspecting the file revealed a hidden congratulatory message:

“Dear student, You reached the end. I am proud of you. Not many can do that.”

A screenshot of a computer

AI-generated content may be incorrect.

“Dear student, You reached the end. I am proud of you. Not many can do that.”

“I am proud of you. Not many can do that.”

“This was only a game, but parts of the real operation were based on the knowledge that you learned.”

” I believe that you are part of the technological edge that keeps us here”

“I wish that you do great things in security, economy, technology and education”

“Pilot, you missed the target. Use your SIGINT to find it”

“Let's remember the hostages and wish all of them will be home soon. Amen.”

“Great job pilot, bombs hit IRGC.”

“Stage 3: Welcome cyber specialist.

Your mission : Penetrate the security system of the supreme leader.

The location of the enriched Uranium is stored there.”

“Your country depends on your skills. We COUNT on you. Good luck.”

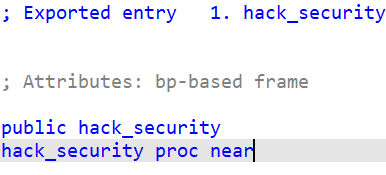
“C:\ReversingCTF\DroneAttack.txt”

“Enter code”

“Wrong code”

“Great work hero, you hacked the system. Prepare for a message from your instructor”

So the next step was to try and run the program (OperatoinLion.exe) with the corrected dll, it didn’t work (it kept overwrite the dll), so the next thing we tried was to write a simple C++ code that would load the dll and call the entry point (we found the entry point when using IDA on the dll



So we tried to run the program and got the following:

A screenshot of a computer

AI-generated content may be incorrect.

So we tried to understand the problem in the program

A computer code with text

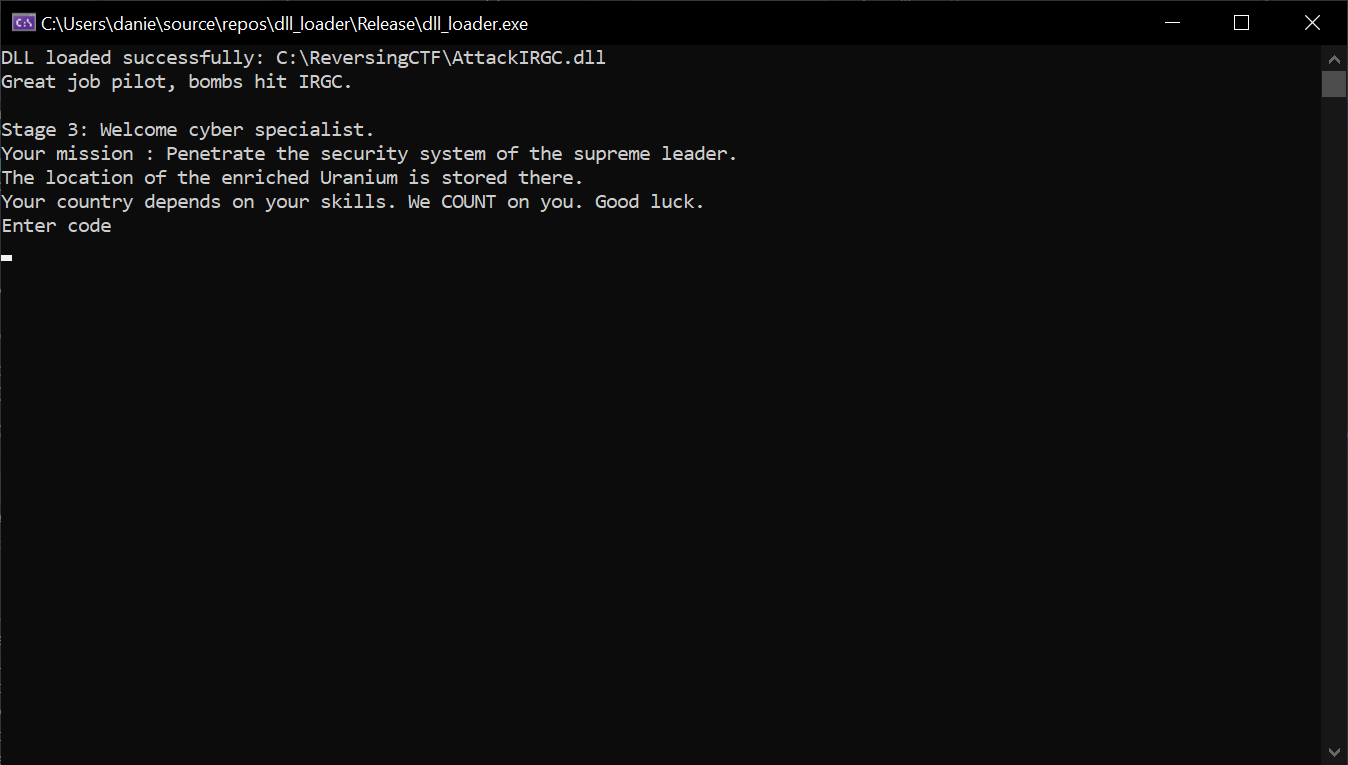
AI-generated content may be incorrect.

So we understood that the entry point is a function with the following signature:

void func(int);

So we gave as input the value of 0x2008 which is equal to 8200 (cyber security unit – the hint)

We tried running it again and got the correct screen.



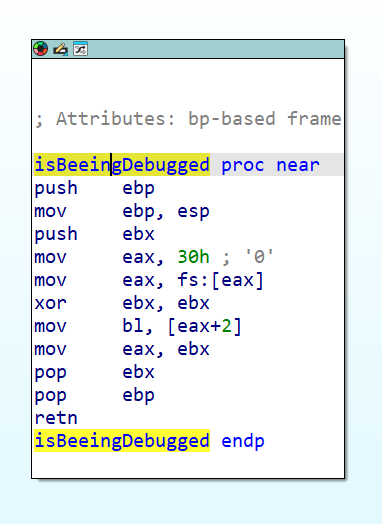
The program:

A screen shot of a computer program

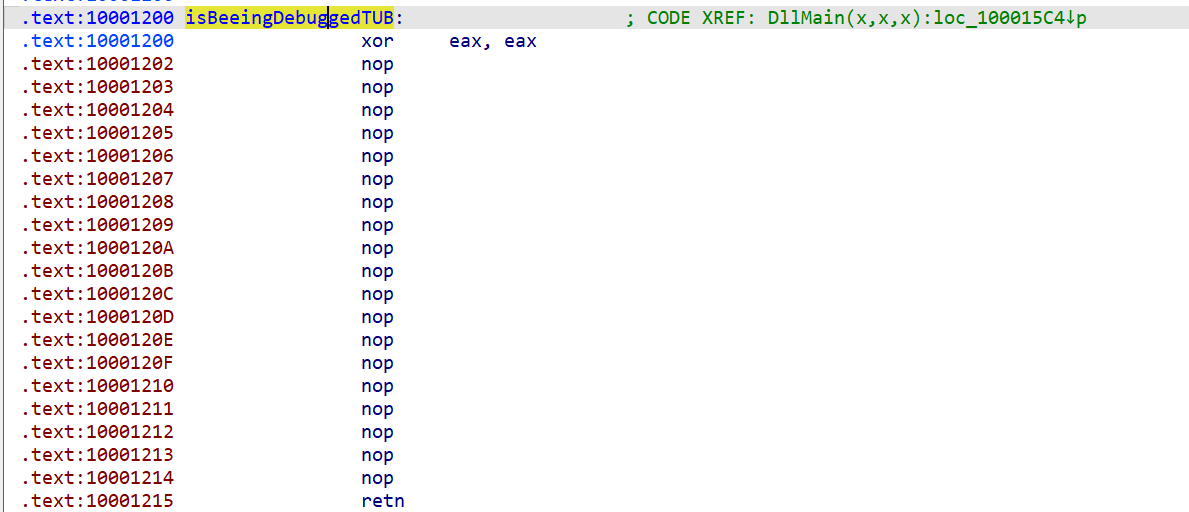
AI-generated content may be incorrect.

## Stage 3:

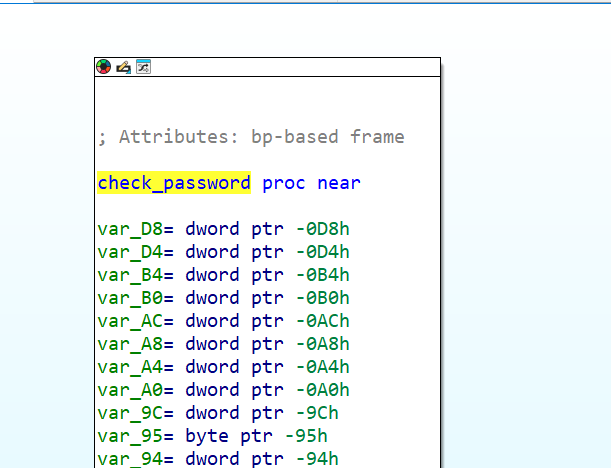
The first step of stage 3 was like stage 1, renaming the functions and finding and disabling anti-debug functions.



Patched with:



The second part was to see where to find the function that check the answer



Which is called in the entry point functions we call from our program.

The second part was to statically analyze the functions and after a lot of work we found the following:

1. The program waits for a string input
2. We check the length and compare it to 8 (answer of 8 char)
3. The string is then separated into char
4. The program check with ds:isdigit to check if the string contains only digits
5. If not, we exit loop
6. If it contains only number between 1-8
7. It then checks if the number is repeating meaning if we use 1 we can’t reuse it (so the answer is a permutation of 1-8)
8. Then we use an array (that values are revealed only using dynamic analysis) and use the input as indices and check array[i] < array[i-1]

I have written the functions in C++ to help me understand it better:

// The function returns 1 if password is valid, 0 otherwise

int check\_password() {

    char input[16] = {0}; // allocate a bit more than 8 for safety

    int used[10] = {0};   // tracks if a digit 1-8 was used

    int vals[8] = {0};    // stores each digit (as int) in order

    int result = 1;

    // 1. Read input

    printf("Enter password: ");

    if (scanf("%8s", input) != 1) return 0;

    // 2. Check length

    if (strlen(input) != 8) return 0;

    // 3. Check all chars are digits 1-8 and no repeats

    for (int i = 0; i < 8; ++i) {

        if (!isdigit(input[i])) return 0;

        int digit = input[i] - '0';

        if (digit < 1 || digit > 8) return 0;

        if (used[digit]) return 0; // already used

        used[digit] = 1;

        vals[i] = digit - 1; // store 0-based for table lookup

    }

    // 4. Validate permutation order using lookup table

    for (int i = 1; i < 8; ++i) {

        int prev\_val = dword\_740F4374[vals[i-1]];

        int curr\_val = dword\_740F4374[vals[i]];

        if (curr\_val >= prev\_val) return 0; // must be strictly decreasing

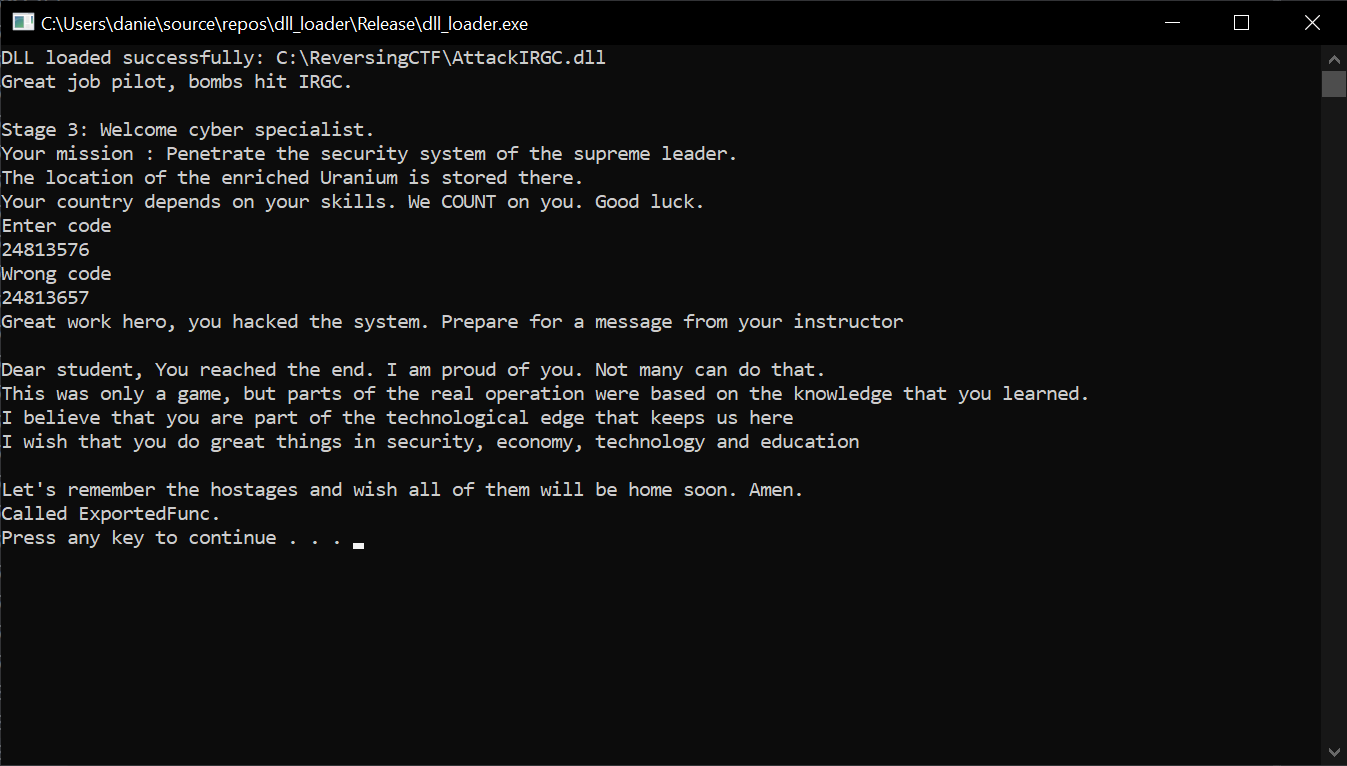
    }

    return 1; // password is valid!

}

So after understanding that we just had to find the permutation such that the condition is true, so we did that and the answer is

24813657



Thank you very much this was fun!!