WRITEUP – Into to Reversing Final Assignment

Submitted by:

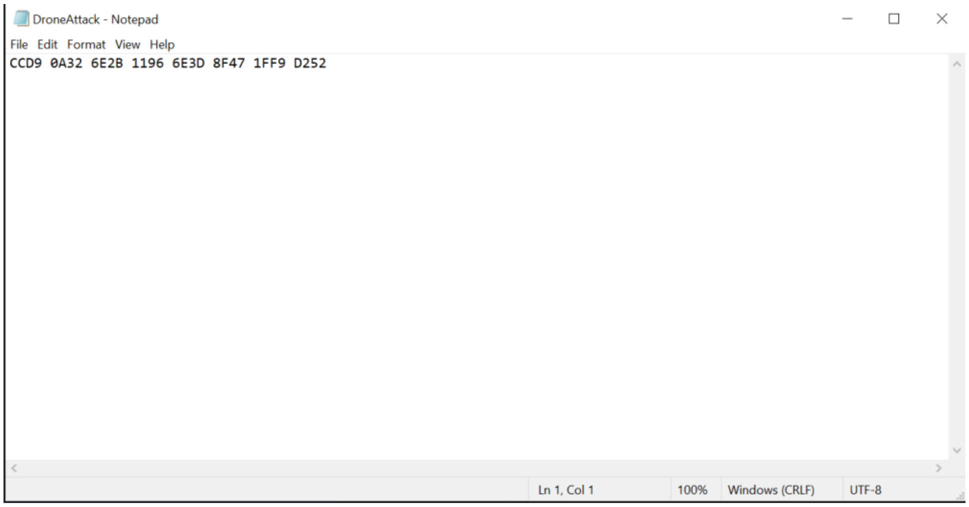
- Daniel Attali 328780879

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



```
CCD9 0A32 6E2B 1196 6E3D 8F47 1FF9 D252
```

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

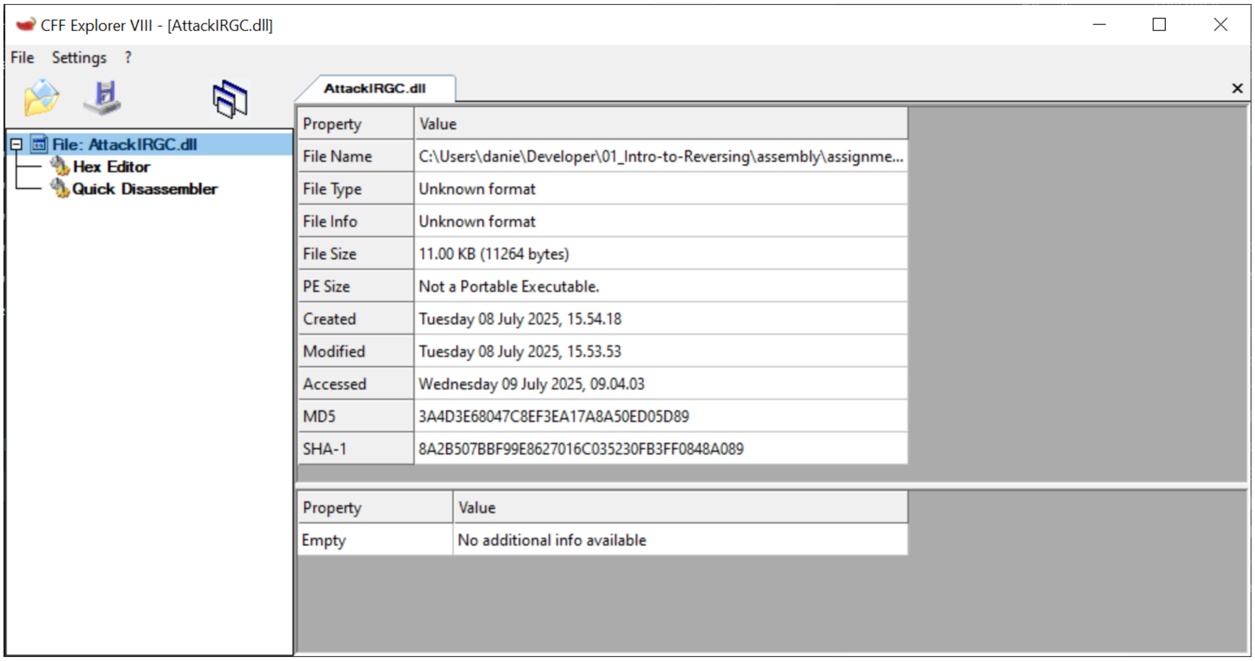
```
Stage 1: You are a special operations expert.

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:

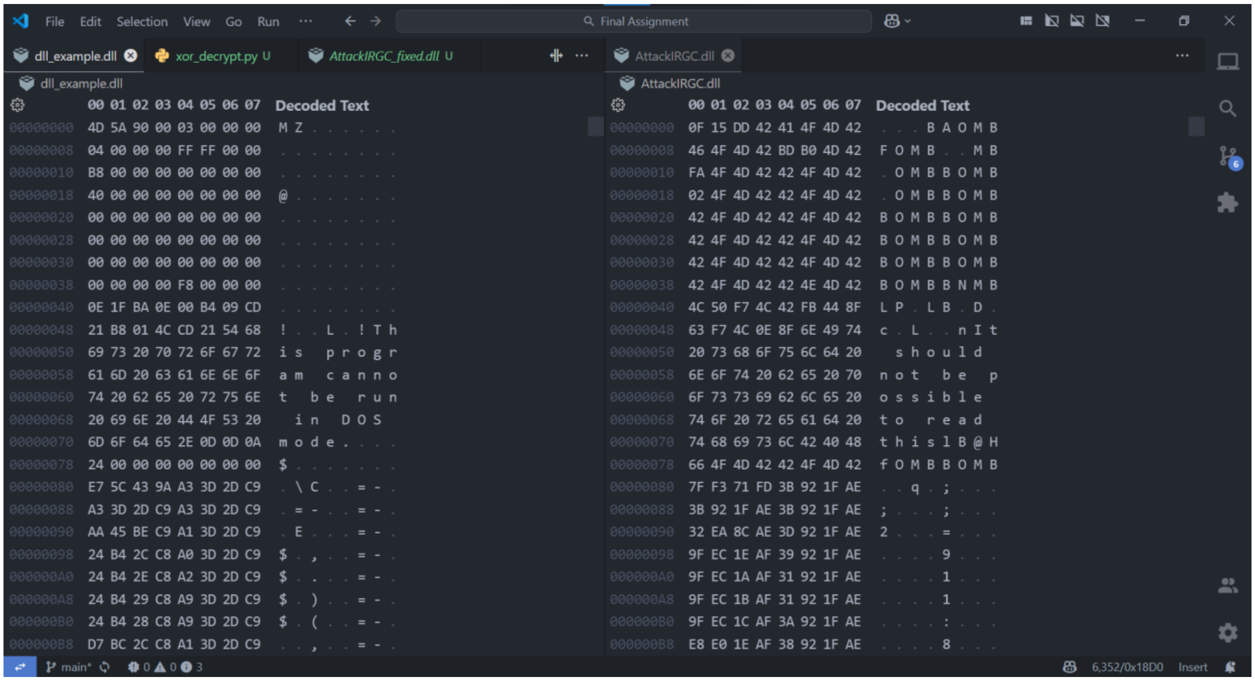
Upon progressing to Stage 2, the executable (`OperationLion.exe`) generated a suspicious DLL file, `AttackIRGC.dll`. Opening this file in CFF Explorer resulted in an error, indicating that the file was not a valid PE format.



Property	Value
File Name	C:\Users\danie\Developer\01_Intro-to-Reversing\assembly\assignme...
File Type	Unknown format
File Info	Unknown format
File Size	11.00 KB (11264 bytes)
PE Size	Not a Portable Executable.
Created	Tuesday 08 July 2025, 15.54.18
Modified	Tuesday 08 July 2025, 15.53.53
Accessed	Wednesday 09 July 2025, 09.04.03
MD5	3A4D3E68047C8EF3EA17A8A50ED05D89
SHA-1	8A2B507BBF99E8627016C035230FB3FF0848A089

Property	Value
Empty	No additional info available

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



```
dll_example.dll
          00 01 02 03 04 05 06 07  Decoded Text
00000000  4D 5A 90 00 03 00 00 00  M Z . . . . . .
00000008  04 00 00 00 FF FF 00 00  . . . . . . . .
00000010  B8 00 00 00 00 00 00 00  . . . . . . . .
00000018  40 00 00 00 00 00 00 00  @ . . . . . . .
00000020  00 00 00 00 00 00 00 00  . . . . . . . .
00000028  00 00 00 00 00 00 00 00  . . . . . . . .
00000030  00 00 00 00 00 00 00 00  . . . . . . . .
00000038  00 00 00 00 F8 00 00 00  . . . . . . . .
00000040  0E 1F BA 0E 00 B4 09 CD  . . . . . . . .
00000048  21 B8 01 4C CD 21 54 68  ! . . L . ! T h
00000050  69 73 20 70 72 6F 67 72  i s   p r o g r
00000058  61 6D 20 63 61 6E 6E 6F  a m   c a n n o
00000060  74 20 62 65 20 72 75 6E  t   b e   r u n
00000068  20 69 6E 20 44 4F 53 20    i n   D O S
00000070  6D 6F 64 65 2E 0D 0D 0A  m o d e . . . .
00000078  24 00 00 00 00 00 00 00  $ . . . . . . .
00000080  E7 5C 43 9A A3 3D 2D C9  . \ C . . = - .
00000088  A3 3D 2D C9 A3 3D 2D C9  . = - . . = - .
00000090  AA 45 BE C9 A1 3D 2D C9  . E . . . = - .
00000098  24 B4 2C C8 A0 3D 2D C9  $ . , . . = - .
000000A0  24 B4 2E C8 A2 3D 2D C9  $ . . . . = - .
000000A8  24 B4 29 C8 A9 3D 2D C9  $ . ) . . = - .
000000B0  24 B4 28 C8 A9 3D 2D C9  $ . ( . . = - .
000000B8  D7 BC 2C C8 A1 3D 2D C9  . . , . . = - .
```

```
AttackIRGC.dll
          00 01 02 03 04 05 06 07  Decoded Text
00000000  0F 15 DD 42 41 4F 4D 42  . . . B A O M B
00000008  46 4F 4D 42 BD B0 4D 42  F O M B . . M B
00000010  FA 4F 4D 42 42 4F 4D 42  . O M B B O M B
00000018  02 4F 4D 42 42 4F 4D 42  . O M B B O M B
00000020  42 4F 4D 42 42 4F 4D 42  B O M B B O M B
00000028  42 4F 4D 42 42 4F 4D 42  B O M B B O M B
00000030  42 4F 4D 42 42 4F 4D 42  B O M B B O M B
00000038  42 4F 4D 42 42 4E 4D 42  B O M B B N M B
00000040  4C 50 F7 4C 42 FB 44 8F  L P . L B . D .
00000048  63 F7 4C 0E 8F 6E 49 74  c . L . . n I t
00000050  20 73 68 6F 75 6C 64 20    s h o u l d
00000058  6E 6F 74 20 62 65 20 70  n o t   b e   p
00000060  6F 73 73 69 62 6C 65 20  o s s i b l e
00000068  74 6F 20 72 65 61 64 20  t o   r e a d
00000070  74 68 69 73 6C 42 40 48  t h i s l B @ H
00000078  66 4F 4D 42 42 4F 4D 42  f O M B B O M B
00000080  7F F3 71 FD 3B 92 1F AE  . . q . ; . . .
00000088  3B 92 1F AE 3B 92 1F AE  ; . . . ; . . .
00000090  32 EA 8C AE 3D 92 1F AE  2 . . . = . . .
00000098  9F EC 1E AF 39 92 1F AE  . . . . 9 . . .
000000A0  9F EC 1A AF 31 92 1F AE  . . . . 1 . . .
000000A8  9F EC 1B AF 31 92 1F AE  . . . . 1 . . .
000000B0  9F EC 1C AF 3A 92 1F AE  . . . . : . . .
000000B8  E8 E0 1E AF 38 92 1F AE  . . . . 8 . . .
```

- 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.

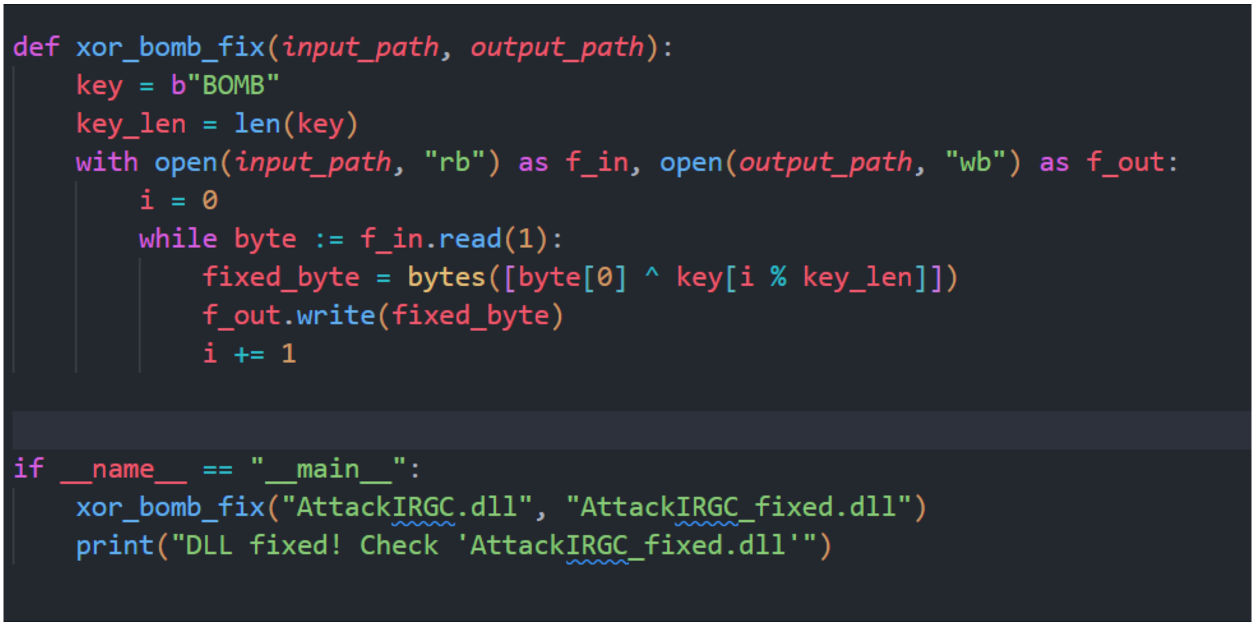
This led me to a byte-by-byte comparison. I noticed that every group of four bytes, starting from the beginning, aligned with the ASCII codes for "B", "O", "M", and "B", repeatedly.

- 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:

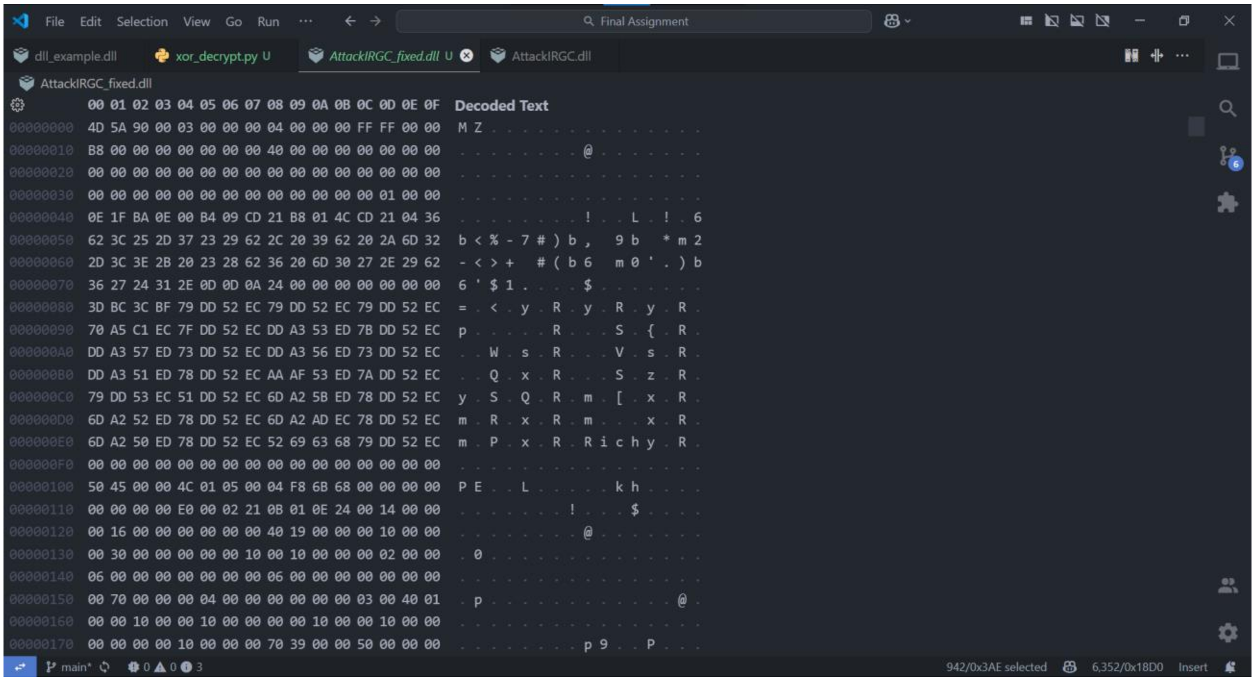
I wrote a Python script to XOR each byte of the corrupted DLL with the repeating "BOMB" key:



```
def xor_bomb_fix(input_path, output_path):
    key = b"BOMB"
    key_len = len(key)
    with open(input_path, "rb") as f_in, open(output_path, "wb") as f_out:
        i = 0
        while byte := f_in.read(1):
            fixed_byte = bytes([byte[0] ^ key[i % key_len]])
            f_out.write(fixed_byte)
            i += 1


if __name__ == "__main__":
    xor_bomb_fix("AttackIRGC.dll", "AttackIRGC_fixed.dll")
    print("DLL fixed! Check 'AttackIRGC_fixed.dll'")
```

Verifying the Solution:

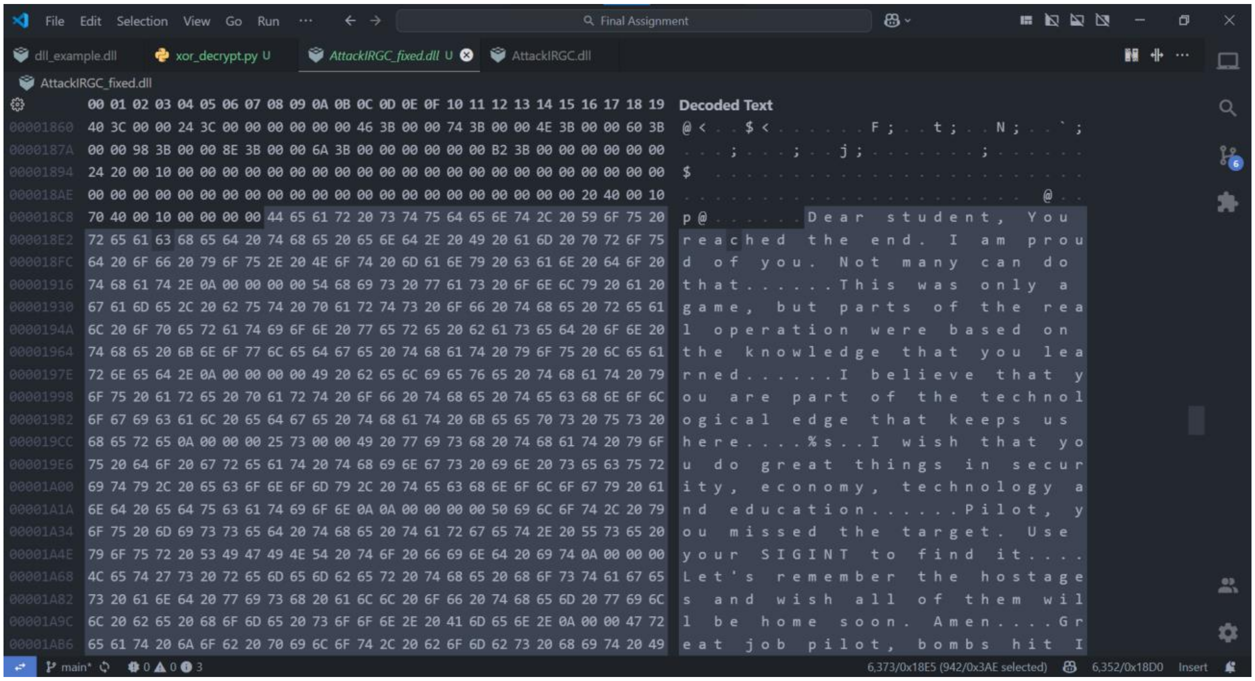
After running the script, the output file (`AttackIRGC_fixed.dll`) was a valid PE file. Opening it in CFF Explorer and a hex editor confirmed the restoration of the correct PE header.



```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded Text
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ........@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 04 36  ........!..L.!.6
00000050  62 3C 25 2D 37 23 29 62 2C 20 39 62 20 2A 6D 32  b<%-7#)b, 9b *m2
00000060  2D 3C 3E 2B 20 23 28 62 36 20 6D 30 27 2E 29 62  -<>+ #(b6 m0'.)b
00000070  36 27 24 31 2E 0D 0D 0A 24 00 00 00 00 00 00 00  6'$1....$.......
00000080  3D BC 3C BF 79 DD 52 EC 79 DD 52 EC 79 DD 52 EC  =.<.y.R.y.R.y.R.
00000090  70 A5 C1 EC 7F DD 52 EC DD A3 53 ED 7B DD 52 EC  p.....R...S.{.R.
000000A0  DD A3 57 ED 73 DD 52 EC DD A3 56 ED 73 DD 52 EC  ..W.s.R...V.s.R.
000000B0  DD A3 51 ED 78 DD 52 EC AA AF 53 ED 7A DD 52 EC  ..Q.x.R...S.z.R.
000000C0  79 DD 53 EC 51 DD 52 EC 6D A2 5B ED 78 DD 52 EC  y.S.Q.R.m.[.x.R.
000000D0  6D A2 52 ED 78 DD 52 EC 6D A2 AD EC 78 DD 52 EC  m.R.x.R.m...x.R.
000000E0  6D A2 50 ED 78 DD 52 EC 52 69 63 68 79 DD 52 EC  m.P.x.R.Richy.R.
000000F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000100  50 45 00 00 4C 01 05 00 04 F8 6B 68 00 00 00 00  PE..L.....kh....
00000110  00 00 00 00 E0 00 02 21 0B 01 0E 24 00 14 00 00  .......!...$....
00000120  00 16 00 00 00 00 00 00 40 19 00 00 00 10 00 00  ........@.......
00000130  00 30 00 00 00 00 00 10 00 10 00 00 00 02 00 00  .0..............
00000140  06 00 00 00 00 00 00 00 06 00 00 00 00 00 00 00  ................
00000150  00 70 00 00 00 04 00 00 00 00 00 00 03 00 40 01  .p............@.
00000160  00 00 10 00 00 10 00 00 00 00 10 00 00 10 00 00  ................
00000170  00 00 00 00 10 00 00 00 70 39 00 00 50 00 00 00  ........p9..P...
```

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.”



```
          00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10 11 12 13 14 15 16 17 18 19  Decoded Text
00001860  40 3C 00 00 24 3C 00 00 00 00 00 00 46 3B 00 00 74 3B 00 00 4E 3B 00 00 60 3B  @<..$<......F;..t;..N;..`;
0000187A  00 00 98 3B 00 00 8E 3B 00 00 6A 3B 00 00 00 00 00 00 B2 3B 00 00 00 00 00 00  ...;...;..j;.......;......
00001894  24 20 00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  $ ........................
000018AE  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 20 40 00 10  ...................... @..
000018C8  70 40 00 10 00 00 00 00 44 65 61 72 20 73 74 75 64 65 6E 74 2C 20 59 6F 75 20  p@......Dear student, You 
000018E2  72 65 61 63 68 65 64 20 74 68 65 20 65 6E 64 2E 20 49 20 61 6D 20 70 72 6F 75  reached the end. I am prou
000018FC  64 20 6F 66 20 79 6F 75 2E 20 4E 6F 74 20 6D 61 6E 79 20 63 61 6E 20 64 6F 20  d of you. Not many can do 
00001916  74 68 61 74 2E 0A 00 00 00 00 54 68 69 73 20 77 61 73 20 6F 6E 6C 79 20 61 20  that......This was only a 
00001930  67 61 6D 65 2C 20 62 75 74 20 70 61 72 74 73 20 6F 66 20 74 68 65 20 72 65 61  game, but parts of the rea
0000194A  6C 20 6F 70 65 72 61 74 69 6F 6E 20 77 65 72 65 20 62 61 73 65 64 20 6F 6E 20  l operation were based on 
00001964  74 68 65 20 6B 6E 6F 77 6C 65 64 67 65 20 74 68 61 74 20 79 6F 75 20 6C 65 61  the knowledge that you lea
0000197E  72 6E 65 64 2E 0A 00 00 00 00 49 20 62 65 6C 69 65 76 65 20 74 68 61 74 20 79  rned......I believe that y
00001998  6F 75 20 61 72 65 20 70 61 72 74 20 6F 66 20 74 68 65 20 74 65 63 68 6E 6F 6C  ou are part of the technol
000019B2  6F 67 69 63 61 6C 20 65 64 67 65 20 74 68 61 74 20 6B 65 65 70 73 20 75 73 20  ogical edge that keeps us 
000019CC  68 65 72 65 0A 00 00 00 25 73 00 00 49 20 77 69 73 68 20 74 68 61 74 20 79 6F  here....%s..I wish that yo
000019E6  75 20 64 6F 20 67 72 65 61 74 20 74 68 69 6E 67 73 20 69 6E 20 73 65 63 75 72  u do great things in secur
00001A00  69 74 79 2C 20 65 63 6F 6E 6F 6D 79 2C 20 74 65 63 68 6E 6F 6C 6F 67 79 20 61  ity, economy, technology a
00001A1A  6E 64 20 65 64 75 63 61 74 69 6F 6E 0A 0A 00 00 00 00 50 69 6C 6F 74 2C 20 79  nd education......Pilot, y
00001A34  6F 75 20 6D 69 73 73 65 64 20 74 68 65 20 74 61 72 67 65 74 2E 20 55 73 65 20  ou missed the target. Use 
00001A4E  79 6F 75 72 20 53 49 47 49 4E 54 20 74 6F 20 66 69 6E 64 20 69 74 0A 00 00 00  your SIGINT to find it....
00001A68  4C 65 74 27 73 20 72 65 6D 65 6D 62 65 72 20 74 68 65 20 68 6F 73 74 61 67 65  Let's remember the hostage
00001A82  73 20 61 6E 64 20 77 69 73 68 20 61 6C 6C 20 6F 66 20 74 68 65 6D 20 77 69 6C  s and wish all of them wil
00001A9C  6C 20 62 65 20 68 6F 6D 65 20 73 6F 6F 6E 2E 20 41 6D 65 6E 2E 0A 00 00 47 72  l be home soon. Amen....Gr
00001AB6  65 61 74 20 6A 6F 62 20 70 69 6C 6F 74 2C 20 62 6F 6D 62 73 20 68 69 74 20 49  eat job pilot, bombs hit I
```

“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.”

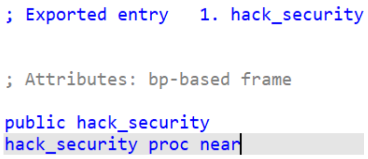
“C:\ReversingCTF\DroneAttack.txt”

“Enter code”

“Wrong code”

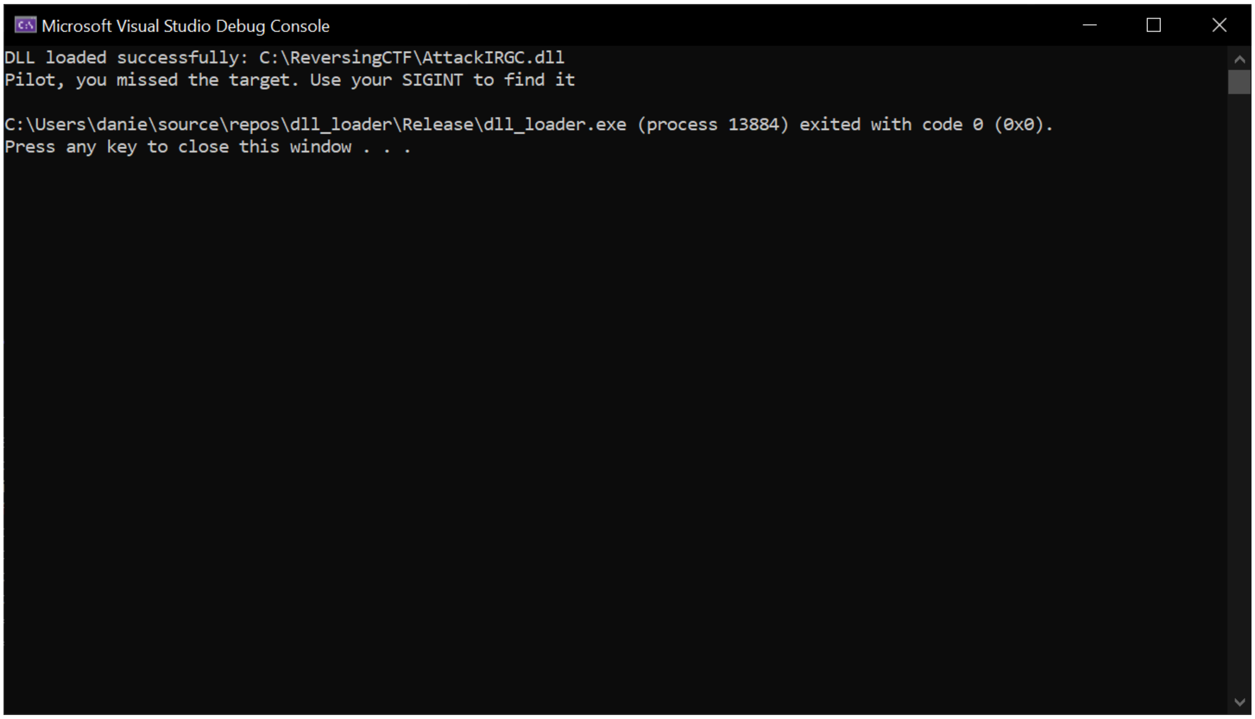
“Great work hero, you hacked the system. Prepare for a message from your instructor”

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



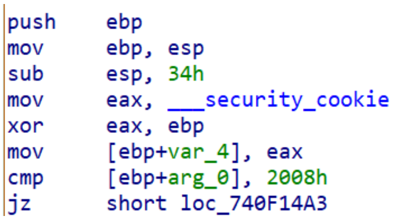
```
; Exported entry   1. hack_security

; Attributes: bp-based frame

public hack_security
hack_security proc near
```

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



```
DLL loaded successfully: C:\ReversingCTF\AttackIRGC.dll
Pilot, you missed the target. Use your SIGINT to find it

C:\Users\danie\source\repos\dll_loader\Release\dll_loader.exe (process 13884) exited with code 0 (0x0).
Press any key to close this window . . .
```

So we tried to understand the problem in the program



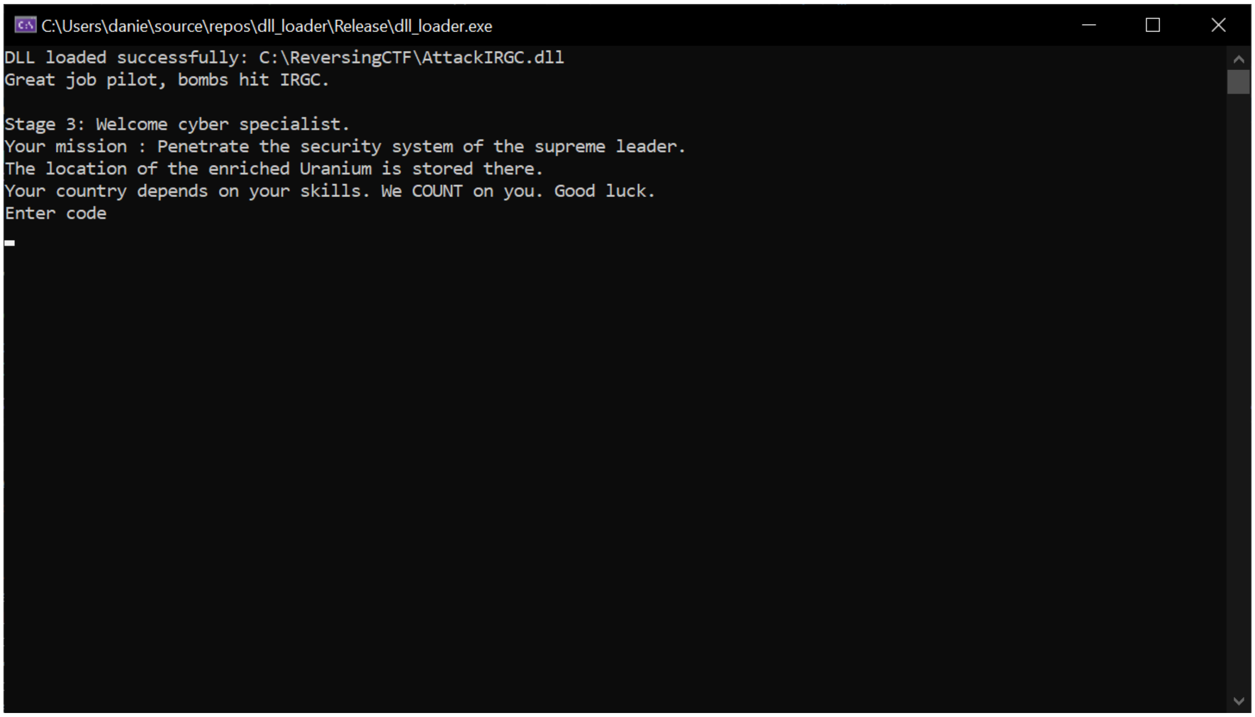
```
push    ebp
mov     ebp, esp
sub     esp, 34h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
cmp     [ebp+arg_0], 2008h
jz      short loc_740F14A3
```

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)

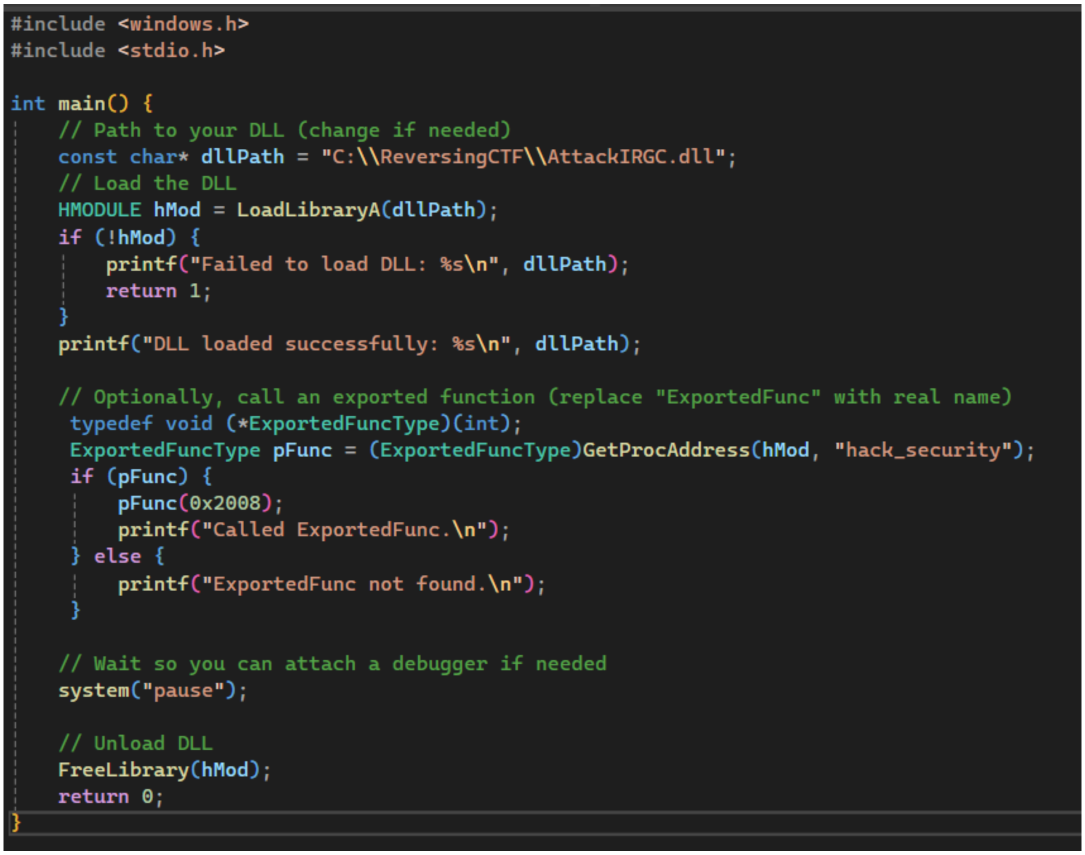
We tried running it again and got the correct screen.



C:\Users\danie\source\repos\dll_loader\Release\dll_loader.exe

```
DLL loaded successfully: C:\ReversingCTF\AttackIRGC.dll
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.
Enter code
```

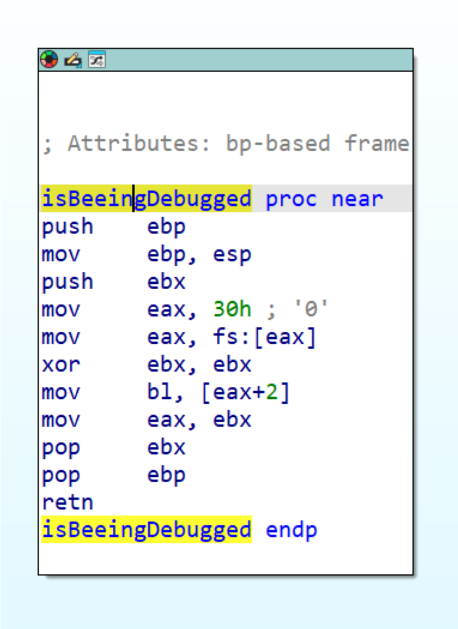
The program:



```c
#include <windows.h>
#include <stdio.h>

int main() {
    // Path to your DLL (change if needed)
    const char* dllPath = "C:\\ReversingCTF\\AttackIRGC.dll";
    // Load the DLL
    HMODULE hMod = LoadLibraryA(dllPath);
    if (!hMod) {
        printf("Failed to load DLL: %s\n", dllPath);
        return 1;
    }
    printf("DLL loaded successfully: %s\n", dllPath);

    // Optionally, call an exported function (replace "ExportedFunc" with real name)
     typedef void (*ExportedFuncType)(int);
     ExportedFuncType pFunc = (ExportedFuncType)GetProcAddress(hMod, "hack_security");
     if (pFunc) {
         pFunc(0x2008);
         printf("Called ExportedFunc.\n");
     } else {
         printf("ExportedFunc not found.\n");
     }

    // Wait so you can attach a debugger if needed
    system("pause");

    // Unload DLL
    FreeLibrary(hMod);
    return 0;
}
```

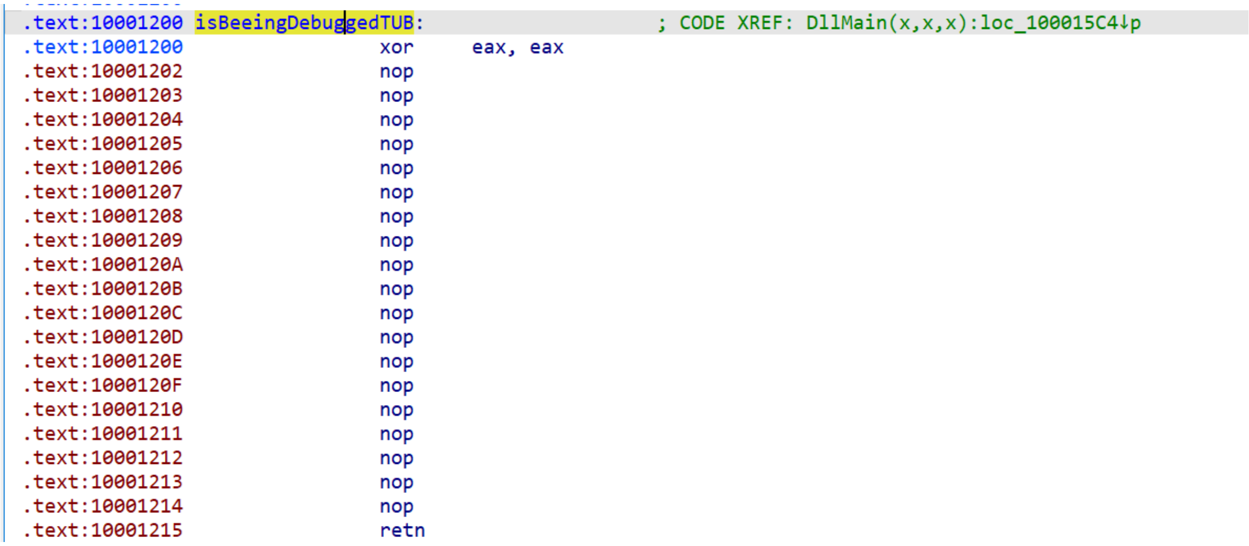
Stage 3:

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



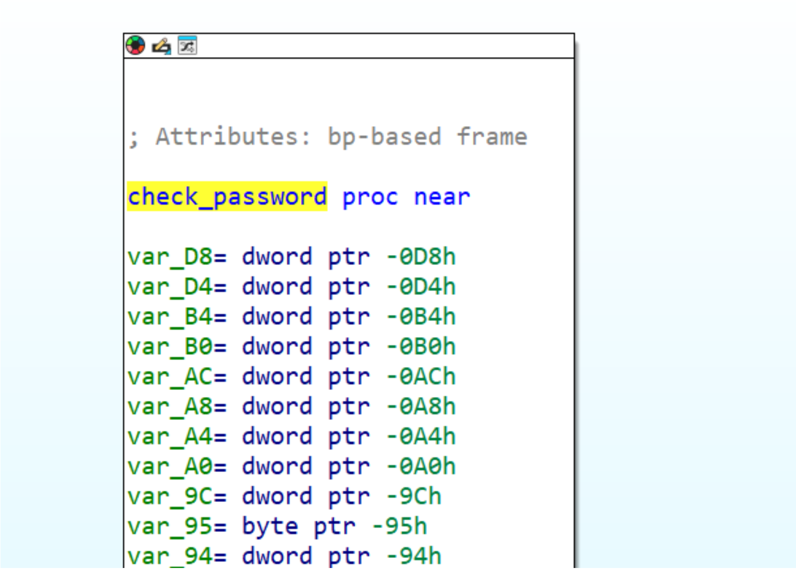
```
; Attributes: bp-based frame

isBeeingDebugged proc near
push    ebp
mov     ebp, esp
push    ebx
mov     eax, 30h ; '0'
mov     eax, fs:[eax]
xor     ebx, ebx
mov     bl, [eax+2]
mov     eax, ebx
pop     ebx
pop     ebp
retn
isBeeingDebugged endp
```

Patched with:



```
.text:10001200 isBeeingDebuggedTUB:                    ; CODE XREF: DllMain(x,x,x):loc_100015C4↓p
.text:10001200                 xor     eax, eax
.text:10001202                 nop
.text:10001203                 nop
.text:10001204                 nop
.text:10001205                 nop
.text:10001206                 nop
.text:10001207                 nop
.text:10001208                 nop
.text:10001209                 nop
.text:1000120A                 nop
.text:1000120B                 nop
.text:1000120C                 nop
.text:1000120D                 nop
.text:1000120E                 nop
.text:1000120F                 nop
.text:10001210                 nop
.text:10001211                 nop
.text:10001212                 nop
.text:10001213                 nop
.text:10001214                 nop
.text:10001215                 retn
```

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



```
; Attributes: bp-based frame

check_password proc near

var_D8= dword ptr -0D8h
var_D4= dword ptr -0D4h
var_B4= dword ptr -0B4h
var_B0= dword ptr -0B0h
var_AC= dword ptr -0ACh
var_A8= dword ptr -0A8h
var_A4= dword ptr -0A4h
var_A0= dword ptr -0A0h
var_9C= dword ptr -9Ch
var_95= byte ptr -95h
var_94= dword ptr -94h
```

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:

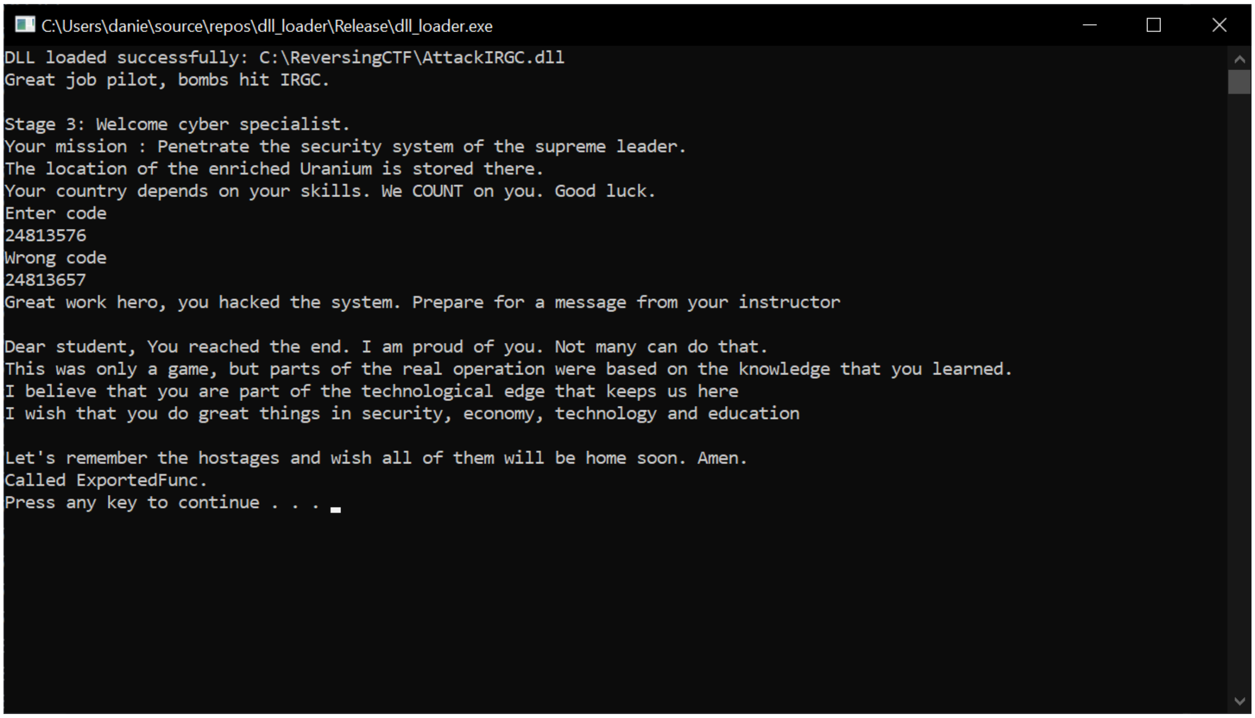
1. The program waits for a string input
2. We check the length and compare it to 8 (answer of 8 char)
3. The string is then separated into char
4. The program check with ds:isdigit to check if the string contains only digits
5. If not, we exit loop
6. If it contains only number between 1-8
7. It then 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:

```c
// 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



C:\Users\danie\source\repos\dll_loader\Release\dll_loader.exe

```
DLL loaded successfully: C:\ReversingCTF\AttackIRGC.dll
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.
Enter code
24813576
Wrong code
24813657
Great work hero, you hacked the system. Prepare for a message from your instructor

Dear student, You reached the end. 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

Let's remember the hostages and wish all of them will be home soon. Amen.
Called ExportedFunc.
Press any key to continue . . .
```

Thank you very much this was fun!!