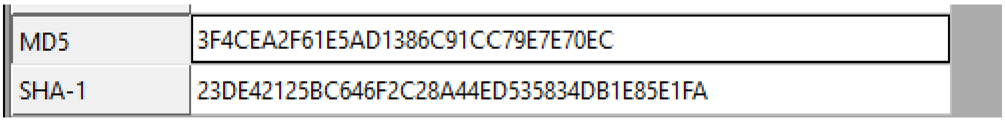
Reverse engineering final assignment - Iyar Gross

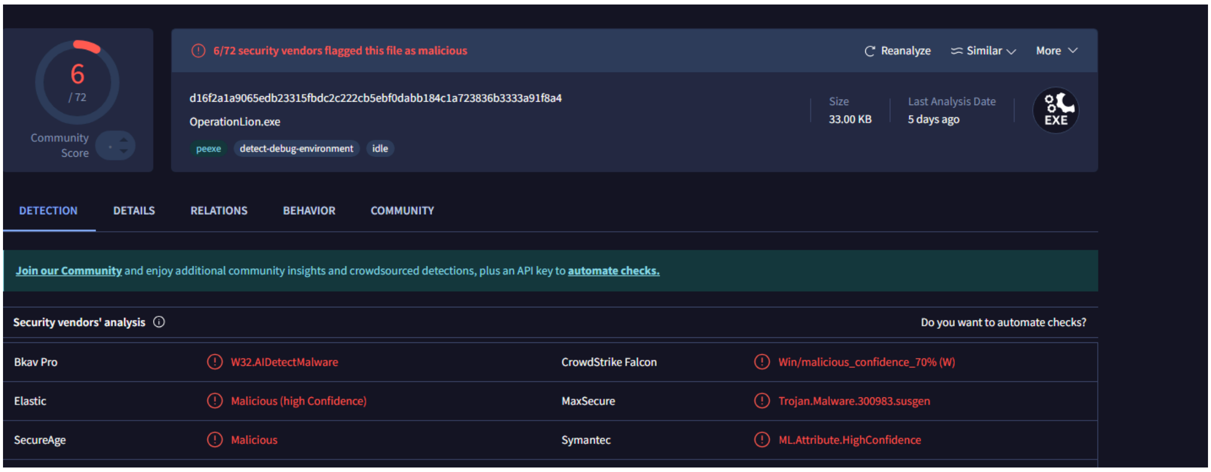
Hello and welcome back!
Today we are analyzing operationlion.exe.
Before diving into IDA, let's gather some initial information about the file using OSINT techniques.

Pre investigation-

I opened the file in CFF Explorer to gather basic information.
This is the file's hash. I searched for it in VirusTotal.



MD5	3F4CEA2F61E5AD1386C91CC79E7E70EC
SHA-1	23DE42125BC646F2C28A44ED535834DB1E85E1FA



6/72 security vendors flagged this file as malicious

d16f2a1a9065edb23315fbdc2c222cb5ebf0dabb184c1a723836b3333a91f8a4
OperationLion.exe

Size 33.00 KB
Last Analysis Date 5 days ago

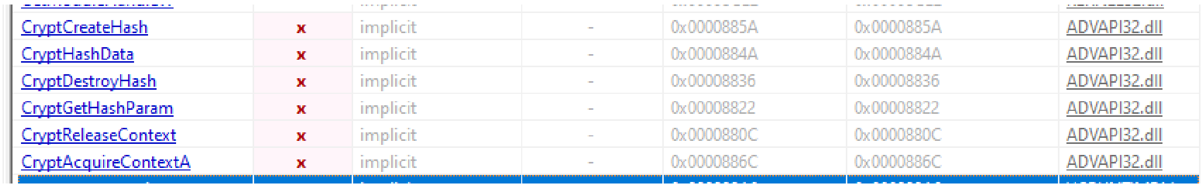
Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis

Do you want to automate checks?

Bkav Pro	W32.AIDetectMalware	CrowdStrike Falcon	Win/malicious_confidence_70% (W)
Elastic	Malicious (high Confidence)	MaxSecure	Trojan.Malware.300983.susgen
SecureAge	Malicious	Symantec	ML.Attribute.HighConfidence

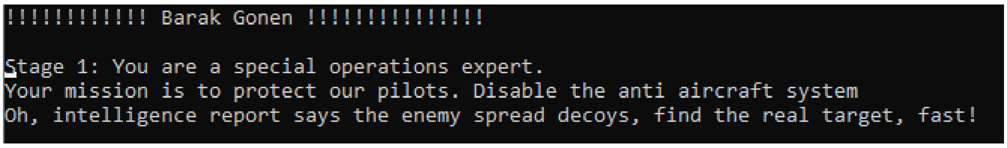
VirusTotal recognized the file and gave it a rating of 6.



CryptCreateHash	x	implicit	-	0x0000885A	0x0000885A	ADVAPI32.dll
CryptHashData	x	implicit	-	0x0000884A	0x0000884A	ADVAPI32.dll
CryptDestroyHash	x	implicit	-	0x00008836	0x00008836	ADVAPI32.dll
CryptGetHashParam	x	implicit	-	0x00008822	0x00008822	ADVAPI32.dll
CryptReleaseContext	x	implicit	-	0x0000880C	0x0000880C	ADVAPI32.dll
CryptAcquireContextA	x	implicit	-	0x0000886C	0x0000886C	ADVAPI32.dll

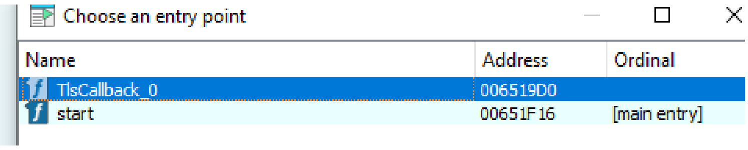
In PE Studio, I identified that the file imports encryption functions.

Step 1 – Tls callback



```
!!!!!!!!!!! Barak Gonen !!!!!!!!!!!!!!

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!
```

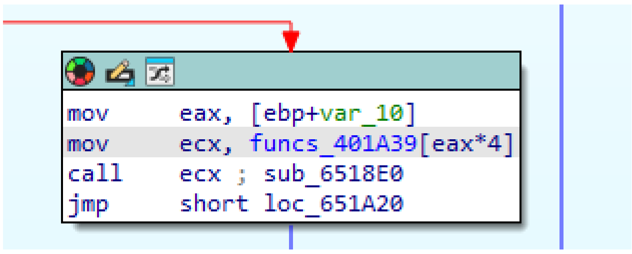
Let's disable the anti aircraft.
After that message the program is shutting off. Let's see why.



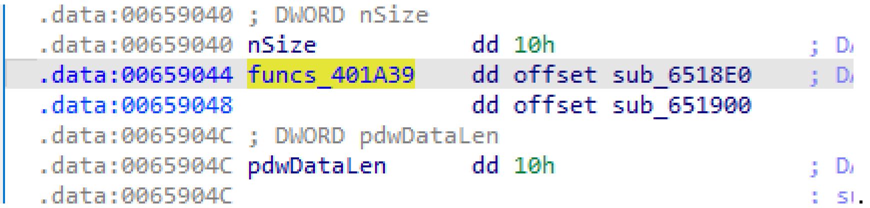
Choose an entry point

Name	Address	Ordinal
TlsCallback_0	006519D0	
start	00651F16	[main entry]

As we can see tls callback is running before the main.



```
mov     eax, [ebp+var_10]
mov     ecx, funcs_401A39[eax*4]
call    ecx ; sub_6518E0
jmp     short loc_651A20
```



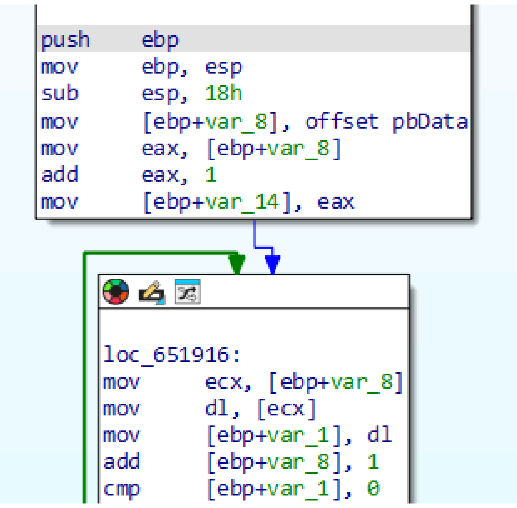
```
.data:00659040 ; DWORD nSize
.data:00659040 nSize           dd 10h                          ; D
.data:00659044 funcs_401A39    dd offset sub_6518E0            ; D
.data:00659048                 dd offset sub_651900
.data:0065904C ; DWORD pdwDataLen
.data:0065904C pdwDataLen      dd 10h                          ; D
.data:0065904C                                                 ; s.
```

Here we have a dispatcher .

A function table (funcs_401A39) is used.

A loop runs twice, calling the first two functions in the table before main executes.

The second function hashes the collected data with MD5 (CryptHashData) and stores the digits in memory (byte_6593D4).



```
push    ebp
mov     ebp, esp
sub     esp, 18h
mov     [ebp+var_8], offset pbData
mov     eax, [ebp+var_8]
add     eax, 1
mov     [ebp+var_14], eax
```

```
loc_651916:
mov     ecx, [ebp+var_8]
mov     dl, [ecx]
mov     [ebp+var_1], dl
add     [ebp+var_8], 1
cmp     [ebp+var_1], 0
```

The TLS loop converts the 16-byte digits to a 32-character uppercase hex string in a buffer.

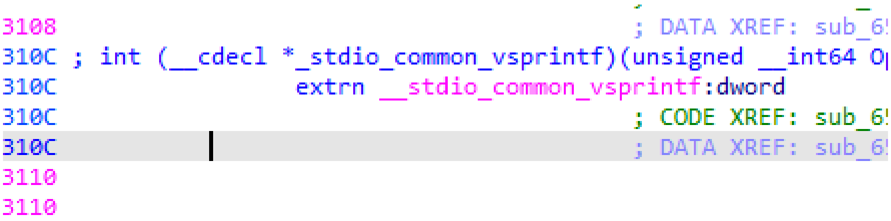
So in conclusion we are dealing with a buffer.

Let's see who calls and uses this buffer.

Step 2 – Print function

I traced the program to see where this buffer is used for printing.

Following the imports and xref graph, I identified the first call to the print function.

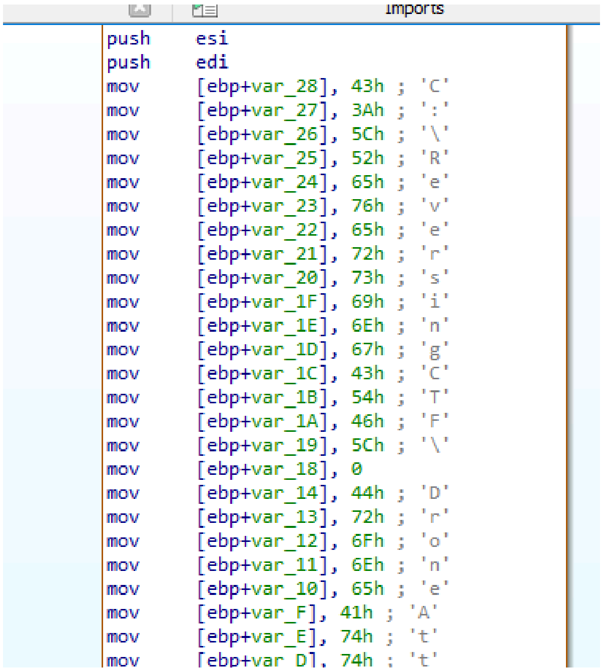


```
3108                                                   ; DATA XREF: sub_6
310C ; int (__cdecl *_stdio_common_vsprintf)(unsigned __int64 O
310C                    extrn __stdio_common_vsprintf:dword
310C                                                   ; CODE XREF: sub_6
310C                                                   ; DATA XREF: sub_6
3110
3110
```

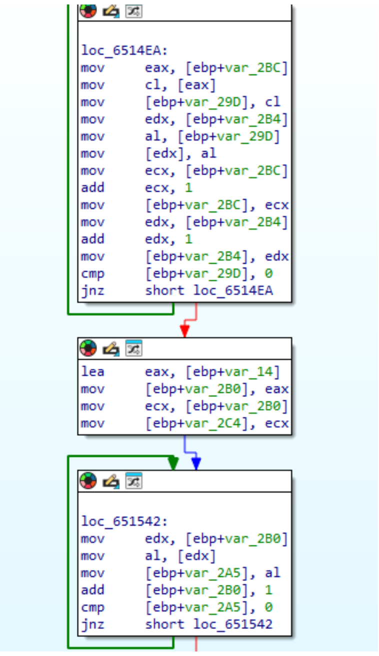
By analyzing buffer usage, I found which function prints the initial output we see.

Step 4 – File Check & Obfuscation

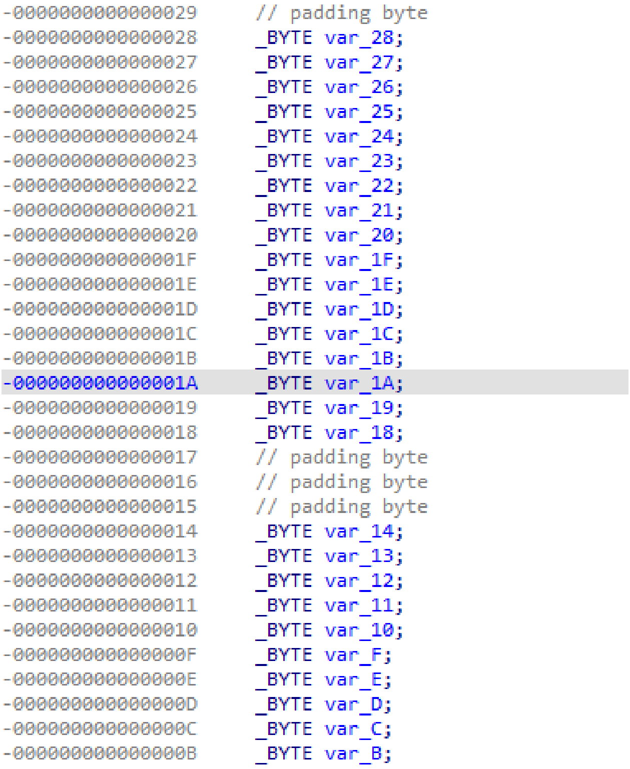
Now we got into the function that prints out our beloved message.



```
push    esi
push    edi
mov     [ebp+var_28], 43h ; 'C'
mov     [ebp+var_27], 3Ah ; ':'
mov     [ebp+var_26], 5Ch ; '\'
mov     [ebp+var_25], 52h ; 'R'
mov     [ebp+var_24], 65h ; 'e'
mov     [ebp+var_23], 76h ; 'v'
mov     [ebp+var_22], 65h ; 'e'
mov     [ebp+var_21], 72h ; 'r'
mov     [ebp+var_20], 73h ; 's'
mov     [ebp+var_1F], 69h ; 'i'
mov     [ebp+var_1E], 6Eh ; 'n'
mov     [ebp+var_1D], 67h ; 'g'
mov     [ebp+var_1C], 43h ; 'C'
mov     [ebp+var_1B], 54h ; 'T'
mov     [ebp+var_1A], 46h ; 'F'
mov     [ebp+var_19], 5Ch ; '\'
mov     [ebp+var_18], 0
mov     [ebp+var_14], 44h ; 'D'
mov     [ebp+var_13], 72h ; 'r'
mov     [ebp+var_12], 6Fh ; 'o'
mov     [ebp+var_11], 6Eh ; 'n'
mov     [ebp+var_10], 65h ; 'e'
mov     [ebp+var_F], 41h ; 'A'
mov     [ebp+var_E], 74h ; 't'
mov     [ebp+var_D], 74h ; 't'
```



```
loc_6514EA:
mov     eax, [ebp+var_2BC]
mov     cl, [eax]
mov     [ebp+var_29D], cl
mov     edx, [ebp+var_2B4]
mov     al, [ebp+var_29D]
mov     [edx], al
mov     ecx, [ebp+var_2BC]
add     ecx, 1
mov     [ebp+var_2BC], ecx
mov     edx, [ebp+var_2B4]
add     edx, 1
mov     [ebp+var_2B4], edx
cmp     [ebp+var_29D], 0
jnz     short loc_6514EA

lea     eax, [ebp+var_14]
mov     [ebp+var_2B0], eax
mov     ecx, [ebp+var_2B0]
mov     [ebp+var_2C4], ecx

loc_651542:
mov     edx, [ebp+var_2B0]
mov     al, [edx]
mov     [ebp+var_2A5], al
add     [ebp+var_2B0], 1
cmp     [ebp+var_2A5], 0
jnz     short loc_651542
```

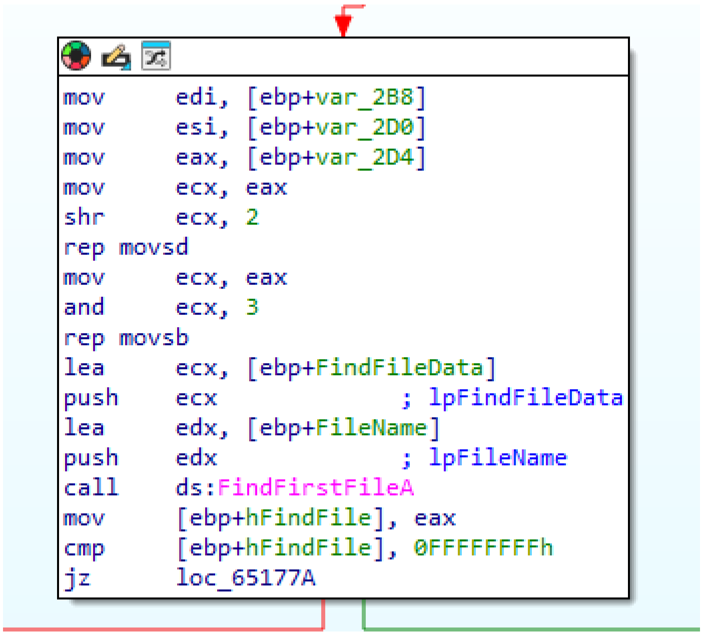


```
-0000000000000029     // padding byte
-0000000000000028     _BYTE var_28;
-0000000000000027     _BYTE var_27;
-0000000000000026     _BYTE var_26;
-0000000000000025     _BYTE var_25;
-0000000000000024     _BYTE var_24;
-0000000000000023     _BYTE var_23;
-0000000000000022     _BYTE var_22;
-0000000000000021     _BYTE var_21;
-0000000000000020     _BYTE var_20;
-000000000000001F     _BYTE var_1F;
-000000000000001E     _BYTE var_1E;
-000000000000001D     _BYTE var_1D;
-000000000000001C     _BYTE var_1C;
-000000000000001B     _BYTE var_1B;
-000000000000001A     _BYTE var_1A;
-0000000000000019     _BYTE var_19;
-0000000000000018     _BYTE var_18;
-0000000000000017     // padding byte
-0000000000000016     // padding byte
-0000000000000015     // padding byte
-0000000000000014     _BYTE var_14;
-0000000000000013     _BYTE var_13;
-0000000000000012     _BYTE var_12;
-0000000000000011     _BYTE var_11;
-0000000000000010     _BYTE var_10;
-000000000000000F     _BYTE var_F;
-000000000000000E     _BYTE var_E;
-000000000000000D     _BYTE var_D;
-000000000000000C     _BYTE var_C;
-000000000000000B     _BYTE var_B;
```

This is an obfuscation.

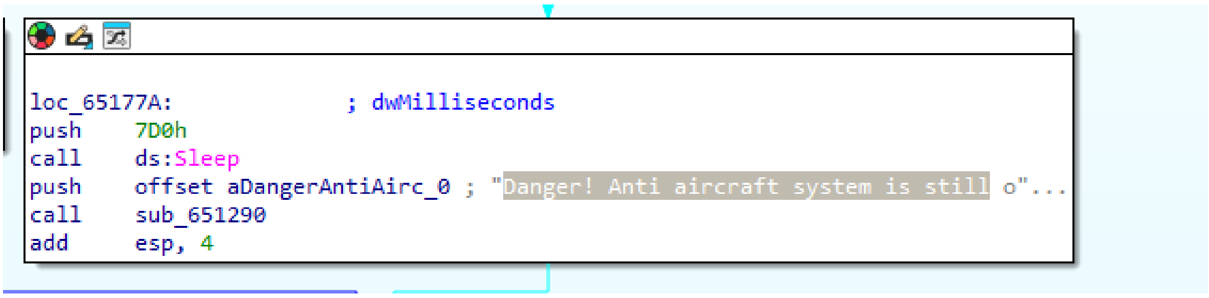
The string -C:\ReversingCTF\DroneAttack.txt

Is being obfuscated ,just a technique to harden our understanding.



```
mov     edi, [ebp+var_2B8]
mov     esi, [ebp+var_2D0]
mov     eax, [ebp+var_2D4]
mov     ecx, eax
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
lea     ecx, [ebp+FindFileData]
push    ecx             ; lpFindFileData
lea     edx, [ebp+FileName]
push    edx             ; lpFileName
call    ds:FindFirstFileA
mov     [ebp+hFindFile], eax
cmp     [ebp+hFindFile], 0FFFFFFFFh
jz      loc_65177A
```

The program checks for the existence of a file.

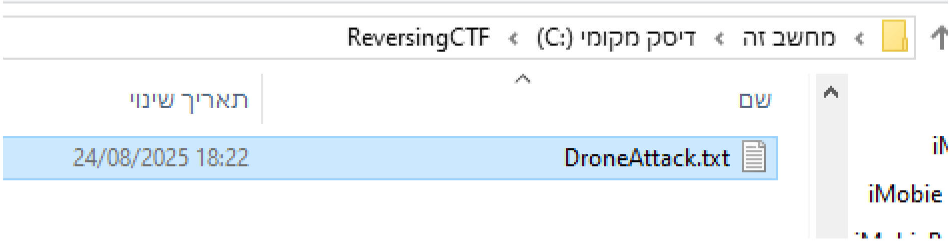


```
loc_65177A:             ; dwMilliseconds
push    7D0h
call    ds:Sleep
push    offset aDangerAntiAirc_0 ; "Danger! Anti aircraft system is still o"...
call    sub_651290
add     esp, 4
```

If the file exists → we continues the operation

If not → jumps to a crash (loc_65177A)

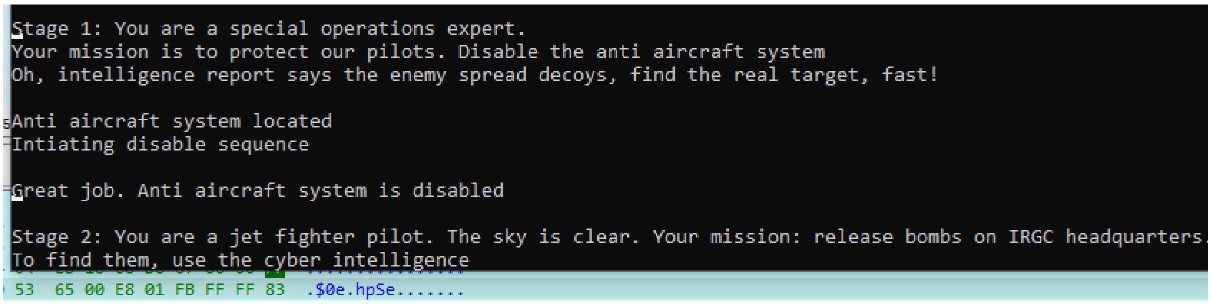
The crash occurs because the program tries to continue without an open file (because the file doesn't exist)



ReversingCTF < (C:) דיסק מקומי < מחשב זה <

שם	תאריך שינוי
DroneAttack.txt	24/08/2025 18:22

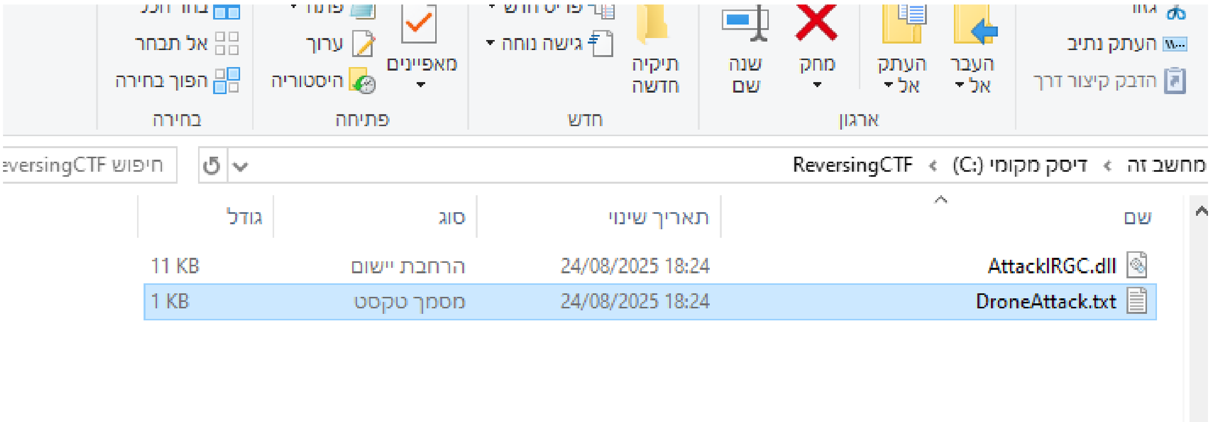
I created the expected file in the target directory (C:\ReversingCTF\DroneAttack.txt) to bypass this crash.



```
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
 53  65 00 E8 01 FB FF FF 83  .$0e.hpSe.......
```

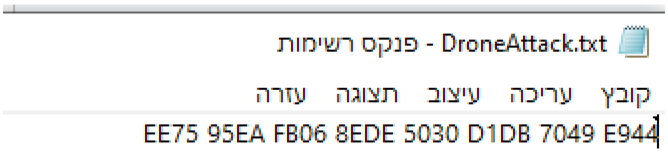
Hurray!

level 2:-Dropped DLL Analysis



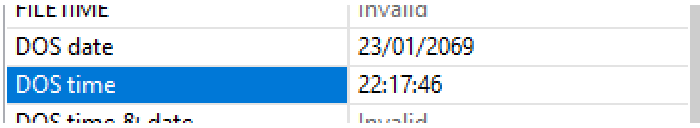
שם	תאריך שינוי	סוג	גודל
AttackIRGC.dll	24/08/2025 18:24	הרחבת יישום	11 KB
DroneAttack.txt	24/08/2025 18:24	מסמך טקסט	1 KB

When operationLion ran it dropped a new DLL- AttackIRGC.dll
And added to our text file a hex dump.



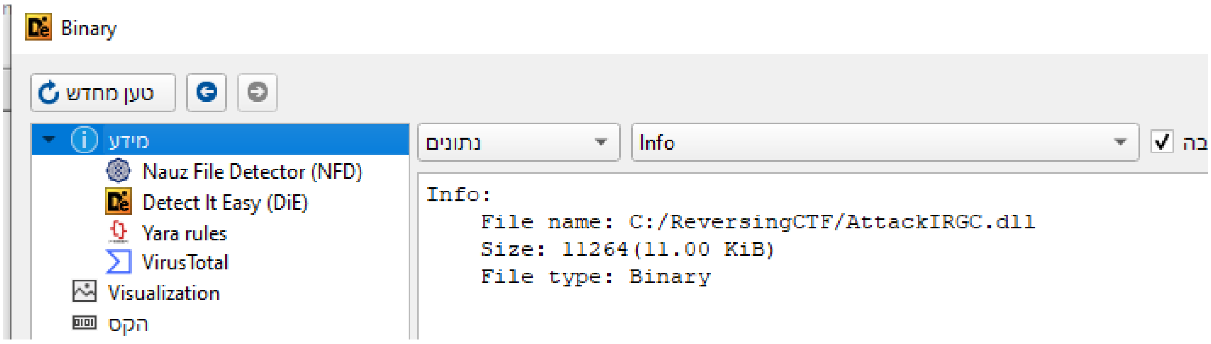
```
EE75 95EA FB06 8EDE 5030 D1DB 7049 E944
```

Step 1 -pre investigation



FILE TIME	Invalid
DOS date	23/01/2069
DOS time	22:17:46
DOS time & date	Invalid

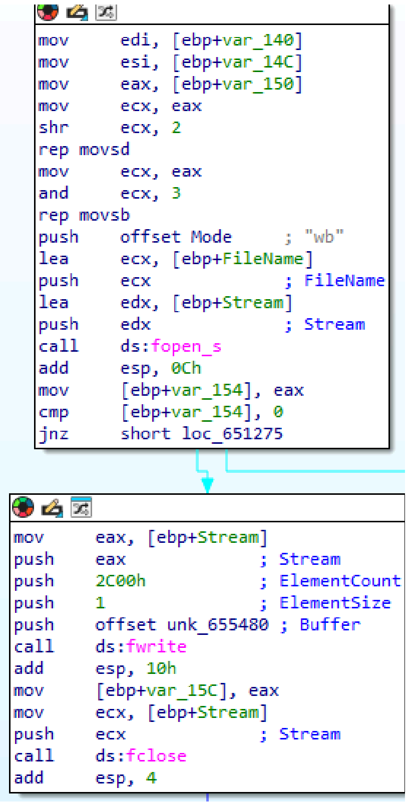
I opened the DLL in DIE-
In this picture we can see that somebody messed up the dos header(it's a bit suspicious).



```
Info:
    File name: C:/ReversingCTF/AttackIRGC.dll
    Size: 11264(11.00 KiB)
    File type: Binary
```

And as I see , the dropped DLL is either encrypted or warped.

I located the function responsible for extracting and writing the DLL:



```
mov     edi, [ebp+var_140]
mov     esi, [ebp+var_14C]
mov     eax, [ebp+var_150]
mov     ecx, eax
shr     ecx, 2
rep movsd
mov     ecx, eax
and     ecx, 3
rep movsb
push    offset Mode      ; "wb"
lea     ecx, [ebp+FileName]
push    ecx              ; FileName
lea     edx, [ebp+Stream]
push    edx              ; Stream
call    ds:fopen_s
add     esp, 0Ch
mov     [ebp+var_154], eax
cmp     [ebp+var_154], 0
jnz     short loc_651275
```

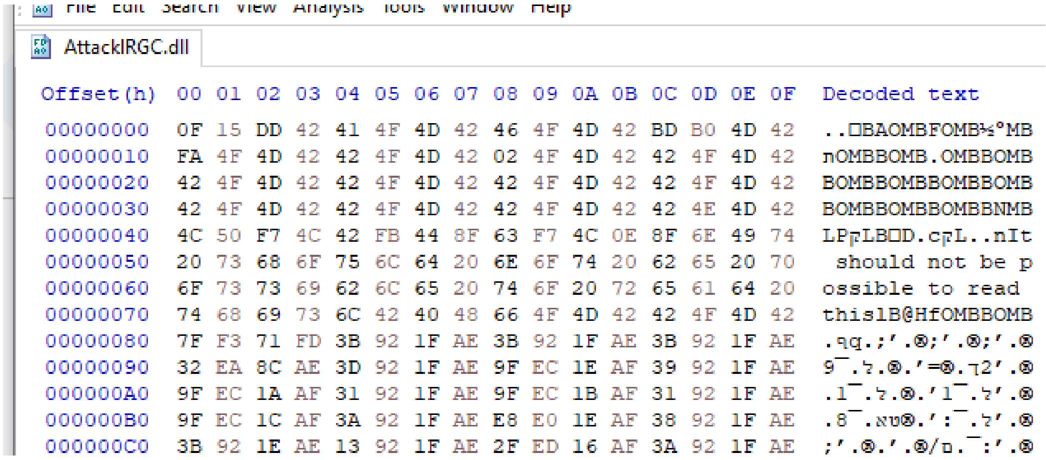
```
mov     eax, [ebp+Stream]
push    eax              ; Stream
push    2C00h            ; ElementCount
push    1                ; ElementSize
push    offset unk_655480 ; Buffer
call    ds:fwrite
add     esp, 10h
mov     [ebp+var_15C], eax
mov     ecx, [ebp+Stream]
push    ecx              ; Stream
call    ds:fclose
add     esp, 4
```

sub_651080 Function:

Extracts a blob from our exe (unk_655480)

Writes it to a file: AttackIRGC.dll

Step 2– Decrypting the DLL



```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text

00000000  0F 15 DD 42 41 4F 4D 42 46 4F 4D 42 BD B0 4D 42  ..□BAOMBFOMB½°MB
00000010  FA 4F 4D 42 42 4F 4D 42 02 4F 4D 42 42 4F 4D 42  ןOMBBOMB.OMBBOMB
00000020  42 4F 4D 42 42 4F 4D 42 42 4F 4D 42 42 4F 4D 42  BOMBBOMBBOMBBOMB
00000030  42 4F 4D 42 42 4F 4D 42 42 4F 4D 42 42 4E 4D 42  BOMBBOMBBOMBBNMB
00000040  4C 50 F7 4C 42 FB 44 8F 63 F7 4C 0E 8F 6E 49 74  LPקLB□D.cקL..nIt
00000050  20 73 68 6F 75 6C 64 20 6E 6F 74 20 62 65 20 70   should not be p
00000060  6F 73 73 69 62 6C 65 20 74 6F 20 72 65 61 64 20  ossible to read 
00000070  74 68 69 73 6C 42 40 48 66 4F 4D 42 42 4F 4D 42  thislB@HfOMBBOMB
00000080  7F F3 71 FD 3B 92 1F AE 3B 92 1F AE 3B 92 1F AE  .ףq.;'.®;'.®;'.®
00000090  32 EA 8C AE 3D 92 1F AE 9F EC 1E AF 39 92 1F AE  2ך.®='.®.ל.¯9'.®
000000A0  9F EC 1A AF 31 92 1F AE 9F EC 1B AF 31 92 1F AE  .ל.¯1'.®.ל.¯1'.®
000000B0  9F EC 1C AF 3A 92 1F AE E8 E0 1E AF 38 92 1F AE  .ל.¯:'.®רא.¯8'.®
000000C0  3B 92 1E AE 13 92 1F AE 2F ED 16 AF 3A 92 1F AE  ;'.®.'.®/ם.¯:'.®
```

I opened the DLL in hex editor , I noticed that the DLL did not start with MZ which confirms that there is an encryption

I noticed that there are repeated occurrences of the word "BOMB" when usually in hex there are lots of “0” .

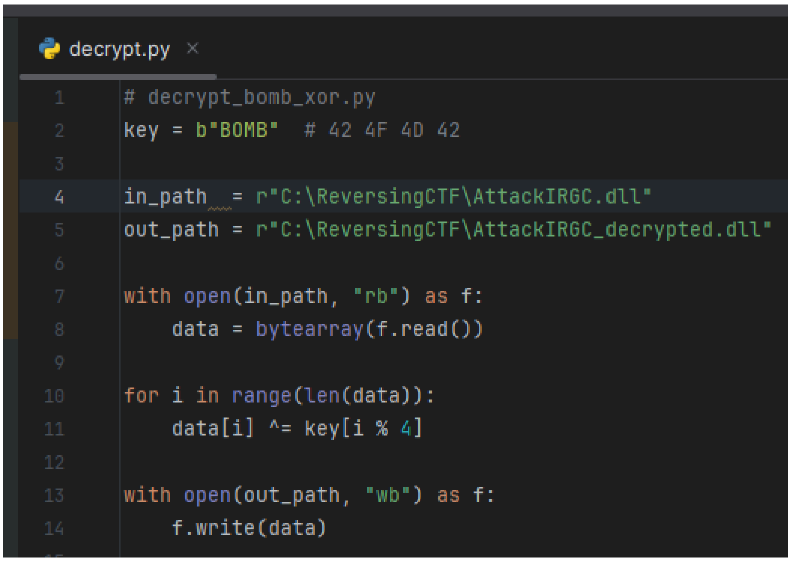
 I started to suspect that the encryption is XOR, a known and commonly used encryption . According to my theory if ”BOMB” is supposed to be 0 then the key is “BOMB” because only a xor a=0 .

I wanted to check if my theory is correct by trying to decrypt the first bytes of the DLL which are supposed to be 4D 5A- MZ

`'M'` = `4D`: `0F ^ k0 = 4D` → `k0 = 0F ^ 4D = 42` → `'B'`

`'Z'` = `5A`: `15 ^ k1 = 5A` → `k1 = 15 ^ 5A = 4F` → `'O'`

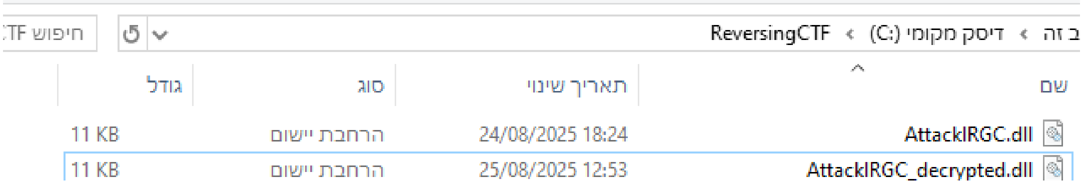
As I saw this I understood my theory was verified .
XOR key: "BOMB" is used to encrypt/decrypt the DLL so that the first bytes produce the proper MZ header.



decrypt.py

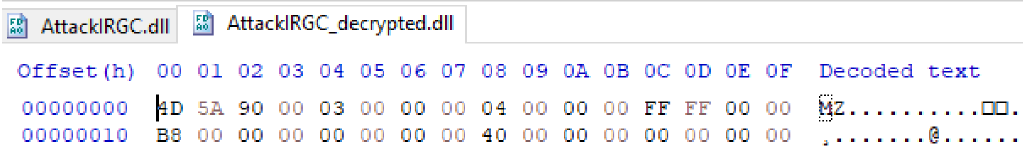
```python
# decrypt_bomb_xor.py
key = b"BOMB"  # 42 4F 4D 42

in_path  = r"C:\ReversingCTF\AttackIRGC.dll"
out_path = r"C:\ReversingCTF\AttackIRGC_decrypted.dll"

with open(in_path, "rb") as f:
    data = bytearray(f.read())

for i in range(len(data)):
    data[i] ^= key[i % 4]

with open(out_path, "wb") as f:
    f.write(data)
```

I Wrote a Python script to decrypt the file.
I successfully obtained the decrypted DLL/



ב זה < דיסק מקומי (C:) < ReversingCTF

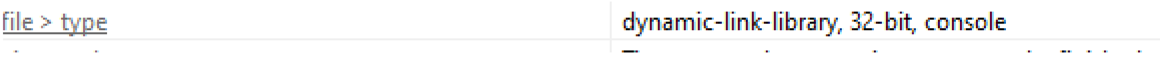
שם	תאריך שינוי	סוג	גודל
AttackIRGC.dll	24/08/2025 18:24	הרחבת יישום	11 KB
AttackIRGC_decrypted.dll	25/08/2025 12:53	הרחבת יישום	11 KB



AttackIRGC.dll | AttackIRGC_decrypted.dll

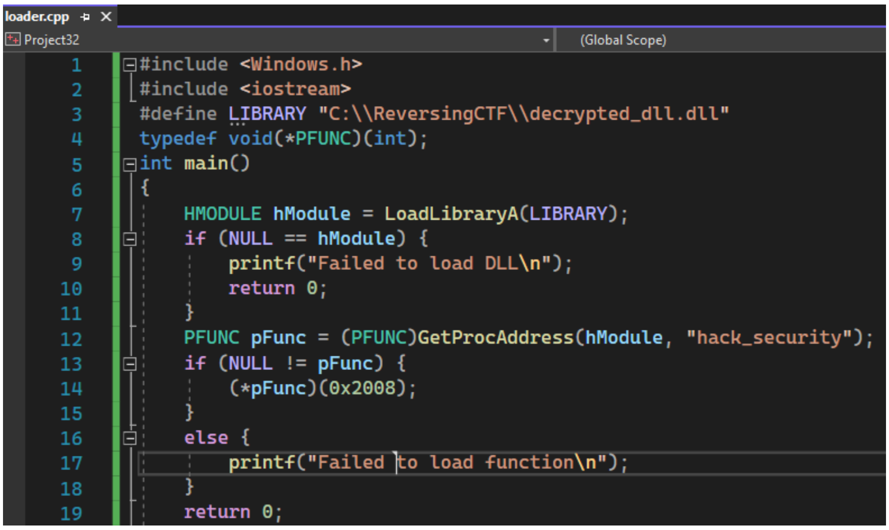
```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
```

Now the decrypted DLL begins with MZ and now we can analyze it in IDA



file > type dynamic-link-library, 32-bit, console

The DLL is 32-bit .
The DLL cannot run by itself therefore it requires a loader.
I adapted a simple DLL loader that we learned in class to execute the decrypted DLL.



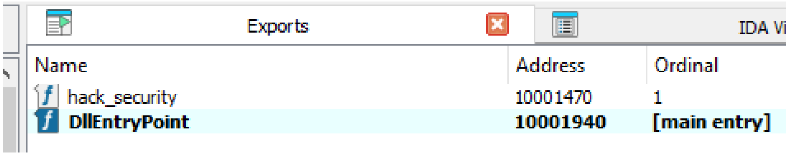
```
#include <Windows.h>
#include <iostream>
#define LIBRARY "C:\\ReversingCTF\\decrypted_dll.dll"
typedef void(*PFUNC)(int);
int main()
{
    HMODULE hModule = LoadLibraryA(LIBRARY);
    if (NULL == hModule) {
        printf("Failed to load DLL\n");
        return 0;
    }
    PFUNC pFunc = (PFUNC)GetProcAddress(hModule, "hack_security");
    if (NULL != pFunc) {
        (*pFunc)(0x2008);
    }
    else {
        printf("Failed to load function\n");
    }
    return 0;
```

I modified the code so it'll fit our DLL .

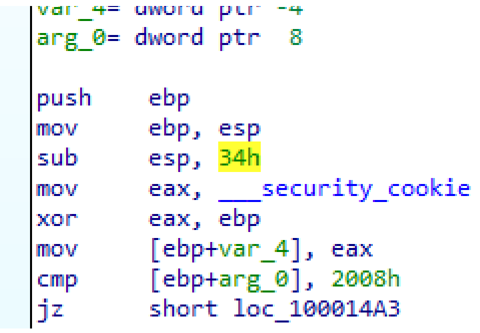
I Checked in the decrypted DLL exports to look for the function that our exe is calling.

-hack_security

And I noticed that hack_security is expecting a number -2008

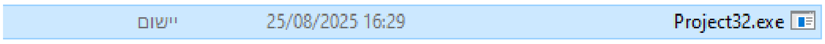


Name	Address	Ordinal
hack_security	10001470	1
DllEntryPoint	**10001940**	**[main entry]**



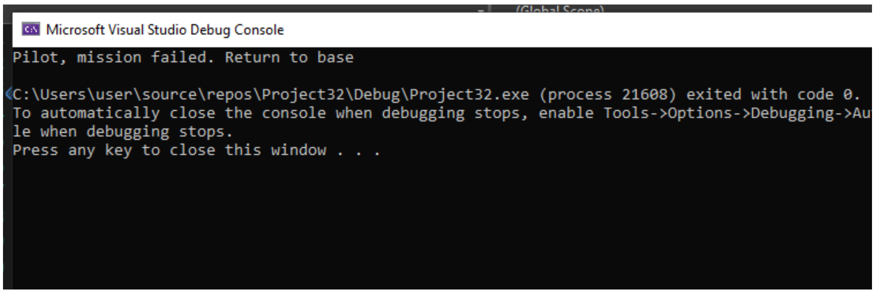
```
var_4= dword ptr -4
arg_0= dword ptr  8

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_100014A3
```

I compiled my program by visual studio to an exe file .



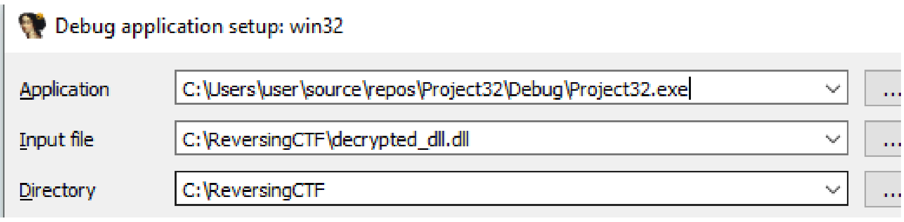
יישום	25/08/2025 16:29	Project32.exe

This is what happens when we run the loader.exe by itself.



```
Pilot, mission failed. Return to base

C:\Users\user\source\repos\Project32\Debug\Project32.exe (process 21608) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Au
le when debugging stops.
Press any key to close this window . . .
```

I loaded the loader to ida .



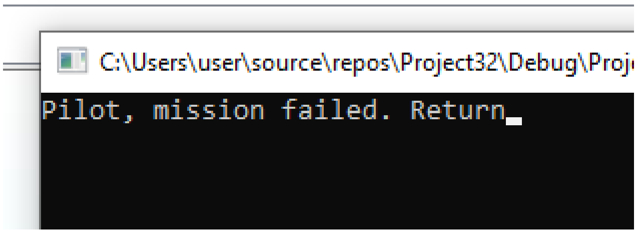
Debug application setup: win32

Application C:\Users\user\source\repos\Project32\Debug\Project32.exe

Input file C:\ReversingCTF\decrypted_dll.dll

Directory C:\ReversingCTF

Now we have the same message as before but now the program is crashing.

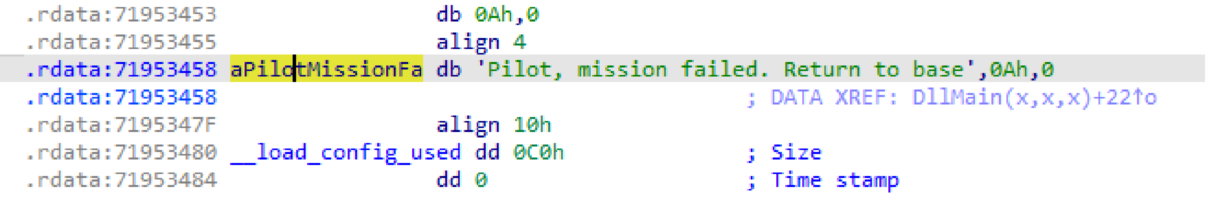


C:\Users\user\source\repos\Project32\Debug\Proj

```
Pilot, mission failed. Return_
```

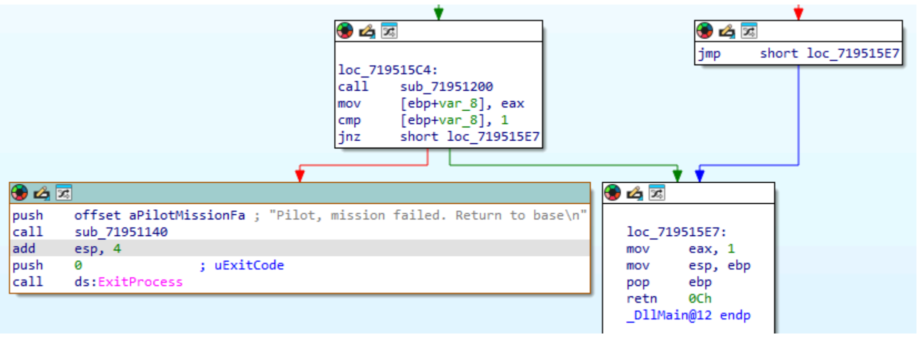
Let's understand why.

I looked for the specific message that I'm getting in ida.



```
.rdata:71953453                 db 0Ah,0
.rdata:71953455                 align 4
.rdata:71953458 aPilotMissionFa db 'Pilot, mission failed. Return to base',0Ah,0
.rdata:71953458                                         ; DATA XREF: DllMain(x,x,x)+22↑o
.rdata:7195347F                 align 10h
.rdata:71953480 __load_config_used dd 0C0h              ; Size
.rdata:71953484                 dd 0                    ; Time stamp
```

I went to the function that calls our string (by xref)



```
loc_719515C4:
call    sub_71951200
mov     [ebp+var_8], eax
cmp     [ebp+var_8], 1
jnz     short loc_719515E7
```

```
jmp     short loc_719515E7
```

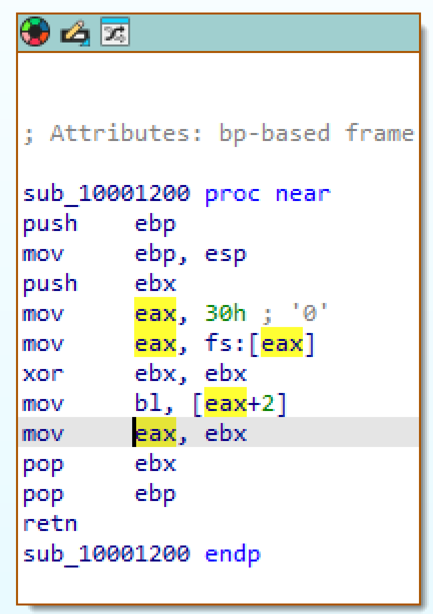
```
push    offset aPilotMissionFa ; "Pilot, mission failed. Return to base\n"
call    sub_71951140
add     esp, 4
push    0               ; uExitCode
call    ds:ExitProcess
```

```
loc_719515E7:
mov     eax, 1
mov     esp, ebp
pop     ebp
retn    0Ch
_DllMain@12 endp
```

Hello buddy!

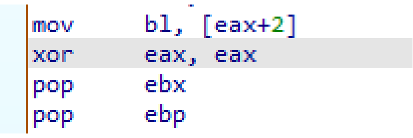
I've found an anti debug:)

The output of sub_71951200 is what determines if the process is finished.

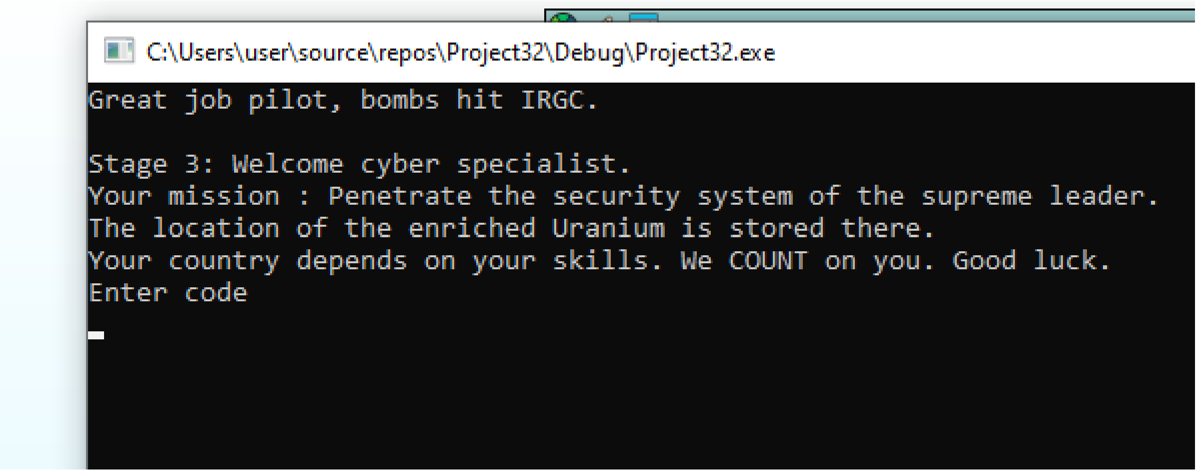


```
; Attributes: bp-based frame

sub_10001200 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
sub_10001200 endp
```

I patched the program so that the anti debug is diffused.



```
mov     bl, [eax+2]
xor     eax, eax
pop     ebx
pop     ebp
```



C:\Users\user\source\repos\Project32\Debug\Project32.exe

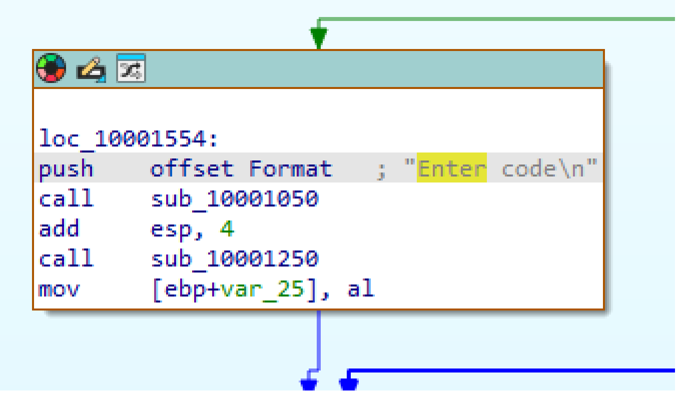
```
Great job pilot, bombs hit IRGC.

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
```

Hurray!

level 3:

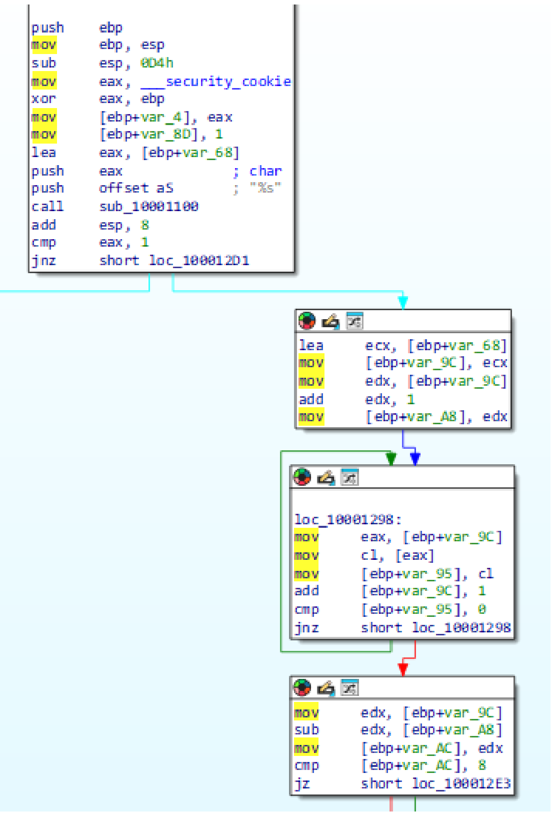
Now we have a code to find , let’s explore☺

I’ve searched where in the file there is a use of the word Enter.



```
loc_10001554:
push    offset Format   ; "Enter code\n"
call    sub_10001050
add     esp, 4
call    sub_10001250
mov     [ebp+var_25], al
```

This is our message and a call for print, but what does sub_10001250 do ?



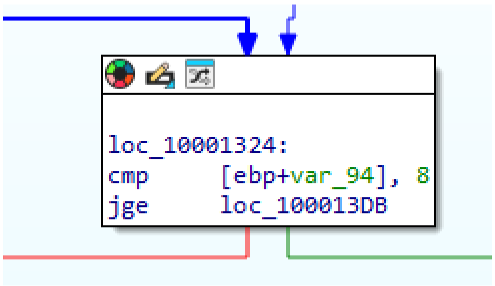
```
push    ebp
mov     ebp, esp
sub     esp, 0D4h
mov     eax, ___security_cookie
xor     eax, ebp
mov     [ebp+var_4], eax
mov     [ebp+var_8D], 1
lea     eax, [ebp+var_68]
push    eax             ; char
push    offset aS       ; "%s"
call    sub_10001100
add     esp, 8
cmp     eax, 1
jnz     short loc_100012D1
```

```
lea     ecx, [ebp+var_68]
mov     [ebp+var_9C], ecx
mov     edx, [ebp+var_9C]
add     edx, 1
mov     [ebp+var_A8], edx
```

```
loc_10001298:
mov     eax, [ebp+var_9C]
mov     cl, [eax]
mov     [ebp+var_95], cl
add     [ebp+var_9C], 1
cmp     [ebp+var_95], 0
jnz     short loc_10001298
```

```
mov     edx, [ebp+var_9C]
sub     edx, [ebp+var_A8]
mov     [ebp+var_AC], edx
cmp     [ebp+var_AC], 8
jz      short loc_100012E3
```

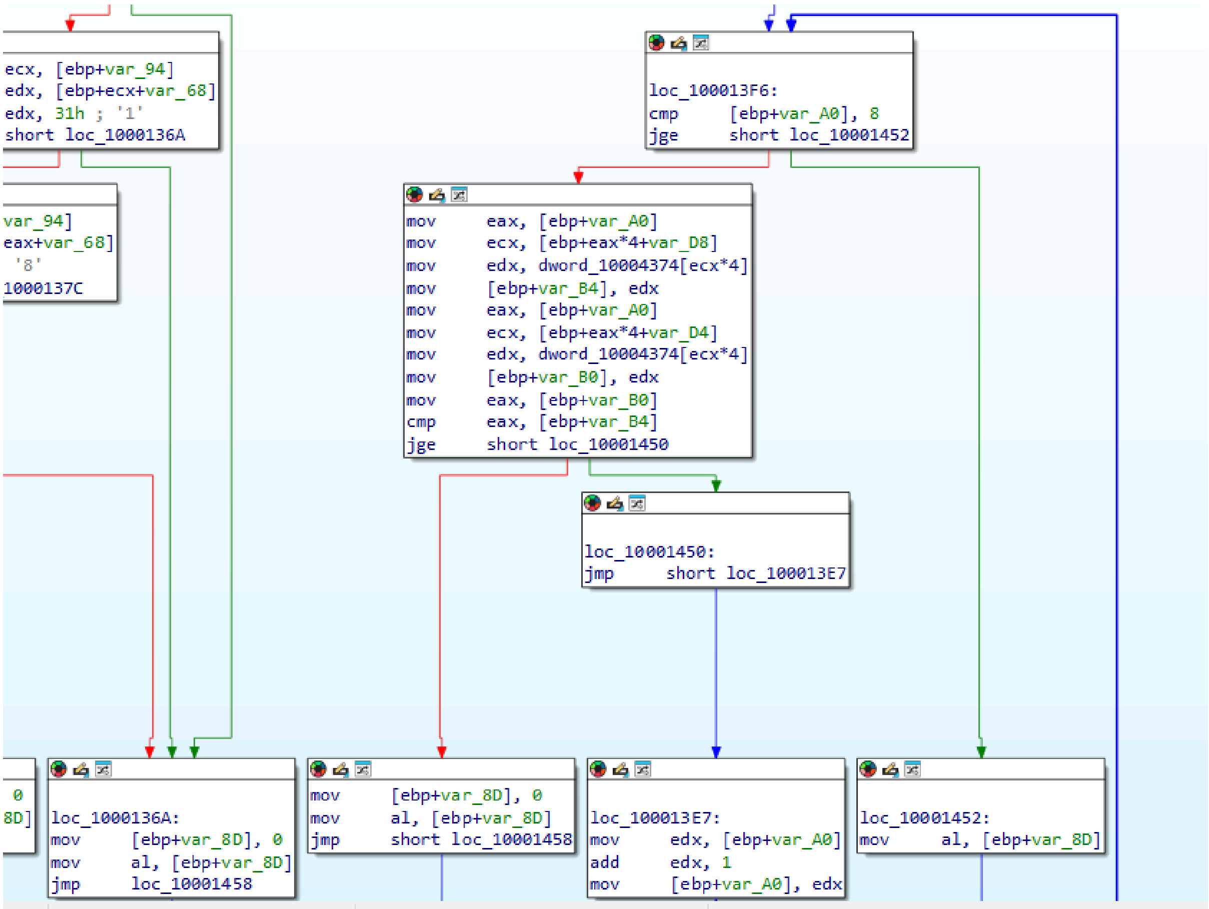
In this part we can see that the function is looking for an input that is 8 numbers long.



```
loc_10001324:
cmp     [ebp+var_94], 8
jge     loc_100013DB
```

Our function goes over our 8 numbers with var_94 and checks if each number is between 1-8

After each of our inputted numbers are verified there is another verification.
Each of the numbers we entered were an index for the hex dump that we got from level 2
In the file dorneAttack.txt.



```
ecx, [ebp+var_94]
edx, [ebp+ecx+var_68]
edx, 31h ; '1'
short loc_1000136A
```

```
var_94]
eax+var_68]
 '8'
1000137C
```

```
loc_100013F6:
cmp     [ebp+var_A0], 8
jge     short loc_10001452
```

```
mov     eax, [ebp+var_A0]
mov     ecx, [ebp+eax*4+var_D8]
mov     edx, dword_10004374[ecx*4]
mov     [ebp+var_B4], edx
mov     eax, [ebp+var_A0]
mov     ecx, [ebp+eax*4+var_D4]
mov     edx, dword_10004374[ecx*4]
mov     [ebp+var_B0], edx
mov     eax, [ebp+var_B0]
cmp     eax, [ebp+var_B4]
jge     short loc_10001450
```

```
loc_10001450:
jmp     short loc_100013E7
```

```
 0
8D]
```

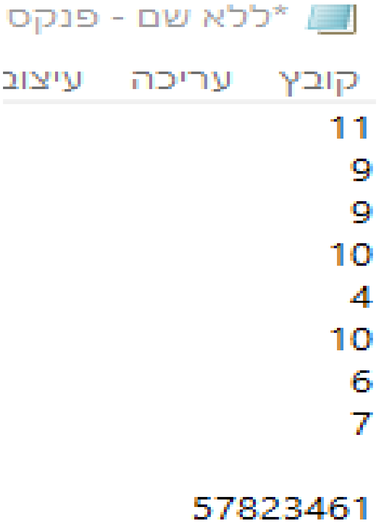
```
loc_1000136A:
mov     [ebp+var_8D], 0
mov     al, [ebp+var_8D]
jmp     loc_10001458
```

```
mov     [ebp+var_8D], 0
mov     al, [ebp+var_8D]
jmp     short loc_10001458
```

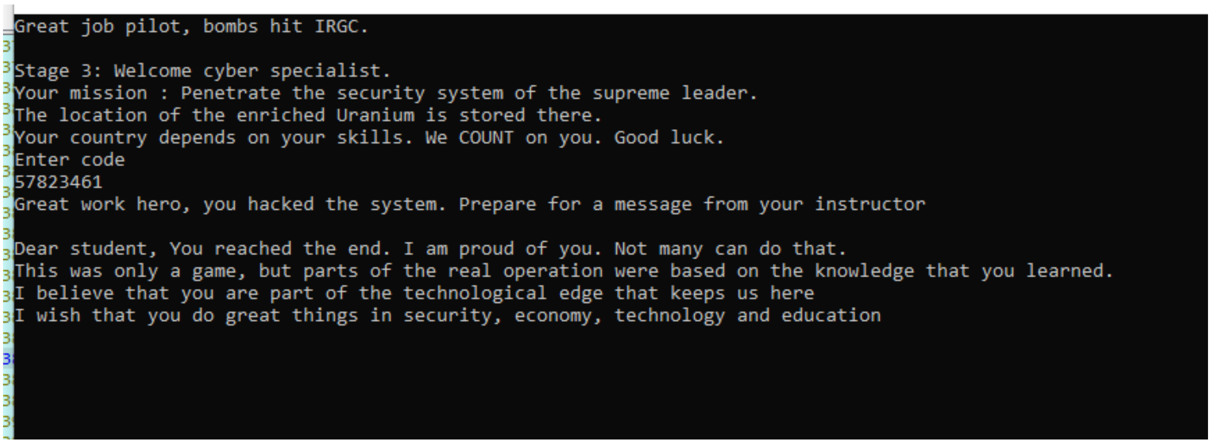
```
loc_100013E7:
mov     edx, [ebp+var_A0]
add     edx, 1
mov     [ebp+var_A0], edx
```

```
loc_10001452:
mov     al, [ebp+var_8D]
```

I added a breakpoint to the function and cautiously calculated the order of the indexes where the numbers were organized in ascending order. (in the function B0 and B4 ,we always check if B0>=B4)
we check in this function if we have 8 digit string-> each of them supposed to be a number->each number is between 1-8 -> that we don't have doubles
 each number indicates an index of dword_10001450 where there are numbers that we need to organize by ascending order(according to B0 B4)



```
      11
       9
       9
      10
       4
      10
       6
       7

57823461
```



```
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
57823461
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
```

Hurray!
(I didn't use any of the clues in the moodle)
Thank you:)