Mission Name
The Library Card

Historical Context
In their quest for knowledge and assistance, Ethan and Claire encounter the Librarian, a gatekeeper of information. To gain his cooperation, they are tasked with obtaining a special card. The Librarian has provided them access to a computer system where this card is hidden.

Technical Synopsis
To secure the Librarian's aid, Ethan is to navigate through the provided computer system, employing techniques to escalate his privileges. The objective is to locate and retrieve the Library Card concealed within the system's files.

Mission Brief
The Librarian has set a condition for his assistance: the acquisition of a crucial card. This card lies hidden within the system he has allowed access to. Your mission involves navigating this system, overcoming security measures to escalate your privileges, and ultimately securing the card for the Librarian. Embark on this task with diligence and acuity. Best of luck in your endeavor!

Detailed Assignment
Ethan and Claire's journey has led them to a pivotal moment where the assistance of the Librarian is within reach, contingent upon the procurement of a specific card. This card, essential for progressing in their quest, is somewhere within the confines of a computer system provided by the Librarian himself.

Tools

IP

Questions

What is the Librarian ID?

- 8F-BE-85-BD-AC-8E

What is the other word that is also reversed?

- CC-00-FF-EE

What is the time expired setted (int)?

- 924100600

Hints

1. .Net reflector decompiler can help, otherwise try to disassembly and debug.
2. Search for the operations done with the strings builded
3. Adjust your localtime for expiration license

Categories

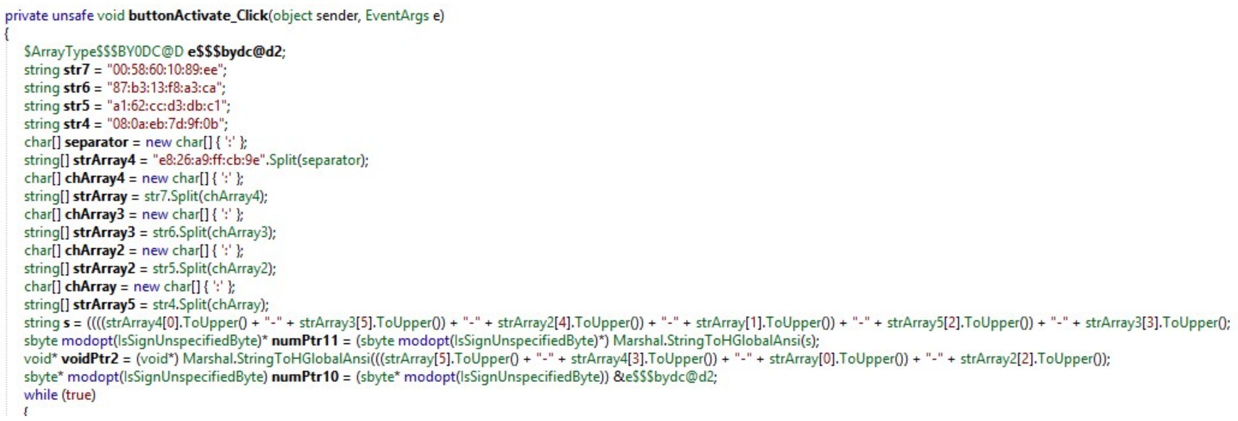
- Windows Reversing
- Decompile
- .Net

Write Up

Step 1: Hardcoded Strings and Manipulation

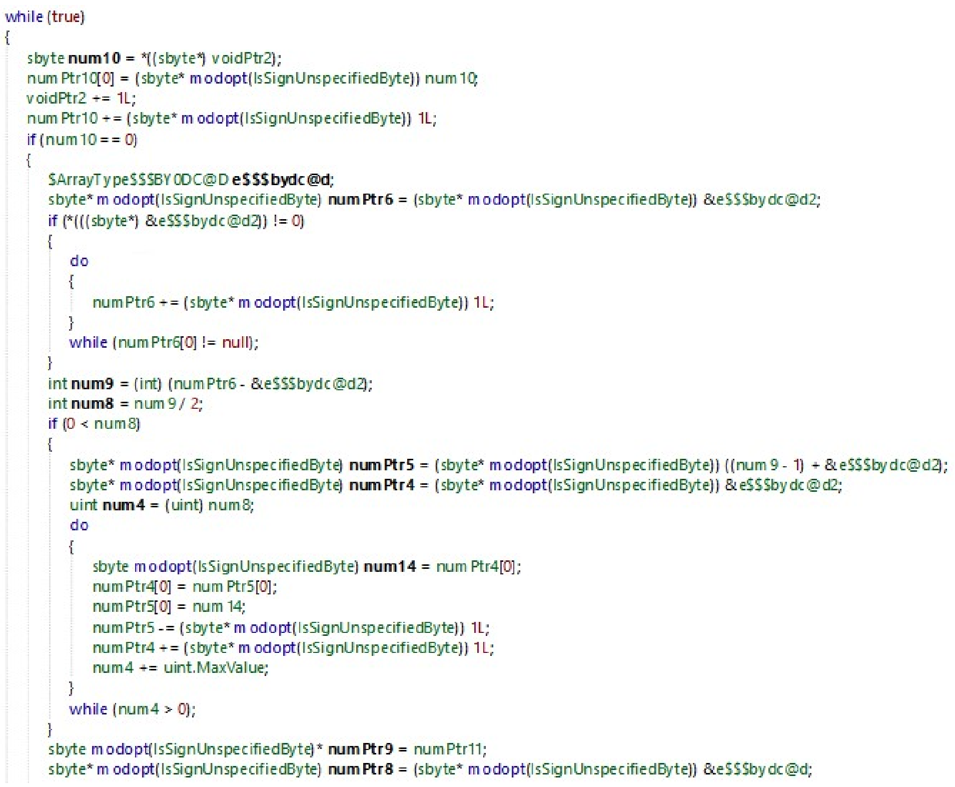
The application contains five hardcoded strings split into arrays of characters using the delimiter ":". From these arrays, two new words are formed using parts of these arrays:

- The first word formed is **E8-CA-DB-58-EB-F8**.
- The second word formed is **EE-FF-00-CC**.



```
private unsafe void buttonActivate_Click(object sender, EventArgs e)
{
    $ArrayType$$$BY0DC@D e$$$bydc@d2;
    string str7 = "00:58:60:10:89:ee";
    string str6 = "87:b3:13:f8:a3:ca";
    string str5 = "a1:62:cc:d3:db:c1";
    string str4 = "08:0a:eb:7d:9f:0b";
    char[] separator = new char[] { ':' };
    string[] strArray4 = "e8:26:a9:ff:cb:9e".Split(separator);
    char[] chArray4 = new char[] { ':' };
    string[] strArray = str7.Split(chArray4);
    char[] chArray3 = new char[] { ':' };
    string[] strArray3 = str6.Split(chArray3);
    char[] chArray2 = new char[] { ':' };
    string[] strArray2 = str5.Split(chArray2);
    char[] chArray = new char[] { ':' };
    string[] strArray5 = str4.Split(chArray);
    string s = ((((strArray4[0].ToUpper() + "-" + strArray3[5].ToUpper()) + "-" + strArray2[4].ToUpper()) + "-" + strArray[1].ToUpper()) + "-" + strArray5[2].ToUpper()) + "-" + strArray3[3].ToUpper();
    sbyte modopt(IsSignUnspecifiedByte)* numPtr11 = (sbyte modopt(IsSignUnspecifiedByte)*) Marshal.StringToHGlobalAnsi(s);
    void* voidPtr2 = (void*) Marshal.StringToHGlobalAnsi(((strArray[5].ToUpper() + "-" + strArray4[3].ToUpper()) + "-" + strArray[0].ToUpper()) + "-" + strArray2[2].ToUpper());
    sbyte* modopt(IsSignUnspecifiedByte) numPtr10 = (sbyte* modopt(IsSignUnspecifiedByte)) &e$$$bydc@d2;
    while (true)
    {
```

Figure 1

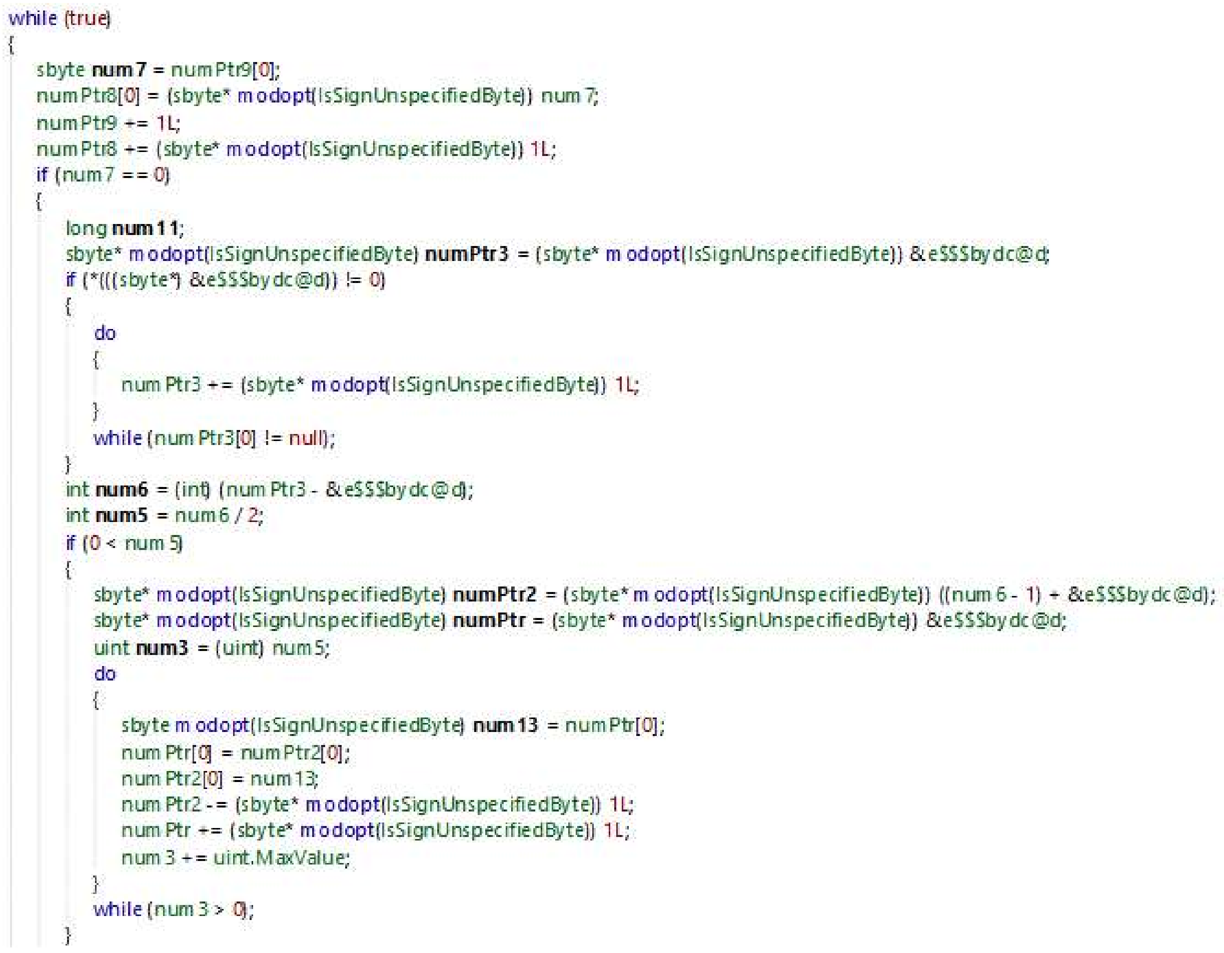


```
while (true)
{
    sbyte num10 = *((sbyte*) voidPtr2);
    numPtr10[0] = (sbyte* modopt(IsSignUnspecifiedByte)) num10;
    voidPtr2 += 1L;
    numPtr10 += (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
    if (num10 == 0)
    {
        $ArrayType$$$BY0DC@D e$$$bydc@d;
        sbyte* modopt(IsSignUnspecifiedByte) numPtr6 = (sbyte* modopt(IsSignUnspecifiedByte)) &e$$$bydc@d2;
        if (*(((sbyte*) &e$$$bydc@d2)) != 0)
        {
            do
            {
                numPtr6 += (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
            }
            while (numPtr6[0] != null);
        }
        int num9 = (int) (numPtr6 - &e$$$bydc@d2);
        int num8 = num9 / 2;
        if (0 < num8)
        {
            sbyte* modopt(IsSignUnspecifiedByte) numPtr5 = (sbyte* modopt(IsSignUnspecifiedByte)) ((num9 - 1) + &e$$$bydc@d2);
            sbyte* modopt(IsSignUnspecifiedByte) numPtr4 = (sbyte* modopt(IsSignUnspecifiedByte)) &e$$$bydc@d2;
            uint num4 = (uint) num8;
            do
            {
                sbyte modopt(IsSignUnspecifiedByte) num14 = numPtr4[0];
                numPtr4[0] = numPtr5[0];
                numPtr5[0] = num14;
                numPtr5 -= (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
                numPtr4 += (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
                num4 += uint.MaxValue;
            }
            while (num4 > 0);
        }
        sbyte modopt(IsSignUnspecifiedByte)* numPtr9 = numPtr11;
        sbyte* modopt(IsSignUnspecifiedByte) numPtr8 = (sbyte* modopt(IsSignUnspecifiedByte)) &e$$$bydc@d;
```

Figure 2

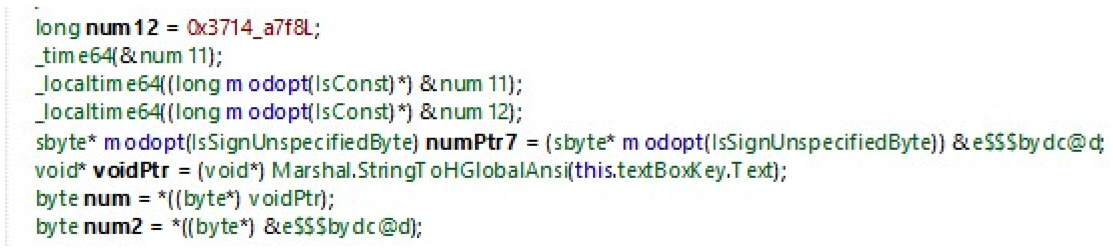
Step 2: String Reversal

The application then reverses the second word, resulting in CC-00-FF-EE, and similarly reverses the first word, resulting in 8F-BE-85-BD-AC-8E.



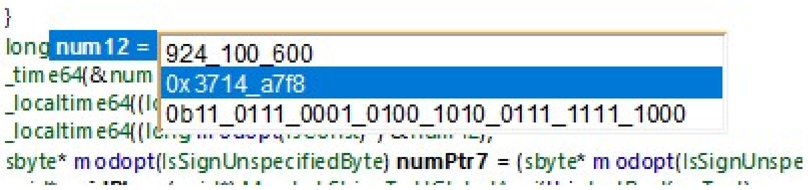
```
while (true)
{
    sbyte num7 = numPtr9[0];
    numPtr8[0] = (sbyte* modopt(IsSignUnspecifiedByte)) num7;
    numPtr9 += 1L;
    numPtr8 += (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
    if (num7 == 0)
    {
        long num11;
        sbyte* modopt(IsSignUnspecifiedByte) numPtr3 = (sbyte* modopt(IsSignUnspecifiedByte)) &e$$$bydc@d;
        if (*(((sbyte*) &e$$$bydc@d)) != 0)
        {
            do
            {
                numPtr3 += (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
            }
            while (numPtr3[0] != null);
        }
        int num6 = (int) (numPtr3 - &e$$$bydc@d);
        int num5 = num6 / 2;
        if (0 < num5)
        {
            sbyte* modopt(IsSignUnspecifiedByte) numPtr2 = (sbyte* modopt(IsSignUnspecifiedByte)) ((num6 - 1) + &e$$$bydc@d);
            sbyte* modopt(IsSignUnspecifiedByte) numPtr = (sbyte* modopt(IsSignUnspecifiedByte)) &e$$$bydc@d;
            uint num3 = (uint) num5;
            do
            {
                sbyte modopt(IsSignUnspecifiedByte) num13 = numPtr[0];
                numPtr[0] = numPtr2[0];
                numPtr2[0] = num13;
                numPtr2 -= (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
                numPtr += (sbyte* modopt(IsSignUnspecifiedByte)) 1L;
                num3 += uint.MaxValue;
            }
            while (num3 > 0);
        }
```

Figure 3



```
long num12 = 0x3714_a7f8L;
_time64(&num11);
_localtime64((long modopt(IsConst)*) &num11);
_localtime64((long modopt(IsConst)*) &num12);
sbyte* modopt(IsSignUnspecifiedByte) numPtr7 = (sbyte* modopt(IsSignUnspecifiedByte)) &e$$$bydc@d;
void* voidPtr = (void*) Marshal.StringToHGlobalAnsi(this.textBoxKey.Text);
byte num = *((byte*) voidPtr);
byte num2 = *((byte*) &e$$$bydc@d);
```

Figure 4



```
}
long num12 =
_time64(&num
_localtime64((l
_localtime64((l
sbyte* modopt(IsSignUnspecifiedByte) numPtr7 = (sbyte* modopt(IsSignUnspe
```

```
924_100_600
0x3714_a7f8
0b11_0111_0001_0100_1010_0111_1111_1000
```

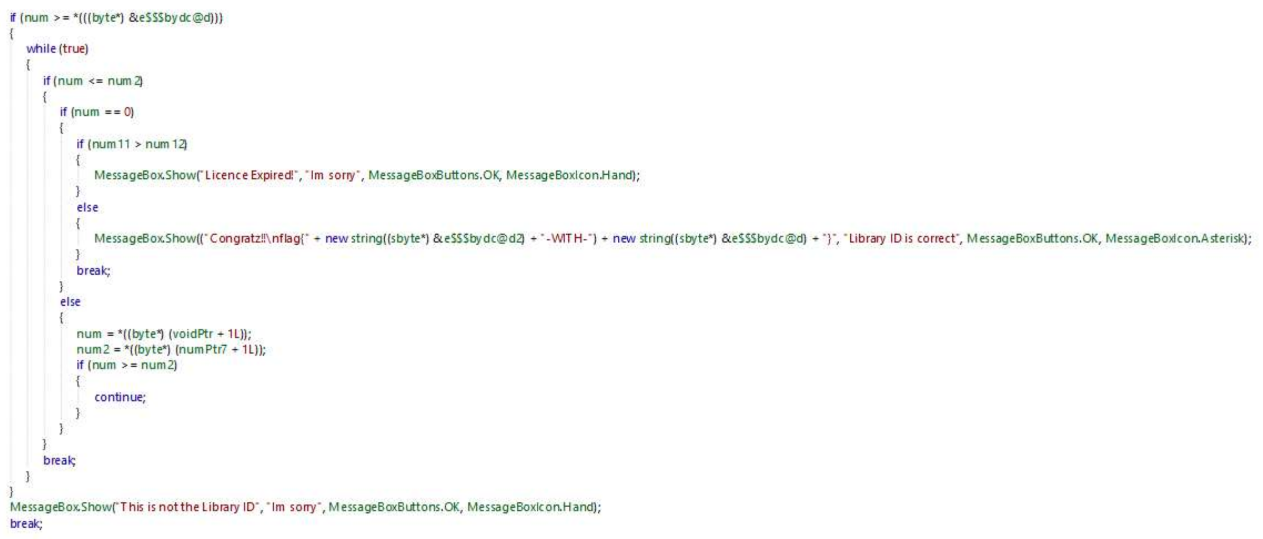
Figure 5

Step 3: Time Comparison

The program compares the current local time on the machine against a hardcoded timestamp, specifically checking if the current year is before 1991.

Step 4: Library ID Input and Validation

For successful validation, the user must input the reversed first string, 8F-BE-85-BD-AC-8E, as the Library ID. Additionally, the system's date must be set to a year before 1991 to pass the time check.



```
if (num >= *(((byte*) &e$$$bydc@d)))
{
    while (true)
    {
        if (num <= num2)
        {
            if (num == 0)
            {
                if (num11 > num12)
                {
                    MessageBox.Show("Licence Expired!", "Im sorry", MessageBoxButtons.OK, MessageBoxIcon.Hand);
                }
                else
                {
                    MessageBox.Show(("Congratz!!\nflag{" + new string((sbyte*) &e$$$bydc@d2) + "-WITH-") + new string((sbyte*) &e$$$bydc@d) + "}", "Library ID is correct", MessageBoxButtons.OK, MessageBoxIcon.Asterisk);
                }
                break;
            }
            else
            {
                num = *((byte*) (voidPtr + 1L));
                num2 = *((byte*) (numPtr7 + 1L));
                if (num >= num2)
                {
                    continue;
                }
            }
        }
        break;
    }
}
MessageBox.Show("This is not the Library ID", "Im sorry", MessageBoxButtons.OK, MessageBoxIcon.Hand);
break;
```

Figure 6

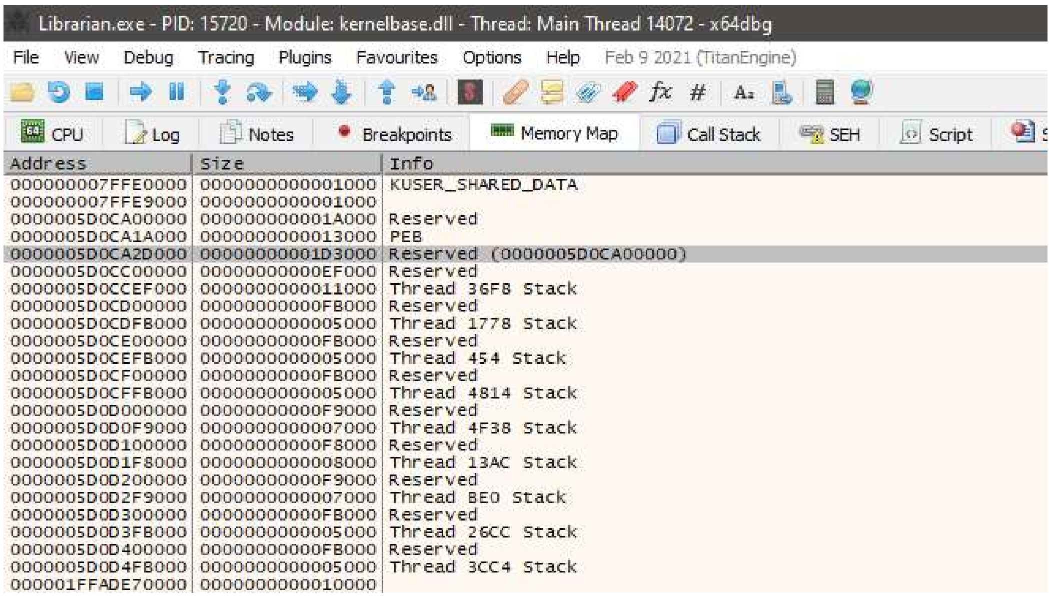
Step 5: Obtaining the Flag

If both checks are passed (the Library ID and the date check), the application generates a flag in the format:

flag{CC-00-FF-EE-WITH-8F-BE-85-BD-AC-8E}

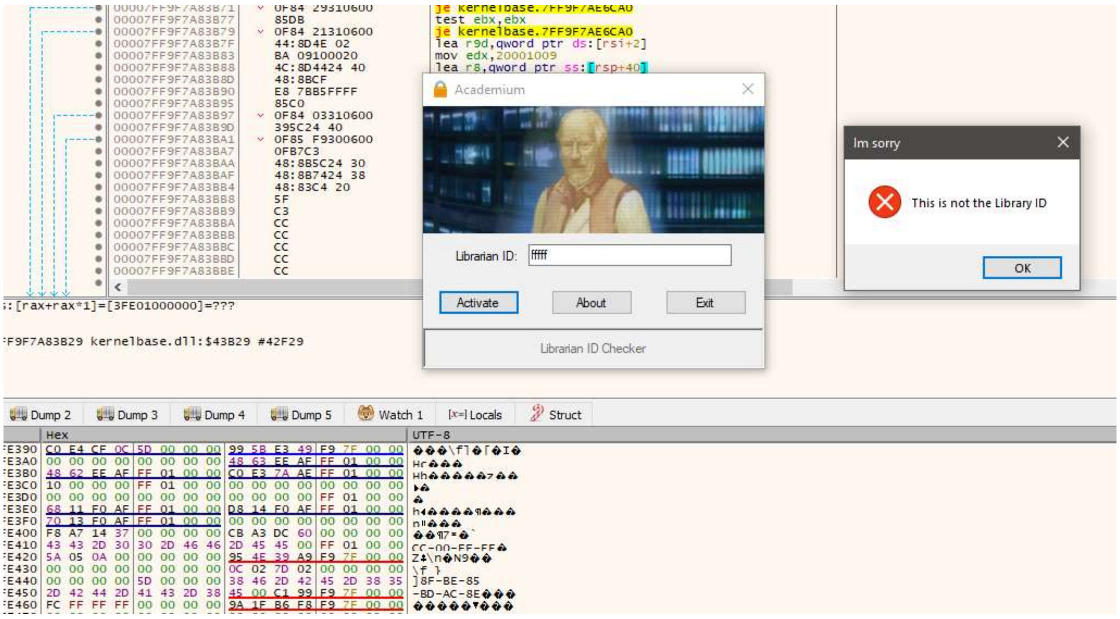
Debugging and Memory Dump

To observe the ID in execution time and potentially bypass the validation checks, one could debug the application and inspect the memory dump. Initiating the program, entering any key, and inspecting the memory before dismissing any error messages can reveal the ID. However, after proceeding past the error message, the relevant data will be cleared from memory.



```
Address          Size             Info
000000007FFE0000 0000000000001000 KUSER_SHARED_DATA
000000007FFE9000 0000000000001000
0000005D0CA00000 000000000001A000 Reserved
0000005D0CA1A000 0000000000013000 PEB
0000005D0CA2D000 00000000001D3000 Reserved (0000005D0CA00000)
0000005D0CC00000 00000000000EF000 Reserved
0000005D0CCEF000 0000000000011000 Thread 36F8 Stack
0000005D0CD00000 00000000000FB000 Reserved
0000005D0CDFB000 0000000000005000 Thread 1778 Stack
0000005D0CE00000 00000000000FB000 Reserved
0000005D0CEFB000 0000000000005000 Thread 454 Stack
0000005D0CF00000 00000000000FB000 Reserved
0000005D0CFFB000 0000000000005000 Thread 4814 Stack
0000005D0D000000 00000000000F9000 Reserved
0000005D0D0F9000 0000000000007000 Thread 4F38 Stack
0000005D0D100000 00000000000F8000 Reserved
0000005D0D1F8000 0000000000008000 Thread 13AC Stack
0000005D0D200000 00000000000F9000 Reserved
0000005D0D2F9000 0000000000007000 Thread BE0 Stack
0000005D0D300000 00000000000FB000 Reserved
0000005D0D3FB000 0000000000005000 Thread 26CC Stack
0000005D0D400000 00000000000FB000 Reserved
0000005D0D4FB000 0000000000005000 Thread 3CC4 Stack
000001FFADE70000 0000000000010000
```

Figure 7



```
00007FF9F7A83B71   0F84 29310600    je kernelbase.7FF9F7AE6CA0
00007FF9F7A83B77   85DB             test ebx,ebx
00007FF9F7A83B79   0F84 21310600    je kernelbase.7FF9F7AE6CA0
00007FF9F7A83B7F   44:8D4E 02       lea r9d,qword ptr ds:[rsi+2]
00007FF9F7A83B83   BA 09100020      mov edx,20001009
00007FF9F7A83B88   4C:8D4424 40     lea r8,qword ptr ss:[rsp+40]
00007FF9F7A83B8D   48:8BCF
00007FF9F7A83B90   E8 7BB5FFFF
00007FF9F7A83B95   85C0
00007FF9F7A83B97   0F84 03310600
00007FF9F7A83B9D   395C24 40
00007FF9F7A83BA1   0F85 F9300600
00007FF9F7A83BA7   0FB7C3
00007FF9F7A83BAA   48:8B5C24 30
00007FF9F7A83BAF   48:8B7424 38
00007FF9F7A83BB4   48:83C4 20
00007FF9F7A83BB8   5F
00007FF9F7A83BB9   C3
00007FF9F7A83BBA   CC
00007FF9F7A83BBB   CC
00007FF9F7A83BBC   CC
00007FF9F7A83BBD   CC
00007FF9F7A83BBE   CC

:[rax+rax*1]=[3FE01000000]=???

FF9F7A83B29 kernelbase.dll:$43B29 #42F29
```

Librarian ID: ffffff

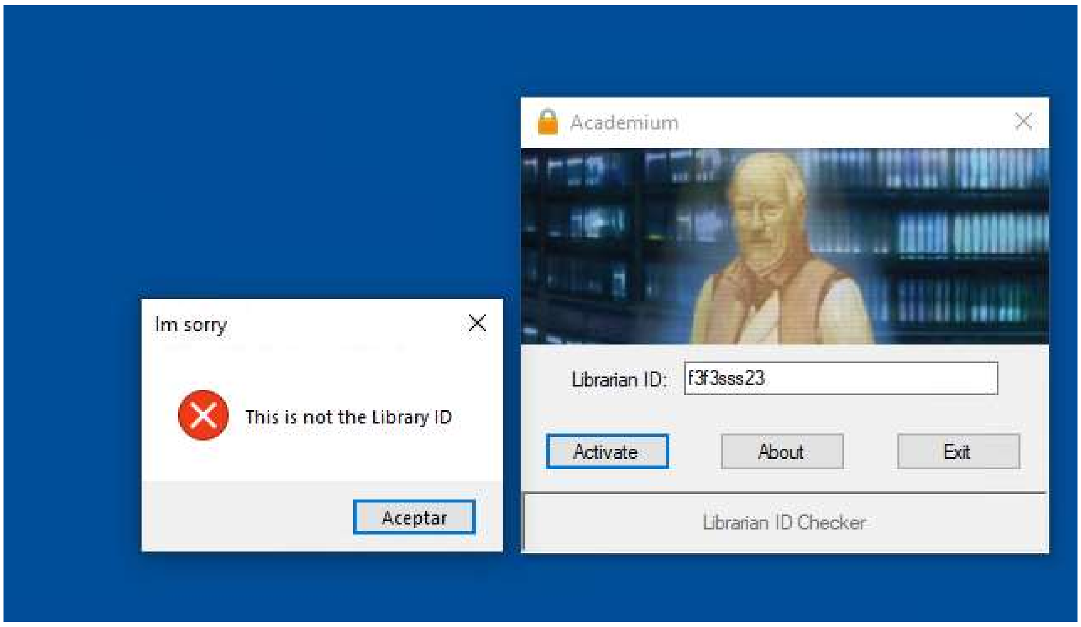
This is not the Library ID

```
      Hex                                               UTF-8
FE390 C0 E4 CF 0C 5D 00 00 00 99 5B E3 49 F9 7F 00 00  ���\f]�[�I�
FE3A0 00 00 00 00 00 00 00 00 48 63 EE AF FF 01 00 00  Hc���
FE3B0 48 62 EE AF FF 01 00 00 C0 E3 7A AE FF 01 00 00  Hh�����z��
FE3C0 10 00 00 00 FF 01 00 00 00 00 00 00 00 00 00 00  ▸�
FE3D0 00 00 00 00 00 00 00 00 00 00 00 00 FF 01 00 00  �
FE3E0 68 11 F0 AF FF 01 00 00 D8 14 F0 AF FF 01 00 00  h◂����¶���
FE3F0 70 13 F0 AF FF 01 00 00 00 00 00 00 00 00 00 00  p‼���
FE400 F8 A7 14 37 00 00 00 00 CB A3 DC 60 00 00 00 00  ��¶7*�`
FE410 43 43 2D 30 30 2D 46 46 2D 45 45 00 FF 01 00 00  CC-00-FF-EE�
FE420 5A 05 0A 00 00 00 00 00 95 4E 39 A9 F9 7F 00 00  Z♣\n�N9��
FE430 00 00 00 00 00 00 00 00 0C 02 7D 02 00 00 00 00  \f }
FE440 00 00 00 00 5D 00 00 00 38 46 2D 42 45 2D 38 35  ]8F-BE-85
FE450 2D 42 44 2D 41 43 2D 38 45 00 C1 99 F9 7F 00 00  -BD-AC-8E���
FE460 FC FF FF FF 00 00 00 00 9A 1F B6 F8 F9 7F 00 00  �����▼���
```

Figure 8

Disassembly Analysis

Although disassembling the application with tools like IDA Pro offers another approach to understanding the code, it may present more complexity in interpreting the logic and flow compared to direct decompilation and analysis.

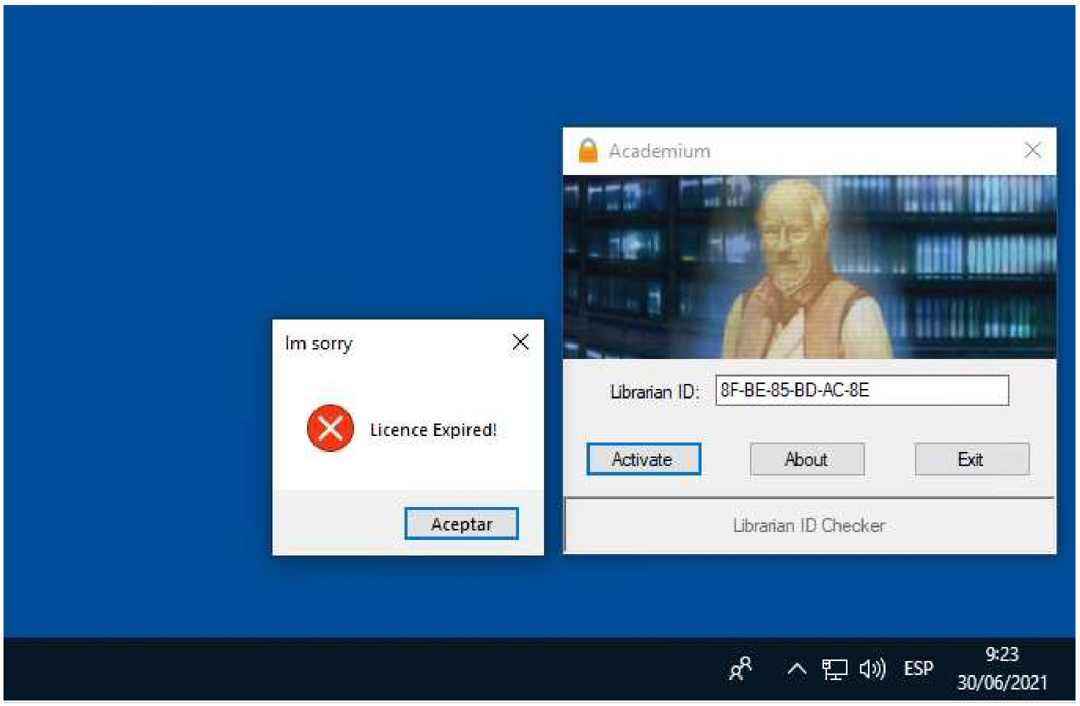


Im sorry

This is not the Library ID

Librarian ID: f3f3sss23

Figure 9

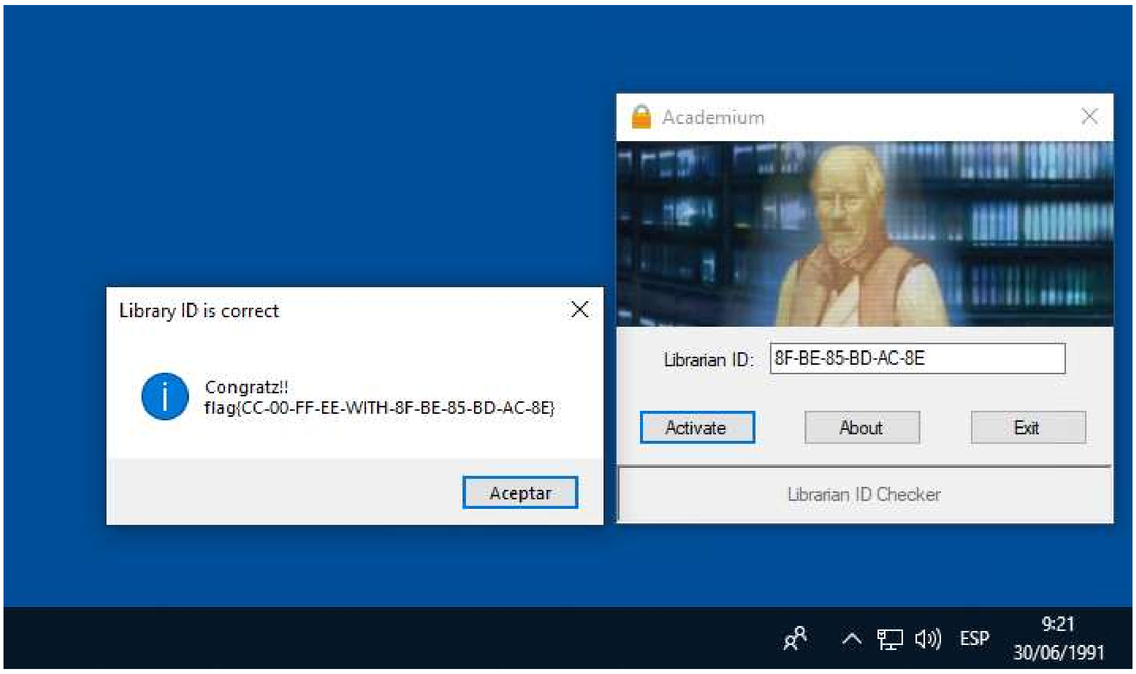


Im sorry

Licence Expired!

Librarian ID: 8F-BE-85-BD-AC-8E

Figure 10



Library ID is correct

Congratz!!
flag{CC-00-FF-EE-WITH-8F-BE-85-BD-AC-8E}

Librarian ID: 8F-BE-85-BD-AC-8E

Figure 11

Executing the Program for the Flag

Incorrect ID Case: The application rejects invalid Library IDs.

Correct ID but Incorrect Date Case: The correct ID without the appropriate date setting will also result in rejection.

Successful Flag Retrieval: Setting the system's date to a year before 1991 and providing the correct Library ID will result in the application displaying the flag.
This analysis showcases the importance of understanding application logic, string manipulation, and condition checks in reverse engineering efforts. Through methodical examination and testing, one can decipher the necessary steps to satisfy an application's conditions and reveal hidden or protected information like flags in cybersecurity challenges.

Flag Information

flag{CC-00-FF-EE-WITH-8F-BE-85-BD-AC-8E}