SPaASM Assignment 3 - Documentation

Ondrej Krajčovič

Slovenská technická univerzita v Bratislave

Fakulta informatiky a informačných technológií

Systémové programovanie a asmeblery

xkrajcovico@stuba.sk

29. 4 2025

IV.Semester Bc Štúdia

Input Length, Acceptance String and understanding the assignment

At the beginning, i found the string J#ki80Ys and i thought that would be the value needed to pass the geek2 window. Since it was not the right answer and it was 8 characters long, i thought it would be a trace of the password and it would be encoded. I tried to crack it as hash, md5, crc, caesareu’s cipher, base 32, base 64 encoding, and i still couldn’t move on. Then i realised that the J#ki80Ys was just garbage data for allocation of the 8 char string, and each letter was replaced by mapping of the longer string in function FUN_00401146. Since then it was really easy to crack and replace encoding of the password. The original password is FIITgeek, I recreated if from finding indexes from the longer string as they were being assigned in FUN_00401146.

Section .data from strstr.exe:

```
                    //
                    // .data
                    // ram:00403000-ram:004031ff
                    //
                    s_Right_!_00403000                XREF[3]:     00400214(*), 004010d1(*),
                                                                   004010d6(*)
    00403000 52 69 67        ds        "Right !"
         68 74 20
         21 00
                    s_Wrong_!_00403008                 XREF[2]:     004010dd(*), 004010e2(*)
    00403008 57 72 6f        ds        "Wrong !"
         6e 67 20
         21 00
                    s_I4561AsEmblerySuPOhodicka2x3Xzgv_00403010     XREF[1]:     004010ae(*)
    00403010 49 34 35        ds        "I4561AsEmblerySuPOhodicka2x3XzgvwpqLJfBCDnFMH
         36 31 41
         73 45 6d
                    s_J#ki80Ys_0040304f                XREF[1]:     004010a9(*)
    0040304f 4a 23 6b        ds        "J#ki80Ys"
         69 38 30
         59 73 00
```

Decompiled mapping function:
undefined1 * FUN_00401146(undefined1 *param_1,undefined1 *param_2)

```
{
  *param_2 = param_1[0x12];
  param_2[1] = *param_1;
  param_2[2] = param_1[0x24];
  param_2[3] = param_1[0x18];
  param_2[4] = param_1[0x16];
  param_2[5] = param_1[0x17];
  param_2[6] = param_1[0xb];
  param_2[7] = param_1[0x14];
  return param_2;
}
```

Function Calls And Return Values of Windows API

DialogBoxParam: Creates a modal dialog box using a template from resources.

- Arguments: hInstance, lpTemplate, hWndParent, lpDialogFunc, dwInitParam.
- Return value: INT_PTR (typically the result of the dialog).

GetDlgItemText: Gets text from a specific dialog element (e.g. a text field).

- Arguments: hDlg, nIDDlgItem, lpString, cchMax.
- Return value: UINT (length of the copied string).

MessageBox: Displays an information box with text.

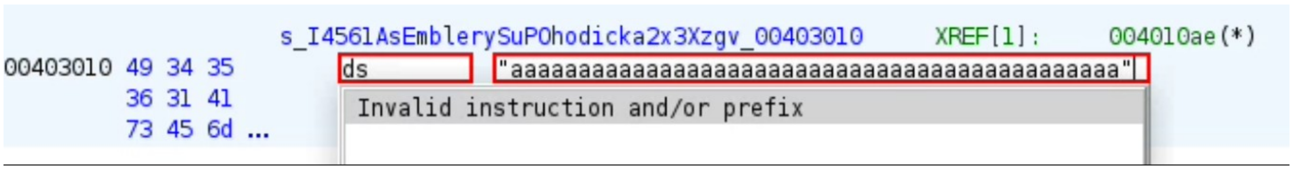
- Arguments: hWnd, lpText, lpCaption, uType.
- Return value: int (ID of the pressed button).

1. Calling of the function GetDlgItemText:
    - Call address: 00401097.
    - Gets the text from the element with ID 0x65 (101) in the dialog (probably a text field) and stores it in DAT_00403058.

2. Calling of the function DialogBoxParam:
    - Call address: 0040101e.
    - Creates a dialog box from the template with ID 0x64 (100) and sets the message handler to LAB_0040102a.

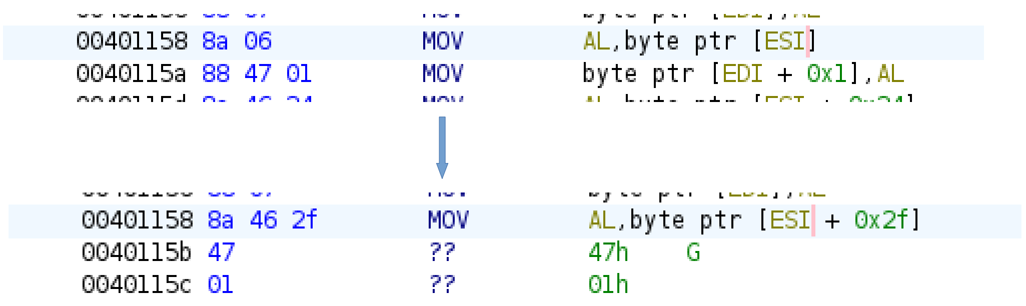
3. Calling of the function CorrectMessageBox:
    - Call address: 004010ea.
    - Displayed if the input string is correct.
    - Text: "Right !" (address 00403000)

Custom Accepted String Replacement



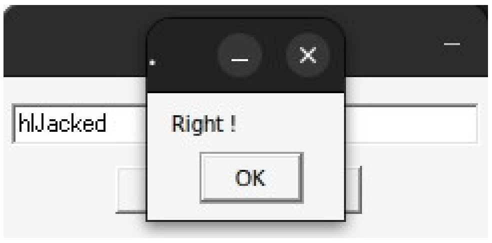
```
                        s_I4561AsEmblerySuP0hodicka2x3Xzgv_00403010     XREF[1]:     004010ae(*)
00403010 49 34 35       ds          "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
         36 31 41       Invalid instruction and/or prefix
         73 45 6d ...
```

Since I was unable to change the longer (mapping) string (ghidra did not allow me to change the *ds* command), I decided to use the original. I searched for a suitable word of length 8 characters. Since I was using the original string, I tried to implement the password as my name *Ondrej+ko*. But after trying to change the *FUN_00401146*, I realized I would need a lot of workaround to be able to replace the second character (uppercase I) It had some kind of problem with how address and required to add two more commands after it to assign correct amount of memory. I couldn't solve it. So my third option, *hIJacked*, was implemented. To implement I just replaced the offset in each character assigning in the *FUN_00401146*.



```
00401158 8a 06          MOV       AL,byte ptr [ESI]
0040115a 88 47 01       MOV       byte ptr [EDI + 0x1],AL
```

```
00401158 8a 46 2f       MOV       AL,byte ptr [ESI + 0x2f]
0040115b 47             ??        47h    G
0040115c 01             ??        01h
```



Modified FUN_00401146 function:

```
                **************************************************************
                *                          FUNCTION                          *
                **************************************************************
                undefined1 * __stdcall FUN_00401146(undefined1 * param_1
     undefined1 *     EAX:4        <RETURN>
     undefined1 *     Stack[0x4]:4   param_1                        XREF[1]:     0040114d(R)
     undefined1 *     Stack[0x8]:4   param_2                        XREF[1]:     00401150(R)
                FUN_00401146                           XREF[1]:     004010b3(c)
  00401146 55             PUSH      EBP
  00401147 8b ec          MOV       EBP,ESP
  00401149 56             PUSH      ESI
  0040114a 57             PUSH      EDI
  0040114b 33 c0          XOR       EAX,EAX
  0040114d 8b 75 08       MOV       ESI,dword ptr [EBP + param_1]          //set parameter_1 for long string
  00401150 8b 7d 0c       MOV       EDI,dword ptr [EBP + param_2]          //set parameter_2 for short string
  00401153 8a 46 12       MOV       AL,byte ptr [ESI + 0x12]               //h
  00401156 88 07          MOV       byte ptr [EDI],AL
  00401158 8a 06          MOV       AL,byte ptr [ESI]                      //I
  0040115a 88 47 01       MOV       byte ptr [EDI + 0x1],AL
  0040115d 8a 46 24       MOV       AL,byte ptr [ESI + 0x24]               //J
  00401160 88 47 02       MOV       byte ptr [EDI + 0x2],AL
  00401163 8a 46 18       MOV       AL,byte ptr [ESI + 0x18]               //a
  00401166 88 47 03       MOV       byte ptr [EDI + 0x3],AL
  00401169 8a 46 16       MOV       AL,byte ptr [ESI + 0x16]               //c
  0040116c 88 47 04       MOV       byte ptr [EDI + 0x4],AL
  0040116f 8a 46 17       MOV       AL,byte ptr [ESI + 0x17]               //k
  00401172 88 47 05       MOV       byte ptr [EDI + 0x5],AL
  00401175 8a 46 0b       MOV       AL,byte ptr [ESI + 0xb]                //e
  00401178 88 47 06       MOV       byte ptr [EDI + 0x6],AL
  0040117b 8a 46 14       MOV       AL,byte ptr [ESI + 0x14]               //d
  0040117e 88 47 07       MOV       byte ptr [EDI + 0x7],AL
  00401181 8b c7          MOV       EAX,EDI
  00401183 5f             POP       EDI
  00401184 5e             POP       ESI
  00401185 c9             LEAVE
  00401186 c2 08 00       RET       0x8
     assume FS_OFFSET = 0xffdff000
  00401189 cc             ??        CCh
```

Table of Contents