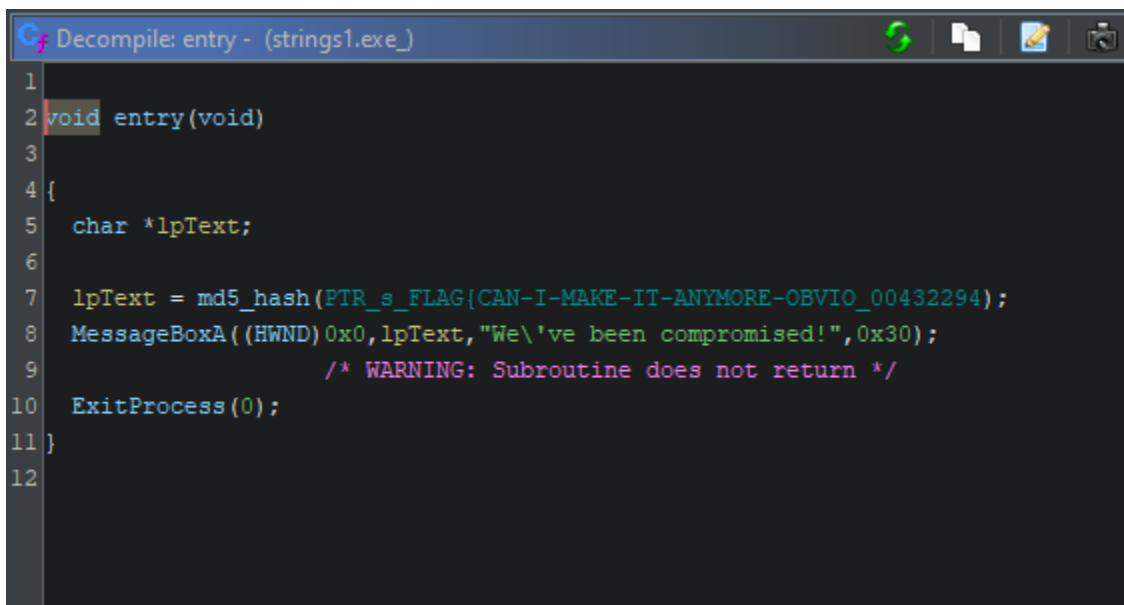


Basic Malware RE Task 1:

We are required to only use static reverse engineering.



Detect it easy shows that this is a C++ application. I then load this into Ghidra and start the analysis process.

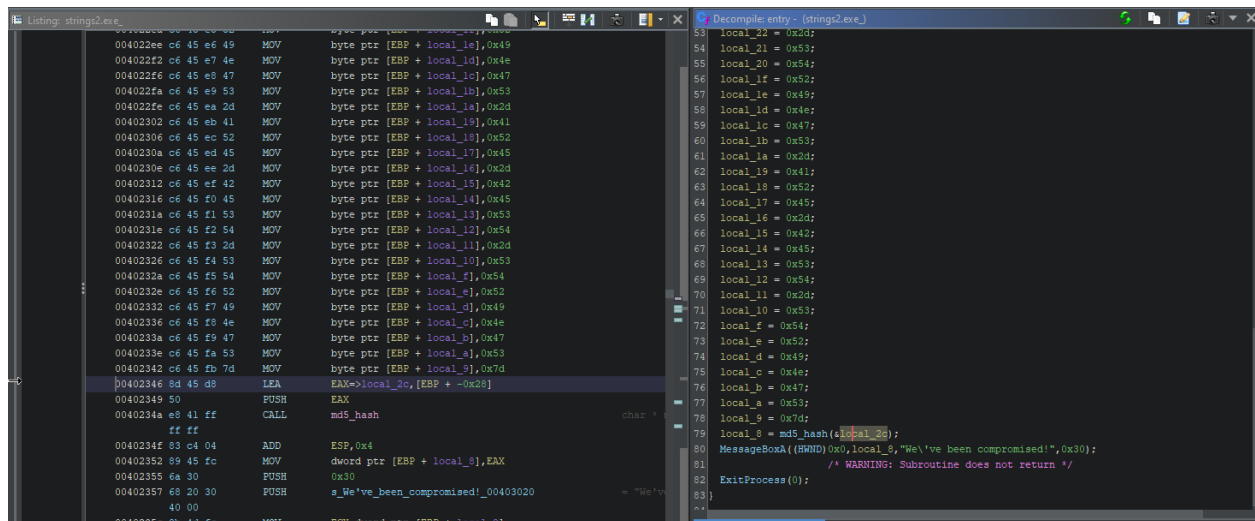


Click on the ptr to flag and we get

```
00424828 46 4c 41 ds s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOUS} "FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOUS}"
47 7b 43
41 4e 2d ...
```

Task 2

Do the same load it into Ghidra

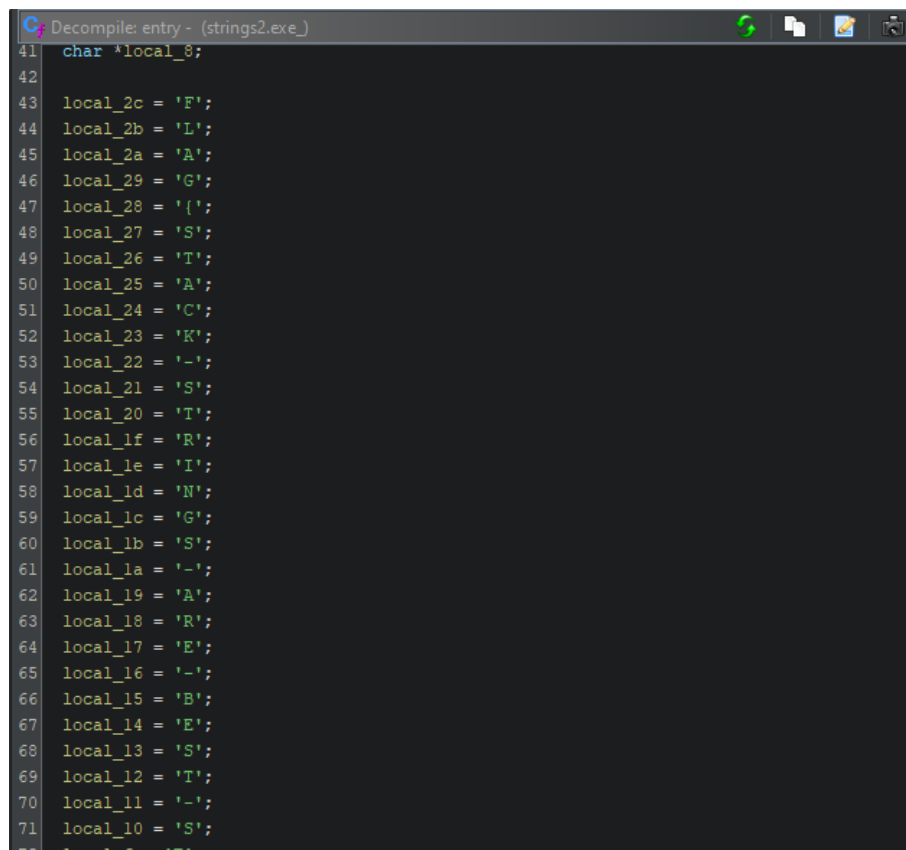


The screenshot shows the Ghidra interface with two panes. The left pane displays assembly code for 'Listing: strings2.exe'. The right pane displays the decompiled code for 'Decompile entry - (strings2.exe)'. The assembly code shows a series of MOV instructions loading values from memory into registers, followed by a CALL instruction to md5_hash. The decompiled code shows the corresponding high-level operations, including the initialization of local variables and the call to md5_hash.

```
Listing: strings2.exe
004022ee c6 45 e6 49 MOV byte ptr [EBP + local_1e], 0x49
004022f2 c6 45 e7 4e MOV byte ptr [EBP + local_1d], 0x4e
004022f6 c6 45 e8 47 MOV byte ptr [EBP + local_1c], 0x47
004022fa c6 45 e9 53 MOV byte ptr [EBP + local_1b], 0x53
004022fe c6 45 ea 2d MOV byte ptr [EBP + local_1a], 0x2d
00402302 c6 45 eb 41 MOV byte ptr [EBP + local_19], 0x41
00402306 c6 45 ec 52 MOV byte ptr [EBP + local_18], 0x52
0040230a c6 45 ed 45 MOV byte ptr [EBP + local_17], 0x45
0040230e c6 45 ee 2d MOV byte ptr [EBP + local_16], 0x2d
00402312 c6 45 ef 42 MOV byte ptr [EBP + local_15], 0x42
00402316 c6 45 f0 45 MOV byte ptr [EBP + local_14], 0x45
0040231a c6 45 f1 53 MOV byte ptr [EBP + local_13], 0x53
0040231e c6 45 f2 54 MOV byte ptr [EBP + local_12], 0x54
00402322 c6 45 f3 2d MOV byte ptr [EBP + local_11], 0x2d
00402326 c6 45 f4 53 MOV byte ptr [EBP + local_10], 0x53
0040232a c6 45 f5 54 MOV byte ptr [EBP + local_0f], 0x54
0040232e c6 45 f6 52 MOV byte ptr [EBP + local_0e], 0x52
00402332 c6 45 f7 49 MOV byte ptr [EBP + local_0d], 0x49
00402336 c6 45 f8 4e MOV byte ptr [EBP + local_0c], 0x4e
0040233a c6 45 f9 47 MOV byte ptr [EBP + local_0b], 0x47
0040233e c6 45 fa 53 MOV byte ptr [EBP + local_0a], 0x53
00402342 c6 45 fb 7d MOV byte ptr [EBP + local_09], 0x7d
00402346 bd 45 d8 LEA EAX>local_2c, [EBP + -0x28]
00402349 50 PUSH EAX
0040234a e8 41 ff CALL md5_hash
0040234f ff ff
00402352 83 c4 04 ADD ESP, 0x4
0040235d 89 45 fc MOV dword ptr [EBP + local_09], EAX
00402365 6a 30 PUSH 0x30
00402367 68 20 30 PUSH s_We've_been_compromised!_00403020
00402369 40 00
0040236c 8b 4d fc MOV ECX, dword ptr [EBP + local_01]

Decompile entry - (strings2.exe)
53 local_22 = 0x2d;
54 local_21 = 0x53;
55 local_20 = 0x54;
56 local_1f = 0x52;
57 local_1e = 0x49;
58 local_1d = 0x4e;
59 local_1c = 0x47;
60 local_1b = 0x53;
61 local_1a = 0x2d;
62 local_19 = 0x41;
63 local_18 = 0x52;
64 local_17 = 0x45;
65 local_16 = 0x2d;
66 local_15 = 0x42;
67 local_14 = 0x45;
68 local_13 = 0x53;
69 local_12 = 0x54;
70 local_11 = 0x2d;
71 local_10 = 0x53;
72 local_0f = 0x54;
73 local_0e = 0x52;
74 local_0d = 0x49;
75 local_0c = 0x4e;
76 local_0b = 0x47;
77 local_0a = 0x53;
78 local_09 = 0x7d;
79 local_08 = md5_hash(local_2c);
80 MessageBoxA(HWND 0x0, local_08, "We've been compromised!", 0x30);
81 /* WARNING: Subroutine does not return */
82 ExitProcess(0);
83
```

We are loading hex values into the md5_hash function. Therefore all we have to do is decode the hex values.



The screenshot shows the Ghidra interface with the decompiled code for 'Decompile entry - (strings2.exe)'. The code shows the initialization of local variables and the call to md5_hash.

```
Decompile entry - (strings2.exe)
41 char *local_08;
42
43 local_2c = 'F';
44 local_2b = 'L';
45 local_2a = 'A';
46 local_29 = 'G';
47 local_28 = '{';
48 local_27 = 'S';
49 local_26 = 'T';
50 local_25 = 'A';
51 local_24 = 'C';
52 local_23 = 'K';
53 local_22 = '-';
54 local_21 = 'S';
55 local_20 = 'T';
56 local_1f = 'R';
57 local_1e = 'I';
58 local_1d = 'N';
59 local_1c = 'G';
60 local_1b = 'S';
61 local_1a = '-';
62 local_19 = 'A';
63 local_18 = 'R';
64 local_17 = 'E';
65 local_16 = '-';
66 local_15 = 'B';
67 local_14 = 'E';
68 local_13 = 'S';
69 local_12 = 'T';
70 local_11 = '-';
71 local_10 = 'S';
```

This is the flag.

Task 3

Exact same process load into ghidra and analyze.

```
Decompile: entry - (strings3.exe_)
1
2 void entry(void)
3
4 {
5     CHAR terminator_str;
6     undefined local_4a3 [1027];
7     char *local_a0;
8     MD5 md5Hash [144];
9     HRSRC local_c;
10    undefined4 local_8;
11
12    MD5::MD5(md5Hash);
13    terminator_str = '\0';
14    memset(local_4a3,0,0x3ff);
15    local_8 = 0;
16    local_c = FindResourceA((HMODULE)0x0,"rc.rc",(LPCSTR)0x6);
17    local_8 = 0x110;
18    LoadStringA((HINSTANCE)0x0,0x110,&terminator_str,0x3ff);
19    local_a0 = MD5::digestString(md5Hash,&terminator_str);
20    MessageBoxA((HWND)0x0,local_a0,"We've been compromised!",0x30);
21    /* WARNING: Subroutine does not return */
22    ExitProcess(0);
23 }
24
```

First we begin to do some tidying up. We see MD5::MD5. This means a Object of MD5 is calling the MD5 function. Most likely this is an object that will hold the hash. We then see \0 which is a terminator char. Below is the declaration for memset.

Following is the declaration for memset() function.

```
void *memset(void *str, int c, size_t n)
```

- **str** – This is a pointer to the block of memory to fill.
- **c** – This is the value to be set. The value is passed as an int, but the function fills the block of memory using the unsigned char conversion of this value.
- **n** – This is the number of bytes to be set to the value.

As we can see we are filling the local_4a3 buffer with zeros.

The below declaration of FindResourceA.

Determines the location of a resource with the specified type and name in the specified module.

To specify a language, use the [FindResourceEx](#) function.

Syntax

```
C++  
  
HRSRC FindResourceA(  
    [in, optional] HMODULE hModule,  
    [in]           LPCSTR  lpName,  
    [in]           LPCSTR  lpType  
);
```

Copy

Parameters

[in, optional] hModule

Type: HMODULE

A handle to the module whose portable executable file or an accompanying MUI file contains the resource. If this parameter is NULL, the function searches the module used to create the current process.

[in] lpName

Type: LPCTSTR

The name of the resource. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the resource. For more information, see the Remarks section below.

[in] lpType

Type: LPCTSTR

The resource type. Alternately, rather than a pointer, this parameter can be MAKEINTRESOURCE(ID), where ID is the integer identifier of the given resource type. For standard resource types, see [Resource Types](#). For more information, see the Remarks section below.

Shows that we are finding the resource rc.rc

```
15 0c CALL dword ptr [->USER32.DLL::LoadStringA] = 0"FLAG{RESOURCES-ARE-POPULAR-F...  
40 00 = 0000312a  
85 60 LEA EAX=>terminator_str,[EBP + 0xf6667b60]  
17 zero = 0x110;  
18 LoadStringA((HINSTANCE) 0x0,0x110,&terminator_str,0x3ff);  
19 local_a0 = MD5::digestString(md5Hash,&terminator_str);
```

A call to loadString shows that we are loading the string
FLAG{RESOURCES-ARE-POPULAR-FOR-MALWARE}.

However how does ghidra know this?

Well essentially we are loading a string from the offset 0x110.

Loads a string resource from the executable file associated with a specified module and either copies the string into a buffer with a terminating null character or returns a read-only pointer to the string resource itself.

Syntax

C++

 Copy

```
int LoadStringA(  
    [in, optional] HINSTANCE hInstance,  
    [in]           UINT       uID,  
    [out]          LPSTR      lpBuffer,  
    [in]           int        cchBufferMax  
);
```

Parameters

[in, optional] **hInstance**

Type: HINSTANCE

A handle to an instance of the module whose executable file contains the string resource. To get the handle to the application itself, call the [GetModuleHandle](#) function with NULL.

[in] **uID**

Type: UINT

The identifier of the string to be loaded.

[out] **lpBuffer**

Type: LPTSTR

The buffer to receive the string (if *cchBufferMax* is non-zero) or a read-only pointer to the string resource itself (if *cchBufferMax* is zero). Must be of sufficient length to hold a pointer (8 bytes).

[in] **cchBufferMax**

Type: int

The size of the buffer, in characters. The string is truncated and null-terminated if it is longer than the number of characters specified. If this parameter is 0, then *lpBuffer* receives a read-only pointer to the string resource itself.

0x110 to hex is 272 in decimal therefore looking for the string resource id 272 will also give us the flag.

```
Rsrc_StringTable_l2_409                                XREF[1]:      entry:004022ff(*)  
0040aef0 27 00 46      p_unicode  u"FLAG{RESOURCES-ARE-POPULAR-FOR-MALWARE}"  Rsrc String ID 272  
00 4c 00  
41 00 47 ...
```