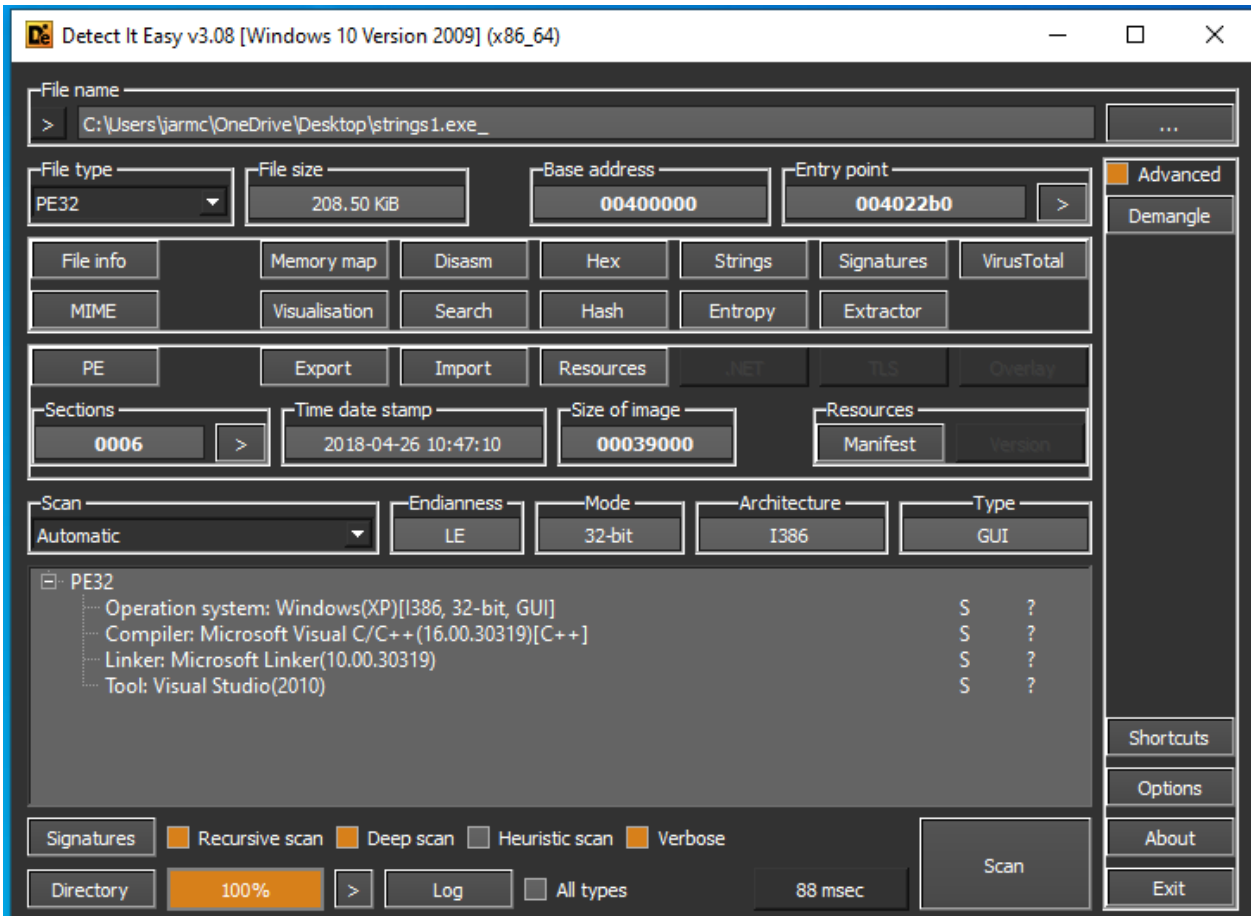
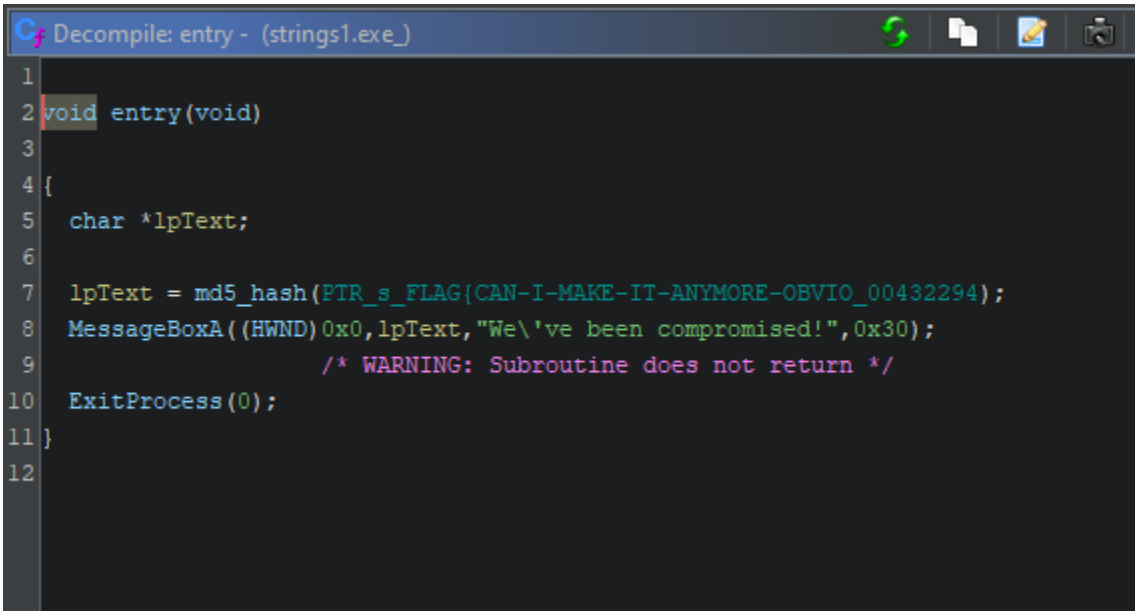
Basic Malware RE Task 1:

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.

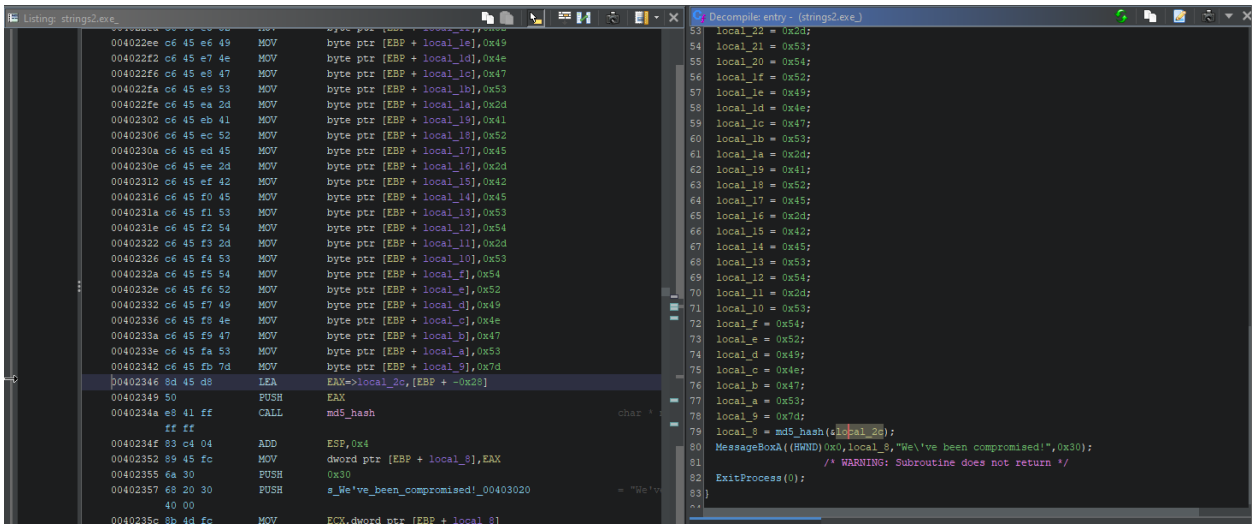


Click on the ptr to flag and we get

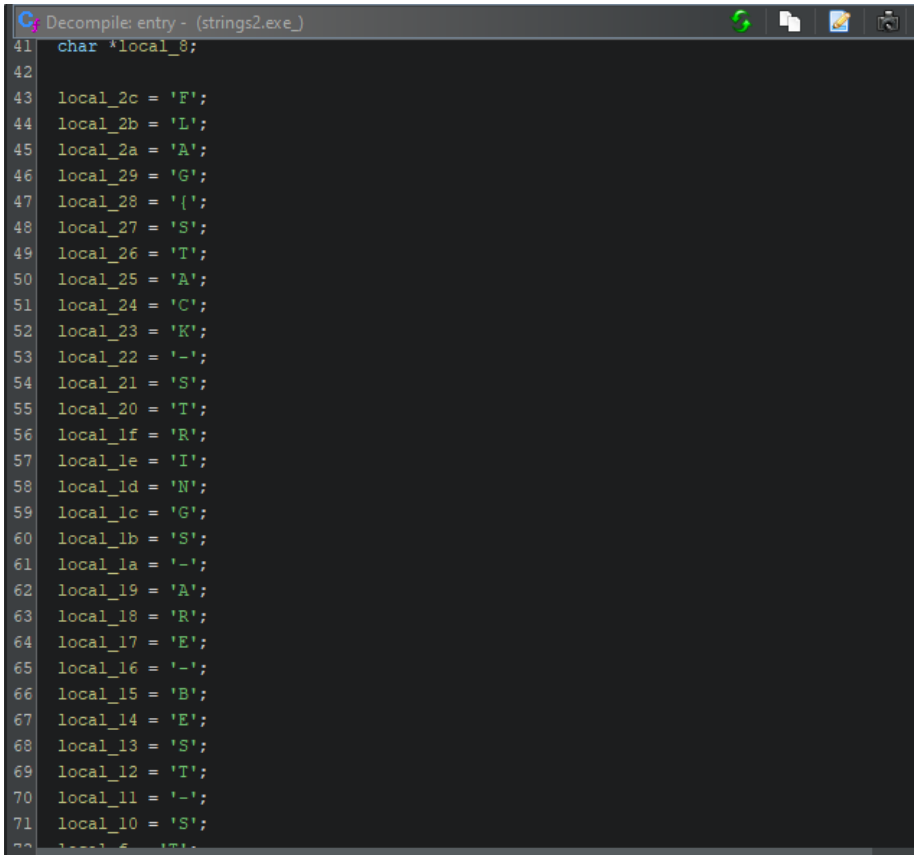
```
                    s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_00424828      XREF[2]:     entry:004022b9(*), 00432294(*)
00424828 46 4c 41        ds          "FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOUS}"
         47 7b 43
         41 4e 2d ...
```

Task 2

Do the same load it into Ghidra



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



This is the flag.

Task 3

Exact same process load into ghidra and analyze.

Decompile: entry - (strings3.exe_)

```
void entry(void)

{
  CHAR terminator_str;
  undefined local_4a3 [1027];
  char *local_a0;
  MD5 md5Hash [144];
  HRSRC local_c;
  undefined4 local_8;

  MD5::MD5(md5Hash);
  terminator_str = '\0';
  memset(local_4a3,0,0x3ff);
  local_8 = 0;
  local_c = FindResourceA((HMODULE)0x0,"rc.rc",(LPCSTR)0x6);
  local_8 = 0x110;
  LoadStringA((HINSTANCE)0x0,0x110,&terminator_str,0x3ff);
  local_a0 = MD5::digestString(md5Hash,&terminator_str);
  MessageBoxA((HWND)0x0,local_a0,"We\'ve been compromised!",0x30);
                    /* WARNING: Subroutine does not return */
  ExitProcess(0);
}
```

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++ Copy

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

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]    = u"FLAG{RESOURCES-ARE-POPULAR-F...
40 00                                                     = 0000312a
85 60    LEA     EAX=>terminator_str,[EBP + 0xffffffb60]
```

```
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

```cpp
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_12_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 ...
```