This room is made with challenges aimed to learning more on static analysis, the challenges are Windows executables , yeah scary , but luckily, you don't need a windows machine to solve this room ; it has 3 challenges namely strings1 , strings2, and strings3!

I'll be using Ghidra , and Cutter for all of these challenges!

String 1

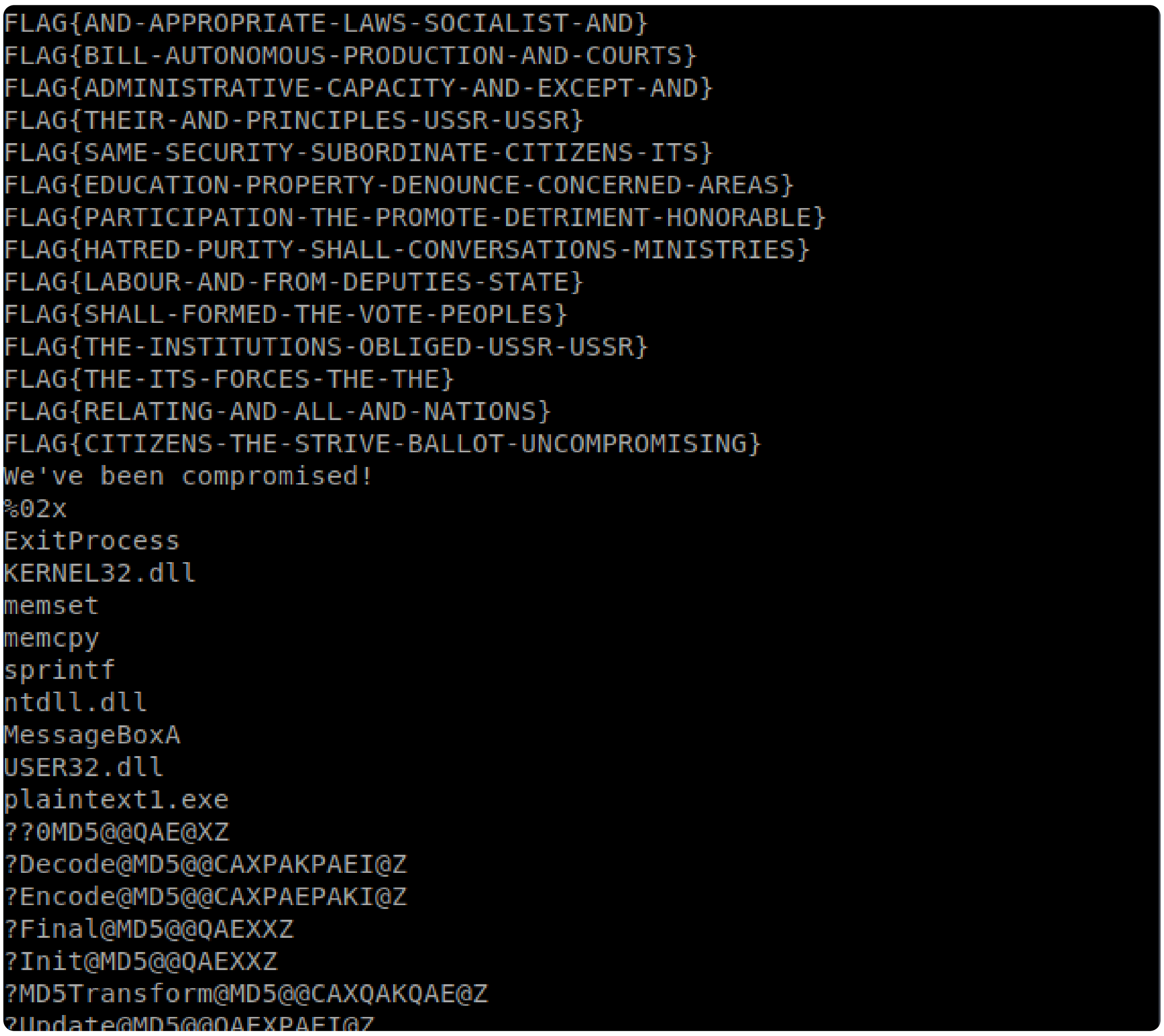
Description:

```
This executable prints an MD5 Hash on the screen when executed. Can you grab the exact

Note: You don't need to run the executable!
```

We are given an executable and it prints an MD5 hash when executed , can we grab the exact flag?:) well yes we can !

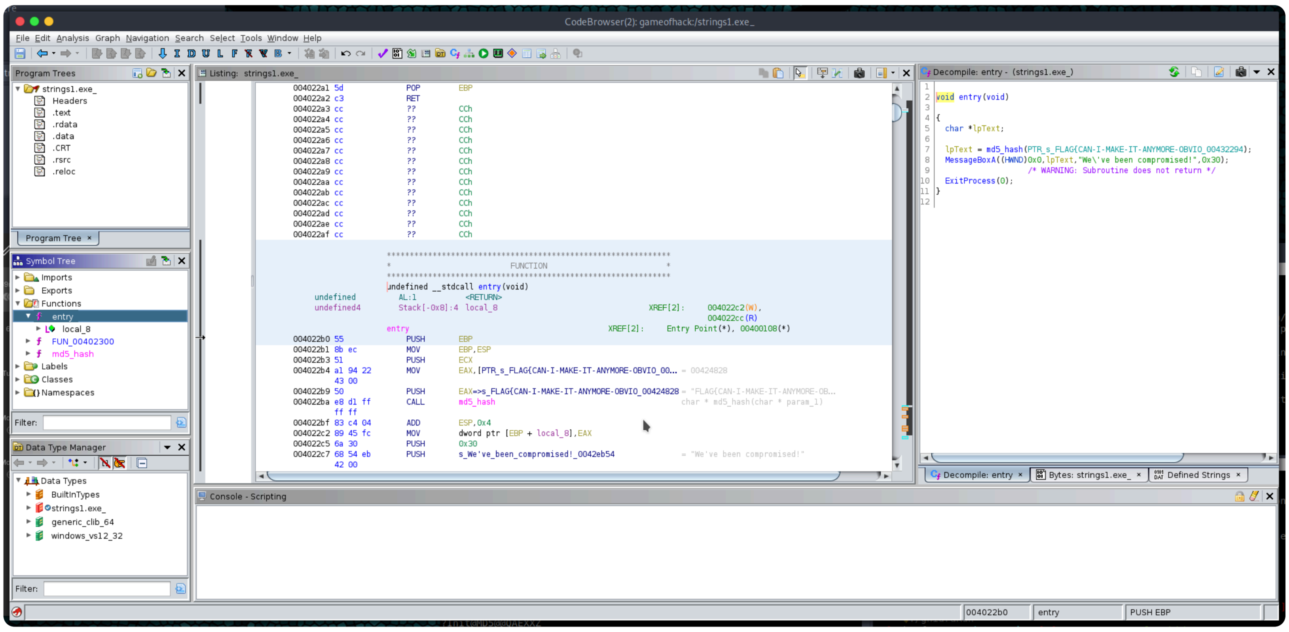
I first downloaded the executable file and then try checking strings , since the challenge name is strings :) and I found alot of flags , didn't expect that one lol!

```
strings strings1.exe_
```



```
FLAG{AND-APPROPRIATE-LAWS-SOCIALIST-AND}
FLAG{BILL-AUTONOMOUS-PRODUCTION-AND-COURTS}
FLAG{ADMINISTRATIVE-CAPACITY-AND-EXCEPT-AND}
FLAG{THEIR-AND-PRINCIPLES-USSR-USSR}
FLAG{SAME-SECURITY-SUBORDINATE-CITIZENS-ITS}
FLAG{EDUCATION-PROPERTY-DENOUNCE-CONCERNED-AREAS}
FLAG{PARTICIPATION-THE-PROMOTE-DETRIMENT-HONORABLE}
FLAG{HATRED-PURITY-SHALL-CONVERSATIONS-MINISTRIES}
FLAG{LABOUR-AND-FROM-DEPUTIES-STATE}
FLAG{SHALL-FORMED-THE-VOTE-PEOPLES}
FLAG{THE-INSTITUTIONS-OBLIGED-USSR-USSR}
FLAG{THE-ITS-FORCES-THE-THE}
FLAG{RELATING-AND-ALL-AND-NATIONS}
FLAG{CITIZENS-THE-STRIVE-BALLOT-UNCOMPROMISING}
We've been compromised!
%02x
ExitProcess
KERNEL32.dll
memset
memcpy
sprintf
ntdll.dll
MessageBoxA
USER32.dll
plaintext1.exe
??0MD5@@QAE@XZ
?Decode@MD5@@CAXPAKPAEI@Z
?Encode@MD5@@CAXPAEPAKI@Z
?Final@MD5@@QAEXXZ
?Init@MD5@@QAEXXZ
?MD5Transform@MD5@@CAXQAKQAE@Z
?Update@MD5@@QAEXPAEI@Z
```

but not usefull , so I upload the executable to ghidra so as I can read the disassembled code and the pseudocode ! And I was able to find some interesting line of codes in the `entry` function!



```
004022a1 5d          POP      EBP
004022a2 c3          RET
004022a3 cc          ??       CCh
004022a4 cc          ??       CCh
004022a5 cc          ??       CCh
004022a6 cc          ??       CCh
004022a7 cc          ??       CCh
004022a8 cc          ??       CCh
004022a9 cc          ??       CCh
004022aa cc          ??       CCh
004022ab cc          ??       CCh
004022ac cc          ??       CCh
004022ad cc          ??       CCh
004022ae cc          ??       CCh
004022af cc          ??       CCh

             **************************************************************
             *                          FUNCTION                          *
             **************************************************************
             undefined __stdcall entry(void)
   undefined       AL:1          <RETURN>
   undefined4      Stack[-0x8]:4 local_8                       XREF[2]:     004022c2(W),
                                                                            004022cc(R)
             entry                                          XREF[2]:     Entry Point(*), 00400108(*)
004022b0 55          PUSH     EBP
004022b1 8b ec       MOV      EBP,ESP
004022b3 51          PUSH     ECX
004022b4 a1 94 22    MOV      EAX,[PTR_s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_00... = 00424828
         43 00
004022b9 50          PUSH     EAX=>s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_00424828 = "FLAG{CAN-I-MAKE-IT-ANYMORE-OB...
004022ba e8 d1 ff    CALL     md5_hash                                  char * md5_hash(char * param_1)
         ff ff
004022bf 83 c4 04    ADD      ESP,0x4
004022c2 89 45 fc    MOV      dword ptr [EBP + local_8],EAX
004022c5 6a 30       PUSH     0x30
004022c7 68 54 eb    PUSH     s_We've_been_compromised!_0042eb54         = "We've been compromised!"
         42 00
```

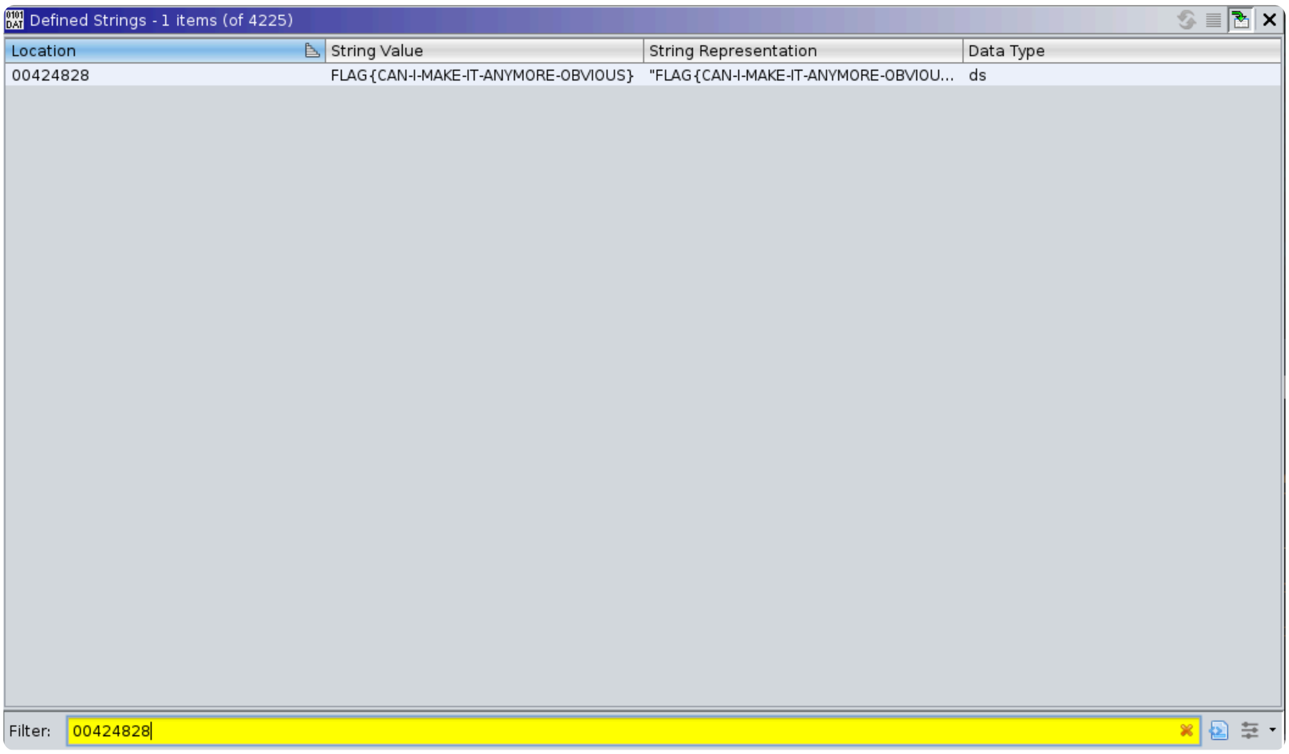
```c

void entry(void)

{
  char *lpText;

  lpText = md5_hash(PTR_s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_00432294);
  MessageBoxA((HWND)0x0,lpText,"We\'ve been compromised!",0x30);
                    /* WARNING: Subroutine does not return */
  ExitProcess(0);
}

```

```
void entry(void)

{
  char *lpText;

  lpText = md5_hash(PTR_s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_00432294);
  MessageBoxA((HWND)0x0,lpText,"We\'ve been compromised!",0x30);
                    /* WARNING: Subroutine does not return */
  ExitProcess(0);
}
```

We can now understand what's being hashed to MD5, so looking again at the disassembled code , I find this :

```
        004022b4 a1 94 22        MOV        EAX,[PTR_s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO
                 43 00
        004022b9 50              PUSH       EAX=>s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_004
        004022ba e8 d1 ff        CALL       md5_hash
                 ff ff
```

and this line: `EAX,[PTR_s_FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIO_00 = 00424828` shows the location of the flag which is `00424828` , so what I do next is to find that location and see what the full flag is :



Defined Strings - 1 items (of 4225)

Location	String Value	String Representation	Data Type
00424828	FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOUS}	"FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOU...	ds

Filter: 00424828

And there we have our flag:

```
FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOUS}
```

String 2

Description:

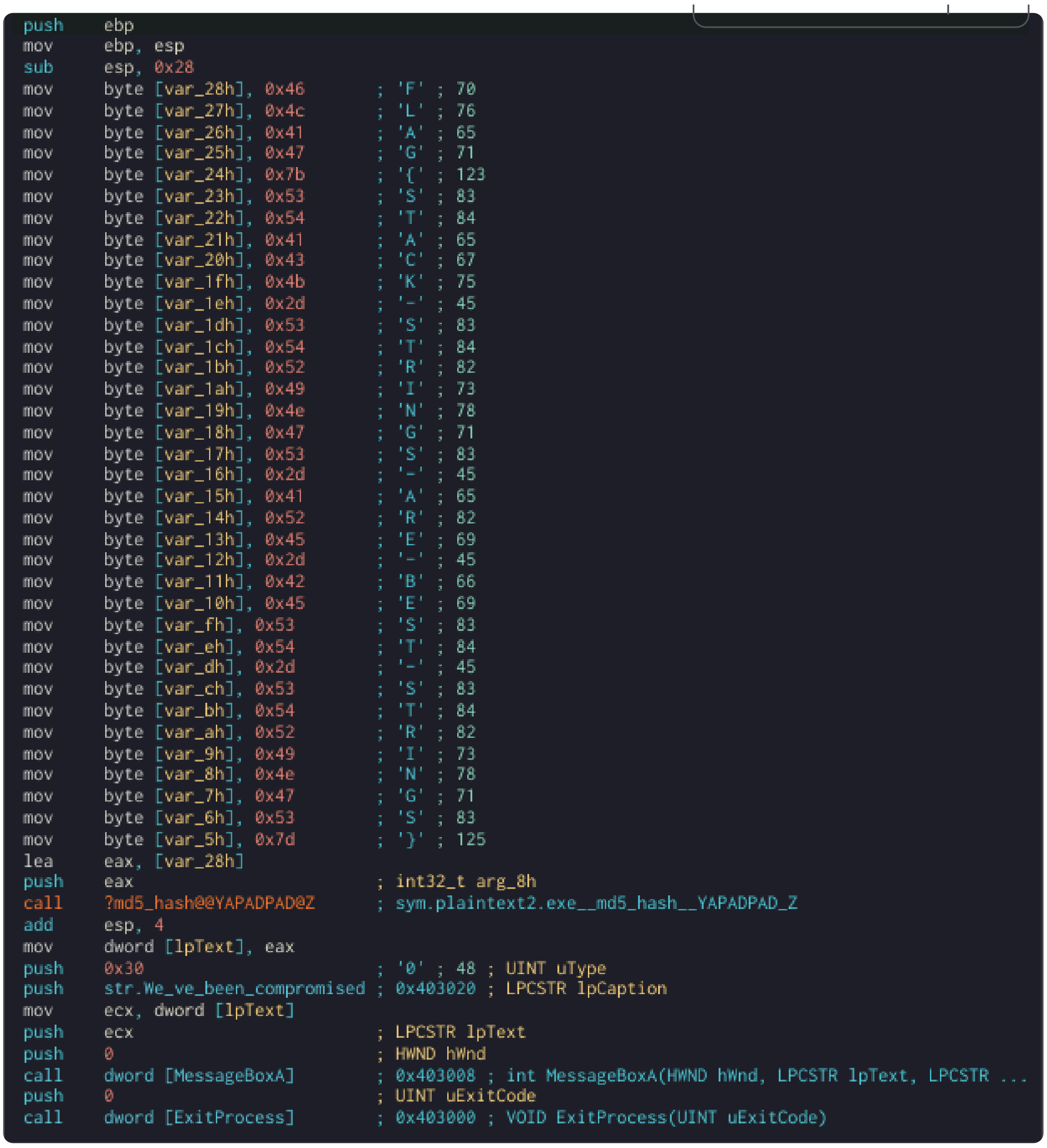
```
This executable prints an MD5 Hash on the screen when executed. Can you grab the exact

Note: You don't need to run the executable!
```

So same thing as the first one !, I tried running strings to see if I can find something but nothing important at all, so I upload the executable to ghidra again, and reading the pseudo code I noticed something really interesting, the variables that were assigned with hex values , when you convert them to ASCII you get a letter for example `local_2b = 0x4c` convert the 0x4c to ASCII and you get `L` and `local_2c = 'F'` which means the variables contain the letters which when all put together they make the flag!

```
local_2c = 'F';
local_2b = 0x4c;
local_2a = 0x41;
local_29 = 0x47;
local_28 = 0x7b;
local_27 = 0x53;
local_26 = 0x54;
local_25 = 0x41;
local_24 = 0x43;
local_23 = 0x4b;
local_22 = 0x2d;
local_21 = 0x53;
local_20 = 0x54;
local_1f = 0x52;
local_1e = 0x49;
local_1d = 0x4e;
local_1c = 0x47;
local_1b = 0x53;
local_1a = 0x2d;
local_19 = 0x41;
local_18 = 0x52;
local_17 = 0x45;
local_16 = 0x2d;
local_15 = 0x42;
local_14 = 0x45;
local_13 = 0x53;
local_12 = 0x54;
local_11 = 0x2d;
local_10 = 0x53;
local_f = 0x54;
local_e = 0x52;
local_d = 0x49;
local_c = 0x4e;
local_b = 0x47;
local_a = 0x53;
local_9 = 0x7d;
local_8 = md5_hash(&local_2c);
MessageBoxA((HWND)0x0,local_8,"We\'ve been compromised!",0x30);
                    /* WARNING: Subroutine does not return */
```

So you could take all those hex values and then convert them manually or you can just open up cutter if you have it installed and it'll show the flag right off the bat!

Comment



```
push    ebp
mov     ebp, esp
sub     esp, 0x28
mov     byte [var_28h], 0x46       ; 'F' ; 70
mov     byte [var_27h], 0x4c       ; 'L' ; 76
mov     byte [var_26h], 0x41       ; 'A' ; 65
mov     byte [var_25h], 0x47       ; 'G' ; 71
mov     byte [var_24h], 0x7b       ; '{' ; 123
mov     byte [var_23h], 0x53       ; 'S' ; 83
mov     byte [var_22h], 0x54       ; 'T' ; 84
mov     byte [var_21h], 0x41       ; 'A' ; 65
mov     byte [var_20h], 0x43       ; 'C' ; 67
mov     byte [var_1fh], 0x4b       ; 'K' ; 75
mov     byte [var_1eh], 0x2d       ; '-' ; 45
mov     byte [var_1dh], 0x53       ; 'S' ; 83
mov     byte [var_1ch], 0x54       ; 'T' ; 84
mov     byte [var_1bh], 0x52       ; 'R' ; 82
mov     byte [var_1ah], 0x49       ; 'I' ; 73
mov     byte [var_19h], 0x4e       ; 'N' ; 78
mov     byte [var_18h], 0x47       ; 'G' ; 71
mov     byte [var_17h], 0x53       ; 'S' ; 83
mov     byte [var_16h], 0x2d       ; '-' ; 45
mov     byte [var_15h], 0x41       ; 'A' ; 65
mov     byte [var_14h], 0x52       ; 'R' ; 82
mov     byte [var_13h], 0x45       ; 'E' ; 69
mov     byte [var_12h], 0x2d       ; '-' ; 45
mov     byte [var_11h], 0x42       ; 'B' ; 66
mov     byte [var_10h], 0x45       ; 'E' ; 69
mov     byte [var_fh], 0x53        ; 'S' ; 83
mov     byte [var_eh], 0x54        ; 'T' ; 84
mov     byte [var_dh], 0x2d        ; '-' ; 45
mov     byte [var_ch], 0x53        ; 'S' ; 83
mov     byte [var_bh], 0x54        ; 'T' ; 84
mov     byte [var_ah], 0x52        ; 'R' ; 82
mov     byte [var_9h], 0x49        ; 'I' ; 73
mov     byte [var_8h], 0x4e        ; 'N' ; 78
mov     byte [var_7h], 0x47        ; 'G' ; 71
mov     byte [var_6h], 0x53        ; 'S' ; 83
mov     byte [var_5h], 0x7d        ; '}' ; 125
lea     eax, [var_28h]
push    eax                        ; int32_t arg_8h
call    ?md5_hash@@YAPADPAD@Z      ; sym.plaintext2.exe__md5_hash__YAPADPAD_Z
add     esp, 4
mov     dword [lpText], eax
push    0x30                       ; '0' ; 48 ; UINT uType
push    str.We_ve_been_compromised ; 0x403020 ; LPCSTR lpCaption
mov     ecx, dword [lpText]
push    ecx                        ; LPCSTR lpText
push    0                          ; HWND hWnd
call    dword [MessageBoxA]        ; 0x403008 ; int MessageBoxA(HWND hWnd, LPCSTR lpText, LPCSTR ...
push    0                          ; UINT uExitCode
call    dword [ExitProcess]        ; 0x403000 ; VOID ExitProcess(UINT uExitCode)
```

And we have the flag :

```
FLAG{STACK-STRINGS-ARE-BEST-STRINGS}
```

String 3

Description:

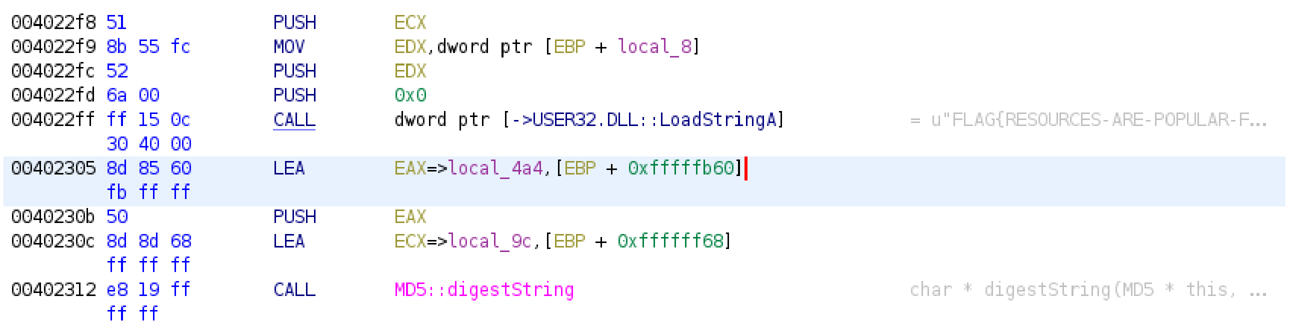
```
This executable prints an MD5 Hash on the screen when executed. Can you grab the exact

Note: You don't need to run the executable!
```

And same description again, wow , so this time I didn't wanna waste my time checking strings lol, so I just shoot it up straight to ghidra!

It has only one function shown in ghidra and that is `entry` but taking a look at the pseudo code I see something interesting :

```
void entry(void)

{
  CHAR local_4a4;
  undefined local_4a3 [1027];
  char *local_a0;
  MD5 local_9c [144];
  HRSRC local_c;
  undefined4 local_8;

  MD5::MD5(local_9c);
  local_4a4 = '\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,&local_4a4,0x3ff);
  local_a0 = MD5::digestString(local_9c,&local_4a4);
  MessageBoxA((HWND)0x0,local_a0,"We\'ve been compromised!",0x30);
                    /* WARNING: Subroutine does not return */
  ExitProcess(0);
}
```

the defined-functions `LoadStringA()` and `FindResourceA()` are quiet interesting , but taking a look at the disassembled code to see how the `LoadStringA()` worked and as shown below it called the flag, but not only the flag it called the flag from a known location!



```
004022f8 51          PUSH     ECX
004022f9 8b 55 fc    MOV      EDX,dword ptr [EBP + local_8]
004022fc 52          PUSH     EDX
004022fd 6a 00       PUSH     0x0
004022ff ff 15 0c    CALL     dword ptr [->USER32.DLL::LoadStringA]          = u"FLAG{RESOURCES-ARE-POPULAR-F...
         30 40 00
00402305 8d 85 60    LEA      EAX=>local_4a4,[EBP + 0xfffffb60]
         fb ff ff
0040230b 50          PUSH     EAX
0040230c 8d 8d 68    LEA      ECX=>local_9c,[EBP + 0xffffff68]
         ff ff ff
00402312 e8 19 ff    CALL     MD5::digestString                              char * digestString(MD5 * this, ...
         ff ff
```

```
        004022ff ff 15 0c        CALL       dword ptr [->USER32.DLL::LoadStringA]
                 30 40 00
```

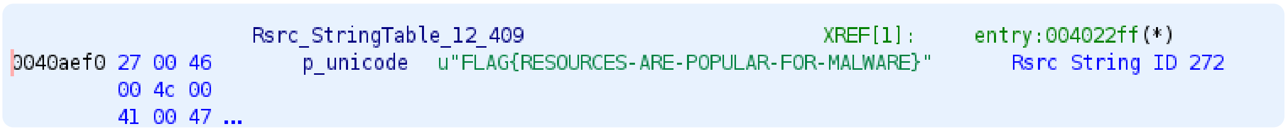
so we have to locate the string from where it's called since we have the ID of the string it won't be hard:

```
    LoadStringA((HINSTANCE)0x0,0x110,&local_4a4,0x3ff);
```

the address as shown here is `0x110` coverting that to an integer:

```
 ┌─[tahaafarooq@cyberwarriors]─[~/Desktop/tryhackme/basicmalware_re]
 └──╼ $python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> hex = 0x110
>>> print(int(0x110))
272
```

it gives us 272 so that's the string ID , now I just search it up:



```
                          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 ...
```

And the flag is :

```
FLAG{RESOURCES-ARE-POPULAR-FOR-MALWARE}
```

Contact

Twitter : tahaafarooq (https://twitter.com/tahaafarooq)

Github : tahaafarooq (https://github.com/tahaafarooq)

Email : tahacodez@gmail.com (mailto:tahacodez@gmail.com)