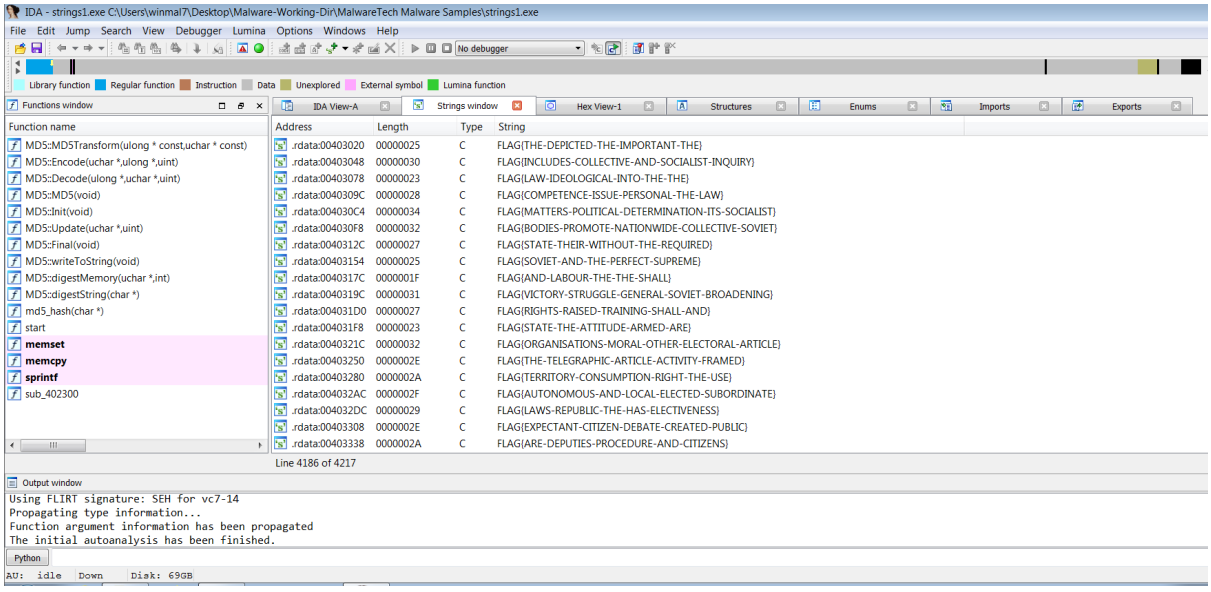
Marcus Hutchins RE Challenges

written by Ryan Ng

Strings1

- In IDA, `View > Open Subview > Strings` shows the strings in the binary, similar to the Linux “strings” command

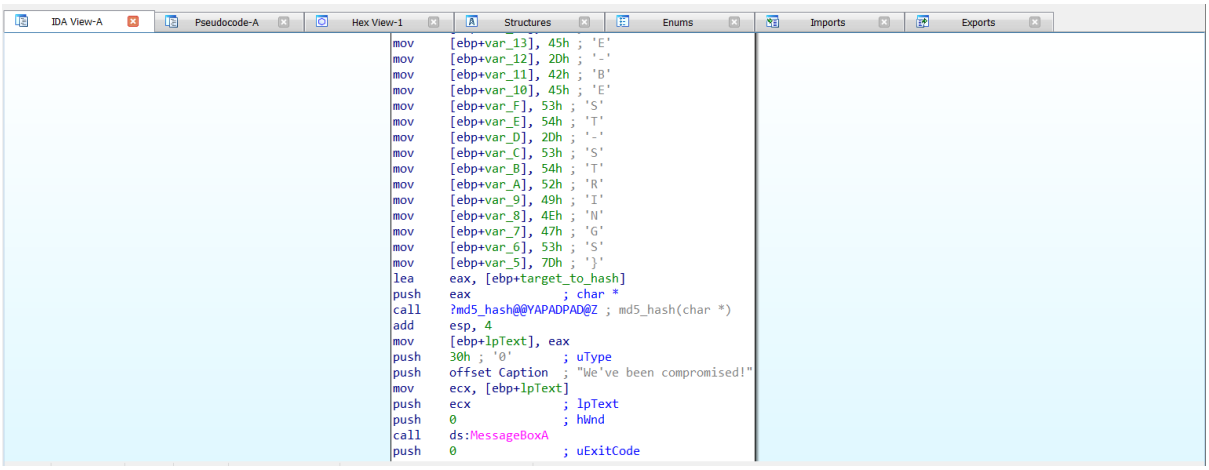


- There’s a million inaccurate “strings” so, not useful to use strings in this case

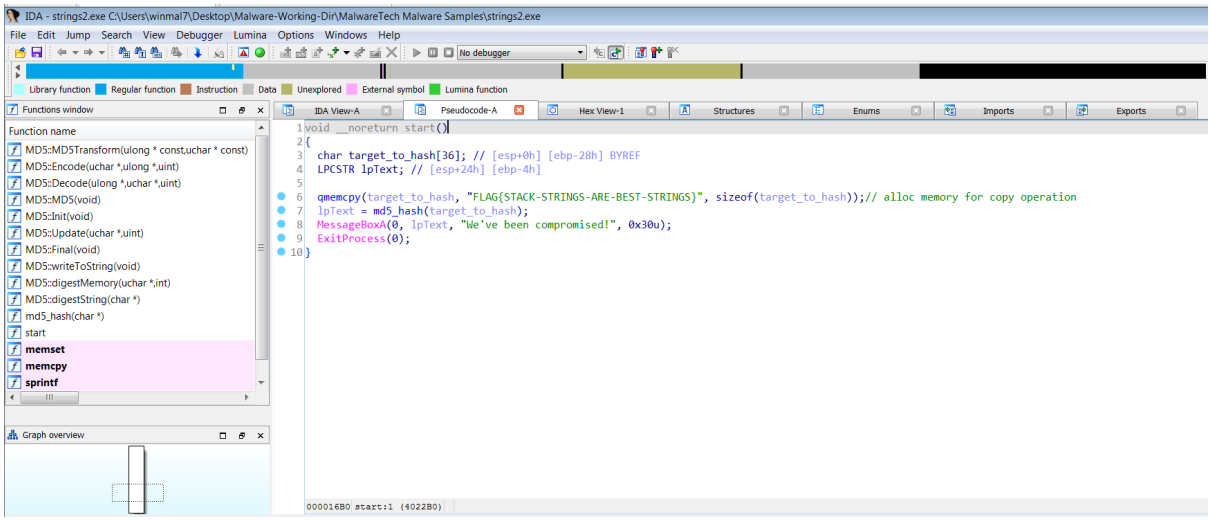
```
public start
start proc near

lpText= dword ptr -4

push    ebp
mov     ebp, esp
push    ecx
mov     eax, off_432294 ; "FLAG{CAN-I-MAKE-IT-ANYMORE-OBVIOUS}"
push    eax             ; char *
call    ?md5_hash@@YAPADPAD@Z ; md5_hash(char *)
add     esp, 4
mov     [ebp+lpText], eax
push    30h ; '0'       ; uType
push    offset Caption  ; "We've been compromised!"
mov     ecx, [ebp+lpText]
push    ecx             ; lpText
push    0               ; hWnd
call    ds:MessageBoxA
push    0               ; uExitCode
call    ds:ExitProcess
start endp
```

- `off_4322294` points to the flag (just click on it to show where it references)

Strings2



- Call `lea` on `ebp+28` (start of stack string, which loads the stack string)
  - For IDA Free, may need to use the `r` key (when mouseover) to convert hex to ascii (can also use ascii converter or table, but more tedious



- In the decompiled pseudocode, we can see that the `qmemcpy` operation is used to allocate memory for the stack strings to be push onto the stack (size of `36` bytes)

C library function - memcpy()

Previous Page Next Page

Description

The C library function **void *memcpy(void *dest, const void *src, size_t n)** copies **n** characters from memory area **src** to memory area **dest**.

Declaration

Following is the declaration for memcpy() function.

```
void *memcpy(void *dest, const void * src, size_t n)
```

Parameters

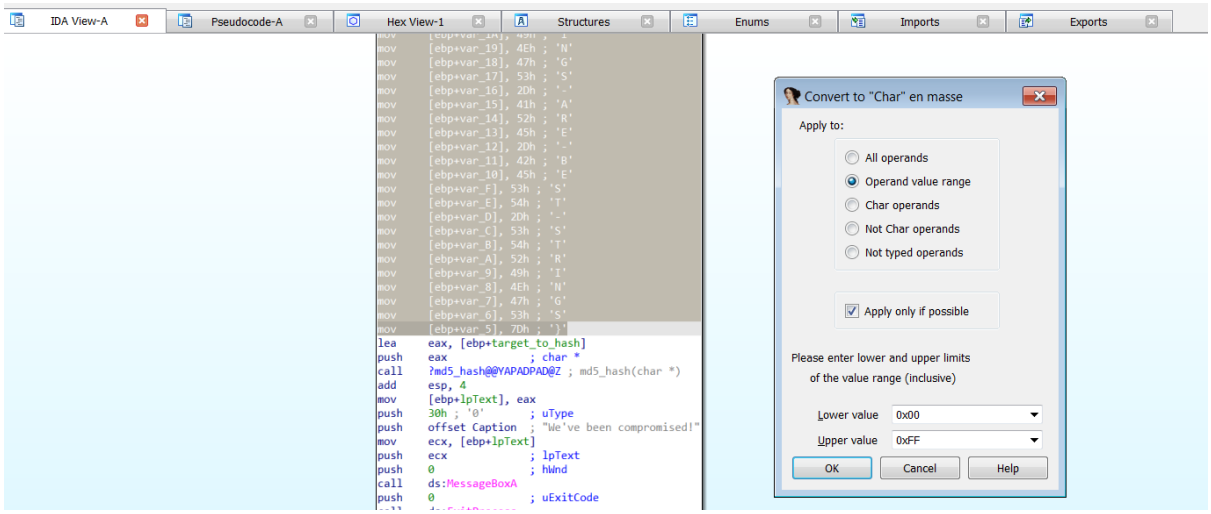
- **dest** − This is pointer to the destination array where the content is to be copied, type-casted to a pointer of type void*.
- **src** − This is pointer to the source of data to be copied, type-casted to a pointer of type void*.
- **n** − This is the number of bytes to be copied.

Return Value

This function returns a pointer to destination, which is str1.

- `qmemcpy` docs above

- ***Tip:*** to mass convert bytes, select the stack string and press `r`, set the boundaries to `0x00` and `0xFF` (decimal `255`) or `0x80` (decimal `128`) → correspond to ascii and extended-ascii table respectively
    - `0x80` also works in this case since all the characters in the flag are bound by the limits of the ascii table



Strings3

- Some of the APIs used are FindResourceA (find resource probably within executable) and LoadStringA (load string contained within the resource)

FindResourceA function (winbase.h)

Article • 10/13/2021 • 2 minutes to read

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.

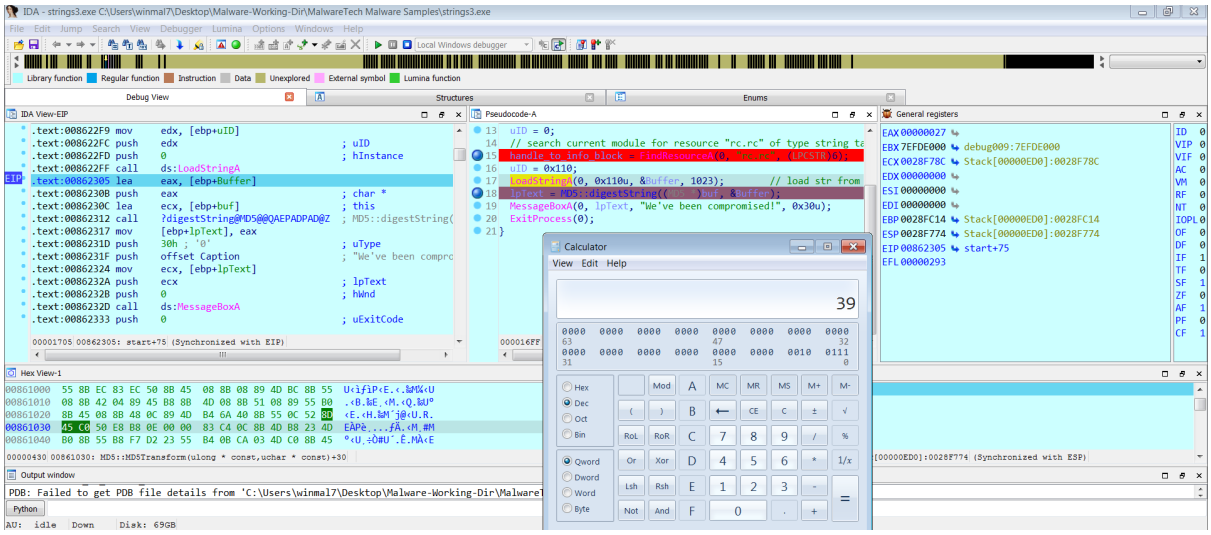
Return value

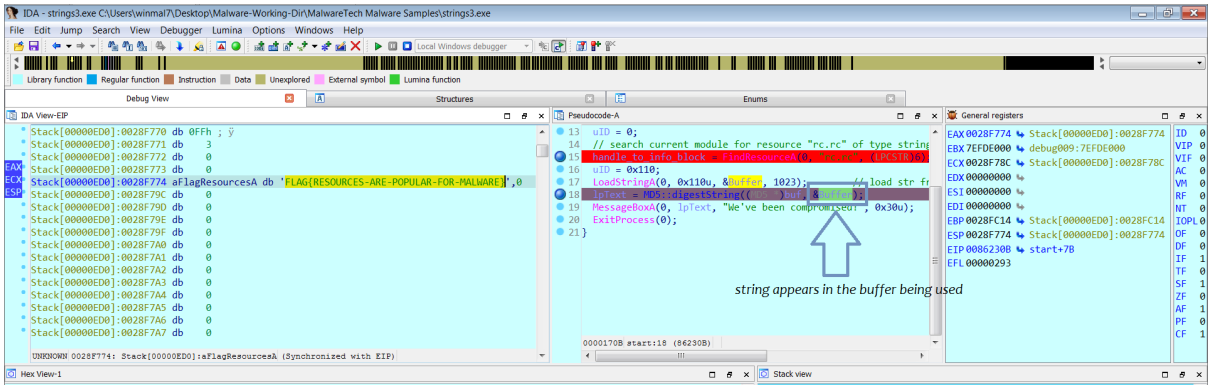
Type: **HRSRC**

If the function succeeds, the return value is a handle to the specified resource's information block. To obtain a handle to the resource, pass this handle to the LoadResource function.

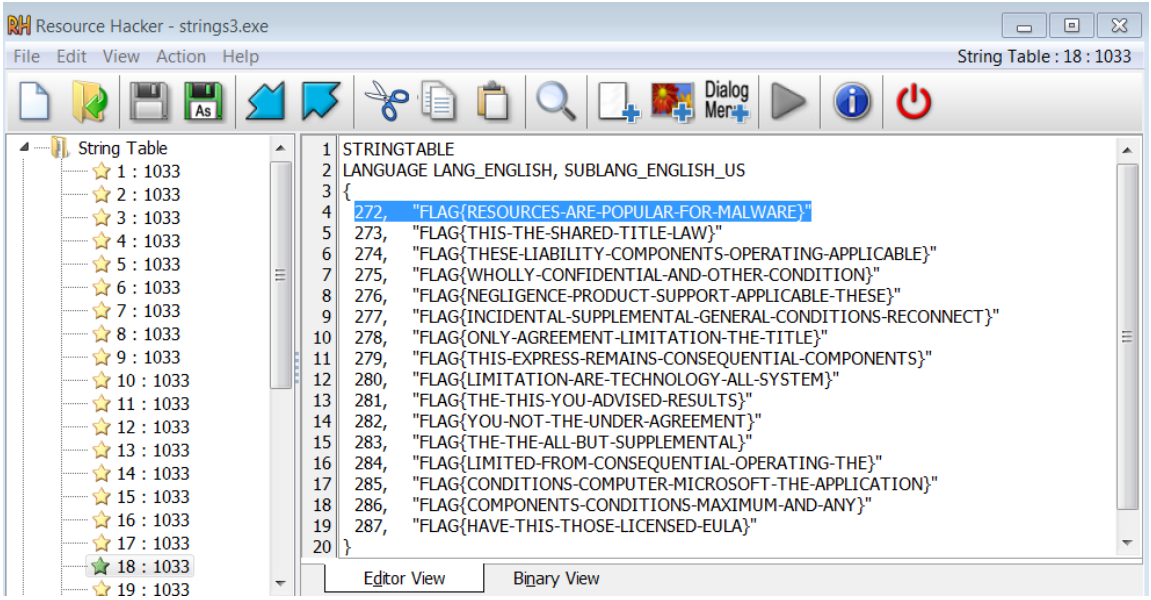
If the function fails, the return value is **NULL**. To get extended error information, call GetLastError.

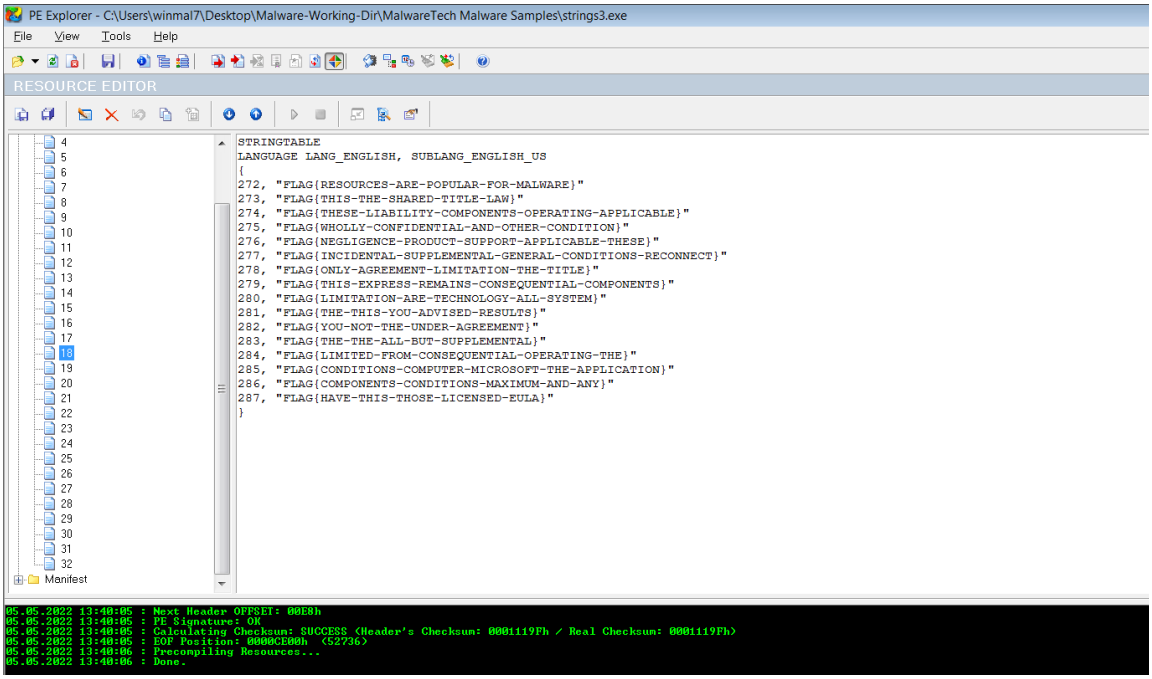
- *hex 27 → dec 39 (size of string copied into eax)*





- alternative method is using calculators to “mimic” the behaviour of the registers (eventually loads resource `272` / `0x110`)
  - flag is found using Resource Hacker’s “Editor View”/PE Explorer’s Resource Editor
  - malware often uses `LoadResourceA` instead of `LoadStringA` in this instance to load malicious code into memory




PE Explorer - C:\Users\winmal7\Desktop\Malware-Working-Dir\MalwareTech Malware Samples\strings3.exe
File View Tools Help
RESOURCE EDITOR
4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
Manifest
STRINGTABLE
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
{
272, "FLAG{RESOURCES-ARE-POPULAR-FOR-MALWARE}"
273, "FLAG{THIS-THE-SHARED-TITLE-LAW}"
274, "FLAG{THESE-LIABILITY-COMPONENTS-OPERATING-APPLICABLE}"
275, "FLAG{WHOLLY-CONFIDENTIAL-AND-OTHER-CONDITION}"
276, "FLAG{NEGLIGENCE-PRODUCT-SUPPORT-APPLICABLE-THESE}"
277, "FLAG{INCIDENTAL-SUPPLEMENTAL-GENERAL-CONDITIONS-RECONNECT}"
278, "FLAG{ONLY-AGREEMENT-LIMITATION-THE-TITLE}"
279, "FLAG{THIS-EXPRESS-REMAINS-CONSEQUENTIAL-COMPONENTS}"
280, "FLAG{LIMITATION-ARE-TECHNOLOGY-ALL-SYSTEM}"
281, "FLAG{THE-THIS-YOU-ADVISED-RESULTS}"
282, "FLAG{YOU-NOT-THE-UNDER-AGREEMENT}"
283, "FLAG{THE-THE-ALL-BUT-SUPPLEMENTAL}"
284, "FLAG{LIMITED-FROM-CONSEQUENTIAL-OPERATING-THE}"
285, "FLAG{CONDITIONS-COMPUTER-MICROSOFT-THE-APPLICATION}"
286, "FLAG{COMPONENTS-CONDITIONS-MAXIMUM-AND-ANY}"
287, "FLAG{HAVE-THIS-THOSE-LICENSED-EULA}"
}
05.05.2022 13:40:05 : Next Header OFFSET: 00E8h
05.05.2022 13:40:05 : PE Signature: OK
05.05.2022 13:40:05 : Calculating Checksum: SUCCESS (Header's Checksum: 0001119Fh / Real Checksum: 0001119Fh)
05.05.2022 13:40:05 : EOF Position: 0000CE00h (52736)
05.05.2022 13:40:06 : Precompiling Resources...
05.05.2022 13:40:06 : Done.