Abusing undocumented features to spoof PE section headers

secret.club/2023/06/05/spoof-pe-sections.html

x86matthew June 5, 2023

x86matthew

Jun 5, 2023

Introduction

Some time ago, I accidentally came across some interesting behaviour in PE files while debugging an unrelated project. I noticed that setting the `SectionAlignment` value in the NT header to a value lower than the page size (4096) resulted in significant differences in the way that the image is mapped into memory. Rather than following the usual procedure of parsing the section table to construct the image in memory, the loader appeared to map the entire file, including the headers, into memory with read-write-execute (RWX) permissions - the individual section headers were completely ignored.

As a result of this behaviour, it is possible to create a PE executable without any sections, yet still capable of executing its own code. The code can even be self-modifying if necessary due to the write permissions that are present by default.

One way in which this mode could potentially be abused would be to create a fake section table - on first inspection, this would appear to be a normal PE module containing read-write/read-only data sections, but when launched, the seemingly NX data becomes executable.

While I am sure that this technique will have already been discovered (and potentially abused) in the past, I have been unable to find any documentation online describing it. MSDN does briefly mention that the `SectionAlignment` value can be less than the page size, but it doesn’t elaborate any further on the implications of this.

Inside the Windows kernel

A quick look in the kernel reveals what is happening. Within `MiCreateImageFileMap`, we can see the parsing of PE headers - notably, if the `SectionAlignment` value is less than 0x1000, an undocumented flag (0x200000) is set prior to mapping the image into memory:

```
        if(v29->SectionAlignment < 0x1000)
        {
                if((SectionFlags & 0x80000) != 0)
                {
                        v17 = 0xC000007B;
                        MiLogCreateImageFileMapFailure(v36, v39, *(unsigned int *)
(v29 + 64), DWORD1(v99));
                        ImageFailureReason = 55;
                        goto LABEL_81;
                }
                if(!MiLegacyImageArchitecture((unsigned __int16)v99))
                {
                        v17 = 0xC000007B;
                        ImageFailureReason = 56;
                        goto LABEL_81;
                }
                SectionFlags |= 0x200000;
        }
        v40 = MiBuildImageControlArea(a3, v38, v29, (unsigned int)&v99, SectionFlags,
(__int64)&FileSize, (__int64)&v93);
```

If the aforementioned flag is set, `MiBuildImageControlArea` treats the entire file as one single section:

```
        if((SectionFlags & 0x200000) != 0)
        {
                SectionCount = 1;
        }
        else
        {
                SectionCount = a4->NumberOfSections + 1;
        }
        v12 = MiAllocatePool(64, 8 * (7 * SectionCount + (((unsigned __int64)
(unsigned int)MiFlags >> 13) & 1)) + 184, (SectionFlags & 0x200000) != 0 ? 0x61436D4D
: 0x69436D4D);
```

As a result, the raw image is mapped into memory with all PTEs assigned `MM_EXECUTE_READWRITE` protection. As mentioned previously, the `IMAGE_SECTION_HEADER` list is ignored, meaning a PE module using this mode can have a `NumberOfSections` value of 0. There are no obvious size restrictions on PE modules using this mode either - the loader will allocate memory based on the `SizeOfImage` field and copy the file contents accordingly. Any excess memory beyond the size of the file will remain blank.

Demonstration #1 - Executable PE with no sections

The simplest demonstration of this technique would be to create a generic “loader” for position-independent code. I have created the following sample headers by hand for testing:

```
// (64-bit EXE headers)
BYTE bHeaders64[328] =
{
        0x4D, 0x5A, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x40, 0x00, 0x00, 0x00, 0x50, 0x45, 0x00, 0x00, 0x64, 0x86, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0xF0, 0x00, 0x22, 0x00, 0x0B, 0x02, 0x0E, 0x1D, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x01, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x01, 0x00, 0x00, 0x00,
        0x00, 0x02, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x10, 0x00, 0x48, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x02, 0x00, 0x60, 0x81, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00,

        // (code goes here)
};

BYTE bHeaders32[304] =
{
        0x4D, 0x5A, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x40, 0x00, 0x00, 0x00, 0x50, 0x45, 0x00, 0x00, 0x4C, 0x01, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0xE0, 0x00, 0x02, 0x01, 0x0B, 0x01, 0x0E, 0x1D, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x30, 0x01, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x40, 0x00,
        0x00, 0x02, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x10, 0x00, 0x30, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x02, 0x00, 0x40, 0x81, 0x00, 0x00, 0x10, 0x00, 0x00, 0x10, 0x00, 0x00,
        0x00, 0x00, 0x10, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
```

```
        0x10, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00,

        // (code goes here)
};
```

These headers contain a `SectionAlignment` value of 0x200 (rather than the usual 0x1000), a `SizeOfImage` value of 0x100000 (1MB), a blank section table, and an entry-point positioned immediately after the headers. Aside from these values, there is nothing special about the remaining fields:

```
(DOS Header)
   e_magic                         : 0x5A4D
   ...
   e_lfanew                        : 0x40
(NT Header)
   Signature                       : 0x4550
   Machine                         : 0x8664
   NumberOfSections                : 0x0
   TimeDateStamp                   : 0x0
   PointerToSymbolTable            : 0x0
   NumberOfSymbols                 : 0x0
   SizeOfOptionalHeader            : 0xF0
   Characteristics                 : 0x22
   Magic                           : 0x20B
   MajorLinkerVersion              : 0xE
   MinorLinkerVersion              : 0x1D
   SizeOfCode                      : 0x0
   SizeOfInitializedData           : 0x0
   SizeOfUninitializedData         : 0x0
   AddressOfEntryPoint             : 0x148
   BaseOfCode                      : 0x0
   ImageBase                       : 0x140000000
   SectionAlignment                : 0x200
   FileAlignment                   : 0x200
   MajorOperatingSystemVersion     : 0x6
   MinorOperatingSystemVersion     : 0x0
   MajorImageVersion               : 0x0
   MinorImageVersion               : 0x0
   MajorSubsystemVersion           : 0x6
   MinorSubsystemVersion           : 0x0
   Win32VersionValue               : 0x0
   SizeOfImage                     : 0x100000
   SizeOfHeaders                   : 0x148
   CheckSum                        : 0x0
   Subsystem                       : 0x2
   DllCharacteristics              : 0x8160
   SizeOfStackReserve              : 0x100000
   SizeOfStackCommit               : 0x1000
   SizeOfHeapReserve               : 0x100000
   SizeOfHeapCommit                : 0x1000
   LoaderFlags                     : 0x0
   NumberOfRvaAndSizes             : 0x10
   DataDirectory[0]                : 0x0, 0x0
   ...
   DataDirectory[15]               : 0x0, 0x0
(Start of code)
```

For demonstration purposes, we will be using some position-independent code that calls `MessageBoxA`. As the base headers lack an import table, this code must locate and load all dependencies manually - user32.dll in this case. This same payload can be used in both 32-bit and 64-bit environments:

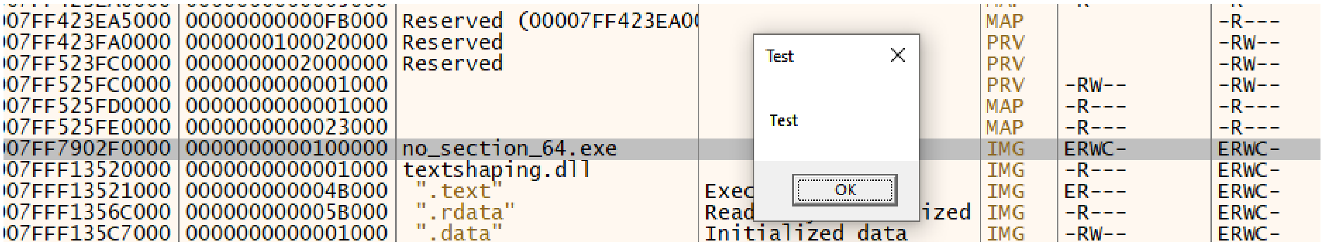
```
BYTE bMessageBox[939] =
{
        0x8B, 0xC4, 0x6A, 0x00, 0x2B, 0xC4, 0x59, 0x83, 0xF8, 0x08, 0x0F, 0x84,
        0xA0, 0x01, 0x00, 0x00, 0x55, 0x8B, 0xEC, 0x83, 0xEC, 0x3C, 0x64, 0xA1,
        0x30, 0x00, 0x00, 0x00, 0x33, 0xD2, 0x53, 0x56, 0x57, 0x8B, 0x40, 0x0C,
        0x33, 0xDB, 0x21, 0x5D, 0xF0, 0x21, 0x5D, 0xEC, 0x8B, 0x40, 0x1C, 0x8B,
        0x00, 0x8B, 0x78, 0x08, 0x8B, 0x47, 0x3C, 0x8B, 0x44, 0x38, 0x78, 0x03,
        0xC7, 0x8B, 0x48, 0x24, 0x03, 0xCF, 0x89, 0x4D, 0xE8, 0x8B, 0x48, 0x20,
        0x03, 0xCF, 0x89, 0x4D, 0xE4, 0x8B, 0x48, 0x1C, 0x03, 0xCF, 0x89, 0x4D,
        0xF4, 0x8B, 0x48, 0x14, 0x89, 0x4D, 0xFC, 0x85, 0xC9, 0x74, 0x5F, 0x8B,
        0x70, 0x18, 0x8B, 0xC1, 0x89, 0x75, 0xF8, 0x33, 0xC9, 0x85, 0xF6, 0x74,
        0x4C, 0x8B, 0x45, 0xE8, 0x0F, 0xB7, 0x04, 0x48, 0x3B, 0xC2, 0x74, 0x07,
        0x41, 0x3B, 0xCE, 0x72, 0xF0, 0xEB, 0x37, 0x8B, 0x45, 0xE4, 0x8B, 0x0C,
        0x88, 0x03, 0xCF, 0x74, 0x2D, 0x8A, 0x01, 0xBE, 0x05, 0x15, 0x00, 0x00,
        0x84, 0xC0, 0x74, 0x1F, 0x6B, 0xF6, 0x21, 0x0F, 0xBE, 0xC0, 0x03, 0xF0,
        0x41, 0x8A, 0x01, 0x84, 0xC0, 0x75, 0xF1, 0x81, 0xFE, 0xFB, 0xF0, 0xBF,
        0x5F, 0x75, 0x74, 0x8B, 0x45, 0xF4, 0x8B, 0x1C, 0x90, 0x03, 0xDF, 0x8B,
        0x75, 0xF8, 0x8B, 0x45, 0xFC, 0x42, 0x3B, 0xD0, 0x72, 0xA9, 0x8D, 0x45,
        0xC4, 0xC7, 0x45, 0xC4, 0x75, 0x73, 0x65, 0x72, 0x50, 0x66, 0xC7, 0x45,
        0xC8, 0x33, 0x32, 0xC6, 0x45, 0xCA, 0x00, 0xFF, 0xD3, 0x8B, 0xF8, 0x33,
        0xD2, 0x8B, 0x4F, 0x3C, 0x8B, 0x4C, 0x39, 0x78, 0x03, 0xCF, 0x8B, 0x41,
        0x20, 0x8B, 0x71, 0x24, 0x03, 0xC7, 0x8B, 0x59, 0x14, 0x03, 0xF7, 0x89,
        0x45, 0xE4, 0x8B, 0x41, 0x1C, 0x03, 0xC7, 0x89, 0x75, 0xF8, 0x89, 0x45,
        0xE8, 0x89, 0x5D, 0xFC, 0x85, 0xDB, 0x74, 0x7D, 0x8B, 0x59, 0x18, 0x8B,
        0x45, 0xFC, 0x33, 0xC9, 0x85, 0xDB, 0x74, 0x6C, 0x0F, 0xB7, 0x04, 0x4E,
        0x3B, 0xC2, 0x74, 0x22, 0x41, 0x3B, 0xCB, 0x72, 0xF3, 0xEB, 0x5A, 0x81,
        0xFE, 0x6D, 0x07, 0xAF, 0x60, 0x8B, 0x75, 0xF8, 0x75, 0x8C, 0x8B, 0x45,
        0xF4, 0x8B, 0x04, 0x90, 0x03, 0xC7, 0x89, 0x45, 0xEC, 0xE9, 0x7C, 0xFF,
        0xFF, 0xFF, 0x8B, 0x45, 0xE4, 0x8B, 0x0C, 0x88, 0x03, 0xCF, 0x74, 0x35,
        0x8A, 0x01, 0xBE, 0x05, 0x15, 0x00, 0x00, 0x84, 0xC0, 0x74, 0x27, 0x6B,
        0xF6, 0x21, 0x0F, 0xBE, 0xC0, 0x03, 0xF0, 0x41, 0x8A, 0x01, 0x84, 0xC0,
        0x75, 0xF1, 0x81, 0xFE, 0xB4, 0x14, 0x4F, 0x38, 0x8B, 0x75, 0xF8, 0x75,
        0x10, 0x8B, 0x45, 0xE8, 0x8B, 0x04, 0x90, 0x03, 0xC7, 0x89, 0x45, 0xF0,
        0xEB, 0x03, 0x8B, 0x75, 0xF8, 0x8B, 0x45, 0xFC, 0x42, 0x3B, 0xD0, 0x72,
        0x89, 0x33, 0xC9, 0xC7, 0x45, 0xC4, 0x54, 0x65, 0x73, 0x74, 0x51, 0x8D,
        0x45, 0xC4, 0x88, 0x4D, 0xC8, 0x50, 0x50, 0x51, 0xFF, 0x55, 0xF0, 0x6A,
        0x7B, 0x6A, 0xFF, 0xFF, 0x55, 0xEC, 0x5F, 0x5E, 0x5B, 0xC9, 0xC3, 0x90,
        0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90, 0x90,
        0x48, 0x89, 0x5C, 0x24, 0x08, 0x48, 0x89, 0x6C, 0x24, 0x10, 0x48, 0x89,
        0x74, 0x24, 0x18, 0x48, 0x89, 0x7C, 0x24, 0x20, 0x41, 0x54, 0x41, 0x56,
        0x41, 0x57, 0x48, 0x83, 0xEC, 0x40, 0x65, 0x48, 0x8B, 0x04, 0x25, 0x60,
        0x00, 0x00, 0x00, 0x33, 0xFF, 0x45, 0x33, 0xFF, 0x45, 0x33, 0xE4, 0x45,
        0x33, 0xC9, 0x48, 0x8B, 0x48, 0x18, 0x48, 0x8B, 0x41, 0x30, 0x48, 0x8B,
        0x08, 0x48, 0x8B, 0x59, 0x10, 0x48, 0x63, 0x43, 0x3C, 0x8B, 0x8C, 0x18,
        0x88, 0x00, 0x00, 0x00, 0x48, 0x03, 0xCB, 0x8B, 0x69, 0x24, 0x44, 0x8B,
        0x71, 0x20, 0x48, 0x03, 0xEB, 0x44, 0x8B, 0x59, 0x1C, 0x4C, 0x03, 0xF3,
        0x8B, 0x71, 0x14, 0x4C, 0x03, 0xDB, 0x85, 0xF6, 0x0F, 0x84, 0x80, 0x00,
        0x00, 0x00, 0x44, 0x8B, 0x51, 0x18, 0x33, 0xC9, 0x45, 0x85, 0xD2, 0x74,
        0x69, 0x48, 0x8B, 0xD5, 0x0F, 0x1F, 0x40, 0x00, 0x0F, 0xB7, 0x02, 0x41,
        0x3B, 0xC1, 0x74, 0x0D, 0xFF, 0xC1, 0x48, 0x83, 0xC2, 0x02, 0x41, 0x3B,
        0xCA, 0x72, 0xED, 0xEB, 0x4D, 0x45, 0x8B, 0x04, 0x8E, 0x4C, 0x03, 0xC3,
        0x74, 0x44, 0x41, 0x0F, 0xB6, 0x00, 0x33, 0xD2, 0xB9, 0x05, 0x15, 0x00,
```

```
        0x00, 0x84, 0xC0, 0x74, 0x35, 0x0F, 0x1F, 0x00, 0x6B, 0xC9, 0x21, 0x8D,
        0x52, 0x01, 0x0F, 0xBE, 0xC0, 0x03, 0xC8, 0x42, 0x0F, 0xB6, 0x04, 0x02,
        0x84, 0xC0, 0x75, 0xEC, 0x81, 0xF9, 0xFB, 0xF0, 0xBF, 0x5F, 0x75, 0x08,
        0x41, 0x8B, 0x3B, 0x48, 0x03, 0xFB, 0xEB, 0x0E, 0x81, 0xF9, 0x6D, 0x07,
        0xAF, 0x60, 0x75, 0x06, 0x45, 0x8B, 0x23, 0x4C, 0x03, 0xE3, 0x41, 0xFF,
        0xC1, 0x49, 0x83, 0xC3, 0x04, 0x44, 0x3B, 0xCE, 0x72, 0x84, 0x48, 0x8D,
        0x4C, 0x24, 0x20, 0xC7, 0x44, 0x24, 0x20, 0x75, 0x73, 0x65, 0x72, 0x66,
        0xC7, 0x44, 0x24, 0x24, 0x33, 0x32, 0x44, 0x88, 0x7C, 0x24, 0x26, 0xFF,
        0xD7, 0x45, 0x33, 0xC9, 0x48, 0x8B, 0xD8, 0x48, 0x63, 0x48, 0x3C, 0x8B,
        0x94, 0x01, 0x88, 0x00, 0x00, 0x00, 0x48, 0x03, 0xD0, 0x8B, 0x7A, 0x24,
        0x8B, 0x6A, 0x20, 0x48, 0x03, 0xF8, 0x44, 0x8B, 0x5A, 0x1C, 0x48, 0x03,
        0xE8, 0x8B, 0x72, 0x14, 0x4C, 0x03, 0xD8, 0x85, 0xF6, 0x74, 0x77, 0x44,
        0x8B, 0x52, 0x18, 0x0F, 0x1F, 0x44, 0x00, 0x00, 0x33, 0xC0, 0x45, 0x85,
        0xD2, 0x74, 0x5B, 0x48, 0x8B, 0xD7, 0x66, 0x0F, 0x1F, 0x44, 0x00, 0x00,
        0x0F, 0xB7, 0x0A, 0x41, 0x3B, 0xC9, 0x74, 0x0D, 0xFF, 0xC0, 0x48, 0x83,
        0xC2, 0x02, 0x41, 0x3B, 0xC2, 0x72, 0xED, 0xEB, 0x3D, 0x44, 0x8B, 0x44,
        0x85, 0x00, 0x4C, 0x03, 0xC3, 0x74, 0x33, 0x41, 0x0F, 0xB6, 0x00, 0x33,
        0xD2, 0xB9, 0x05, 0x15, 0x00, 0x00, 0x84, 0xC0, 0x74, 0x24, 0x66, 0x90,
        0x6B, 0xC9, 0x21, 0x8D, 0x52, 0x01, 0x0F, 0xBE, 0xC0, 0x03, 0xC8, 0x42,
        0x0F, 0xB6, 0x04, 0x02, 0x84, 0xC0, 0x75, 0xEC, 0x81, 0xF9, 0xB4, 0x14,
        0x4F, 0x38, 0x75, 0x06, 0x45, 0x8B, 0x3B, 0x4C, 0x03, 0xFB, 0x41, 0xFF,
        0xC1, 0x49, 0x83, 0xC3, 0x04, 0x44, 0x3B, 0xCE, 0x72, 0x92, 0x45, 0x33,
        0xC9, 0xC7, 0x44, 0x24, 0x20, 0x54, 0x65, 0x73, 0x74, 0x4C, 0x8D, 0x44,
        0x24, 0x20, 0xC6, 0x44, 0x24, 0x24, 0x00, 0x48, 0x8D, 0x54, 0x24, 0x20,
        0x33, 0xC9, 0x41, 0xFF, 0xD7, 0xBA, 0x7B, 0x00, 0x00, 0x00, 0x48, 0xC7,
        0xC1, 0xFF, 0xFF, 0xFF, 0xFF, 0x41, 0xFF, 0xD4, 0x48, 0x8B, 0x5C, 0x24,
        0x60, 0x48, 0x8B, 0x6C, 0x24, 0x68, 0x48, 0x8B, 0x74, 0x24, 0x70, 0x48,
        0x8B, 0x7C, 0x24, 0x78, 0x48, 0x83, 0xC4, 0x40, 0x41, 0x5F, 0x41, 0x5E,
        0x41, 0x5C, 0xC3
};
```

As a side note, several readers have asked how I created this sample code (previously used in another project) which works correctly in both 32-bit and 64-bit modes. The answer is very simple: it begins by storing the original stack pointer value, pushes a value onto the stack, and compares the new stack pointer to the original value. If the difference is 8, the 64-bit code is executed - otherwise, the 32-bit code is executed. While there are certainly more efficient approaches to achieve this outcome, this method is sufficient for demonstration purposes:

```
mov eax, esp    ; store stack ptr
push 0          ; push a value onto the stack
sub eax, esp    ; calculate difference
pop ecx         ; restore stack
cmp eax, 8      ; check if the difference is 8
je 64bit_code
32bit_code:
xxxx
64bit_code:
xxxx
```

By appending this payload to the original headers above, we can generate a valid and functional EXE file. The provided PE headers contain a hardcoded `SizeOfImage` value of 0x100000 which allows for a maximum payload size of almost 1MB, but this can be increased if necessary. Running this program will display our message box, despite the fact that the PE headers lack any executable sections, or any sections at all in this case:



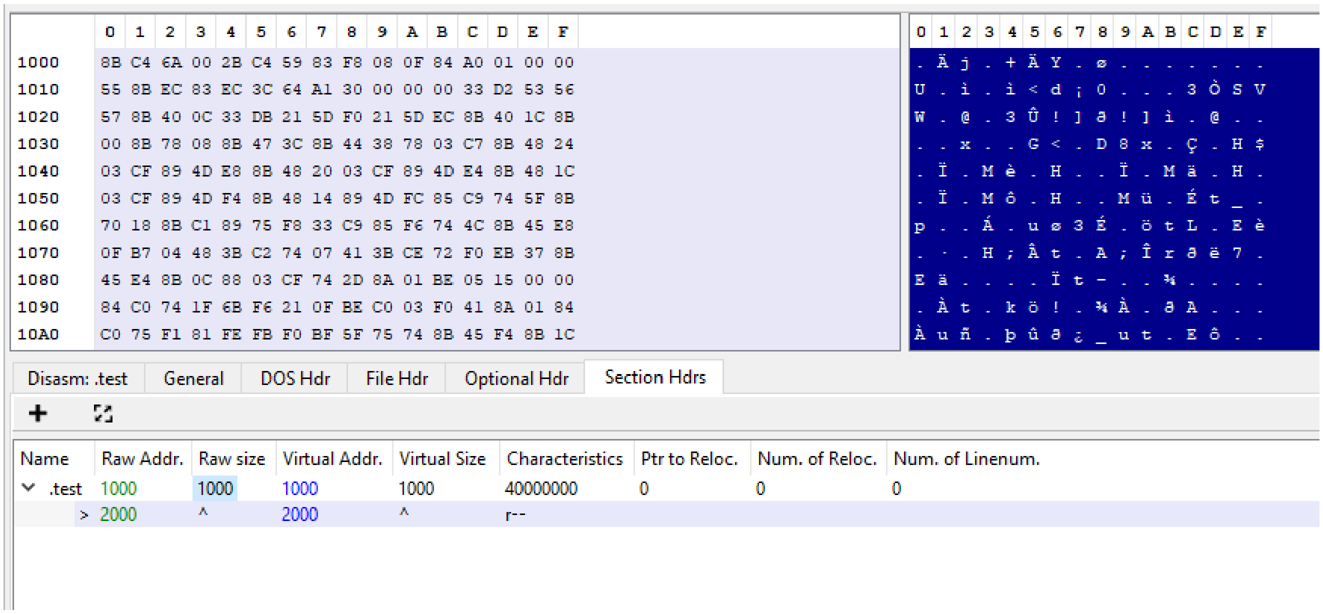
```
07FF423EA5000 00000000000FB000 Reserved (00007FF423EA0(                          MAP            -R---
07FF423FA0000 0000000100020000 Reserved                                          PRV            -RW--
07FF523FC0000 0000000002000000 Reserved                                          PRV            -RW--
07FF525FC0000 0000000000001000                                                   PRV   -RW--    -RW--
07FF525FD0000 0000000000001000                                                   MAP   -R---    -R---
07FF525FE0000 0000000000023000                                                   MAP   -R---    -R---
07FF7902F0000 0000000000100000 no_section_64.exe                                 IMG   ERWC-    ERWC-
07FFF13520000 0000000000001000 textshaping.dll                                   IMG   -R---    ERWC-
07FFF13521000 000000000004B000  ".text"            Exec                          IMG   ER---    ERWC-
07FFF1356C000 000000000005B000  ".rdata"           Read                   ized   IMG   -R---    ERWC-
07FFF135C7000 0000000000001000  ".data"            Initialized data              IMG   -RW--    ERWC-
```

Test

Test

Demonstration #2 - Executable PE with spoofed sections

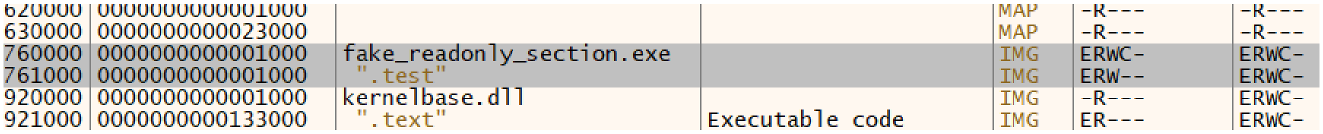
Perhaps more interestingly, it is also possible to create a fake section table using this mode as mentioned earlier. I have created another EXE which follows a similar format to the previous samples, but also includes a single read-only section:



```
      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F    0 1 2 3 4 5 6 7 8 9 A B C D E F
1000  8B C4 6A 00 2B C4 59 83 F8 08 0F 84 A0 01 00 00   . Ä j . + Ä Y . ø . . . . . . .
1010  55 8B EC 83 EC 3C 64 A1 30 00 00 00 33 D2 53 56   U . ì . ì < d ¡ 0 . . . 3 Ò S V
1020  57 8B 40 0C 33 DB 21 5D F0 21 5D EC 8B 40 1C 8B   W . @ . 3 Û ! ] ð ! ] ì . @ . .
1030  00 8B 78 08 8B 47 3C 8B 44 38 78 03 C7 8B 48 24   . . x . . G < . D 8 x . Ç . H $
1040  03 CF 89 4D E8 8B 48 20 03 CF 89 4D E4 8B 48 1C   . Ï . M è . H . . Ï . M ä . H .
1050  03 CF 89 4D F4 8B 48 14 89 4D FC 85 C9 74 5F 8B   . Ï . M ô . H . . M ü . É t _ .
1060  70 18 8B C1 89 75 F8 33 C9 85 F6 74 4C 8B 45 E8   p . . Á . u ø 3 É . ö t L . E è
1070  0F B7 04 48 3B C2 74 07 41 3B CE 72 F0 EB 37 8B   . · . H ; Â t . A ; Î r ð ë 7 .
1080  45 E4 8B 0C 88 03 CF 74 2D 8A 01 BE 05 15 00 00   E ä . . . . Ï t - . . ¾ . . . .
1090  84 C0 74 1F 6B F6 21 0F BE C0 03 F0 41 8A 01 84   . À t . k ö ! . ¾ À . ð A . . .
10A0  C0 75 F1 81 FE FB F0 BF 5F 75 74 8B 45 F4 8B 1C   À u ñ . þ û ð ¿ _ u t . E ô . .
```

Name	Raw Addr.	Raw size	Virtual Addr.	Virtual Size	Characteristics	Ptr to Reloc.	Num. of Reloc.	Num. of Linenum.
.test	1000	1000	1000	1000	40000000	0	0	0
>	2000	^	2000	^	r--			

The main payload has been stored within this read-only section and the entry-point has been updated to 0x1000. Under normal circumstances, you would expect the program to crash immediately with an access-violation exception due to attempting to execute read-only memory. However, this doesn’t occur here - the target memory region contains RWX permissions and the payload is executed successfully:



```
620000 0000000000001000                                                  MAP   -R---    -R---
630000 0000000000023000                                                  MAP   -R---    -R---
760000 0000000000001000 fake_readonly_section.exe                        IMG   ERWC-    ERWC-
761000 0000000000001000  ".test"                                         IMG   ERW--    ERWC-
920000 0000000000001000 kernelbase.dll                                   IMG   -R---    ERWC-
921000 0000000000133000  ".text"                  Executable code        IMG   ER---    ERWC-
```

Notes

The sample EXE files can be downloaded here.

The proof-of-concepts described above involve appending the payload to the end of the NT headers, but it is also possible to embed executable code within the headers themselves using this technique. The module will fail to load if the `AddressOfEntryPoint` value is less than the `SizeOfHeaders` value, but this can easily be bypassed since the `SizeOfHeaders` value is not strictly enforced. It can even be set to 0, allowing the entry-point to be positioned anywhere within the file.

It is possible that this feature was initially designed to allow for very small images, enabling the headers, code, and data to fit within a single memory page. As memory protection is applied per-page, it makes sense to apply RWX to all PTEs when the virtual section size is lower than the page size - it would otherwise be impossible to manage protections correctly if multiple sections resided within a single page.

I have tested these EXE files on various different versions of Windows from Vista to 10 with success in all cases. Unfortunately it has very little practical use in the real world as it won’t deceive any modern disassemblers - nonetheless, it remains an interesting concept.