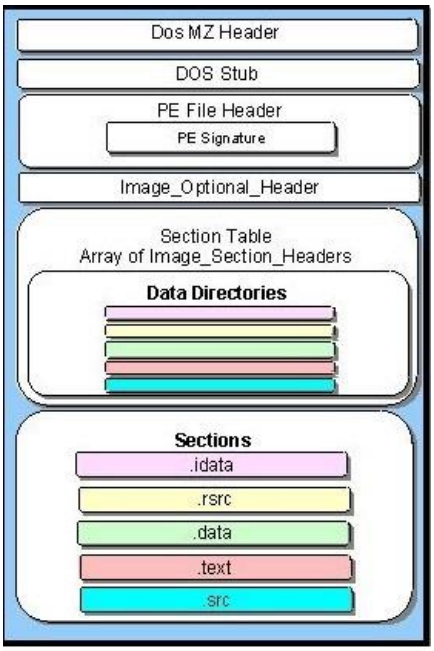
Cách dump PE file từ bộ nhớ bằng IDA

Mã độc từ lâu đã được trang bị các kĩ thuật lẩn tránh nhằm mục đích qua mặt các giải pháp bảo mật. Một trong những kĩ thuật phổ biến nhất hiện nay mà các loại mã độc thường sử dụng đó là giải nén module/payload cuối của chúng vào bộ nhớ, sau đó thực thi module/payload này.

Có nhiều cách thức khác nhau để lấy được module/payload, nội dung dưới đây tập trung vào việc sử dụng công cụ IDA để dump payload từ bộ nhớ.

1. Cơ bản về PE file

Bài viết sẽ không trình bày chi tiết về cấu trúc của PE file bởi đã có rất nhiều tài liệu viết về nó. Các tài liệu đều cung cấp các thông tin giải thích chi tiết về cấu trúc cũng như ý nghĩa của từng trường/giá trị trong PE file. Hình dưới đây minh họa tổng quan nhất về cấu trúc thông thường của một PE file:



Nguồn: https://resources.infosecinstitute.com/2-malware-researchers-handbook-demystifying-pe-file

Như vậy, về cơ bản thì PE file sẽ gồm một loạt các khối bộ nhớ đặt liền kề nhau. Trong đó, PE header chứa nhiều thông tin quan trọng như: địa chỉ **entry point** của file, thời gian biên dịch, số lượng section, địa chỉ các sections, thông tin về kích thước, bảng IAT v.v…

Để phục vụ cho mục tiêu của bài viết, cần quan tâm đến thông tin các sections, vì khi tính tổng địa chỉ của một section với kích thước của nó, sẽ nhận được địa chỉ của section tiếp theo. Và nếu lấy tổng của địa chỉ và kích thước của section cuối cùng, sẽ có được kích thước của file.

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00000DB1	00001000	00000E00	00000400	00000000	00000000	0000	0000	60000020
.rdata	00000BBC	00002000	00000C00	00001200	00000000	00000000	0000	0000	40000040
.data	00000384	00003000	00000200	00001E00	00000000	00000000	0000	0000	C0000040
.gfids	00000020	00004000	00000200	00002000	00000000				40000040
.rsrc	000001E0	00005000	00000200	00002200	00000000				40000040
.reloc	00000154	00006000	00000200	00002400	00000000	00000000	0000	0000	42000040

Size: 9.50 KB (9,728 bytes)

Hình 1: Thông tin về các sections của PE file

2. Phân tích file

Giả sử trong quá trình phân tích malware, gặp đoạn code thực hiện giải nén một PE file mới vào vùng nhớ như sau:

```
UCHAR *__cdecl f_DecompressNewPEfile(PULONG FinalUncompressedSize)
{
  NTSTATUS status; // [esp+0h] [ebp-24h]
  UCHAR *UncompressedBuffer; // [esp+8h] [ebp-1Ch]
  unsigned int l; // [esp+Ch] [ebp-18h]
  unsigned int k; // [esp+10h] [ebp-14h]
  unsigned int j; // [esp+14h] [ebp-10h]
  unsigned int i; // [esp+18h] [ebp-Ch]
  UCHAR *CompressedBuffer; // [esp+1Ch] [ebp-8h]

  CompressedBuffer = f_AllocateHeapMemory();
  UncompressedBuffer = f_AllocateHeapMemory();
  for ( i = 0; i < 0x2A04; i += 4 )
    CompressedBuffer[i] = *(i + 0x402048) ^ 0x68;          // XOR first byte
  for ( j = 1; j < 0x2A04; j += 4 )
    CompressedBuffer[j] = *(j + 0x402048) ^ 0x8A;          // XOR second byte
  for ( k = 2; k < 0x2A04; k += 4 )
    CompressedBuffer[k] = *(k + 0x402048) ^ 0x49;          // XOR third byte
  for ( l = 3; l < 0x2A04; l += 4 )
    CompressedBuffer[l] = *(l + 0x402048) ^ 0xEC;          // XOR fourth byte
  // Decompress a new PE file
  status = RtlDecompressBuffer(COMPRESSION_FORMAT_LZNT1, UncompressedBuffer, 0xD214u, CompressedBuffer, 0x2A04u, FinalUncompressedSize);
  au_func_HeapFree(CompressedBuffer);
  if ( !status )
    return UncompressedBuffer;
  au_func_HeapFree(UncompressedBuffer);
  UncompressedBuffer = 0;
  *FinalUncompressedSize = 0;
  return UncompressedBuffer;
}
```

Hình 2: Đoạn mã thực hiện giải nén một PE file mới vào bộ nhớ

Sau khi thực hiện đoạn code trên, kết quả có được một PE file mới trong bộ nhớ:

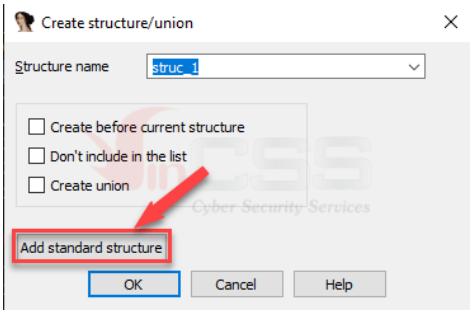
```
00448480  00 00 00 00 00 00 00 00  43 DF 54 2F EA A7 00 1C  ........CßT/ê§..
00448490  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
004484A0  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ........@.......
004484B0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
004484C0  00 00 00 00 00 00 00 00  00 00 00 00 D0 00 00 00  ............Ð...
004484D0  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
004484E0  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is·program·canno
004484F0  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t·be·run·in·DOS·
00448500  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
00448510  FD 65 BE 38 B9 04 D0 6B  B9 04 D0 6B B9 04 D0 6B  ýe¾8¹.Ðk¹.Ðk¹.Ðk
00448520  9E C2 AB 6B BB 04 D0 6B  B0 7C 43 6B B4 04 D0 6B  žÂ«k».Ðk°|Ck´.Ðk
00448530  B9 04 D1 6B FB 04 D0 6B  0C 9A 31 6B BE 04 D0 6B  ¹.Ñkû.Ðk.š1k¾.Ðk
00448540  0C 9A 0E 6B B8 04 D0 6B  52 69 63 68 B9 04 D0 6B  .š.k¸.ÐkRich¹.Ðk
00448550  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00448560  50 45 00 00 4C 01 04 00  DC 32 FC 5B 00 00 00 00  PE..L...Ü2ü[....

UNKNOWN 004484C0: debug060:004484C0
```

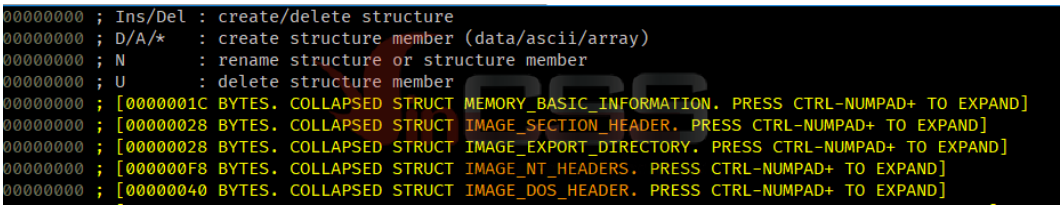
Hình 3: PE file mới sau khi được giải nén

Với thông tin trên, ta biết được địa chỉ bắt đầu của PE file trong bộ nhớ (ví dụ trong hình là **`0x448490`**). Nhiệm vụ tiếp theo là cần phải phân tích các cấu trúc của file mới để tính ra kích thước. Công việc này được thực hiện một cách nhanh chóng nhờ vào sự hỗ trợ của IDA. Để phân tích, cần phải nạp các struct cần thiết liên quan tới PE file, bao gồm **`IMAGE_DOS_HEADER`**, **`IMAGE_NT_HEADERS`** và **`IMAGE_SECTION_HEADER`**.

Tại IDA, Nhấn **Shift + F9** để chuyển tới tab **Structures**, sau đó nhấn **Insert** để thực hiện thêm các struct mới. Tại màn hình **Create structure/union** chọn **Add standard structure** để lựa chọn các cấu trúc chuẩn mà IDA đã định nghĩa trước và thêm các struct đã đề cập ở trên:



Hình 4: Tạo một struct mới trong IDA



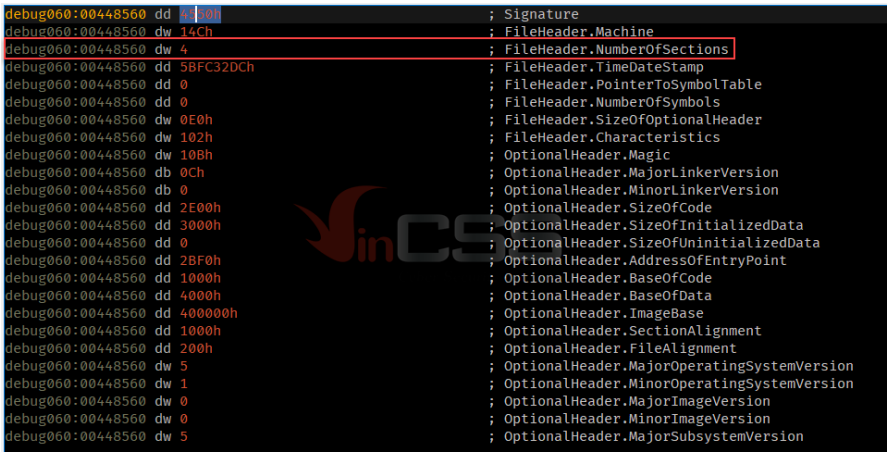
Hình 5: Bổ sung các struct cần thiết của PE file

Như đã giải thích ở phần đầu, khi lấy địa chỉ của section cuối cùng cộng với kích thước của nó sẽ có được kích thước của file. Theo cách tổ chức của PE file thì các sections được bố trí nằm sau PE header. PE header được tìm thấy bằng cách cộng địa chỉ của **MZ header** (địa chỉ bắt đầu của PE file) với giá trị của trường **e_lfanew**.

Chuyển tới địa chỉ bắt đầu của file trong bộ nhớ (theo minh họa trong bài viết là **0x448490**), đặt con trỏ tại **MZ** (DOS) header, nhấn **Alt + Q**, chọn **IMAGE_DOS_HEADER**. Kết quả sẽ tương tự như hình dưới đây:

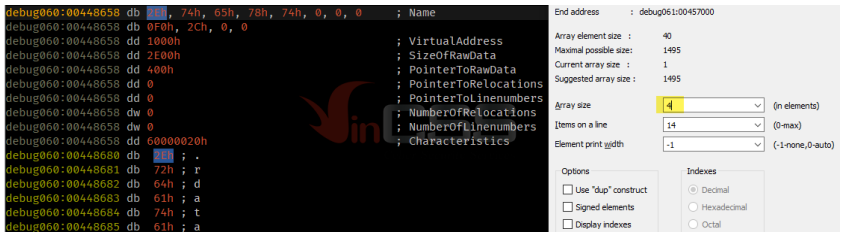
Hình 6: Kết quả sau khi áp IMAGE_DOS_HEADER

PE header luôn bắt đầu tại **MZ + e_lfanew**, do vậy lấy địa chỉ bắt đầu (**0x00448490**) cộng với **e_lfanew** (trong ví dụ này là **0xD0**) ra được địa chỉ của PE header tại **0x448560**. Chuyển tới địa chỉ này, áp cấu trúc **IMAGE_NT_HEADERS** cho nó. Kết quả có được như sau:



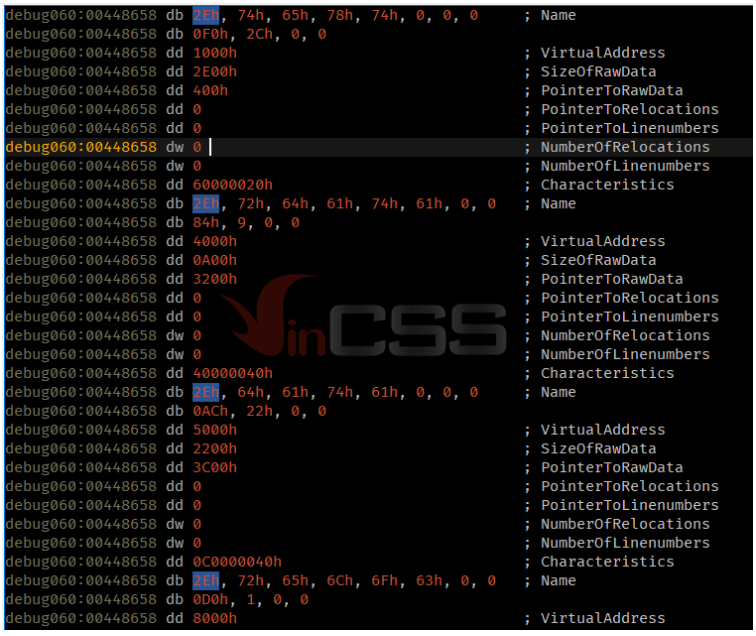
Hình 7: Kết quả sau khi áp IMAGE_NT_HEADERS

Lưu ý về số lượng các sections như đã khoanh trên hình, cuộn xuống cuối cấu trúc sẽ thấy một loạt các bytes liền kề nhau, tất cả đều thuộc **IMAGE_SECTION_HEADER**. Trước tiên, đặt chuột tại byte đầu tiên và áp **IMAGE_SECTION_HEADER** cho nó. Căn cứ vào **FileHeader.NumberOfSections**, trong trường hợp này là **4**, tiếp theo nhấn ***** để thực hiện tạo một mảng. Tại **Array Size** nhập thông tin là **4**:



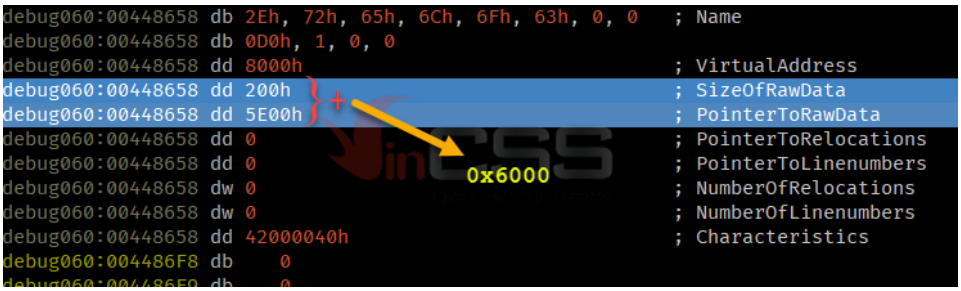
Hình 8: Kết quả sau khi áp IMAGE_SECTION_HEADER

Kết quả cuối cùng có được:



Hình 9: Thông tin đầy đủ các sections của PE File

Tìm tới section cuối cùng, lấy giá trị tại các trường **`PointerToRawData`** và **`SizeOfRawData`** cộng với nhau sẽ có được kích thước của file cần dump:



Hình 10: Tính toán kích thước của PE File

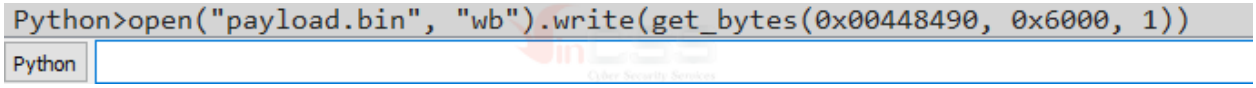
3. Dump file ra disk

Để thực hiện dump file từ bộ nhớ ra disk, sử dụng câu lệnh IDAPython như sau:

```
open("payload.bin", "wb").write(get_bytes(mz_addr, size, 1))
```

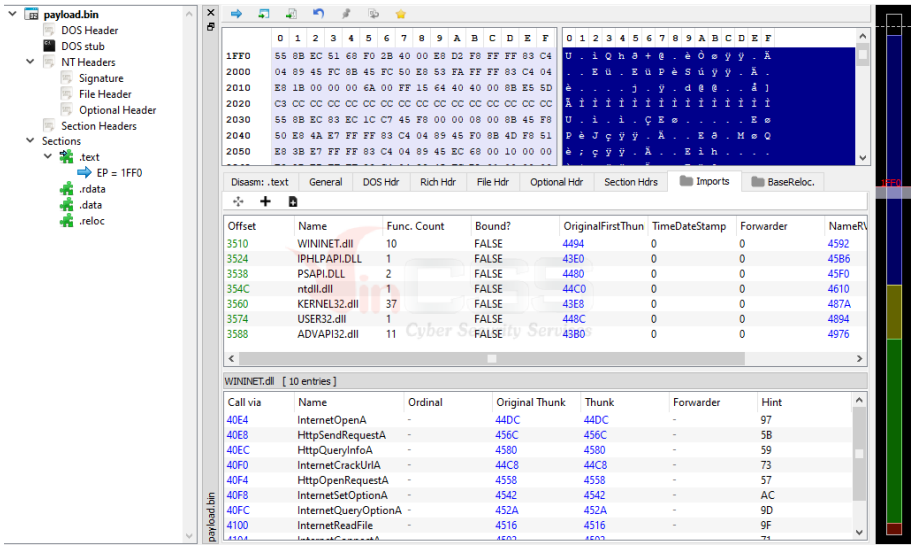
Trong đó:

- **`mz_addr`**: địa chỉ bắt đầu của MZ header (trong ví dụ này: **`0x00448490`**)
- **`size`**: kích thước tính toán được ở trên (trong ví dụ này: **`0x6000`**)



Hình 11: Thực hiện dump file sau khi có đủ thông tin

Cuối cùng, kiểm tra lại file đã dump bằng một công cụ PE bất kì:



Hình 12: Kiểm tra lại file đã dump