2013

[Cracking with OllyDbg]

Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)

www.reaonline.net

kienmanowar

19/07/2013

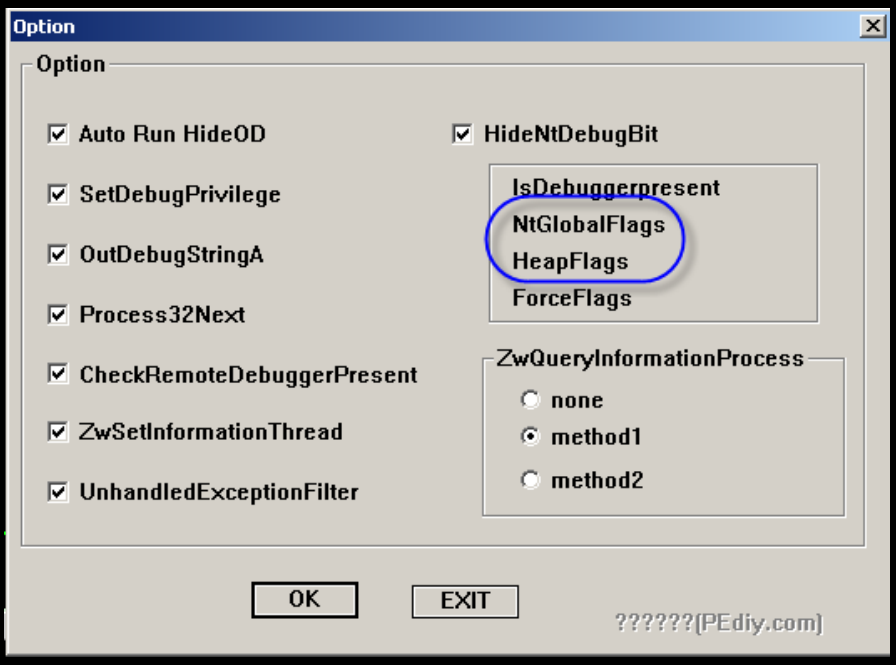
Mục Lục

I. Giới thiệu chung

Khà khà, lâu quá nên chẳng biết lời mở đầu phải bắt đầu từ đâu và phải bắt đầu như thế nào. Tính ra cũng phải hơn 3 năm kể từ ngày tôi dừng lại ở phần 22. Quãng thời gian 3 năm cũng không phải là dài, cũng không phải là quá ngắn. Diễn đàn REA thì cũng đóng rồi, do anh em BQT nhận thấy thời của 4rum cũng xuống dần, cộng với cơm áo gạo tiền và bộn bề cuộc sống, không ai còn đủ thời gian để quản lý, điều hành cũng như hàng ngày phải mất công "ban nick" một tá những thằng ất ơ vào spam và quấy rồi, giờ chủ yếu là lên face chém gió, hẹn hò nhậu nhẹt ...Blah blah. Trong các phần trước tôi đã giới thiệu tới các bạn các chủ đề liên quan tới Anti-Debug bằng cách sử dụng các phương pháp như `IsDebuggerPresent`, `CreateToolhelp32Snapshot`, `Process32First`, `Process32Next`, `SetUnhandledExceptionFilter`, `UnhandledExceptionFilter`, `ZwQueryInformationProcess` `v..v..` cũng như các các trick để bypass, phần 23 này sẽ khép lại chủ đề Anti-Debug với phương pháp sử dụng `NtGlobalFlag` và `HeapFlags`. Trên thực tế, các trình packer đình đám như EXECryptor cũng đều áp dụng các phương pháp trên, song bên cạnh đó chúng còn có các thủ thuật, phương pháp tinh vi khác, mà cứ mỗi version mới ra đời thì lại có các kĩ thuật nâng cao và quái chiêu hơn. Tuy nhiên, các cụ nhà ta thường có câu: "Vỏ quýt dày móng tay nhọn". N0w let's g0……☺

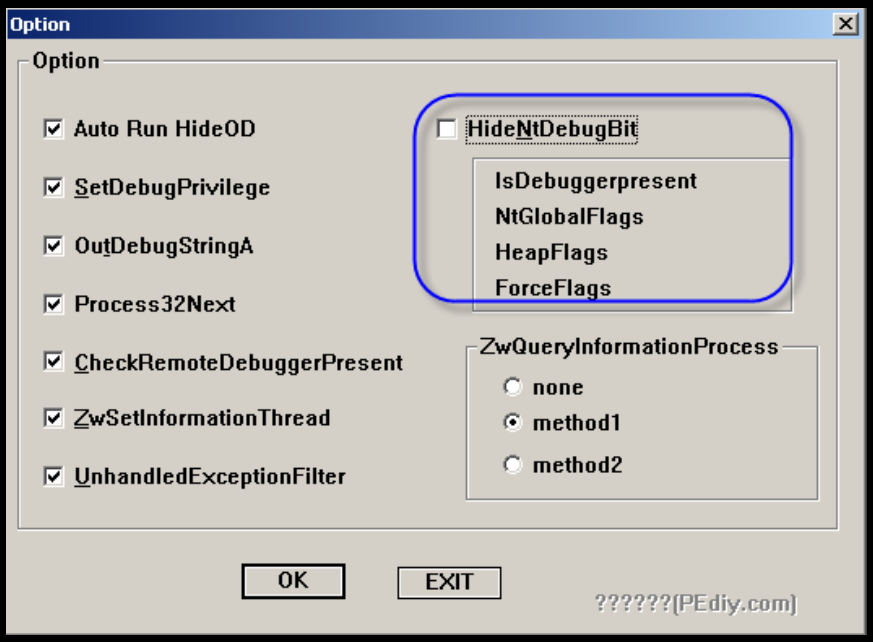
II. Phân tích và xử lý target

Như các bạn đã biết, với các phương pháp mới đề cập ở trên thì plugin HideOD của lão kanxue bên PeiDy đều hỗ trợ để bypass:

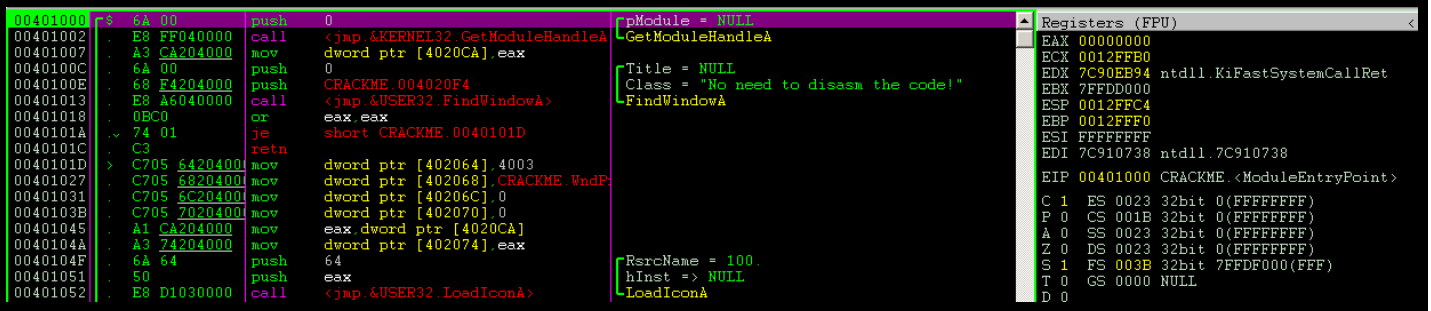


Tuy nhiên, với bài viết này tôi sẽ cùng với các bạn tìm hiểu và vượt qua các cơ chế này một cách thủ công nhằm hiểu rõ hơn cách thức mà chúng áp dụng. Về cơ bản, hai Flag này được sử dụng để giúp phát hiện Process đang bị debug, rất dễ để tìm ra vị trí của các Flag này do chúng có liên quan với cấu trúc PEB, nếu như bạn không còn nhớ thì

hãy xem lại phần 19, liên quan tới việc tìm kiếm vùng byte được sử dụng bởi hàm IsDebuggerPresent.
Để bắt đầu, trước tiên ta bỏ chọn trong plugin HideOD, mục đích là không muốn plugin tự động fix:



Target được dùng để thực hành với những Flag trên chính là CRUEHEAD CRACKME v1.0, thực tế crackme này không có Anti-debug gì cả, ta chỉ mượn nó để thực hành mà thôi.
Sau khi cấu hình lại Plugin HideOD như trên, tiến hành load crackme vào OllyDbg, ta dừng lại tại EP của crackme:



Tiếp theo, chúng ta sẽ tìm cách để xác định vị trí và thay đổi giá trị của các Flag. Thông qua Google và tài liệu, tôi có được thông tin về NtGlobalFlag như sau:

NtGlobalFlag

NtGlobalFlag is a DWORD value inside the process PEB. This value contains many flags set by the Operating System that affects the way the process runs. When a process is being debugged, the flags FLG_HEAP_ENABLE_TAIL_CHECK (0x10), FLG_HEAP_ENABLE_FREE_CHECK(0x20), and FLG_HEAP_VALIDATE_PARAMETERS(0x40) are set for the process, and we can use this to our advantage to identify if our process is being debugged.

Collapse | Copy Code

```
unsigned long NtGlobalFlags = 0;

__asm {

    mov eax, fs:[30h]
    mov eax, [eax + 68h]
    mov NtGlobalFlags, eax
}


if(NtGlobalFlags & 0x70)
// 0x70 =  FLG_HEAP_ENABLE_TAIL_CHECK |
//         FLG_HEAP_ENABLE_FREE_CHECK |
//         FLG_HEAP_VALIDATE_PARAMETERS
{
    // Debugger is present
    MessageBox(NULL, TEXT("Please close your debugging " +
               "application and restart the program"),
               TEXT("Debugger Found!"), 0);
    ExitProcess(0);
}
// Normal execution
```

Nguồn: http://www.codeproject.com/Articles/29469/Introduction-Into-Windows-Anti-Debugging

Checking NTGlobalFlag

Since processes run slightly differently when started with a debugger, they create memory heaps differently. The information that the system uses to determine how to create heap structures is stored at an undocumented location in the PEB at offset 0x68. If the value at this location is 0x70, we know that we are running in a debugger.

The value of 0x70 is a combination of the following flags when a heap is created by a debugger. These flags are set for the process if it is started from within a debugger.

```
(FLG_HEAP_ENABLE_TAIL_CHECK | FLG_HEAP_ENABLE_FREE_CHECK | FLG_HEAP_VALIDATE_PARAMETERS)
```

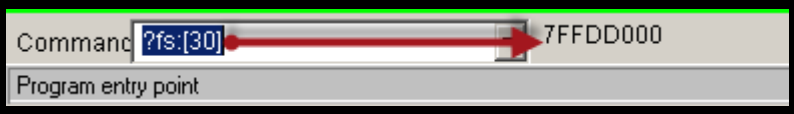
Listing 16-4 shows the assembly code for performing this check.

```
mov eax, large fs:30h
cmp dword ptr ds:[eax+68h], 70h
jz DebuggerDetected
```

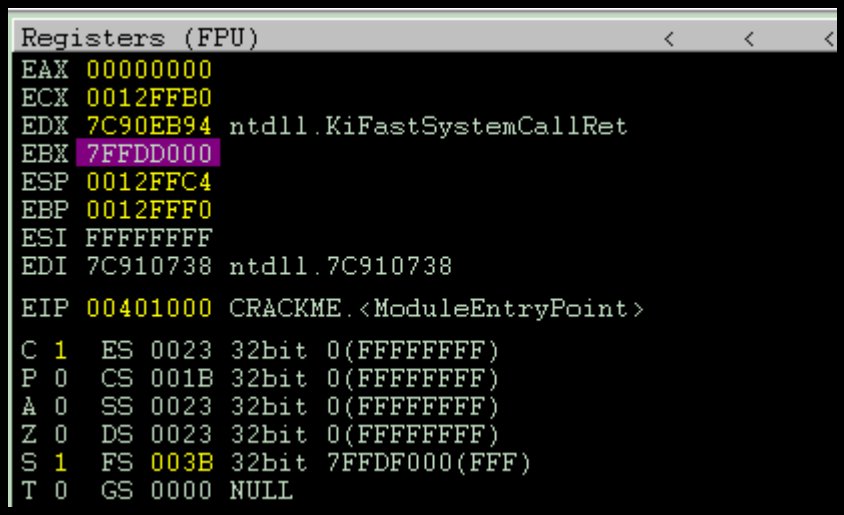
Nguồn: Tài liệu Practical Malware Analysis, tác giả Michael Sikorski và Andrew Honig.

Qua hai thông tin trên, ta có được kết luận: `NtGlobalFlag` nằm ở offset `0x68` của PEB, tương tự như `BeingDebugged` (đề cập trong phần 19) nó cũng được thiết lập là `0x0` nếu như không bị Debug và sẽ là `0x70` trong trường hợp bị Debug.

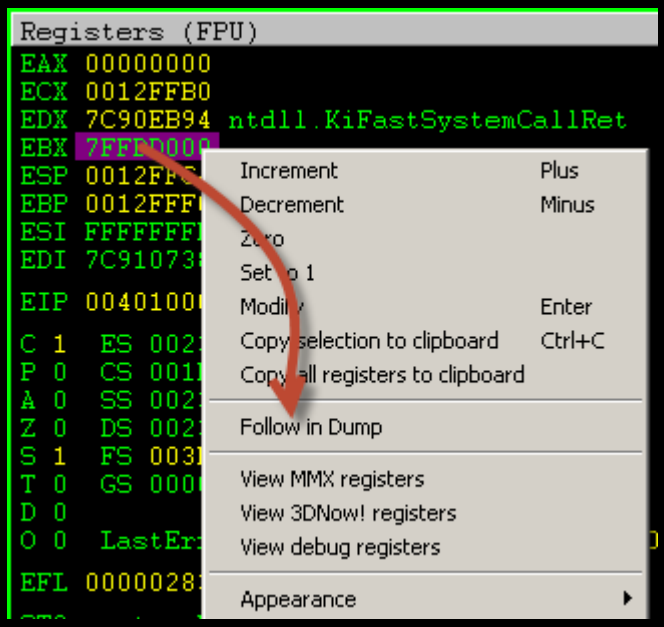
Ok, với thông tin có được ở trên, ta đi tìm giá trị của `FS:[30]` bằng cách sử dụng Command Bar và gõ lệnh sau:



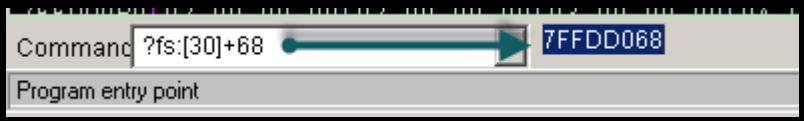
Quan sát trên cửa sổ Registers, ta thấy giá trị trên trùng khớp với thanh ghi `EBX`:



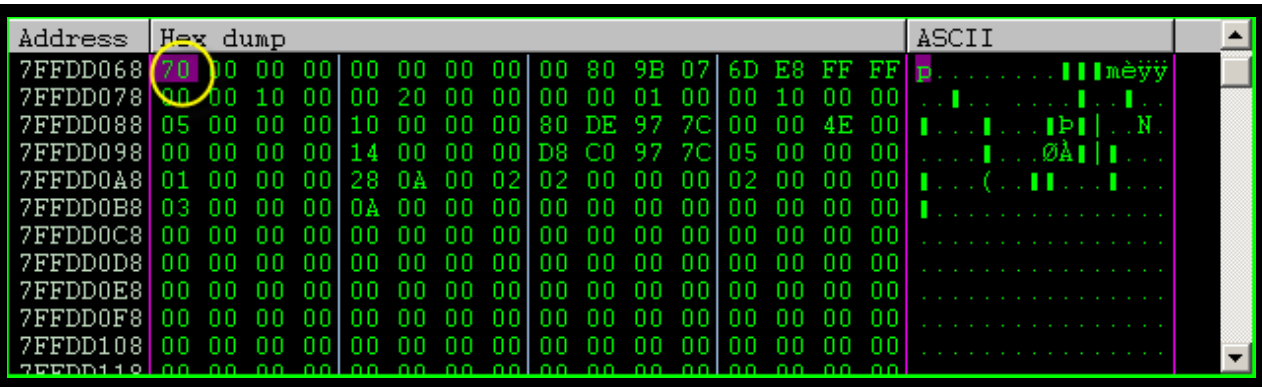
Chuột phải tại thanh ghi `EBX` và chọn Follow in Dump:



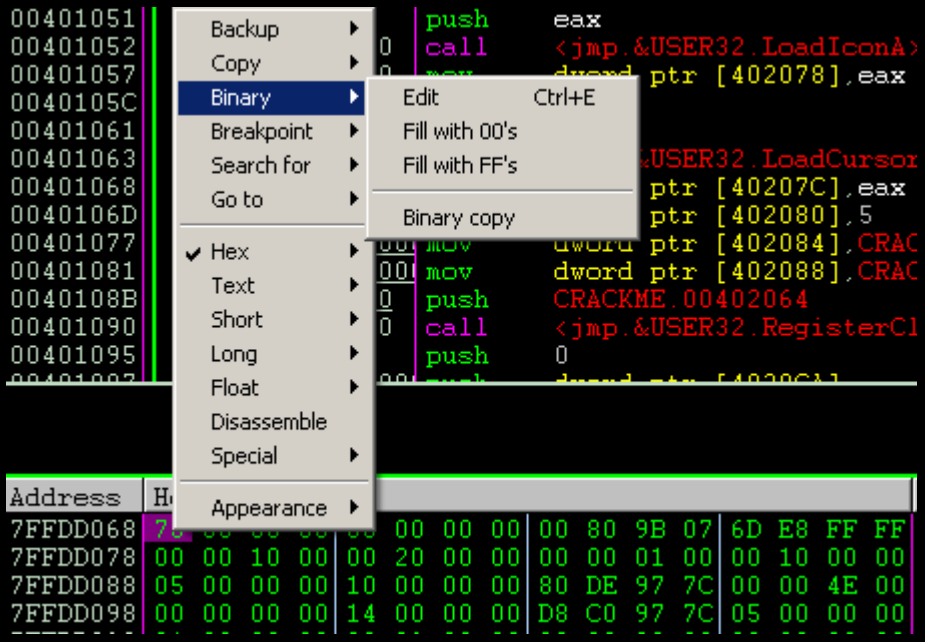
Như vậy ta đã biết được giá trị `FS:[30]` là `0x7FFDD000`, theo lý thuyết ở trên thì `NtGlobalFlag` nằm ở offset `0x68` của PEB. Ta lấy `0x7FFDD000 + 0x68 = 0x7FFDD068`:



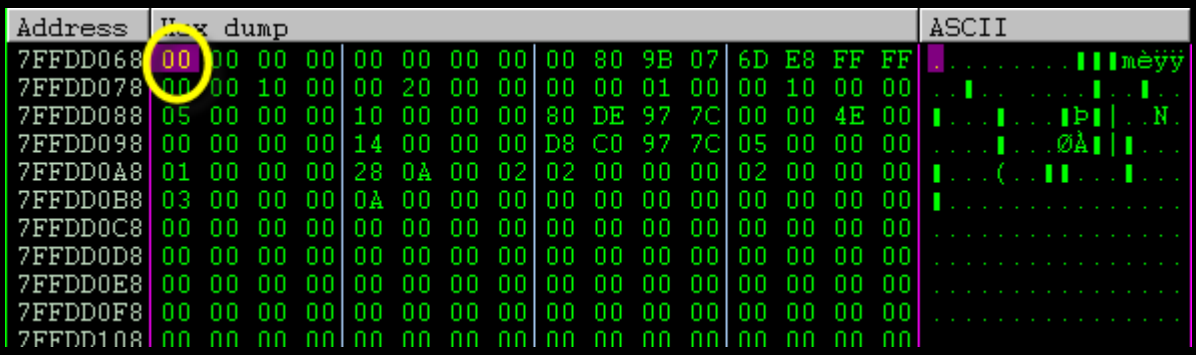
Tại cửa sổ Dump, nhấn Ctrl+G và nhập giá trị trên ta được thông tin như sau:



Ta thấy, giá trị ở đây là `0x70`, tương ứng với crackme đang bị debug. Để bypass được cơ chế Anti-Debug này, đơn giản ta chỉ việc sửa `0x70` thành `0x0`. Chuột phải tại `0x70` và chọn Binary > Edit:



Kết quả sau khi sửa lại tương tự như hình sau:



Phù, như vậy là ta đã tìm được vị trí và chỉnh sửa giá trị của `NtGlobalFlag` để vượt qua Anti-Debug. Tiếp theo chúng ta sẽ tìm `ProcessHeap Flag`. Thông tin về `ProcessHeap Flag` ta có được ở các tài liệu như sau:

Checking the ProcessHeap Flag

An undocumented location within the `Reserved4` array (shown in Listing 16-2), known as `ProcessHeap`, is set to the location of a process's first heap allocated by the loader. `ProcessHeap` is located at 0x18 in the PEB structure. This first heap contains a header with fields used to tell the kernel whether the heap was created within a debugger. These are known as the `ForceFlags` and `Flags` fields.

Offset 0x10 in the heap header is the `ForceFlags` field on Windows XP, but for Windows 7, it is at offset 0x44 for 32-bit applications. Malware may also look at offset 0x0C on Windows XP or offset 0x40 on Windows 7 for the `Flags` field. This field is almost always equal to the `ForceFlags` field, but is usually ORed with the value 2.

Listing 16-3 shows the assembly code for this technique. (Note that two separate dereferences must occur.)

```
mov eax, large fs:30h
mov eax, dword ptr [eax+18h]
cmp dword ptr ds:[eax+10h], 0
jne DebuggerDetected
```

Listing 16-3: Manual ProcessHeap flag check

The best way to overcome this technique is to change the `ProcessHeap` flag manually or to use a hide-debug plug-in for your debugger. If you are using WinDbg, you can start the program with the debug heap disabled. For example, the command `windbg -hd notepad.exe` will start the heap in normal mode as opposed to debug mode, and the flags we've discussed won't be set.

Nguồn: Tài liệu Practical Malware Analysis, tác giả Michael Sikorski và Andrew Honig.

E. PEB ProcessHeap Flag Debugger

There are additional flags within the PEB structure that are used to indicate to the kernel how heap memory should be allocated. In addition to the NtGlobalFlag, the ProcessHeap Flag can also be used to determine if a process was started with a debugger attached. By dereferencing a pointer to the first heap located at offset 0x18 in the PEB structure, we can then locate the heap flags which are located at offset 0x10 within the heap header. These heap flags are used to indicate to the kernel that the heap was created within a debugger. If the value at this location is non-zero then the heap was created within a debugger.

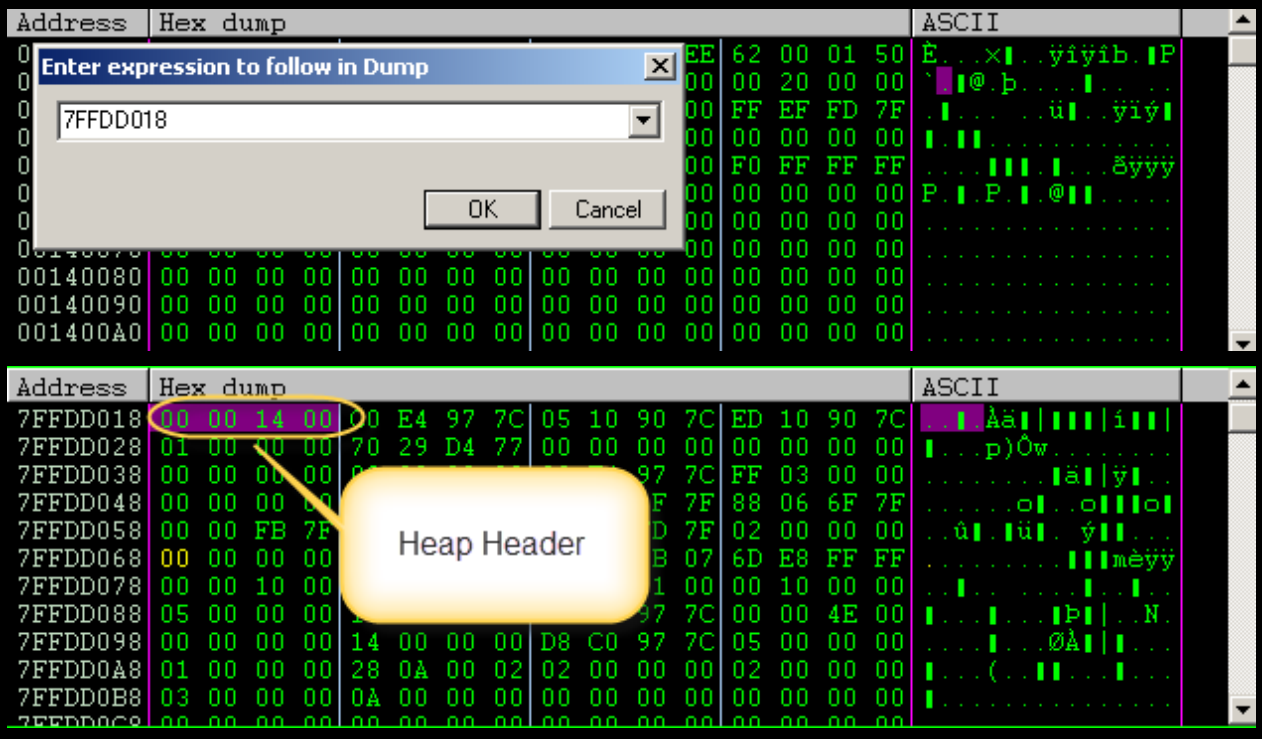
```
base = (char *)pPIB.PebBaseAddress;
procHeap = base+0x18;
procHeap = *procHeap;
heapFlag = (char*) procHeap+0x10;
last = (DWORD*) heapFlag;

if(*heapFlag != 0x00) {
    MessageBox(NULL, L"Debugger Detected Using PEB
Process Heap Flag", L"Debugger Detected", MB_OK);
    } else {
    MessageBox(NULL, L"No Debugger Detected", L"No
Debugger Detected", MB_OK);
}
```

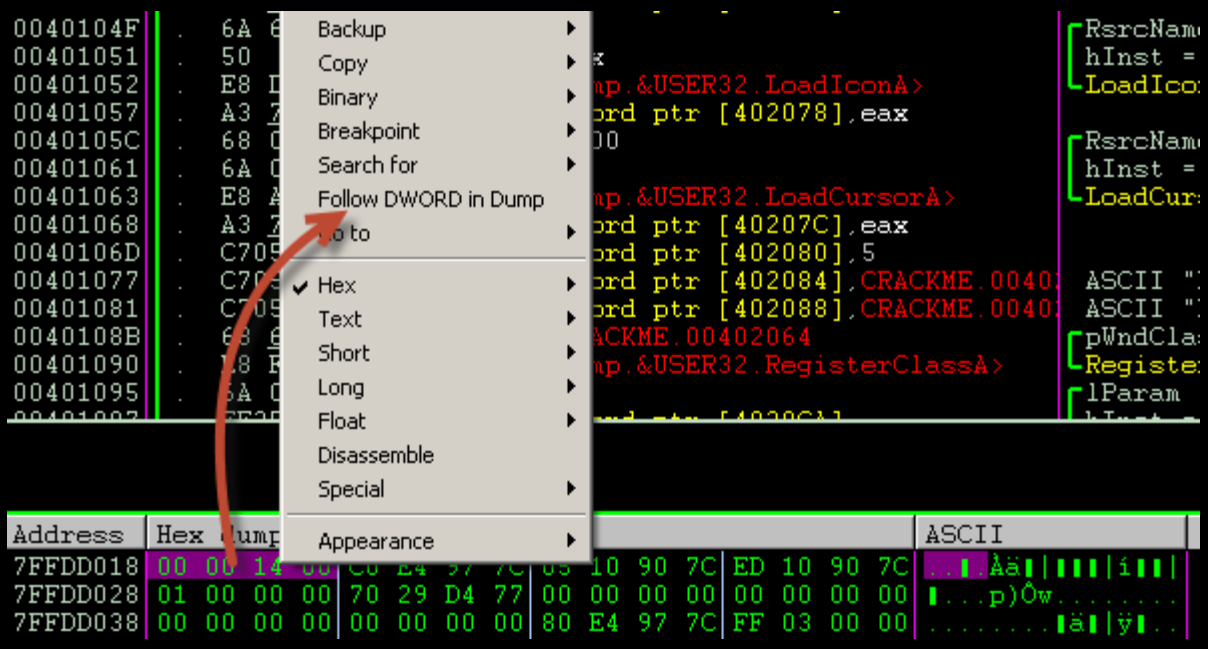
Nguồn: Tài liệu Anti-Debugging – A Developers View, tác giả Tyler Shields

Tóm lại, ProcessHeap nằm ở offset 0x18 của PEB, và tại offset này sẽ chứa giá trị Heap header. Offset thứ 0x10 tại Heap Header chính là Heap Flags. Có vẻ hơi lằng nhằng nhỉ, ta sẽ lần lượt đi từng bước.
Đầu tiên, ta xác định ProcessHeap bằng cách lấy fs:[30]+18 = 0x7FFDD000 + 0x18 = 0x7FFDD018

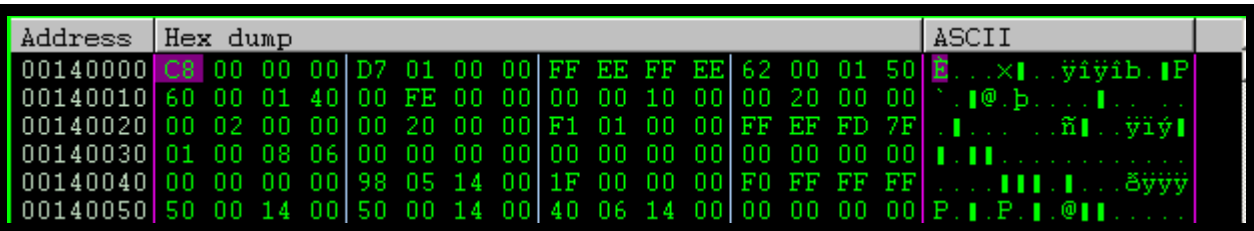
Tại cửa sổ Dump, ta tới giá trị đã tính toán ở trên:



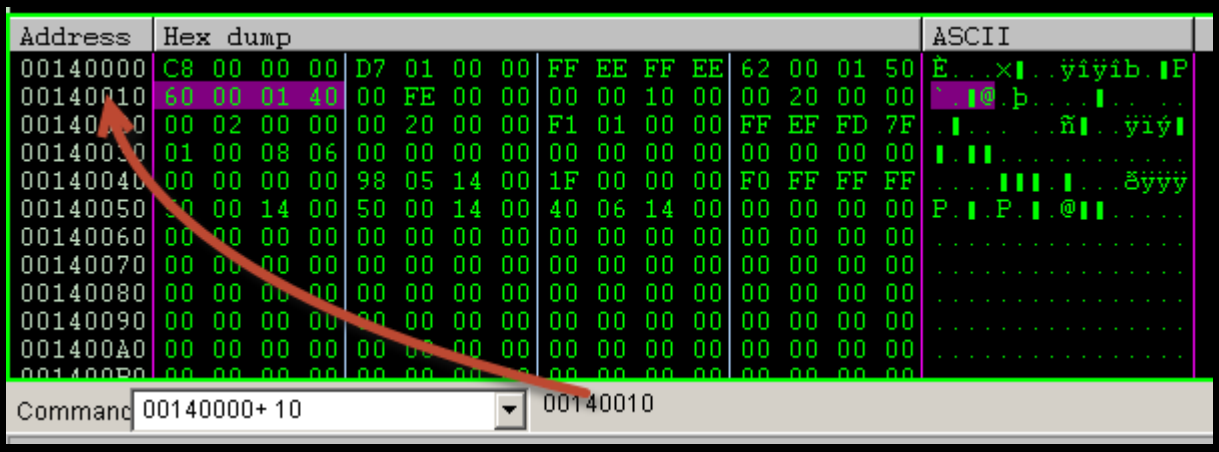
Giá chị heap header: 0x00140000. Follow DWORD in Dump tại giá trị này:



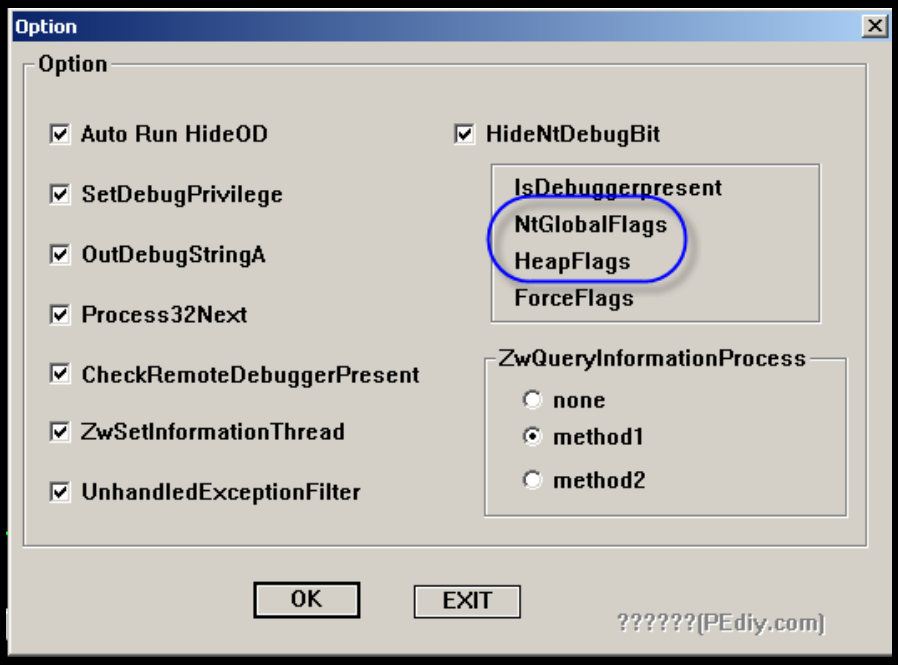
Ta đến đây trong cửa sổ dump:



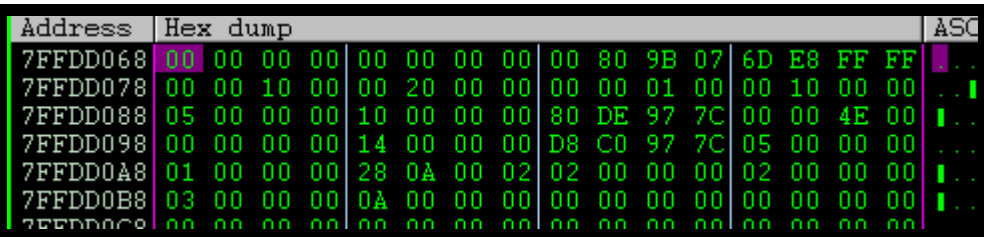
Offset thứ `0x10` của heap header chính là `0x00140000 + 0x10 = 0x00140010`:



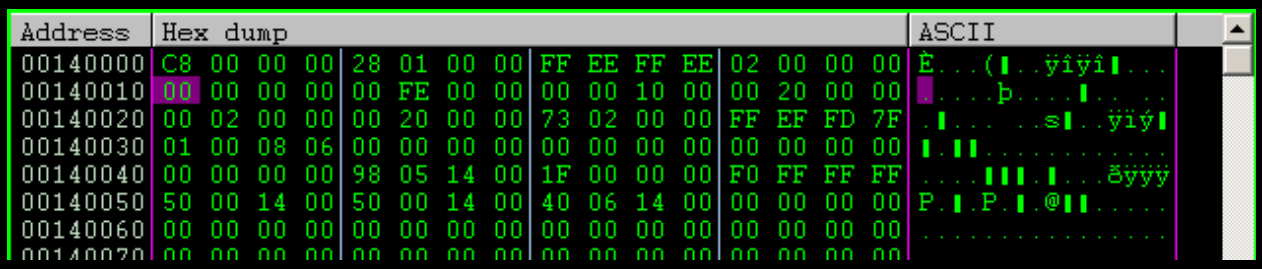
Ta thấy, nếu giá trị ở đây khác `0x0` thì chương trình đang bị Debug. OK, như vậy ta vừa tìm hiểu xong `NtGlobalFlag` và `HeapFlags`. Bây giờ, cấu hình lại plugin HideOD như dưới đây và restart Olly:



Load lại crackme, tiến hành kiểm tra `NtGlobalFlag`, ta thấy nó đã được fix thành giá trị `0x0`:



Tiếp theo ta tìm thông tin về `HeapFlags`, ta thấy nó cũng mang giá trị `0x0`:



Qua đó, có thể thấy rằng plugin HideOD hoạt động rất chính xác.

III. Kết luận

OK, toàn bộ bài 23 đến đây là kết thúc, có vẻ cũng không quá phức tạp với các bạn. Ngoài ra, để phục vụ cho các bạn thực hành, bác Ricardo có cung cấp thêm một crackme có tên là Antisocial1.exe, theo bác Ricardo thì crackme này hội tụ đủ các tricks mà các bạn đã gặp trong các bài viết có liên quan đến Anti-Debug, cộng thêm một vài trick mới ☺. Mục tiêu là làm sao thực thi được crackme đó trong OllyDBG. Hẹn gặp lại các bạn ở bài 24 hi vọng sẽ mang lại nhiều điều thú vị khác!

PS: Tài liệu này chỉ mang tính tham khảo, tác giả không chịu trách nhiệm nếu người đọc sử dụng nó vào bất kì mục đích nào.

Best Regards
[Kienmanowar]

--++--==[Greatz Thanks To]==--++--
My family, Computer_Angel, Moonbaby , Zombie_Deathman, Littleboy, Benina, QHQCrker, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dqtln, ARTEAM .... all my friend, and YOU.

--++--==[Thanks To]==--++--
iamidiot, WhyNotBar, trickyboy, dzungltvn, takada, hurt_heart, haule_nth, hytkl, moth, XIANUA, nhc1987, 0xdie, Unregistered!, akira, mranglex v..v.. các bạn đã

đóng góp rất nhiều cho REA. Hi vọng các bạn sẽ tiếp tục phát huy ☺

I want to thank **Teddy Roggers** for his great site, Reversing.be folks(especially **haggar**), Arteam folks(**Shub-Nigurrath, MaDMAn_H3rCuL3s**) and all folks on crackmes.de, thank to all members of **unpack.cn** (especially **fly** and **linhanshi**). Great thanks to **lena151**(I like your tutorials). And finally, thanks to **RICARDO NARVAJA** and all members on **CRACKSLATINOS.**

>>>> If you have any suggestions, comments or corrections email me:
kienbigmummy[at]gmail.com