

2014

[Cracking with OllyDbg]

Based on OllyDbg tuts of Ricardo Narvaja (CrackLatinos Team)



www.reaonline.net

kienmanowar



27/10/2014

Mục Lục

I. Giới thiệu chung.....	2
II. Phân tích và xử lý target.....	2
1. Tổng quan	2
2. Thực hành	4
III. Kết luận	21

I. Giới thiệu chung

Trong phần 24, tôi đã cùng các bạn phân tích một crackme rất dị có tên là **Antisocial1.exe**, crackme này hội tụ lung tung các tricks Anti-Debug, và hi vọng nó không làm các bạn bức mình. Kết thúc phần 24 cũng đồng thời là chủ đề về Anti-Debug xin được dừng lại để dành đất cho các chủ đề khác. Kể từ phần 25 này và các phần tới đây (nếu tôi không bận 🤪) chúng ta sẽ dành thời gian để tìm hiểu về Unpacking.

'Packer' được xem như là một chương trình nén, được sử dụng để nén một file thực thi (executable file). Các chương trình này ra đời bắt nguồn từ mục đích muốn làm giảm kích thước của file, làm cho việc tải file nhanh hơn, tương tự các trình nén file như WinZip/Winrar. Tuy nhiên, bên cạnh việc nén, các trình packer cũng thường che dấu file gốc (original file) và gây nhiều khó khăn trong việc phân tích một file đã bị packed.

Rất nhiều các coder hay các hãng phần mềm sử dụng packer nhằm mục đích khiến các tay cracker/reverser phải khó khăn hơn và tốn thời gian hơn trong việc crack hoặc reverse phần mềm của họ. Đối với những kẻ chuyên phát tán các phần mềm độc hại (malicious software) thì còn một mục đích nữa là kéo dài thời gian tồn tại của phần

mềm càng lâu bị phát hiện càng tốt 🤪. Theo thời gian, các trình packers ngày càng trở nên phức tạp hơn, kéo theo đó là việc có rất nhiều thủ thuật nâng cao nhằm để bảo vệ chương trình. Phần 25 này tôi sẽ giới thiệu tới các bạn một số kiến thức cơ bản, là tiền đề tiếp nối cho các phần tiếp theo.

Now let's go..... 🤪

II. Phân tích và xử lý target

1. Tổng quan

Trong phần đầu tiên này, chúng ta sẽ xem xét một số khái niệm cơ bản cùng một số ý tưởng nhằm phục vụ trong quá trình làm việc với unpacking, để sau đó ở các phần tiếp theo chúng ta sẽ thực hành unpacking thông qua các ví dụ cụ thể và trực quan hơn.

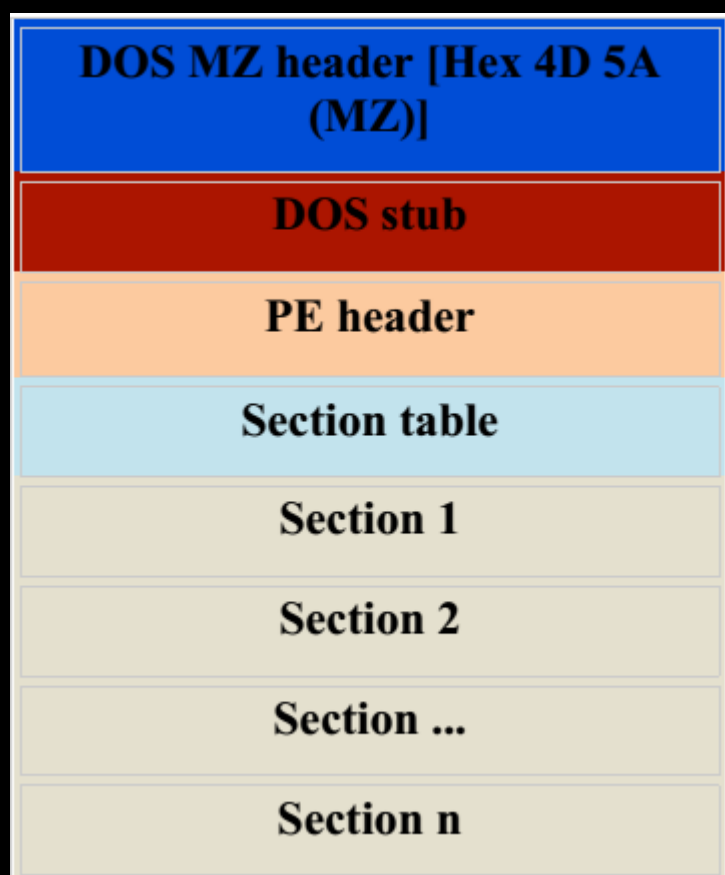
Trước tiên, ta tìm hiểu về ý tưởng về việc pack (nôm na tiếng Việt là "đóng gói") một chương trình là như thế nào? Về cơ bản chắc các bạn cũng đồng ý với tôi rằng, một chương trình không bị pack rất dễ dàng để chỉnh sửa/thay đổi code, đơn giản là vì ta có thể truy cập các bytes của chương trình ngay từ đầu, các bytes này cũng không thay đổi (khi chương trình đang trong quá trình thực thi), và bất cứ lúc nào ta cũng có thể sửa đổi một hoặc nhiều bytes bất kỳ và lưu các thay đổi đó mà không gặp khó khăn. Nếu

một chương trình có khả năng tự chỉnh sửa/thay đổi khi nó thực thi, thì lúc đó sẽ khó khăn hơn để ta có thể patch nó.

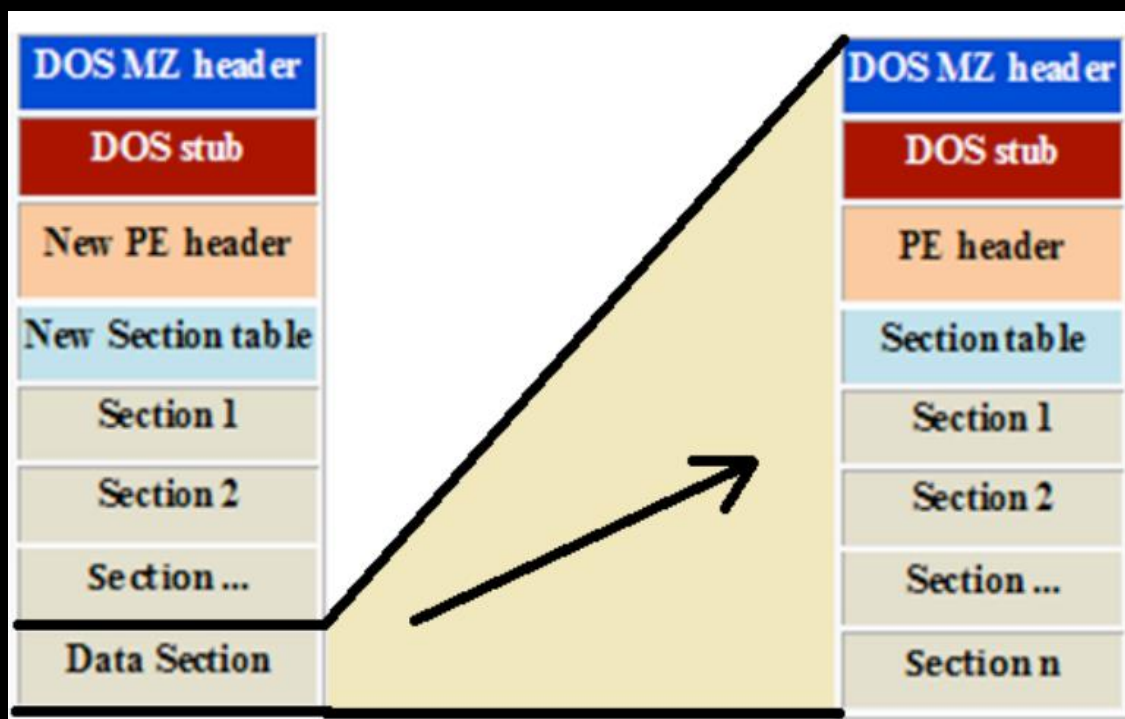
Do vậy, một chương trình khi bị packed, sẽ làm cho chúng ta không thấy được code gốc ban đầu. Các code quan trọng của chương trình gốc cũng không thể thay đổi được một cách dễ dàng như trước, vì chúng ta không tìm được nó, và nó đã bị nén (compressed), bị mã hóa (encrypted), hay nói cách khác là bị biến đổi, làm cho khó khăn hơn để có thể nhận biết hoặc RE.

Vấn đề ở đây là khi ta pack một ứng dụng bằng một trình packer cụ thể, packer sẽ nén, mã hóa và lưu hoặc dấu code gốc của chương trình, tự động bổ sung một hoặc nhiều sections, sau đó sẽ thêm đoạn code (hay còn được gọi là unpacking stub) và chuyển hướng (redirect) Entry Point (EP) tới vùng code này. *(Note: Bình thường một file Nonpacked sẽ được load bởi OS. Với file bị pack thì unpacking stub sẽ được load bởi OS, sau đó unpacking stub này sẽ load chương trình gốc. Lúc này code entry point của file thực thi sẽ trở tới unpacking stub thay vì trở vào original code. Chương trình ban đầu thường được lưu vào một hoặc nhiều sections (do packer tự thêm vào) của file).*

Layout cơ bản của một PE file khi chưa bị pack sẽ tương tự như hình minh họa dưới đây:



Khi file bị pack thì layout sẽ tương tự như hình minh họa dưới đây:



Kết quả là, sẽ không thấy được code của chương trình ban đầu. Khi ta load chương trình OllyDBG, sau quá trình phân tích OllyDBG sẽ dừng lại tại EP (Entry Point) mà Packer đã thực hiện redirect khi pack chương trình. Do vậy, khi tiến hành thực thi chương trình, thì từ EP đó unpacking stub sẽ tìm kiếm các thông tin được lưu của mã ban đầu đã bị mã hóa, thực hiện giải mã nó, sau khi quá trình giải mã kết thúc sẽ nhảy tới OEP hay Original Entry Point, đây chính là EP gốc của chương trình trước khi bị pack, hoặc chính là nơi dòng đầu tiên của mã ban đầu sẽ được thực thi.

2. Thực hành

Ta sẽ thực hành với một trong những trình packer đơn giản và phổ biến nhất, đó chính là UPX, phiên bản GUI được sử dụng trong bài viết này là GUIPeX. Các bạn có thể download tại link dưới đây:

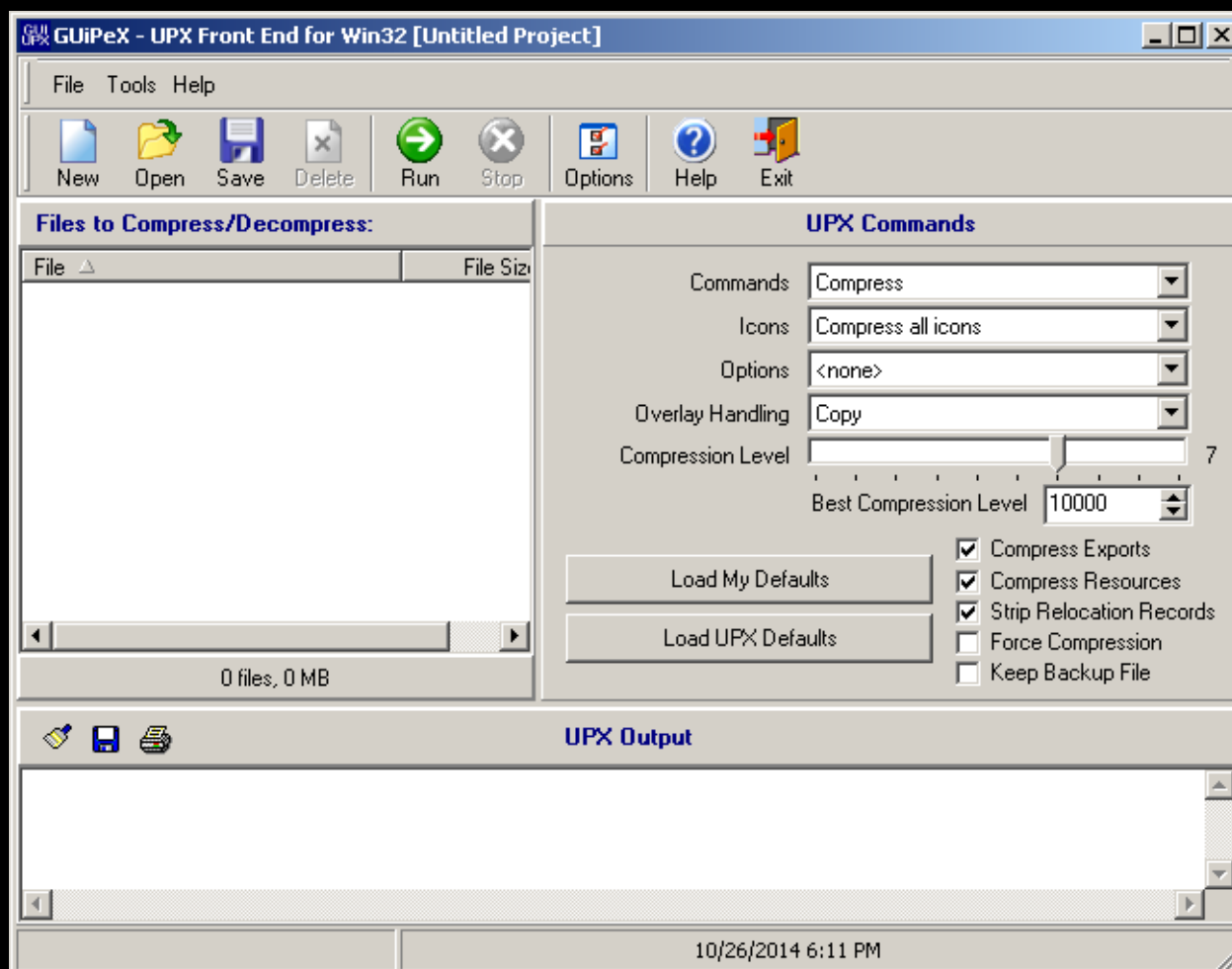
<http://www.freepcstuff.com/appinfo/GUIPeX/index.html>

GUIPeX is a name taken from GUI (Graphical User) and UPX.

UPX, a free DOS command-line , is a portable, extendable, high-performance executable packer for several different executable formats. It achieves an excellent compression ratio and offers very fast decompression. Your executables suffer no memory overhead or other drawbacks. Since UPX is a DOS , a simple GUI front-end for use in Windows was needed.

Download [Click Here](#)

Tải về và tiến hành cài đặt trên máy. Chương trình sau khi chạy có giao diện như sau:



Như các bạn thấy, chúng ta đã có một trình packer với giao diện khá đơn giản. Và để minh họa cho bài viết này, tôi vẫn tiếp tục sử dụng CRUEHEAD crackme (đã được sử dụng để làm ví dụ trong các bài viết trước). Đầu tiên, ta cần xem trước khi bị pack thì code gốc của crackme này sẽ như thế nào, tiến hành load crackme vào OllyDBG, ta dừng lại tại Entry Point của crackme:

The screenshot shows the Phantom debugger window with the following assembly code and variable values:

Address	Disassembly	Comment
00401000	6A 00	push 0x0
00401002	E8 FF040000	call <jmp.&KERNEL32.GetModuleHandleA>
00401007	A3 CA204000	mov dword ptr [0x4020CA], eax
0040100C	6A 00	push 0x0
0040100E	68 F4204000	push 004020F4
00401013	E8 A6040000	call <jmp.&USER32.FindWindowA>
00401018	0BC0	or eax, eax
0040101A	74 01	je short 0040101D
0040101C	C3	retn
0040101D	C705 64204000	mov dword ptr [0x402064], 0x4003
00401027	C705 68204000	mov dword ptr [0x402068], WndProc
00401031	C705 6C204000	mov dword ptr [0x40206C], 0x0
0040103B	C705 70204000	mov dword ptr [0x402070], 0x0
00401045	A1 CA204000	mov eax, dword ptr [0x4020CA]
0040104A	A3 74204000	mov dword ptr [0x402074], eax
0040104F	6A 64	push 0x64
00401051	50	push eax
00401052	E8 D1030000	call <jmp.&USER32.LoadIconA>

Variable values on the right:

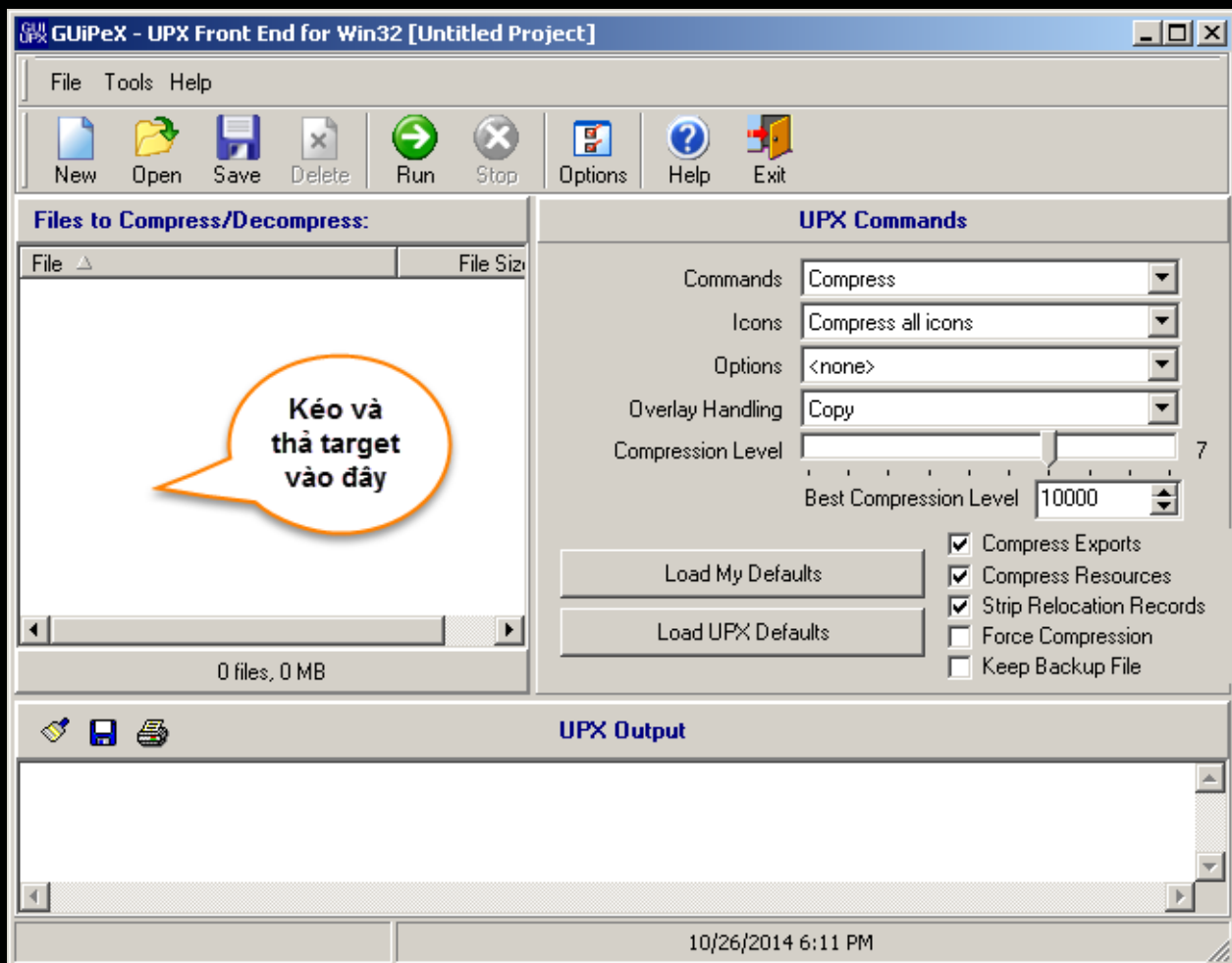
- pModule = NULL
- GetModuleHandleA
- Title = NULL
- Class = "No need to disasm the code!"
- FindWindowA
- RsrcName = 100.
- hInst => NULL
- LoadIconA

Như các bạn thấy trong hình, EP của crackme là 401000, điều này có nghĩa là nếu ta thực thi crackme này thì đây sẽ là dòng code đầu tiên được thực hiện. Vậy, nếu ta tiến hành pack crackme này bằng GUIPeX thì nó sẽ thêm và thay đổi các sections, mã hóa các code ban đầu, sau đó thay đổi địa chỉ EP trở đến vùng unpacking stub.

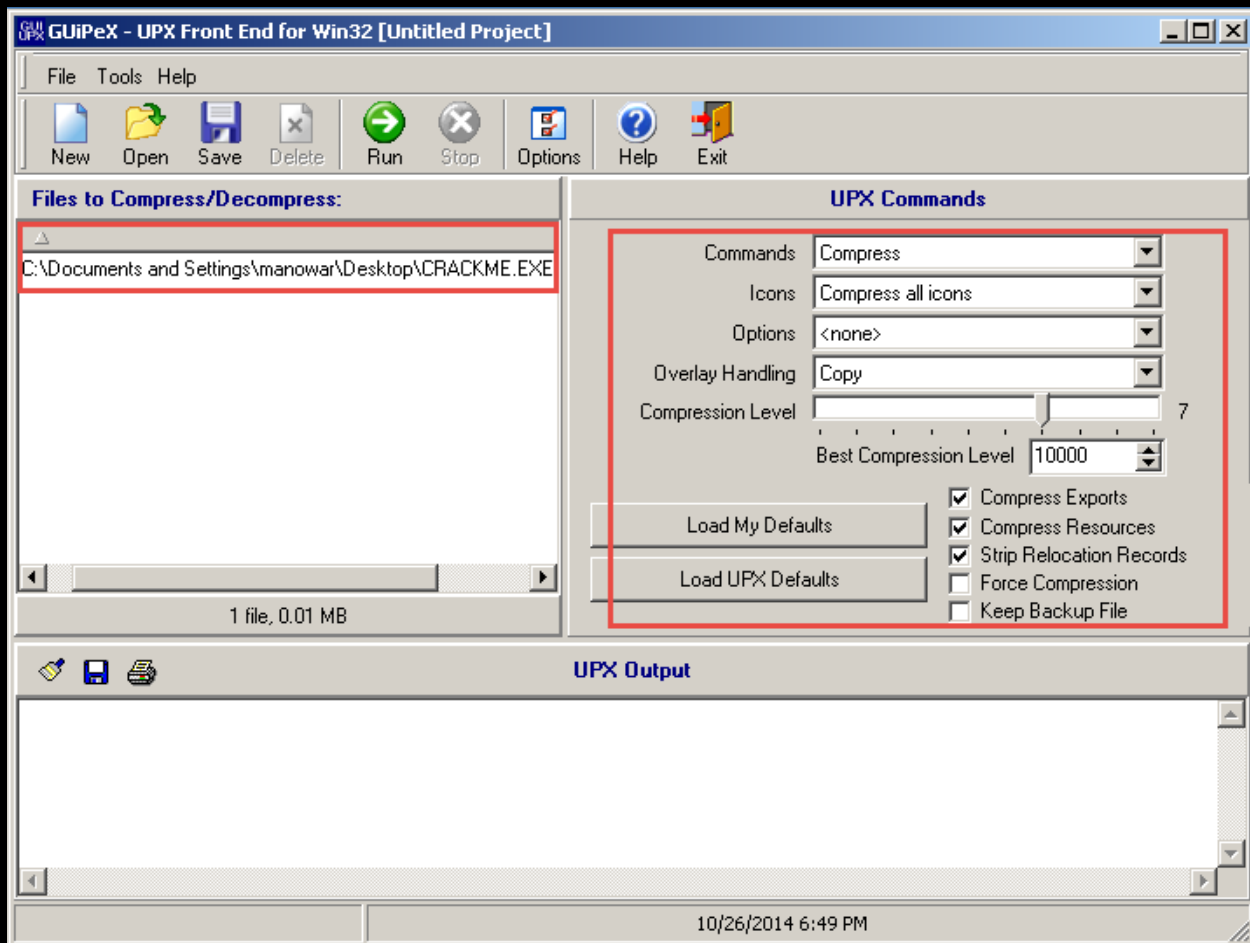
Khi ta cho thực thi crackme, unpacking stub sẽ được thực thi để tìm ra mã ban đầu bị mã hóa, giải mã nó. Sau khi giải mã xong sẽ nhảy tới dòng đầu tiên của code ban đầu, là nơi mà chương trình gốc thực sự bắt đầu thực thi, đó chính là **OEP** hay **Original Entry Point** (một thuật ngữ mà các bạn sẽ gặp trong hàng loạt các tut trên net), trong trường hợp cụ thể này của chúng ta, đó chính là địa chỉ 401000.


Thông thường, khi chúng ta thực hiện việc unpack một chương trình bị pack, chúng ta sẽ không có được file gốc ban đầu (file trước khi pack) để so sánh và tìm ra đâu là OEP của chương trình. Chính vì điều này chúng ta phải tìm hiểu, nghiên cứu các kỹ thuật khác nhau để làm sao có thể tìm thấy OEP. Trước khi đi vào các phần cụ thể hơn, đầu tiên chúng ta thực hành với CC (Cruehead Crackme, viết tắt cho nó ngắn 😊) đã. Đầu tiên, lưu một bản sao của CC ở một vị trí an toàn trước khi tiến hành chỉnh sửa và đây cũng được xem là file gốc để chúng ta so sánh với file bị pack.

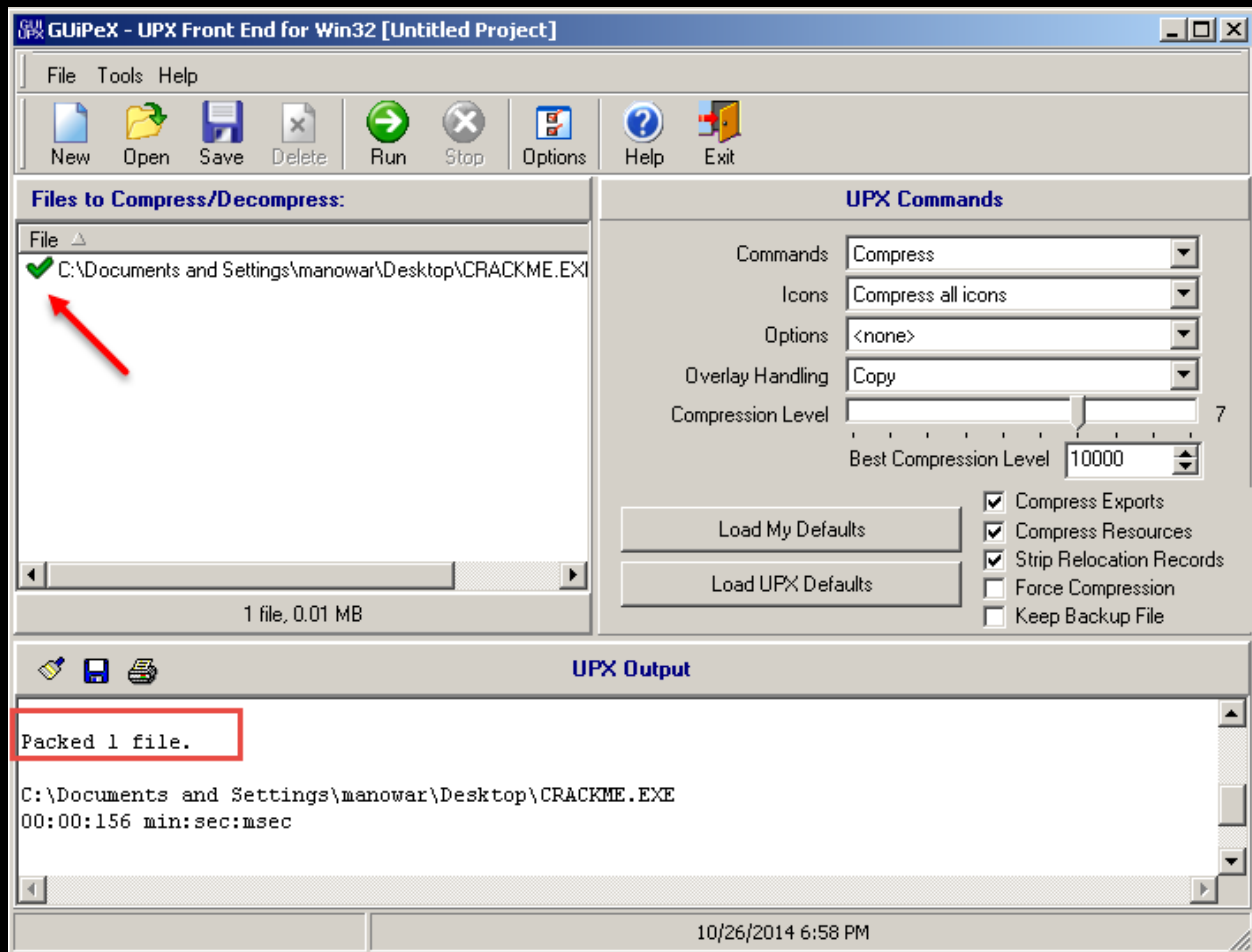
Các bạn vẫn còn mở GUIPeX chứ, nếu chưa thì chạy lại nhé:



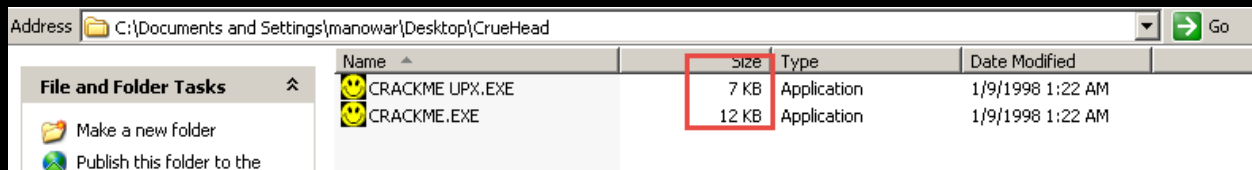
Kéo và thả crackme vào vị trí như trên hình minh họa:



Sau khi kéo thả crackme vào, ta thấy đường dẫn tới Crackme xuất hiện tại **"File to Compress/Decompress"**. Kế bên là **"UPX Commands"**, nơi ta lựa chọn các thiết lập cho việc pack file. Tiến hành cấu hình như hình minh họa, sau khi cấu hình xong nhấn nút Run () để GUIPeX thực hiện việc pack file:



Kết quả sau khi pack thành công sẽ tương tự như hình minh họa ở trên. Để đỡ bị nhầm lẫn với file gốc, ta đổi tên của file sau khi pack thành **"CRACKME UPX.EXE"**:



Như chúng ta quan sát trên hình, file crackme sau khi pack có kích thước nhỏ hơn so với file gốc (thông thường là như vậy), tuy nhiên hiện nay các trình packer/protector còn thêm cả các code để bảo vệ chương trình, khiến cho file sau khi pack lại có kích thước lớn hơn so với file gốc. Nếu chúng ta cho thực thi file bị pack, ta thấy nó cũng thực thi bình thường như ta thực thi file gốc:



Bây giờ chúng ta hãy phân tích một chút, ta load cả hai crackme vào OllyDbg để so sánh xem sự khác biệt trước và sau pack là như thế nào nhé:

ENTRY POINT OF CRACKME.exe:

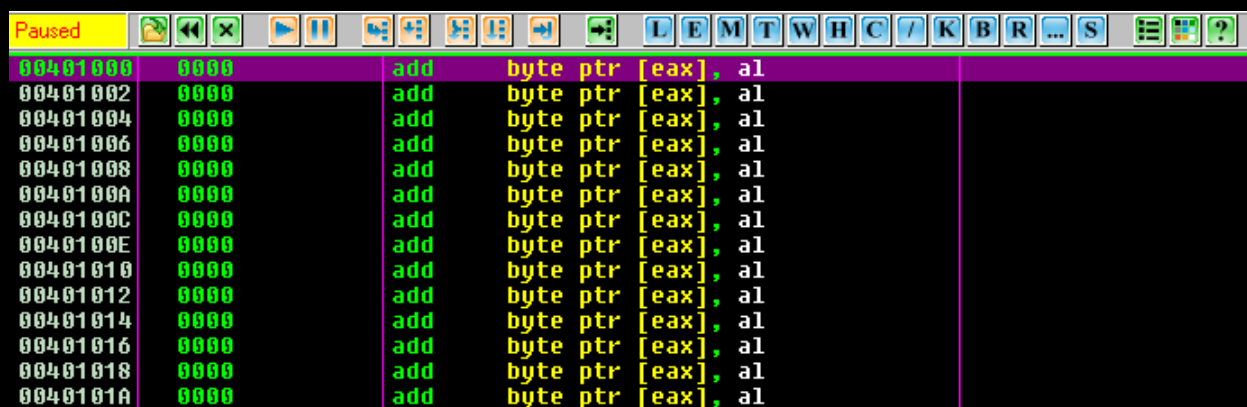
Address	Disassembly	Comment
00401000	6A 00	push 0x0
00401002	E8 FF040000	call <jmp.&KERNEL32.GetModuleHandleA>
00401007	A3 CA204000	mov dword ptr [0x4020CA], eax
0040100C	6A 00	push 0x0
0040100E	68 F4204000	push 004020F4
00401013	E8 A6040000	call <jmp.&USER32.FindWindowA>
00401018	0BC0	or eax, eax
0040101A	74 01	je short 0040101D
0040101C	C3	ret

ENTRY POINT OF CRACKME UPX.exe

Address	Disassembly	Comment
00409BF0	6A 00	pushad
00409BF1	BE 00904000	mov esi, 00409000
00409BF6	8DBE 0080FFFF	lea edi, dword ptr [esi+0xFFFF8000]
00409BFC	57	push edi
00409BFD	83CD FF	or ebp, 0xFFFFFFFF
00409C00	EB 10	jmp short 00409C12
00409C02	90	nop
00409C03	90	nop
00409C04	90	nop
00409C05	90	nop
00409C06	90	nop
00409C07	90	nop

Theo kết quả so sánh có được trên hình minh họa, ta thấy crackme sau khi được pack bằng UPX thì entry point đã thay đổi thành 409BF0 (nơi unpacking stub sẽ thực thi), nhấn **Ctrl+G** và nhập 401000 để quan sát entry point ban đầu, ta thấy không còn

thông tin về các mã gốc ban đầu của crackme 🤖 :



Address	Disassembly
00401000	add byte ptr [eax], al
00401002	add byte ptr [eax], al
00401004	add byte ptr [eax], al
00401006	add byte ptr [eax], al
00401008	add byte ptr [eax], al
0040100A	add byte ptr [eax], al
0040100C	add byte ptr [eax], al
0040100E	add byte ptr [eax], al
00401010	add byte ptr [eax], al
00401012	add byte ptr [eax], al
00401014	add byte ptr [eax], al
00401016	add byte ptr [eax], al
00401018	add byte ptr [eax], al
0040101A	add byte ptr [eax], al

Như trên hình, chúng ta thấy code tại EP gốc đã bị thay đổi thành một loạt các bytes `0x00`, như vậy có thể thấy rằng packer sau khi mã hóa đã thực hiện loại bỏ toàn bộ các byte và lưu giữ ở đâu đó trong code. Nhìn chung, hầu hết các trình packer thường tạo ra một section riêng biệt và thực thi code từ đó để lấy ra những byte được mã hóa của mã ban đầu, thực hiện fix và giải mã lại code về code gốc ban đầu.

Quay trở lại màn hình OllyDbg khi ta load file bị pack, quan sát đoạn code bên dưới tính từ EP của file bị pack.

Paused			
00409BF0	\$ 60	pushad	
00409BF1	> BE 00904000	mov esi, 00409000	
00409BF6	. 8DBE 0080FFFF	lea edi, dword ptr [esi+0xFFFF8000]	
00409BFC	. 57	push edi	
00409BFD	. 83CD FF	or ebp, 0xFFFFFFFF	
00409C00	~ EB 10	jmp short 00409C12	
00409C02	90	nop	
00409C03	90	nop	
00409C04	90	nop	
00409C05	90	nop	
00409C06	90	nop	
00409C07	90	nop	
00409C08	> 8A06	mov al, byte ptr [esi]	
00409C0A	. 46	inc esi	
00409C0B	. 8807	mov byte ptr [edi], al	
00409C0D	. 47	inc edi	
00409C0E	> 01DB	add ebx, ebx	
00409C10	~ 75 07	jnz short 00409C19	
00409C12	> 8B1E	mov ebx, dword ptr [esi]	
00409C14	. 83EE FC	sub esi, -0x4	
00409C17	. 11DB	adc ebx, ebx	
00409C19	> 72 ED	jb short 00409C08	
00409C1B	. B8 01000000	mov eax, 0x1	
00409C20	> 01DB	add ebx, ebx	
00409C22	~ 75 07	jnz short 00409C2B	
00409C24	. 8B1E	mov ebx, dword ptr [esi]	
00409C26	. 83EE FC	sub esi, -0x4	
00409C29	. 11DB	adc ebx, ebx	
00409C2B	> 11C0	adc eax, eax	
00409C2D	. 01DB	add ebx, ebx	
00409C2F	^ 73 EF	jnb short 00409C20	
00409C31	~ 75 09	jnz short 00409C3C	
00409C33	. 8B1E	mov ebx, dword ptr [esi]	
00409C35	. 83EE FC	sub esi, -0x4	
00409C38	. 11DB	adc ebx, ebx	
00409C3A	^ 73 E4	jnb short 00409C20	

Tiếp tục kéo xuống và quan sát cho đến khi ta thấy được một lệnh nhảy tới OEP:

```

00409CE8 . 80EB E8      sub     bl, 0xE8
00409CEB . 01F0         add     eax, esi
00409CED . 8907         mov     dword ptr [edi], eax
00409CEF . 83C7 05      add     edi, 0x5
00409CF2 . 89D8         mov     eax, ebx
00409CF4 . ^ E2 D9      loopd   short 00409CCF
00409CF6 . 8DBE 0070000 lea     edi, dword ptr [esi+0x7000]
00409CFC > 8B07         mov     eax, dword ptr [edi]
00409CFE . 09C0         or      eax, eax
00409D00 . 74 3C       je      short 00409D3E
00409D02 . 8B5F 04      mov     ebx, dword ptr [edi+0x4]
00409D05 . 8D8430 E0940 lea     eax, dword ptr [eax+esi+0x94E0]
00409D0C . 01F3         add     ebx, esi
00409D0E . 50          push    eax
00409D0F . 83C7 08      add     edi, 0x8
00409D12 . FF96 5895000 call   dword ptr [esi+0x9558]
00409D18 . 95          xchg    eax, ebp
00409D19 > 8A07         mov     al, byte ptr [edi]
00409D1B . 47          inc     edi
00409D1C . 08C0         or      al, al
00409D1E . ^ 74 DC       je      short 00409CFC
00409D20 . 89F9         mov     ecx, edi
00409D22 . 57          push    edi
00409D23 . 48          dec     eax
00409D24 . F2:AE       repne   scas byte ptr es:[edi]
00409D26 . 55          push    ebp
00409D27 . FF96 5C95000 call   dword ptr [esi+0x955C]
00409D2D . 09C0         or      eax, eax
00409D2F . ^ 74 07       je      short 00409D38
00409D31 . 8903         mov     dword ptr [ebx], eax
00409D33 . 83C3 04      add     ebx, 0x4
00409D36 . ^ EB E1      jmp     short 00409D19
00409D38 > FF96 6095000 call   dword ptr [esi+0x9560]
00409D3E > 61          popad
00409D3F . ^ E9 BC72FFFF jmp     00401000

```

Có thể thấy rằng UPX là một Packer đơn giản, chúng ta thấy nó sẽ thực quá trình giải mã, sau khi thực hiện xong sẽ sử dụng một lệnh nhảy để nhảy tới OEP, lệnh nhảy này không hề bị che dấu. Các trình packer sau này sẽ không dễ dàng như thế này.

Ta tiến hành đặt một BP tại lệnh nhảy tới OEP:

```

00409D2F . ^ 74 07       je      short 00409D38
00409D31 . 8903         mov     dword ptr [ebx], eax
00409D33 . 83C3 04      add     ebx, 0x4
00409D36 . ^ EB E1      jmp     short 00409D19
00409D38 > FF96 6095000 call   dword ptr [esi+0x9560]
00409D3E > 61          popad
00409D3F . ^ E9 BC72FFFF jmp     00401000
00409D44 . 00          db      00
00409D45 . 00          db      00
00409D46 . 00          db      00
00409D47 . 00          db      00

```

Đặt BP xong nhấn **F9** để Run, ta sẽ break tại lệnh nhảy này:

```

00409D36 . ^ EB E1      jmp     short 00409D19
00409D38 > FF96 6095000 call   dword ptr [esi+0x9560]
00409D3E > 61          popad
00409D3F . ^ E9 BC72FFFF jmp     00401000
00409D44 . 00          db      00
00409D45 . 00          db      00
00409D46 . 00          db      00

```

Để tới được OEP, ta nhấn **F7**:

```

00401000 6A 00      push     0x0
00401002 E8 FF040000 call     00401506
00401007 A3 C8204000 mov     dword ptr [0x4020C8], eax
0040100C 6A 00      push     0x0
0040100E 68 F4204000 push    004020F4
00401013 E8 A6040000 call     004014BE
00401018 0BC0      or       eax, eax
0040101A 74 01      je       short 0040101D
0040101C C3        retn
0040101D C705 64204000 mov     dword ptr [0x402064], 0x4003
00401027 C705 68204000 mov     dword ptr [0x402068], 00401128
  
```

Sau khi tới được OEP, ta thấy rằng code tại đây giống hệt với code của file crackme gốc. Như vậy, rõ ràng là unpacking stub đã hoàn thành công việc của mình và đã giải mã toàn bộ mã gốc ban đầu. Qua các thông tin ở trên, ta có thể đưa ra được các bước cơ bản khi thực thi một chương trình bị pack như sau:

- 1) Thực thi unpacking stub.
- 2) Giải mã và chỉnh sửa để khôi phục lại Original Code.
- 3) Nhảy tới OEP.
- 4) Thực thi chương trình một cách bình thường.

Các bước nêu trên đã được áp dụng trong nhiều năm và rất nhiều packers đã hoạt động rất tốt, tuy nhiên về sau này các packers có áp dụng thêm các thủ thuật để ẩn (che dấu) OEP, và các thủ thuật này sẽ được đề cập trong các bài viết sau này. OK, quay trở lại với file crackme đã bị pack, restart lại OllyDbg. Ta suy nghĩ một chút, địa chỉ EP gốc của crackme thuộc section **CODE**, nếu section này sau khi pack chứa toàn các byte 0x00 như trong trường hợp crackme của chúng ta hoặc trong trường hợp nó chứa toàn mã rác chẳng hạn thì rõ ràng phải có đoạn code thực hiện để xây dựng lại code gốc tại đó. Vậy để biết đoạn code nào tác động lên section này ta sẽ đặt một **BPM ON ACCESS** lên nó. Nhấn **M** để mở cửa sổ **Memory Map**:

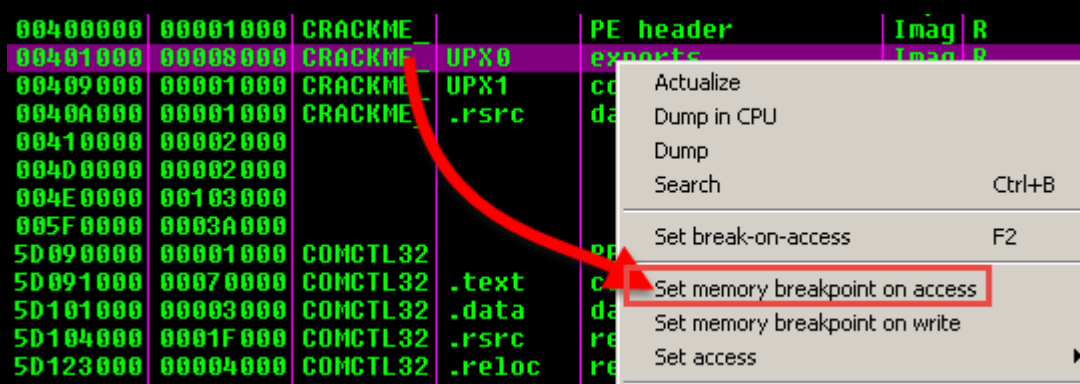
003B0000	00002000				Map	R	R
00400000	00001000	CRACKME		PE header	Imag	R	RWE
00401000	00008000	CRACKME	UPX0	exports	Imag	R	RWE
00409000	00001000	CRACKME	UPX1	code	Imag	R	RWE
0040A000	00001000	CRACKME	.rsrc	data,imports,resou	Imag	R	RWE
00410000	00002000				Map	R E	R E
00410000	00002000				Map	R E	R E

Ta có thể nhận thấy, các sections đã được thay đổi tên cũng như size so với các sections của bản gốc 🤔:

00400000	00001000	CRACKME		PE header	Image	R	RWE
00401000	00001000	CRACKME	CODE	code	Image	R	RWE
00402000	00001000	CRACKME	DATA	data	Image	R	RWE
00403000	00001000	CRACKME	.idata	imports	Image	R	RWE
00404000	00001000	CRACKME	.edata	exports	Image	R	RWE
00405000	00001000	CRACKME	.reloc	relocations	Image	R	RWE
00406000	00002000	CRACKME	.rsrc	resources	Image	R	RWE
00410000	00002000				Map	R E	R E

Section **CODE** của file gốc bắt đầu tại 401000 và có kích thước là 1000 bytes, trong khi đó tại file bị pack bằng UPX, thì section **CODE** đã bị thay đổi và nó bắt đầu tại 409000.

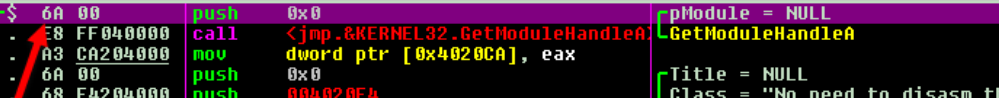
OK ta tiến hành đặt **BPM ON ACCESS** tại section đầu tiên của file bị pack. Sau khi đặt BP xong nhấn **F9** để thực thi.



Ta break tại đoạn code sau:

[illegible]

Quan sát đoạn code ta thấy, giá trị của thanh ghi `AL = 6A` được lưu vào vùng nhớ `401000`. So sánh với code tại EP của crackme không bị pack ta có được như sau:



The screenshot shows OllyDb with the CPU window displaying assembly code and the Disasm window showing the corresponding C++ code. A red arrow points to the instruction at address 00401008, which is a `call` instruction. The assembly code is as follows:

```
00401000 6A 00      push     0x0
00401002 E8 FF040000 call     <jmp.&KERNEL32.GetModuleHandleA>
00401007 A3 CA204000 mov     dword ptr [0x4020CA], eax
0040100C 6A 00      push     0x0
0040100E 68 F4204000 push    004020F4
00401013 E8 A6040000 call     <jmp.&USER32.FindWindowA>
00401018 0BC0      or      eax, eax
0040101A 74 01      je      short 0040101D
0040101C C3        retn
0040101D > C705 64204000 mov     dword ptr [0x402064], 0x4003
00401027 C705 68204000 mov     dword ptr [0x402068], WndProc
```

The C++ code in the Disasm window is as follows:

```
pModule = NULL;
GetModuleHandleA(0);
Title = NULL;
Class = "No need to disasm the code!";
FindWindowA(0, Class, 0, 0);
```

Như vậy ta thấy, byte đầu tiên tại 401000 của file gốc đang chứa giá trị 0x6A. Giờ ta nhấn **F9** một lần nữa và hi vọng byte được lưu tiếp theo sẽ là 0x00, tương tự như ta thấy ở code gốc:

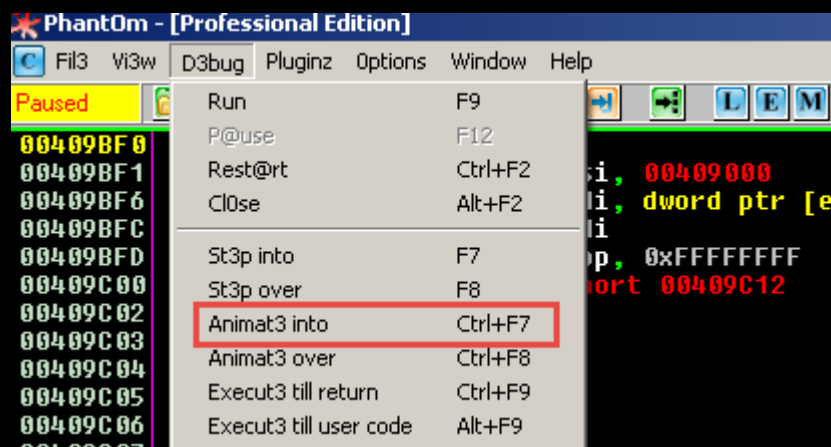
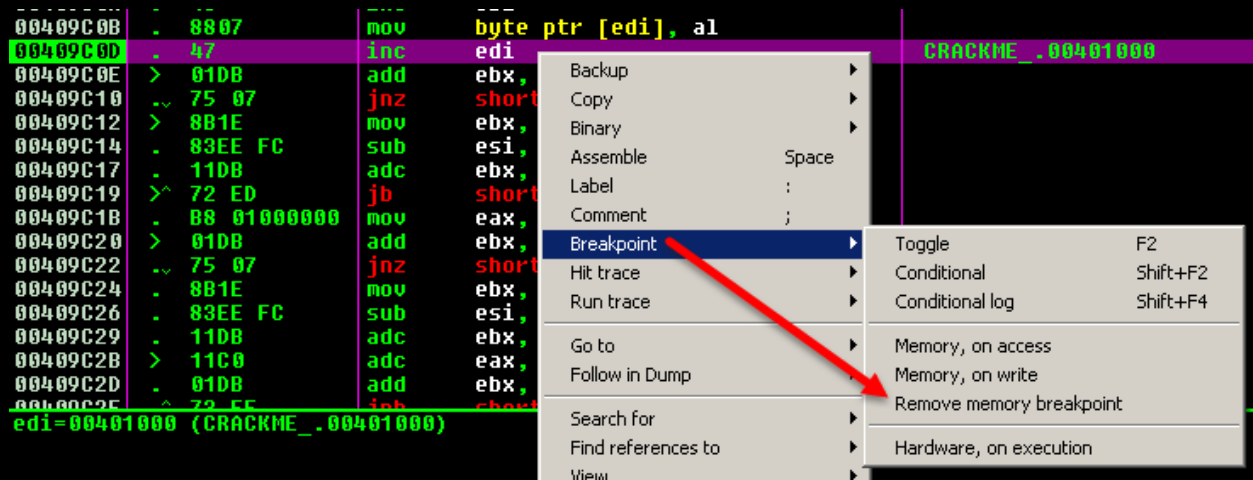
Address	Disassembly
00409C06	90 nop
00409C07	90 nop
00409C08	> 8A06 mov al, byte ptr [esi]
00409C0A	. 46 inc esi
00409C0B	. 8807 mov byte ptr [edi], al
00409C0D	. 47 inc edi
00409C0E	> 01DB add ebx, ebx
00409C10	.. 75 07 jnz short 00409C19
00409C12	> 8B1E mov ebx, dword ptr [esi]
00409C14	. 83EE FC sub esi, -0x4
00409C17	. 11DB adc ebx, ebx
00409C19	> ^ 72 ED jb short 00409C08
00409C1B	. B8 01000000 mov eax, 0x1
00409C20	> 01DB add ebx, ebx
00409C22	.. 75 07 jnz short 00409C2B
00409C24	. 8B1E mov ebx, dword ptr [esi]
00409C26	. 83EE FC sub esi, -0x4
00409C29	. 11DB adc ebx, ebx
00409C2B	> 11C0 adc eax, eax
00409C2D	. 01DB add ebx, ebx
00409C2F	.. 73 EF jnb short 00409C20
00409C31	.. 75 09 jnz short 00409C3C
00409C33	. 8B1E mov ebx, dword ptr [esi]
00409C35	. 83EE FC sub esi, -0x4
00409C38	. 11DB adc ebx, ebx
00409C3A	.. ^ 73 E4 jnb short 00409C20
00409C3C	> 31C9 xor ecx, ecx
00409C3E	. 83E8 03 sub eax, 0x3
00409C41	.. 72 0D jb short 00409C50
00409C43	. C1E0 08 shl eax, 0x8
00409C46	.. 8A06 mov al, byte ptr [esi]

al=00
ds:[00401001]=00

Address	Bytes	Comment
00401000	6A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	j.....
00401010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401020	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Cứ tiếp tục như thế từ từ từng byte một, nó sẽ thực hiện việc lưu byte đã được giải mã và dựng lại code ban đầu, tất nhiên không phải mọi packers đều thực hiện như vậy ngay từ đầu. Nếu như ta thử trace code thì chúng ta sẽ bắt gặp một vòng lặp thực hiện đọc 1 byte từ vùng mã hóa, thực hiện các phép tính toán để giải mã cho tới khi có được giá trị gốc.

Nếu ta bỏ **BPM ON ACCESS**, sau đó nhấn **Ctrl+F7 (Animate Into)**, đợi và quan sát quá trình điền thông tin giá trị gốc vào section đầu tiên, quá trình này sẽ thực hiện liên tục cho tới khi nó dừng lại tại BPX mà ta đã đặt tại lệnh **JMP OEP** trước đó:



Quan sát quá trình thực hiện giải mã section **CODE**:

```

00409C08 > 8A06 mov al, byte ptr [esi]
00409C0A . 46 inc esi
00409C0B . 8807 mov byte ptr [edi], al
00409C0D . 47 inc edi
00409C0E > 01DB add ebx, ebx
00409C10 .. 75 07 jnz short 00409C19
00409C12 > 8B1E mov ebx, dword ptr [esi]
00409C14 . 83EE FC sub esi, -0x4
00409C17 . 11DB adc ebx, ebx
00409C19 > 72 ED jbe short 00409C08
00409C1B . B8 01000000 mov eax, 0x1
00409C20 > 01DB add ebx, ebx
00409C22 .. 75 07 jnz short 00409C2B
00409C24 . 8B1E mov ebx, dword ptr [esi]
00409C26 . 83EE FC sub esi, -0x4
00409C29 . 11DB adc ebx, ebx
00409C2B > 11C0 adc eax, eax
00409C2D . 01DB add ebx, ebx
00409C2F . 73 EF jnb short 00409C20
00409C31 .. 75 09 jnz short 00409C3C
00409C33 . 8B1E mov ebx, dword ptr [esi]
00409C35 . 83EE FC sub esi, -0x4
00409C38 . 11DB adc ebx, ebx
00409C3A . 73 E4 jnb short 00409C20
00409C3C > 31C9 xor ecx, ecx
00409C3E . 83E8 03 sub eax, 0x3
00409C41 .. 72 0D jbe short 00409C50
00409C43 . C1E0 08 shl eax, 0x8
00409C46 . 8A06 mov al, byte ptr [esi]
00409C48 . 46 inc esi
00409C4A . 82F0 FF xor eax, 0xFFFF

```

Jump is taken
00409C19=00409C19

00401000	6A 00 E8 00	00 05 02 A3	CA 20 40 00	6A 00 68 F4	j.è...EE @.j.hô
00401010	20 40 00 E8	00 00 04 BA	0B C0 74 01	C3 C7 05 64	@.è...AtAÇld
00401020	20 40 00 03	40 00 00 C7	05 68 20 40	00 28 11 40	@.a...Çh @.(a
00401030	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00401040	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00401050	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

Toàn bộ quá trình trên cứ dịch chuyển và thực hiện code theo một vòng lặp liên tục


(chạy xong đoạn này cũng toát mồ hôi 🤔), và cứ thế section đầu tiên của chúng ta sẽ được điền đầy đủ các giá trị gốc, cho tới khi kết thúc và dừng lại tại lệnh nhảy JMP tới OEP:

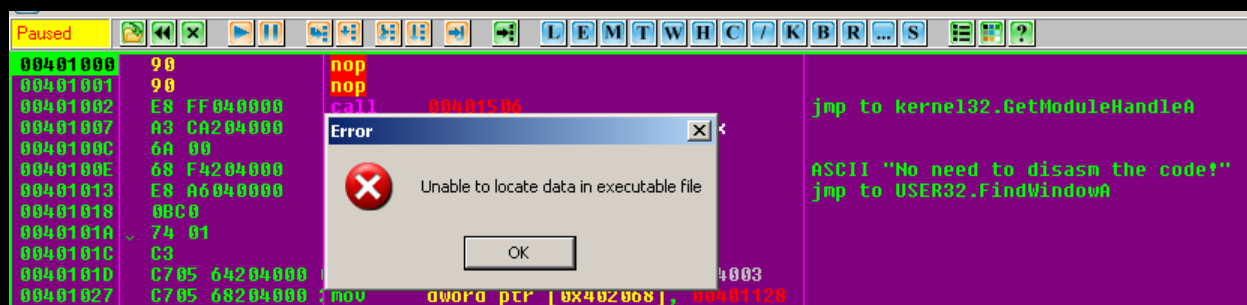
00409D3F	E9 BC72FFFF	jmp	00401000
00409D44	00	db	00
00409D45	00	db	00
00409D46	00	db	00
00409D47	00	db	00
00409D48	00	db	00
00409D49	00	db	00
00401000=00401000			
00401000	6A 00 E8 FF 04 00 00 A3 CA 20 40 00 6A 00 68 F4	j.èÿ...ÈE @.j.hô	
00401010	20 40 00 E8 A6 04 00 00 0B C0 74 01 C3 C7 05 64	@.è!...ÁtAÇnd	
00401020	20 40 00 03 40 00 00 C7 05 68 20 40 00 28 11 40	@.M@..Çh @.(M@	
00401030	00 C7 05 6C 20 40 00 00 00 00 C7 05 70 20 40	.ÇMl @....Çp @	
00401040	00 00 00 00 00 A1 CA 20 40 00 A3 74 20 40 00 6AË @.Et @.j	
00401050	64 50 E8 D1 03 00 00 A3 78 20 40 00 68 00 7F 00	dPèN...Èx @.h..	
00401060	00 6A 00 E8 A2 03 00 00 A3 7C 20 40 00 C7 05 80	.j.èÇ...È @.ÇM	
00401070	20 40 00 05 00 00 00 C7 05 84 20 40 00 10 21 40	@.M...ÇM @.M!@	
00401080	00 C7 05 88 20 40 00 F4 20 40 00 68 64 20 40 00	.ÇM @.ô @.hd @.	
00401090	FA FA A3 AA AA AA AA FF 35 CA 20 40 AA AA AA AA	Pñ...i..i5F @.i.i	

Khi ta dừng lại ở lệnh nhảy, nhấn **F7** để thực hiện lệnh và ta sẽ tới OEP:

00401000	6A 00	push	0x0	
00401002	E8 FF040000	call	00401506	jmp to kernel32.GetModuleHandleA
00401007	A3 CA204000	mov	dword ptr [0x4020CA], eax	
0040100C	6A 00	push	0x0	
0040100E	68 F4204000	push	004020F4	ASCII "No need to disasm the code!"
00401013	E8 A6040000	call	004014BE	jmp to USER32.FindWindowA
00401018	0BC0	or	eax, eax	
0040101A	74 01	je	short 0040101D	
0040101C	C3	ret		
0040101D	C705 64204000	mov	dword ptr [0x402064], 0x4003	
00401027	C705 68204000	mov	dword ptr [0x402068], 00401128	
00401031	C705 6C204000	mov	dword ptr [0x40206C], 0x0	
0040103B	C705 70204000	mov	dword ptr [0x402070], 0x0	
00401045	A1 CA204000	mov	eax, dword ptr [0x4020CA]	
0040104A	A3 74204000	mov	dword ptr [0x402074], eax	
0040104F	6A 64	push	0x64	
00401051	50	push	eax	
00401052	E8 D1030000	call	00401428	jmp to USER32.LoadIconA
00401057	A3 78204000	mov	dword ptr [0x402078], eax	
0040105C	68 007F0000	push	0x7F00	
00401061	6A 00	push	0x0	
00401063	E8 A2030000	call	0040140A	jmp to USER32.LoadCursorA
00401068	A3 7C204000	mov	dword ptr [0x40207C], eax	

OK ta đang dừng tại OEP, theo như phân tích từ đầu bài viết, chúng ta nói rằng

không thể thay đổi mã với OllyDbg và lưu lại thay đổi như chúng ta thường làm . Giờ ví dụ nếu tôi lấy 2 bytes đầu tiên là 6A 00 và muốn thay đổi chúng, ví dụ sửa thành 90 90. Tôi thực hiện thay đổi và cố gắng để lưu lại thay đổi đã thực hiện, nhưng OllyDbg xuất hiện thông báo sau:



Do vậy, tôi không thể thực hiện được, bởi OllyDbg không thể tìm được đoạn mã thực thi, để thay đổi nó.

OK, ta kết thúc bài viết tại đây với một cái nhìn tổng quan ban đầu, trong phần tiếp theo chúng ta sẽ tiếp tục công việc unpacking với crackme CC.

III. Kết luận

Toàn bộ bài 25 đến đây là kết thúc, cảm ơn các bạn đã dành thời gian để theo dõi. Hẹn gặp lại ở phần tiếp theo!

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, dqtn, ARTEAM ... all my friend, and YOU.

--++--==[Thanks To]==--++--

iamidiot, WhyNotBar, trickyboy, dzungltn, 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 Rogers** 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](mailto:kienbigmummy[at]gmail.com)