

INTRODUCTION TO THE CRACKING WITH OLLYDBG

FROM CRACKLATINOS

([_kienmanowar_](#))



Một cái đầu lạnh để vững vàng, một trái tim đỏ lửa để yêu và làm việc hết mình!

I. Giới thiệu chung

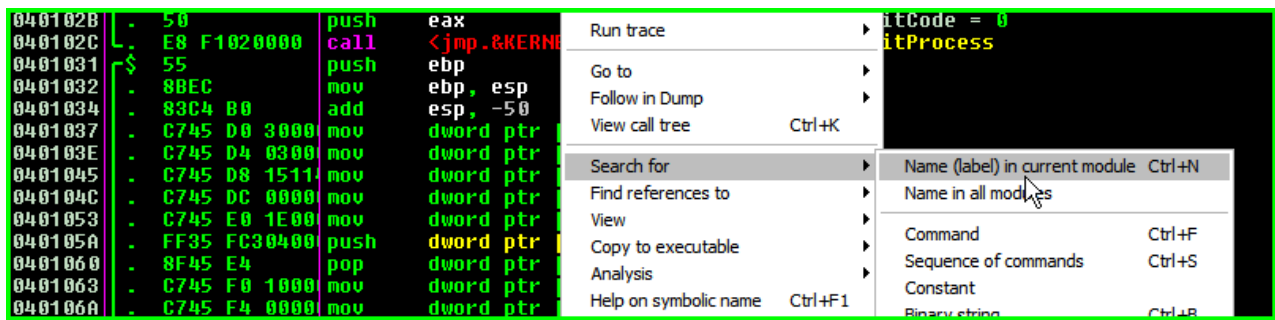
Hi vọng trong mấy ngày vừa qua các bạn đã đọc và thực hành xong với những crackme trong phần 14, hôm nay chúng ta sẽ tiếp tục với phần 15. Trong phần 14 tôi đã cùng các bạn làm việc với dạng Hardcoded Serial đơn giản, phần 15 này ta vẫn sẽ tiếp tục nhưng có khó hơn một chút. Lượn qua site của lão Ricard để down tutor về (các tutor của lão tôi đã down cả và lưu trong CD rồi nhưng ngại mở đĩa ☺), extract tutor và target ra thì lại đòi pass, sặc chẳng hiểu sao lão lại đặt pass và tôi cũng chẳng biết pass để extract là gì ☺. Hehe google thử phát để tìm kiếm thông tin, hóa ra pass để unrar là serial ta tìm được trong cái Detten's crackme. Chà ý tưởng của lão Rincar cũng hay phết, lão muốn người đọc phải giải quyết được target của tutor này mới được xem bài viết ở tutor tiếp theo. Ok nói chuyện vui thế đủ rồi, chúng ta đi vào phần chính luôn nhé ☺. Now....L3t's GO !!!!

II. Fishing Serial ☺

Trước tiên chúng ta lướt qua cách giải quyết Detten's crackme cái đã, vì có giải quyết được crackme này tôi và các bạn mới extract được tutor 15 để đọc. Ta load crackme vào trong Olly :

00401000	\$ 6A 00	push	0	pModule = NULL
00401002	. E8 27030000	call	<jmp.&KERNEL32.GetModuleHandleA>	GetModuleHandleA
00401007	. A3 FC304000	mov	dword ptr [4030FC], eax	
0040100C	. E8 17030000	call	<jmp.&KERNEL32.GetCommandLineA>	GetCommandLineA
00401011	. A3 00314000	mov	dword ptr [403100], eax	
00401016	. 6A 0A	push	0A	Arg4 = 0000000A
00401018	. FF35 00314000	push	dword ptr [403100]	Arg3 = 00000000
0040101E	. 6A 00	push	0	Arg2 = 00000000
00401020	. FF35 FC304000	push	dword ptr [4030FC]	Arg1 = 00000000
00401026	. E8 06000000	call	00401031	crackme.00401031
0040102B	. 50	push	eax	ExitCode = 0
0040102C	. E8 F1020000	call	<jmp.&KERNEL32.ExitProcess>	ExitProcess
00401031	\$ 55	push	ebp	
00401032	. 8BEC	mov	ebp, esp	
00401034	. 83C4 B0	add	esp, -B0	

Ta đang dừng lại tại EP của crackme, tiến hành tìm kiếm thông tin về các hàm APIs được sử dụng trong crackme này :



Kết quả ta có được như sau :

Address	Section	Type	Name	Comment
00402048	.rdata	Import	USER32.CreateWindowExA	
00402028	.rdata	Import	USER32.DefWindowProcA	
00402040	.rdata	Import	USER32.DispatchMessageA	
00402008	.rdata	Import	KERNEL32.ExitProcess	
0040200C	.rdata	Import	KERNEL32.GetCommandLineA	
0040203C	.rdata	Import	USER32.GetMessageA	
00402004	.rdata	Import	KERNEL32.GetModuleHandleA	
00402024	.rdata	Import	USER32.GetWindowTextA	
00402020	.rdata	Import	USER32.LoadCursorA	
00402014	.rdata	Import	USER32.LoadIconA	
00402000	.rdata	Import	KERNEL32.lstrcmpA	
00402018	.rdata	Import	USER32.MessageBoxA	
00401000	.text	Export	<ModuleEntryPoint>	
0040201C	.rdata	Import	USER32.PostQuitMessage	
00402044	.rdata	Import	USER32.RegisterClassExA	
0040204C	.rdata	Import	USER32.SetFocus	
0040202C	.rdata	Import	USER32.SetWindowTextA	
00402030	.rdata	Import	USER32.ShowWindow	
00402034	.rdata	Import	USER32.TranslateMessage	
00402038	.rdata	Import	USER32.UpdateWindow	

Các bạn thấy có khá nhiều hàm API, tuy nhiên để ý sẽ thấy có 3 hàm API quan trọng (các hàm mà tôi đã đánh dấu) có thể chọn làm đầu mối để tiếp cận mục tiêu. Trong 3 hàm này thì hàm **MessageBoxA** các bạn đã quá quen thuộc rồi, chỉ còn hai hàm **GetWindowTextA** và **lstrcmpA**. Ta sẽ lần lượt xem qua ý nghĩa của hai hàm này :

The **GetWindowText** function copies the text of the specified window's title bar (if it has one) into a buffer. If the specified window is a control, the text of the control is copied.

```

int GetWindowText(
    HWND hWnd,           // handle of window or control with text
    LPTSTR lpString,     // address of buffer for text
    int nMaxCount        // maximum number of characters to copy
);

```

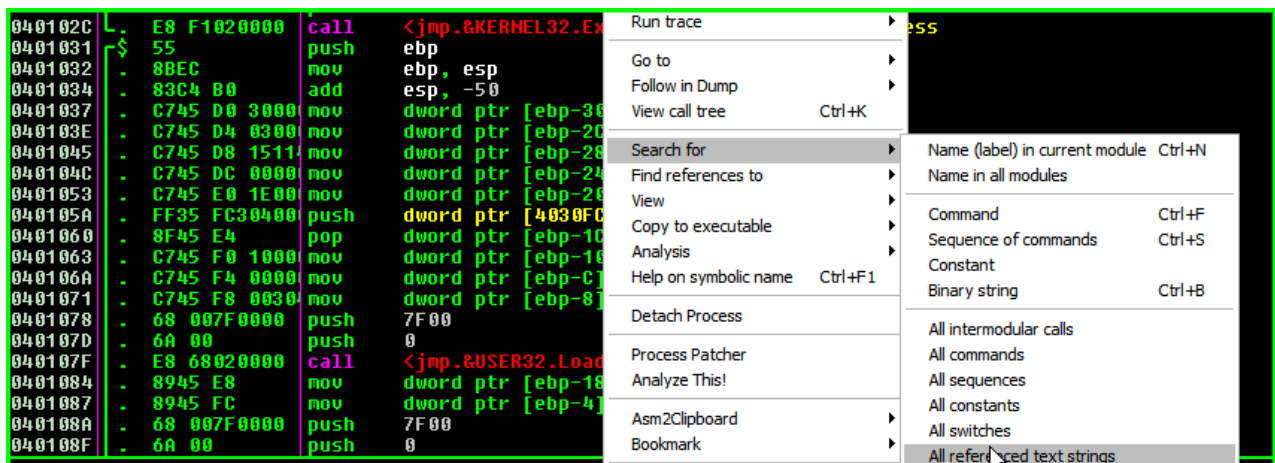
The **lstrcmp** function compares two character strings. The comparison is case sensitive.

```

int lstrcmp(
    LPCTSTR lpString1,    // address of first string
    LPCTSTR lpString2     // address of second string
);

```

Đọc các thông tin cung cấp ở trên các bạn đã phần nào hiểu được mục đích sử dụng và ý nghĩa của từng hàm API rồi chứ. Như vậy các bạn có thể đặt BP tại các hàm API này để giải quyết bài toán. Giờ chúng ta tìm kiếm thử các String quan trọng :



Kết quả ta có được như sau :

Address	Disassembly	Text string
00401000	push 0	(Initial CPU selection)
00401071	mov dword ptr [ebp-8], 00403000	ASCII "Crackme"
0040108C	push 0CF0000	ASCII "Actx "
004010C1	push 00403008	ASCII "Crackme 1.0"
004010C6	push 00403000	ASCII "Crackme"
00401153	push 00403014	ASCII "edit"
0040118F	push 00403020	ASCII "Check"
00401194	push 00403019	ASCII "button"
004011C5	push 004030C2	ASCII "Info"
004011CA	push 004030BB	ASCII "button"
004011F8	push 0040302D	ASCII "Password:"
004011FD	push 00403026	ASCII "static"
0040123E	push 00403055	ASCII "cannabis"
00401253	push 0040307E	ASCII "Correct!"
00401258	push 0040305E	ASCII "You entered the right password!"
00401269	push 00403087	ASCII "Nope!"
0040126E	push 0040308D	ASCII "Maybe, you should try again, it's sooo easy!?"
00401298	push 004030F7	ASCII "Info"
0040129D	push 004030C7	ASCII "Coded in win32asm by Miele. Greetz to Detten!! "

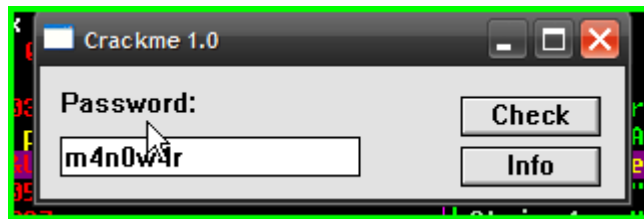
Các String quan trọng show ra rõ như ban ngày ☺. Ta nhấn đúp chuột tại ASCII "You entered the right password!", Olly sẽ đưa ta tới đây :

0040122A	75 5C	jnz short 00401288	Count = 1E (30.)
0040122C	6A 1E	push 1E	Buffer = crackme.00403037
0040122E	68 37304000	push 00403037	hWnd = NULL
00401233	FF35 04314000	push dword ptr [403104]	GetWindowTextA
00401239	E8 A2000000	call <jmp.&USER32.GetWindowTextA>	String2 = "cannabis"
0040123E	68 55304000	push 00403055	String1 = ""
00401243	68 37304000	push 00403037	IstrcmpA
00401248	E8 E7000000	call <jmp.&KERNEL32.IstrcmpA>	
0040124D	0BC0	or eax, eax	
0040124F	75 16	jnz short 00401267	
00401251	6A 00	push 0	Style = MB_OK MB_APPLMODAL
00401253	68 7E304000	push 0040307E	Title = "Correct!"
00401258	68 5E304000	push 0040305E	Text = "You entered the right password!"
0040125D	FF75 08	push dword ptr [ebp+8]	hOwner = 00401000
00401260	E8 8D000000	call <jmp.&USER32.MessageBoxA>	MessageBoxA
00401265	EB 21	jmp short 00401288	
00401267	6A 00	push 0	Style = MB_OK MB_APPLMODAL
00401269	68 87304000	push 00403087	Title = "Nope!"
0040126E	68 8D304000	push 0040308D	Text = "Maybe, you should try again, it's
00401273	FF75 08	push dword ptr [ebp+8]	hOwner = 00401000
00401276	E8 77000000	call <jmp.&USER32.MessageBoxA>	MessageBoxA

Nhìn tổng thể toàn bộ đoạn code trên ta có thể đưa ra những nhận xét cơ bản như sau :

1. Chương trình sử dụng hàm API **GetWindowTextA** để copy chuỗi Serial mà ta nhập vào và lưu vào vùng Buffer.
2. Sau đó chuỗi Fake serial này sẽ được đem đi so sánh với chuỗi Hardcoded serial thông qua hàm API **IstrcmpA**.
3. Phụ thuộc vào kết quả so sánh mà quyết định hiển thị Good boy hay Bad boy.

Để kiểm chứng những gì chúng ta vừa nhận xét ở trên, tôi đặt bp tại **GetWindowTextA**, sau đó nhấn **F9** để run chương trình. Tiến hành nhập Fake Serial :



Sau khi nhập xong nhấn Check, Olly sẽ ice tại BP mà ta đã đặt. Lúc này quan sát cửa sổ Stack ta có thông tin :

0013FC78	000A051C	hWnd = 000A051C (class='Edit',parent=0019054A)
0013FC7C	00403037	Buffer = crackme.00403037
0013FC80	0000001E	Count = 1E (30.)
0013FC84	0013FC80	

Ta chuẩn bị thực hiện hàm **GetWindowTextA**, vùng Buffer sẽ là nơi lưu Serial mà ta nhập vào. Ta có được như sau :

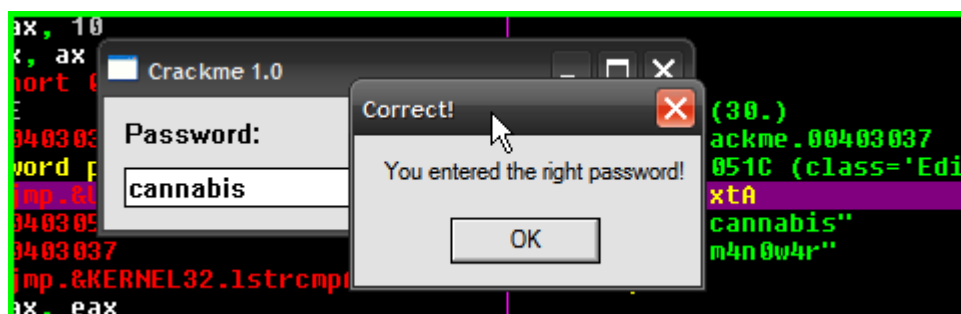
00403037	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
00403047	00 00 00 00	00 00 00 00	00 00 00 00	00 00 63 61ca
00403057	6E 6E 61 62	69 73 00 59	6F 75 20 65	6E 74 65 72	nnabis.You enter
00403067	65 64 20 74	68 65 20 72	69 67 68 74	20 70 61 73	ed the right pas
00403077	73 77 6F 72	64 21 00 43	6F 72 72 65	63 74 21 00	sword!.Correct!.

00403037	6D 34 6E 30	77 34 72 00	00 00 00 00	00 00 00 00	m4n0w4r.....
00403047	00 00 00 00	00 00 00 00	00 00 00 00	00 00 63 61ca
00403057	6E 6E 61 62	69 73 00 59	6F 75 20 65	6E 74 65 72	nnabis.You enter
00403067	65 64 20 74	68 65 20 72	69 67 68 74	20 70 61 73	ed the right pas
00403077	73 77 6F 72	64 21 00 43	6F 72 72 65	63 74 21 00	sword!.Correct!.

Sau khi thực hiện hàm **GetWindowTextA**, ta dừng lại tại đây :

00401239	. E8 A2000000	call <jmp.&USER32.GetWindowTextA>	GetWindowTextA
0040123E	. 68 55304000	push 00403055	String2 = "cannabis"
00401243	. 68 37304000	push 00403037	String1 = "m4n0w4r"
00401248	. E8 E7000000	call <jmp.&KERNEL32.lstrcmpA>	lstrcmpA
0040124D	. 0BC0	or eax, eax	
0040124F	. 75 16	jnz short 00401267	

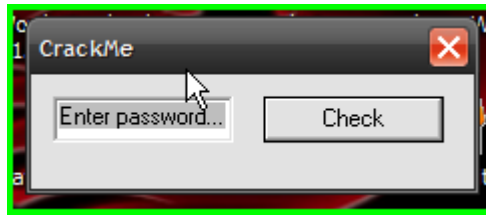
Khả khả vậy không còn nghi ngờ gì nữa, tới đây ta có thể hoàn toàn kết luận rằng chuỗi Hardcoded Serial của crackme này là **"cannabis"**. Ta kiểm tra thử xem thế nào nhé :



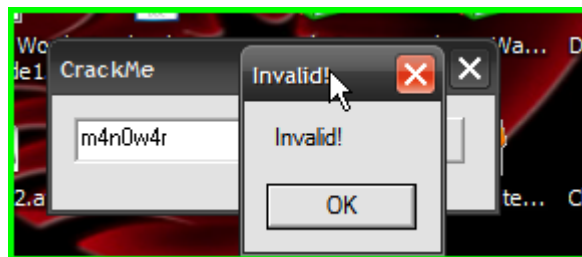
Vậy là xong!! Tiếp theo chúng ta sẽ giải quyết một crackme khác..

LESSON 15 HARDCODED 1: crackmeeasy.exe

Không giống như các crackme trước, crackme này sẽ không thực hiện việc so sánh trực tiếp Fake Serial với Real Serial mà sẽ thực hiện một số bước tính toán trước khi tiến hành thao tác so sánh. Trước khi load vào Olly, ta chạy thử xem mặt mũi cái crackme này thế nào :



Ta thấy có mỗi cái Textbox cho phép nhập password và một button dùng để kiểm tra xem pass nhập vào là đúng hay sai. Ok, ta nhập đại một password vào và nhấn nút Check :



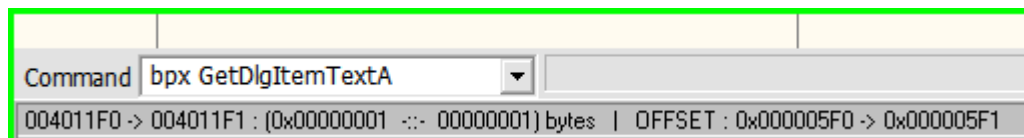
Qua các thao tác trên chúng ta đã thu thập được một số thông tin cần thiết cho việc tiếp cận để giải quyết crackme này. Tiếp theo ta phải “**đi câu**” để tóm được valid Serial ☺. Load cái crackme vào trong Olly :

004011F0	55	push	ebp	
004011F1	89E5	mov	ebp, esp	
004011F3	83EC 08	sub	esp, 8	
004011F6	83C4 F4	add	esp, -0C	
004011F9	6A 02	push	2	
004011FB	A1 24314000	mov	eax, dword ptr [<msvcrt.__set_app_type>]	<msvcrt.__set_app_type>
00401200	FFD0	call	eax	
00401202	E8 79FFFFFF	call	00401180	
00401207	C9	leave		
00401208	C3	retn		
00401209	00	db	00	
0040120A	00	db	00	
0040120B	00	db	00	
0040120C	00	db	00	
0040120D	00	db	00	
0040120E	00	db	00	
0040120F	00	db	00	
00401210	45 6E 74 65	ascii	"Enter password.."	
00401220	2E 00	ascii	".",0	
00401222	31 30 34 34	ascii	"10445678951",0	
0040122E	43 6F 72 72	ascii	"Correct!",0	
00401237	49 6E 76 61	ascii	"Invalid!",0	

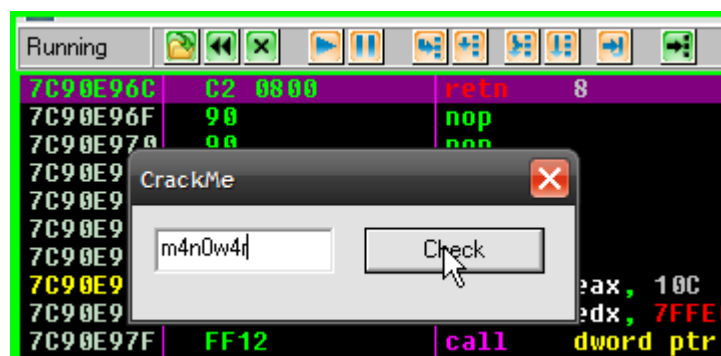
Tiến hành tìm kiếm thông tin về các hàm APIs được sử dụng trong crackme này :

Address	Section	Type	Name
00403110	.idata	Import	msvcrt.atexit
004030F4	.idata	Import	msvcrt._cexit
00403130	.idata	Import	USER32.DialogBoxParamA
00403134	.idata	Import	USER32.EndDialog
004030D4	.idata	Import	KERNEL32.ExitProcess
004030F8	.idata	Import	msvcrt._fileno
004030FC	.idata	Import	msvcrt._fmode
00403100	.idata	Import	msvcrt._fpreset
004030D8	.idata	Import	KERNEL32.GetCommandLineA
00403138	.idata	Import	USER32.GetDlgItem
0040313C	.idata	Import	USER32.GetDlgItemTextA
0040310C	.idata	Import	msvcrt.__getmainargs
004030DC	.idata	Import	KERNEL32.GetModuleHandleA
004030E0	.idata	Import	KERNEL32.GetStartupInfoA
00403140	.idata	Import	USER32.GetWindowTextLengthA
004030E4	.idata	Import	KERNEL32.GlobalAlloc
00403104	.idata	Import	msvcrt._iob
00403118	.idata	Import	msvcrt.memset
00403144	.idata	Import	USER32.MessageBoxA
004011F0	.text	Export	<ModuleEntryPoint>
00403114	.idata	Import	msvcrt._p__environ
00403148	.idata	Import	USER32.SetDlgItemTextA
00403108	.idata	Import	msvcrt._setmode
004030E8	.idata	Import	KERNEL32.SetUnhandledExceptionFilter
00403124	.idata	Import	msvcrt._set_app_type
0040311C	.idata	Import	msvcrt.signal
00403120	.idata	Import	msvcrt.strlen

Quan sát thấy có hàm **GetDlgItemTextA**, chúng ta sẽ dùng hàm này để tiếp cận mục tiêu. Ta đặt BP tại hàm này như sau :



Nhấn **F9** để run chương trình, nhập Fake Serial và nhấn Check :



Olly sẽ dừng lại tại BP, quan sát cửa sổ Stack ta có thông tin như sau :

0240F99C	00090462	hWnd = 00090462 ('CrackMe',class='#32770')
0240F9A0	00000191	ControlID = 191 (401.)
0240F9A4	0006B8F8	Buffer = 0006B8F8
0240F9A8	00000008	Count = 8
0240F9AC	2601097A	
0240F9B0	0240FA3C	
0240F9B4	7E432632	RETURN to USER32.7E432632 from USER32.DrawStateW
0240F9B8	2601097A	

Chú ý vùng **Buffer**, đây sẽ là nơi lưu Fake serial mà ta nhập vào. **Follow in Dump** tại vùng Buffer ta có được kết quả trước khi thực hiện hàm như sau :

00401650	< jmp .&USER32.GetDlgItemTextA >	
0006B8F8	00 00 00 00	00 00 00 00 06 00 02 00 2B 23 14 00
0006B908	1C 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
0006B918	00 00 00 00	00 00 00 00 00 00 00 00 80 00 80 00
0006B928	00 00 00 00	2C 00 82 02 03 00 06 00 2D 23 14 00
0006B938	00 00 00 00	78 01 06 00 00 00 00 00 3C 00 82 02
0006B948	F8 00 03 00	22 23 14 00 AC 07 00 00 00 00 00 00

Ok, ta nhấn **F8** để thực hiện hàm **GetDlgItemTextA** và quan sát vùng Buffer. Kết quả sau khi thực hiện sẽ có được như sau :

00401650	< jmp .&USER32.GetDlgItemTextA >	
Stack ss:[0240FA04]=0240000F		
0006B8F8	6D 34 6E 30	77 34 72 00 06 00 02 00 2B 23 14 00
0006B908	1C 00 00 00	00 00 00 00 00 00 00 00 00 00 00 00
0006B918	00 00 00 00	00 00 00 00 00 00 00 00 80 00 80 00
0006B928	00 00 00 00	2C 00 82 02 03 00 06 00 2D 23 14 00

Như các bạn thấy lúc này vùng **Buffer** đã chứa chuỗi Serial mà chúng ta vừa nhập vào. Ta quay trở lại cửa sổ Code, lúc này ta đang dừng lại tại đây :

004012FE	- E8 4D030000	call < jmp.&USER32.GetDlgItemTextA >	GetDlgItemTextA
00401303	- C745 F0 0000	mov dword ptr [ebp-10], 0	ASCII "10445678951"
0040130A	- 8B 22124000	mov eax, 00401222	
0040130F	- 8B10	mov edx, dword ptr [eax]	
00401311	- 8955 D0	mov dword ptr [ebp-30], edx	
00401314	- 8B50 04	mov edx, dword ptr [eax+4]	
00401317	- 8955 D4	mov dword ptr [ebp-2C], edx	
0040131A	- 8B40 08	mov eax, dword ptr [eax+8]	
0040131D	- 8945 D8	mov dword ptr [ebp-28], eax	
00401320	- 8D45 DC	lea eax, dword ptr [ebp-24]	
00401323	- 83C4 FC	add esp, -4	
00401326	- 6A 08	push 8	
00401328	- 6A 00	push 0	
0040132A	- 50	push eax	
0040132B	- E8 F0020000	call < jmp.&msvcrt.memset >	n = 8
00401330	- 83C4 10	add esp, 10	c = 00
00401333	- C745 CC 0000	mov dword ptr [ebp-34], 0	s = 00000007
0040133A	- 8D06 00000000	lea esi, dword ptr [esi]	memset
00401340	> 83C4 F4	add esp, -0C	
00401343	- 8D45 D0	lea eax, dword ptr [ebp-30]	
00401346	- 50	push eax	
00401347	- E8 DC020000	call < jmp.&msvcrt.strlen >	s = 00000007 ???
0040134C	- 83C4 10	add esp, 10	strlen
0040134F	- 89C0	mov ecx, eax	
00401351	- 8D50 FF	lea edx, dword ptr [eax-1]	
00401354	- 3955 F0	cmp dword ptr [ebp-10], edx	
00401357	- 72 07	jb short 00401360	
00401359	- EB 35	jmp short 00401390	
0040135B	- 90	nop	
0040135C	- 8D7426 00	lea esi, dword ptr [esi]	
00401360	> 8B45 F4	mov eax, dword ptr [ebp-C]	

Chà như các bạn thấy trên hình, chúng ta thấy rằng có một chuỗi mặc định gồm toàn chữ số tham gia vào quá trình tính toán. Không biết nó có phải là Hardcoded serial không nhỉ? Để khẳng định thì các bạn có thể nhập thử chuỗi đó vào và kiểm tra, còn tôi thì đã thử rồi khá khá ☺. Các bạn để ý sẽ thấy giá trị **0x00401222** được đẩy vào thanh ghi eax, mà địa chỉ này lại trỏ tới chuỗi mặc định là **"10445678951"**, vậy tức là thanh ghi eax lúc này sẽ trỏ tới chuỗi mặc định :

```
Registers (FPU)
EAX 00401222 ASCII "10445678951"
ECX 7E43218C USER32.7E43218C
EDX 00060608
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C
```

Follow in Dump tại thanh ghi EAX :

```
00401303 . 8B55 F0 mov     eax, dword ptr [ebp-10]
00401222=00401222 (ASCII "10445678951")
eax=00401222 (crakmeea.00401222), ASCII "10445678951"

00401222 31 30 34 34 35 36 37 38 39 35 31 00 43 6F 72 72 10445678951.Corr
00401232 65 63 74 21 00 49 6E 76 61 6C 69 64 21 00 55 89 ect?.Invalid?.U
00401242 E5 83 EC 68 8B 45 0C 83 F8 10 0F 84 C3 01 00 00 3iñhE.0+00r..
00401252 83 F8 10 77 0E 83 F8 02 0F 84 B5 01 00 00 E9 C3 0+0000r..éA
00401262 01 00 00 3D 10 01 00 00 74 0C 3D 11 01 00 00 74 r..+=r..t.=<r..t
00401272 2D E9 B0 01 00 00 83 C4 FC 68 10 12 40 00 68 91 -é°r..Äü+!@.h'
00401282 01 00 00 8B 45 08 50 E8 AA 03 00 00 83 C4 04 B8 r..E0Pèal..ÄJ
00401292 01 00 00 00 F9 A5 01 00 00 F0 90 01 00 00 0F B7 r..é¥...é...¥.
```

Ở dòng lệnh tiếp theo ta sẽ thấy giá trị tại EAX được chuyển vào EDX :

```
0040130F . 8B10 mov     edx, dword ptr [eax]
```

mà cụ thể ở đây tức là :

```
0040130F . 8B10 mov     edx, dword ptr [401222]
```

```
0040130A . B8 22124000 mov     eax, 00401222 ASCII "10445678951"
0040130F . 8B10 mov     edx, dword ptr [eax]
00401311 . 8955 D0 mov     dword ptr [ebp-30], edx
00401314 . 8B50 04 mov     edx, dword ptr [eax+4]
00401317 . 8955 D4 mov     dword ptr [ebp-2C], edx
0040131A . 8B40 08 mov     eax, dword ptr [eax+8]
```

Vậy kết luận là nội dung tại 0x401222 sẽ được chuyển vào thanh ghi EDX, cụ thể ở đây là Dword tức là 4 bytes đầu tiên tại vùng nhớ 0x401222. Quan sát tại cửa sổ Tip ta sẽ thấy được thông tin cụ thể như sau :

```
00401368 . 0FBE10 movsx   edx, byte ptr [eax]
ds:[00401222]=34343031
edx=00060608
```

Ok phân tích xong ta nhấn **F8** để thực hiện lệnh và quan sát thanh ghi EDX để kiểm chứng kết quả:


```

Registers (FPU)
EAX 00401222 ASCII "10445678951"
ECX 7E43218C USER32.7E43218C
EDX 34343031
EBX 00000000
ESP 0240F9AC
EBP 0240FA14
ESI 00401240 crakmeea.00401240
EDI 0240FA7C

EIP 00401311 crakmeea.00401311

C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL

```

Dòng lệnh tiếp theo sẽ lại chuyển giá trị của EDX vào vùng nhớ [EBP-30] :

```

Paused
0040130A . B8 22124000 mov     eax, 00401222      ASCII "10445678951"
0040130F . 8B10          mov     edx, dword ptr [eax]
00401311 . 8955 D0      mov     dword ptr [ebp-30], edx
00401314 . 8B50 04      mov     edx, dword ptr [eax+4]
00401317 . 8955 D4      mov     dword ptr [ebp-2C], edx
0040131A . 8B40 08      mov     eax, dword ptr [eax+8]

```

Quan sát trên cửa sổ Tip ta sẽ biết được [EBP-30] là vùng nhớ nào và giá trị của nó tại cửa sổ Dump đang là bao nhiêu :

```

edx=34343031
Stack ss:[0240F9E4]=0006038C

```

0240F9E4	8C 03 06 00	00 02 00 00	03 00 00 00	03 00 00 00	W...L...L...
0240F9F4	57 00 00 00	14 00 00 00	11 00 00 00	8C B3 06 00	W...L...L...
0240FA04	00 00 00 00	F8 B8 06 00	06 00 00 00	07 00 00 00	...-...-...+
0240FA14	40 FA 40 02	34 87 41 7E	46 04 12 00	11 01 00 00	@u@_4A~FJ↑.↓r..
0240FA24	01 00 00 00	56 04 0F 00	40 12 40 00	CD AB BA DC	r...UJ%.@I@.I<<@j
0240FA34	00 00 00 00	7C FA 40 02	40 12 40 00	AC FA 40 02 u@_@I@.~u@_

Nhấn **F8** để thực thi lệnh và quan sát cửa sổ Dump :

0240F9E4	31 30 34 34	00 02 00 00	03 00 00 00	03 00 00 00	1044...L...L...
0240F9F4	57 00 00 00	14 00 00 00	11 00 00 00	8C B3 06 00	W...L...L...
0240FA04	00 00 00 00	F8 B8 06 00	06 00 00 00	07 00 00 00	...-...-...+
0240FA14	40 FA 40 02	34 87 41 7E	46 04 12 00	11 01 00 00	@u@_4A~FJ↑.↓r..
0240FA24	01 00 00 00	56 04 0F 00	40 12 40 00	CD AB BA DC	r...UJ%.@I@.I<<@j

Thực hiện xong ta dừng tại lệnh :

```
00401314 . 8B50 04      mov     edx, dword ptr [eax+4]
```

Lệnh này sẽ lấy 4 bytes tiếp theo tại 0x401222 và đẩy vào thanh ghi EDX :

00401368	.	0FBE10	movsx	edx, byte ptr [eax]	
ds:[00401226]=38373635 eax=34343031					
00401222	31 30 34 34	35 36 37 38	39 35 31 00	43 6F 72 72	10445678951.Corr
00401232	65 63 74 21	00 49 6E 76	61 6C 69 64	21 00 55 89	ect?.Invalid?.U

Thực thi lệnh và quan sát thanh ghi EDX :

Registers (FPU)	
EAX	00401222 ASCII "10445678951"
ECX	7E43218C USER32.7E43218C
EDX	38373635
EBX	00000000
ESP	0240F9AC
EBP	0240F914
ESI	00401240 crakmeea.00401240
EDI	0240FA7C

Tiếp theo giá trị của EDX được chuyển vào vùng nhớ [EBP-2C] :

0240F9E4	31 30 34 34	35 36 37 38	03 00 00 00	03 00 00 00	10445678951.L...
0240F9F4	57 00 00 00	14 00 00 00	11 00 00 00	8C B3 06 00	W...?...<...?³-
0240FA04	00 00 00 00	F8 B8 06 00	06 00 00 00	07 00 00 000,-.-...+....

Cuối cùng thực hiện copy nốt 4 bytes vào thanh ghi EAX. Sau đó giá trị của EAX lại được cất vào vùng nhớ [EBP-28] :

00401317	.	8955 D4	mov	dword ptr [ebp-2C], edx	
0040131A	.	8B40 08	mov	eax, dword ptr [eax+8]	
0040131D	.	8945 D8	mov	dword ptr [ebp-28], eax	crakmeea.00401222
00401320	.	8D45 DC	lea	eax, dword ptr [ebp-24]	

0240F9E4	31 30 34 34	35 36 37 38	39 35 31 00	03 00 00 00	10445678951.L...
0240F9F4	57 00 00 00	14 00 00 00	11 00 00 00	8C B3 06 00	W...?...<...?³-
0240FA04	00 00 00 00	F8 B8 06 00	06 00 00 00	07 00 00 000,-.-...+....

Như vậy qua đây ta thấy rằng hai thanh ghi EAX và EDX được sử dụng làm trung gian để copy toàn bộ chuỗi mặc định vào một vùng nhớ do chương trình định ra. Đoạn code tiếp theo chúng ta sẽ thấy chương trình sử dụng hàm **memset**, quan sát ta thấy hàm này nhận vào 3 tham số :

00401323	.	83C4 FC	add	esp, -4	
00401326	.	6A 08	push	8	n = 8 c = 00 s = 0240F9F0 memset
00401328	.	6A 00	push	0	
0040132A	.	50	push	eax	
0040132B	.	E8 F0020000	call	<jmp.&msvcrt.memset>	
00401330	.	83C4 10	add	esp, 10	

Google thử thông tin về hàm này :

NAME

memset - set bytes in memory

SYNOPSIS

```
#include <string.h>

void *memset(void *s, int c, size_t n);
```

DESCRIPTION

The `memset()` function copies `c` (converted to an **unsigned char**) into each of the first `n` bytes of the object pointed to by `s`.

Hàm memset nhận vào 3 biến là `s`, `c`, `n` :

`s` is the starting address

`n` is the number of bytes that are going to fill

`c` is the value with which to fill this area

0240F99C	0240F9F0	s = 0240F9F0
0240F9A0	00000000	c = 00
0240F9A4	00000008	n = 8
0240F9A8	00000000	

Qua những thông tin trên ta kết luận rằng, hàm memset sẽ tiến hành set 8 bytes của vùng nhớ tại `0x0240F9F0` thành giá trị **0x00**. Follow in Dump tại vùng nhớ, sau đó nhấn **F8** để thực hiện hàm ta sẽ thấy kết quả như sau :

0240F9F0	00 00 00 00	00 00 00 00	14 00 00 00	11 00 00 007...<...
0240FA00	8C B3 06 00	00 00 00 00	F8 B8 06 00	06 00 00 00	3-.....0,-.-...
0240FA10	07 00 00 00	40 FA 40 02	34 87 41 7E	46 04 12 00	...@u@,4AA~FJ↑.

Trace tiếp xuống đoạn code bên dưới ta bắt gặp đoạn code sau :

00401343	. 8D45 D0	lea	eax, dword ptr [ebp-30]	
00401346	. 50	push	eax	s = "10445678951"
00401347	. E8 DC020000	call	<jmp.&msvcrt.strlen>	strlen
0040134C	. 83C4 10	add	esp, 10	

Hàm **strlen** sẽ tính toán độ dài của chuỗi `s`, cụ thể ở đây là chuỗi `s = "10445678951"`. Kết quả tính toán được sẽ lưu tại thanh ghi EAX :

Registers (FPU)		< < < < <	
EAX	0000000B		
ECX	0240F9E4 ASCII "10445678951"		
EDX	7F303438		
EBX	00000000		
ESP	0240F99C		
EBP	0240FA14		
ESI	00401240 crakmeea.00401240		
EDI	0240FA7C		

Như quan sát kết quả tại thanh ghi EAX ta thấy giá trị của nó là **0xB**, tương ứng với 11 ở hệ mười. Vậy tức là độ dài chuỗi mặc định là 11. Nhấn **F8** để trace tiếp ta sẽ đến đây :

Dòng lệnh tiếp theo sẽ mov giá trị tại [EBP-10] vào thanh ghi EDX, giá trị của [EBP-10] lúc này đang là 0x0 :

00401363	. 8B55 F0	mov	edx, dword ptr [ebp-10]
00401366	. 01D0	add	eax, edx

Registers (FPU)			
EAX	00066A70	ASCII	"m4n0w4r"
ECX	0240F9E4	ASCII	"10445678951"
EDX	00000000		
EBX	00000000		
ESP	0240F9AC		
EBP	0240FA14		
ESI	00401240	crakmeea.00401240	

Tiếp theo :

00401363	. 8B55 F0	mov	edx, dword ptr [ebp-10]
00401366	. 01D0	add	eax, edx
00401368	. 0FBE10	movsx	edx, byte ptr [eax]
0040136B	. 8D42 EC	lea	eax, dword ptr [edx-14]

EAX hiện tại đang trỏ vào chuỗi Fake Serial, giá trị của EDX đang là 0x0. Điều này làm ta liên tưởng tới việc sẽ có một vòng lặp trong đó thanh ghi EAX giữ nguyên, thanh ghi EDX thay đổi từ 0x0 đến 0xA, vậy tức là sẽ lần lượt lấy ra từng kí tự trong chuỗi Fake Serial (vậy cũng kết luận là chuỗi Serial mà ta nhập vào cũng phải có độ dài là 0xB). Điều này thể hiện ở đoạn code sau :

00401368	. 0FBE10	movsx	edx, byte ptr [eax]	
0040136B	. 8D42 EC	lea	eax, dword ptr [edx-14]	
0040136E	. 8D55 D0	lea	edx, dword ptr [ebp-30]	
00401371	. 8B4D F0	mov	ecx, dword ptr [ebp-10]	
00401374	. 0FBE1411	movsx	edx, byte ptr [ecx+edx]	
00401378	. 39D0	cmp	eax, edx	
0040137A	. 75 0D	jnz	short 00401389	
0040137C	. 8D45 D0	lea	eax, dword ptr [ebp-30]	
0040137F	. 8B55 F0	mov	edx, dword ptr [ebp-10]	
00401382	. C60402 73	mov	byte ptr [edx+eax], 73	
00401386	. FF45 CC	inc	dword ptr [ebp-34]	
00401389	. FF45 F0	inc	dword ptr [ebp-10]	
0040138C	. EB B2	jmp	short 00401340	
0040138E	. 89F6	mov	esi, esi	crakmeea.00401240
00401390	. B8 2E124000	mov	eax, 0040122E	ASCII "Correct!"
00401395	. 8B10	mov	edx, dword ptr [eax]	
00401397	. 8955 B0	mov	dword ptr [ebp-50], edx	
ds:[0006B8F8]=6D ('m')				
edx=00000000				

Lệnh MOVSX sẽ thực hiện copy giá trị từ nguồn vào đích, nguồn ở đây vùng nhớ chứa Fake Serial, tính theo từng byte. Đích ở đây là thanh ghi EDX, tính theo Dword. Trong trường hợp cụ thể của tôi là copy byte đầu tiên, tức là chữ cái “m” và thanh ghi EDX :

Registers (FPU)			
EAX	0006B8F8	ASCII	"m4n0w4r"
ECX	0240F9E4	ASCII	"10445678951"
EDX	0000006D		
EBX	00000000		
ESP	0240F9AC		
EBP	0240FA14		
ESI	00401240	crakmeea.00401240	

Dòng lệnh tiếp theo :

00401368	. 0FBE10	movsx	edx, byte ptr [eax]
0040136B	. 8D42 EC	lea	eax, dword ptr [edx-14]
0040136E	. 8D55 D0	lea	edx, dword ptr [ebp-30]
00401371	. 8B4D F0	mov	ecx, dword ptr [ebp-10]
00401374	. 0FBE1411	movsx	edx, byte ptr [ecx+edx]
00401378	. 39D0	cmp	eax, edx

Ta thấy rằng, giá trị của EDX sau đó được trừ đi 0x14, kết quả được bao nhiêu sẽ lưu vào thanh ghi EAX. Cụ thể của tôi ($0x6D - 0x14 = 0x59$):

Registers (FPU)	
EAX	00000059
ECX	0240F9E4 ASCII "10445678951"
EDX	0000006D
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmeea.00401240
EDI	0240FA7C

Câu lệnh tiếp theo 0040136E .8D55 D0 lea edx, dword ptr [ebp-30], thanh ghi EDX lúc này sẽ trở vào chuỗi mặc định :

Stack address=0240F9E4, (ASCII "10445678951")
edx=0000006D

Registers (FPU)	
EAX	00000059
ECX	0240F9E4 ASCII "10445678951"
EDX	0240F9E4 ASCII "10445678951"
EBX	00000000
ESP	0240F9AC
EBP	0240FA14

Tiếp tục xem xét hai câu lệnh bên dưới :

```
00401371 . 8B4D F0      mov     ecx, dword ptr [ebp-10]
00401374 . 0FBE1411    movsx   edx, byte ptr [ecx+edx]
```

Lệnh đầu tiên thanh ghi ECX sẽ được gán giá trị tại [EBP-10] (hiện tại giá trị tại [EBP-10] = 0x0). Lệnh tiếp theo dùng để copy vào thanh ghi EDX từng kí tự trong chuỗi mặc định (do lúc này thanh ghi ECX đang là 0x0 cho nên thanh ghi EDX sẽ chứa kí tự đầu tiên của chuỗi mặc định). Vậy ta nhận thấy rằng việc di chuyển để copy kí tự tiếp theo trong chuỗi Fake Serial và chuỗi mặc định sẽ phụ thuộc vào giá trị tại [EBP-10] :

Stack ds:[0240F9E4]=31 ('1')
edx=0240F9E4, (ASCII "10445678951")

Registers (FPU)	
EAX	00000059
ECX	00000000
EDX	00000031
EBX	00000000
ESP	0240F9AC
EBP	0240FA14
ESI	00401240 crakmeea.00401240

Kết thúc quá trình lấy kí tự từ hai chuỗi Fake Serial và chuỗi mặc định, chúng ta sẽ tới đoạn code so sánh :

```

00401374 . 0FBE1411 movsx   edx, byte ptr [ecx+edx]
00401378 . 39D0      cmp     eax, edx
0040137A . 75 0D     jnz     short 00401389
0040137C . 8D45 D0   lea     eax, dword ptr [ebp-30]

```

```

edx=00000031
eax=00000059

```

Trong hình minh họa của tôi, các bạn thấy thanh ghi EAX đang có giá trị là 0x59, thanh ghi EAX đang có giá trị là 0x31. Quay ngược lại một chút ta sẽ thấy rằng, giá trị 0x59 có được là do kí tự đầu tiên của chuỗi Fake Serial là “m”, mã hexa là 0x6D trừ đi 0x14. Còn giá trị 0x31 là mã hexa của kí tự đầu tiên trong chuỗi mặc định. Tôi viết lại cụ thể như sau :

cmp eax, edx ⇔ cmp (First byte of FS - 0x14), First byte of Constant string ⇔
cmp 59, 31

Kết quả của việc so sánh này sẽ ảnh hưởng tới lệnh nhảy bên dưới ☺. Tới đây chúng ta không cần quan tâm tới đoạn code bên dưới nữa, chúng ta sẽ lập luận để tìm ra được Real Serial. Như các bạn thấy chúng ta có công thức như sau :

CMP (FIRST BYTE OF MY CORRECT SERIAL - 14), 31

Ở đây là so sánh bằng vậy cho nên ta có :

FIRST BYTE OF MY CORRECT SERIAL - 0x14 = 0x31

Suy ra :

FIRST BYTE OF MY CORRECT SERIAL = 0x31 + 0x14

```

Command ? 31 + 14  HEX: 45 - DEC: 69 - ASCII: E
Start:402000 End:401FFF Value:1

```

Khả khả vậy là ta đã tìm ra được kí tự đầu tiên của chuỗi Real Serial rồi, đó là kí tự ‘E’. Thực hiện tiếp tục theo logic như trên ta sẽ có được chuỗi Real Serial hoàn chỉnh :

```

0240F9E4 31 30 34 34 35 36 37 38 39 35 31 00 00 00 00 00 10445678951
0240F9F4 00 00 00 00 13 00 00 00 11 00 00 00 8C B3 06 00 .....!!...3-
0240FA04 01 00 00 00 F8 B8 06 00 06 00 00 00 07 00 00 00 r--0--...

```

31 + 14 = 45 is the letter E in ASCII

30 + 14 = 44 is the letter D in ASCII

34 + 14 = 48 is the letter H in ASCII

34 + 14 = 48 is the letter H in ASCII

35 + 14 = 49 is the letter I ASCII

36 + 14 = 4A is the letter J in ASCII

37 + 14 = 4B is the letter K in ASCII

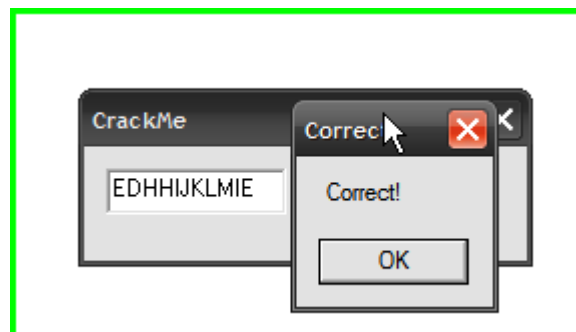
38 + 14 = 4C is the letter L in ASCII

39 + 14 = 4D is the letter M in ASCII

35 + 14 = 48 is the letter I in ASCII

31 + 14 = 45 is the letter E in ASCII

Hola, chuỗi serial cuối cùng của chúng ta là : **EDHHIJKLMIE**. Nhiệm vụ của Olly đã xong, ta tạm thời close Olly, chạy thử crackme và test serial mà ta tìm được :



Ok vậy là phần 15 của loạt tuts về Ollydbg đến đây là kết thúc, qua bài viết này tôi và các bạn đã cùng thực hành với một crackme với mức độ khó hơn một chút, chỉ là một chút thôi ☺. Trong phần tutor này còn có một target khác để các bạn thực hành đó là Splish crackme, có lẽ tôi cũng bắt chiếc lão Ricard, tutor tiếp theo sẽ đặt pass là serial mà các bạn tìm được ở Splish crackme lolz. Hi vọng qua bài viết này của tôi đã cung cấp thêm một số kiến thức cơ bản giúp các bạn làm việc hiệu quả với Olly. Hẹn gặp lại các bạn trong các phần tiếp theo của loạt tutor này, By3 By3!! ☺

Best Regards

[Kienmanowar]



--++--==[**Greatz Thanks To**]==--++--

My family, Computer_Angel, Moonbaby , Zombie_Deathman, Littleboy, Benina, QHQCrk, the_Lighthouse, Merc, Hoadongnoi, Nini ... all REA's members, TQN, HacNho, RongChauA, Deux, tlandn, light.phoenix, dump, 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 **hagggar**), 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).Thanx to Orthodox, kanxue, TiGa and finally, thanks to **RICARDO NARVAJA** and all members on **CRACKSLATINOS**.

>>>> If you have any suggestions, comments or corrections email me:

[kienmanowar\[at\]reaonline.net](mailto:kienmanowar[at]reaonline.net)

