

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

Đợt rồi bận quá, mãi cài đặt và config hệ thống để phục vụ cho dự án upgrade, xem chút nữa thì quên mất viết tiếp bộ tut về Olly. Trong phần 7 tôi đã tập trung giới thiệu tới các bạn về lệnh CALL và RET. Xét một cách tổng quan thì đây là 2 lệnh đơn giản, tuy nhiên khi đi vào chi tiết nhiều người thường khó hiểu đặc biệt là những bạn mới làm quen với ASM. Hôm nay, chúng ta tiếp tục với phần 8 của loạt tuts này, phần này tập trung vào giới thiệu về vòng lặp và các câu lệnh liên quan tới chuỗi. Bài viết sẽ không sa đà vào giải thích cặn kẽ từng cấu trúc của vòng lặp mà sẽ đi nhanh và tập trung nhấn mạnh vào những phần quan trọng. Phần 8 này cũng là phần kết thúc việc giới thiệu các lệnh cơ bản, các phần tiếp theo sẽ tập trung vào những chủ đề, khía cạnh thú vị hơn.

II. Giới thiệu về vòng lặp

Một vòng lặp là một chuỗi các lệnh được lặp lại. Số lần lặp có thể đã được xác định trước hoặc phụ thuộc vào điều kiện. Thông thường trong Asm thì bộ đếm vòng lặp thường là thanh ghi ECX, nó được khởi tạo bằng **số lần lặp**. Sau khi thực hiện xong các lệnh bên trong vòng lặp giá trị của thanh ghi ECX sẽ tự động giảm đi 1. Nếu như giá trị của ECX còn khác 0 thì các lệnh trong vòng lặp còn tiếp tục được thực hiện, còn nếu như ECX bằng 0 thì sẽ thoát khỏi vòng lặp và thực hiện các lệnh tiếp theo bên dưới vòng lặp. Lấy ví dụ minh họa:

```
XOR ECX, ECX \ Đoạn lệnh này thực hiện nhiệm vụ khởi gán cho thanh ghi
ADD ECX, 15 / ECX.Số lần lặp lúc này sẽ là 15 và được lưu trong thanh ghi ECX.
LOOP:
DEC ECX // Giá trị của thanh ghi ECX sẽ được giảm đi 1
..... \ Thực hiện các lệnh trong thân vòng lặp
..... /
TEST ECX, ECX \ Chừng nào giá trị của ECX còn khác 0
JNE LOOP      / thì còn tiếp tục thực hiện
```

Minh họa ví dụ trên trong Olly như sau :

Address	Hex dump	Disassembly	Comment
00401000	33C9	XOR ECX, ECX	
00401002	83C1 15	ADD ECX, 15	
00401005	90	NOP	
00401006	90	NOP	
00401007	49	DEC ECX	
00401008	90	NOP	
00401009	90	NOP	
0040100A	90	NOP	
0040100B	90	NOP	
0040100C	85C9	TEST ECX, ECX	
0040100E	75 F7	JNZ SHORT CRACKME.00401007	
00401010	2040 00	AND BYTE PTR DS:[EAX], AL	
00401012	E8 A0400000	CALL <JMP.@USER32.FindWindowA>	FindWindowA

Vùng mà tôi khoanh vàng ở trên hình đó chính là vòng lặp, các lệnh trong vòng lặp này sẽ được thực hiện lặp lại cho tới khi nào thanh ghi ECX có giá trị 0. Các câu lệnh bắt đầu từ 0x00401008 cho tới 0x0040100C chính là thân của vòng lặp. Do chỉ mang tính chất minh họa nên tôi để toàn lệnh NOP, các bạn có thể thay thế bằng các lệnh khác. Hiện tại chúng ta đang ở tại câu lệnh XOR ECX, ECX, nhấn F7 để trace và quan sát giá trị của thanh ghi ECX.

Lệnh XOR ECX, ECX sẽ xóa thanh ghi ECX về 0 :

```

Registers (FPU)
EAX 00000000
ECX 00000000
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 0(FFFFFFFF)
CS 0008 32bit 0(FFFFFFFF)
  
```

Lệnh ADD ECX, 15 tương đương với việc gán ECX = 0x15 :

```

Registers (FPU)
EAX 00000000
ECX 00000015
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401005 CRACKME.00401005
  
```

Nhấn F7 và trace qua lệnh DEC ECX. Giá trị của ECX lúc này sẽ được giảm đi 1 (ECX = ECX - 1 = 0x15 - 0x1 = 0x14) :

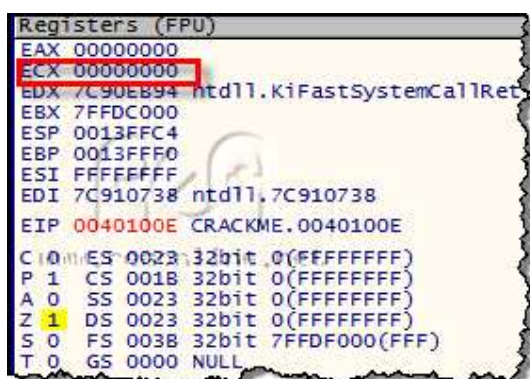
```

Registers (FPU)
EAX 00000000
ECX 00000014
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD8000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401007 CRACKME.00401007
  
```

Tiếp tục thực hiện cho tới khi chúng ta dừng lại tại lệnh TEST ECX, ECX, câu lệnh này sẽ kiểm tra xem giá trị của ECX đã bằng 0 chưa ? Nếu bằng 0 thì cờ Z sẽ được bật lên. Tuy nhiên lúc này giá trị ECX đang là 0x14, do đó khi trace qua lệnh TEST thì cờ Z không được active và vì vậy lệnh JNZ ở bên dưới sẽ nhảy lại về vị trí 0x00401007 (nơi bắt đầu của vòng lặp)

Address	hex	dump	Disassembly
00401000	33C9		XOR ECX, ECX
00401002	83C1 15		ADD ECX, 15
00401005	90		NOP
00401006	90		NOP
00401007	49		DEC ECX
00401008	90		NOP
00401009	90		NOP
0040100A	90		NOP
0040100B	90		NOP
0040100C	85C9		TEST ECX, ECX
0040100E	75 F7		JNZ SHORT CRACKME.00401007

Trace tiếp để thực hiện các lệnh cho tới khi giá trị của thanh ghi ECX = 0x0, lúc này lệnh TEST kiểm tra thấy giá trị của ECX = 0x0 cho nên nó sẽ active cờ Z. Khi cờ Z được active nên câu lệnh JNZ sẽ dựa vào cờ Z và đưa ra quyết định sẽ không nhảy về vị trí 0x00401007.



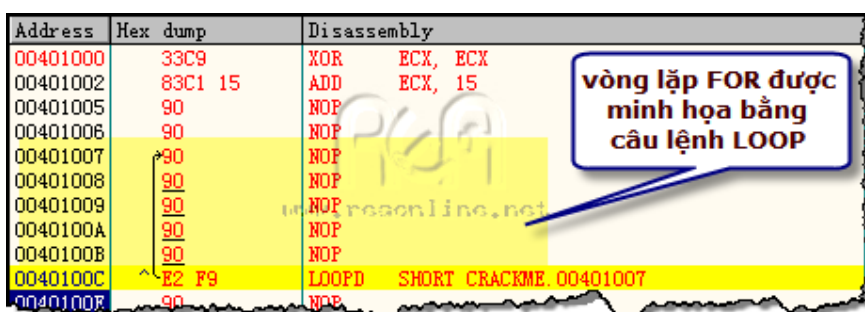
Các bạn sẽ dễ dàng nhận thấy mũi tên tại lệnh JNZ đã chuyển sang màu xám, điều đó có nghĩa là lệnh nhảy sẽ không được thực hiện. Còn nếu như mũi tên này có màu đỏ như hình bên trên thì tức là lệnh nhảy sẽ được thực hiện. Do mũi tên đã chuyển sang màu xám cho nên khi ta tiếp tục trace, chúng ta sẽ thoát ra khỏi vòng lặp và thực hiện các lệnh tiếp theo bên dưới. Đây chỉ là một ví dụ minh họa rất đơn giản để các bạn hình dung về vòng lặp, các cấu trúc lặp như For, While và Repeat các bạn có thể đọc thêm trong các tài liệu về lập trình ASM.

Lệnh LOOP :

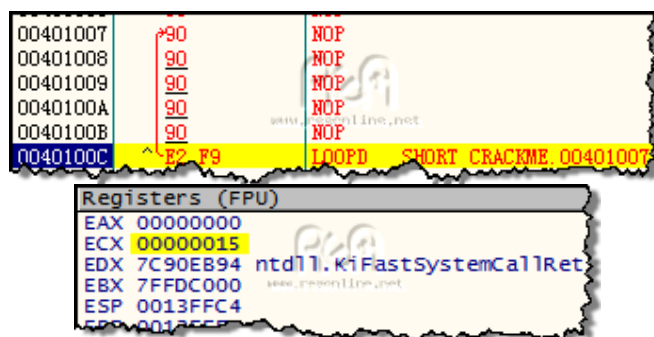
Câu lệnh này thường được sử dụng để thực hiện vòng lặp FOR. Nó có dạng như sau :

LOOP Nhãn

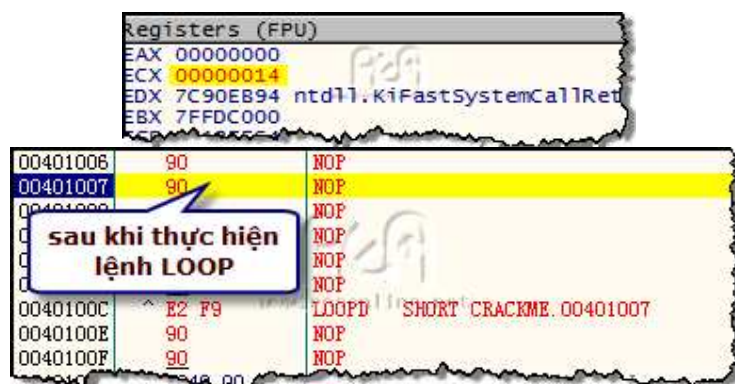
Bộ đếm vòng lặp được khởi tạo trong thanh ghi ECX. Mỗi lần thực hiện lệnh LOOP, thanh ghi ECX sẽ tự động được giảm đi 1, nếu thanh ghi ECX khác 0 thì điều khiển được chuyển tới **Nhãn** (tức là lặp lại lệnh bắt đầu từ vị trí của Nhãn). Nếu ECX = 0, lệnh tiếp theo sau lệnh LOOP sẽ được thực hiện. Với câu lệnh LOOP này ta có thể thay thế các lệnh đã minh họa ở mục trước như sau :



Như đã nói, vì chỉ mang tính chất minh họa cho nên tôi để toàn lệnh NOP cho đơn giản và dễ hiểu. Tương tự như ở trên ta thực hiện lệnh XOR và lệnh ADD để khởi tạo giá trị cho thanh ghi ECX. Sau đó ta trace cho tới câu lệnh LOOP, giá trị của ECX lúc này đang là 0x15.



Bây giờ ta trace để thực hiện lệnh LOOP, ngay lập tức thanh ghi ECX sẽ được giảm xuống (trừ đi 1) và quyền điều khiển được chuyển tới **Nhãn** (địa chỉ 0x401007).



Tiếp tục thực hiện cho tới khi giá trị của thanh ghi ECX được giảm xuống còn 0x1, và trace tới lệnh LOOP. Lúc này khi ta thực hiện lệnh LOOP, ECX sẽ được giảm xuống là 0x0, quyền điều khiển sẽ không được chuyển cho **Nhãn** nữa mà chuyển cho câu lệnh bên dưới lệnh LOOP.

Ngoài lệnh LOOP trên các bạn có thể tham khảo thêm về các lệnh LOOPE/LOOPZ, LOOPNE/LOOPNZ trong các tài liệu.

III. Các lệnh liên quan đến chuỗi

Lệnh MOVS :

Câu lệnh này sẽ sao chép nội dung dữ liệu được định địa chỉ bởi DS:ESI đến vùng dữ liệu được định địa chỉ bởi ES:EDI. Nội dung của chuỗi gốc không bị thay đổi. Sau khi chuỗi được chuyển cả hai thanh ghi ESI và EDI đều được tự động tăng lên 1 nếu như cờ DF = 0 hay giảm đi nếu DF = 1.

Nếu DF = 0, ESI và EDI được xử lý theo chiều tăng của các địa chỉ trong bộ nhớ : từ trái qua phải trong chuỗi. Ngược lại nếu DF = 1, ESI và EDI được xử lý theo chiều giảm dần các địa chỉ bộ nhớ : từ phải qua trái trong chuỗi.

Lệnh MOVS thường không cần bất kì tham số nào, tuy nhiên khi viết trong Olly chúng ta phải chỉ rõ ra như sau : `MOVS DWORD PTR IS:[EDI], DWORD PTR DS:[ESI]`. Lấy một ví dụ nhỏ để minh họa :

Address	Hex dump	Disassembly	Comment
00401000	BE 6C364000	MOV ESI, CRACKME.0040366C	ASCII "lgA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	
0040100A	A5	MOVS DWORD PTR ES:[EDI], DWORD PTR DS:[ESI]	
0040100B	006A 00	ADD BYTE PTR DS:[EDX], CH	
0040100E	68 F4204000	PUSH CRACKME.004020F4	Class = "No need to disasm the code!"
00401013	E8 A6040000	CALL <JMP.@USER32.FindWindowA>	FindWindowA

Câu lệnh đầu tiên sẽ khởi tạo giá trị cho ESI và nội dung dữ liệu sẽ được đọc ra từ đó. EDI cũng được khởi tạo một giá trị, và tại đó dữ liệu sẽ được copy vào. Chúng ta có thể xem nội dung tại hai địa chỉ trên bằng cách vào cửa sổ Dump, nhấn Ctrl+G và gõ vào địa chỉ cần xem nội dung hoặc tại cửa sổ CPU, chuột phải chọn **Follow in Dump > Immediate Constant** :

Address	Hex dump	Disassembly	Comment
00401000	BE 6C364000	MOV ESI, CRACKME.0040366C	ASCII "lgA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	
0040100A	A5	MOVS DWORD PTR ES:[EDI], DWORD PTR DS:[ESI]	
0040100B	006A 00	ADD BYTE PTR DS:[EDX], CH	
0040100E	68 F4204000	PUSH CRACKME.004020F4	No need to disasm the
00401013	E8 A6040000	CALL <JMP.@USER32.FindWindowA>	FindWindowA
00401018	0BC0	OR EAX, EAX	
0040101A	74 01	JE SHORT CRACKME.0040101C	
0040101C	C3	RETN	
0040101D	C705 84204000	MOV DWORD PTR DS:[4], CRACKME.004020F4	
00401027	C705 68204000	MOV DWORD PTR DS:[4], CRACKME.004020F4	
00401031	C705 8C204000	MOV DWORD PTR DS:[4], CRACKME.004020F4	
0040103B	C705 70204000	MOV DWORD PTR DS:[4], CRACKME.004020F4	
00401045	A1 CA204000	MOV EAX, DWORD PTR DS:[4]	
0040104A	A3 74204000	MOV DWORD PTR DS:[4], EAX	
0040104F	6A 64	PUSH 64	
00401051	50	PUSH EAX	
00401052	E8 D1030000	CALL <JMP.@USER32.FindWindowA>	

0040366C=CRACKME.0040366C (ASCII "lgA")
ESI=0040366C (CRACKME.0040366C), ASCII "lgA"

Address	Hex dump	ASCII
0040366C	6C 67 41 00 00 00 00 00 00 00 00 00 00 00 00 00	lgA....
0040367C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040368C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040369C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0040369C=CRACKME.0040369C

Address	Hex dump	ASCII
0040369C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036AC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036BC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

OK vậy là ta đã biết được nội dung của chuỗi gốc và chuỗi đích (Chuỗi gốc là nơi mà ta đọc dữ liệu ra và chuỗi đích là nơi mà ta copy dữ liệu vào) Bây giờ nhấn F7 để trace qua lệnh MOVS, quan sát cửa sổ dump ta sẽ thấy được 4bytes của chuỗi gốc được copy vào chuỗi đích :

DS: [ESI]=[0040366C]=0041676C
ES: [EDI]=[0040369C]=00000000

Address	Hex dump	ASCII
0040369C	6C 67 41 00	lgA.....
004036AC	00 00 00 00
0040368C	00 00 00 00
004036CC	00 00 00 00
004036DC	00 00 00 00

Các bạn tự tìm hiểu thêm các lệnh MOVSB và MOVSW.

Lệnh REP :

Không giống như lệnh MOVS chỉ chuyển một doubleword tại địa chỉ DS:(E)SI tới địa chỉ ES:(E)DI, lệnh REP cho chúng ta chuyển cả chuỗi. Để chuyển cả chuỗi đầu tiên chúng ta phải khởi tạo thanh ghi (E)CX với số N bằng số doubleword trong chuỗi nguồn và thực hiện lệnh như sau :

REP MOVS

Lệnh REP có tác dụng làm cho MOVS được thực hiện N lần. Sau mỗi lệnh MOVS thì (E)CX được giảm đi 1 cho đến khi nó bằng 0. Tôi lấy ví dụ minh họa như sau :

Address	Hex dump	Disassembly	Comment
00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	
0040100A	B9 04000000	MOV ECX, 4	
0040100F	F3:A5	REP MOVS DWORD PTR ES:[EDI], DWORD PTR DS:[ESI]	
00401011	40	INC EAX	
00401012	00E8	ADD AL, CHUCKONLINE.NET	
00401014	? A5	CMPS BYTE PTR DS:[ESI], BYTE PTR ES:[EDI]	
00401015	? 04 00	ADD AL, 0	
00401017	? 000F	ADD BYTE PTR DS:[EBX], CL	

Ta quan sát nội dung tại 0x0040365C, nơi chứa chuỗi gốc :

0040365C=CRACKME.0040365C (ASCII "eNameA")		
ESI=0040366C (CRACKME.0040366C), ASCII "lgA"		
Address	Hex dump	ASCII
0040365C	65 4E 61 6D 65 41 00 00 00 00 50 72 69 6E 74 44	eNameA....PrintD
0040366C	6C 67 41 00 00 00 00 00 00 00 00 00 00 00 00 00	lgA.....
0040367C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Trong đoạn code trên, bạn để ý thấy ECX được khởi tạo giá trị là 4. Vậy tức là sẽ có 4 lần copy từ chuỗi nguồn vào chuỗi đích, mỗi lần copy một doubleword. Bây giờ ta Trace từ từ tới lệnh REP :

ECX=00000004 (decimal 4.)		
DS:[ESI]=[0040365C]=6D614E65		
ES:[EDI]=[0040369C]=00000000		
Address	Hex dump	ASCII
0040369C	65 4E 61 6D 00 00 00 00 00 00 00 00 00 00 00 00	eNam.....

Các thông tin hiện ra trong Tip Window rất đầy đủ, giờ ta nhấn F7 và quan sát nội dung tại chuỗi đích, tức là tại 0x0040369C. Tại sao lại nhấn F7 mà không phải là F8, vì nếu các bạn nhấn F8 lệnh REP MOVS sẽ copy thẳng một mạch toàn bộ 4 doubleword từ chuỗi gốc sang chuỗi đích, đồng thời thanh ghi ECX sẽ được giảm về 0. Ta sử dụng F7 để quan sát từng lần copy một :

Address	Hex dump	ASCII
0040369C	65 4E 61 6D 00 00 00 00 00 00 00 00 00 00 00 00	eNam.....
004036AC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040368C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

Registers (FPU)
EAX 00000000
ECX 00000003
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDB000
ESP 0013FFC4
EBP 0013FFF0
ESI 00403660 ASCII "eA"
EDI 004036A0 CRACKME.004036A0

```

Như quan sát trên hình minh họa, ta thấy 4 bytes đầu tiên được chuyển tới và thành ghi ECX lúc này bị giảm đi 1 ($ECX = 0 \times 4 - 0 \times 1 = 0 \times 3$). Thêm vào đó 2 thanh ghi ESI và EDI được tăng thêm 4 để trở vào vị trí tiếp theo :

```

ECX=00000003 (decimal 3.)
DS:[ESI]=[00403660]=00004165
ES:[EDI]=[004036A0]=00000000

```

Tiếp tục nhấn F7 và quan sát :

Address	Hex dump	ASCII
0040369C	65 4E 61 60 65 41 00 00 00 00 00 00 00 00 00 00	eNameA.....
004036AC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040368C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036DC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00


```

Registers (FPU)
EAX 00000000
ECX 00000002
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDB000
ESP 0013FFC4
EBP 0013FFF0
ESI 00403664 CRACKME.00403664
EDI 004036A4 CRACKME.004036A4

```

Cuối cùng nhấn F7 cho tới khi $ECX = 0 \times 0$, lúc này toàn bộ chuỗi gốc đã được copy sang chuỗi đích. Rất rõ ràng và dễ hiểu!

Address	Hex dump	ASCII
0040367C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040368C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040369C	65 4E 61 60 65 41 00 00 00 50 72 69 6E 74 44	eNameA...PrintD
004036AC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040368C	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036CC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036DC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
004036EC	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Lệnh LODS :

Lệnh LODS nạp vào EAX một double word do ESI chỉ ra trong đoạn DS, sau đó ESI tự động tăng/ giảm để chỉ vào phần tử tiếp theo tùy theo cờ hướng DF. Ví dụ minh họa :

Address	Hex dump	Disassembly	Comment
00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	
0040100A	AD	LODS DWORD PTR DS:[ESI]	
0040100B	90	NOP	
0040100C	90	NOP	
0040100D	90	NOP	
0040100E	90	NOP	
0040100F	90	NOP	

Trace từ từ đến lệnh LODS và quan sát :

DS: [ESI]=[0040365C]=6D614E65

Address	Hex dump	ASCII
0040365C	65 4E 61 6D	eNameA....PrintD
0040366C	00 00 00 00	lgA.....
0040367C	00 00 00 00
0040368C	00 00 00 00
0040369C	00 00 00 00
004036AC	00 00 00 00
004036BC	00 00 00 00
004036CC	00 00 00 00
004036DC	00 00 00 00
004036EC	00 00 00 00

Giá trị sẽ được nạp vào EAX

Nhấn F7 để thực hiện lệnh LODS, kết quả ta sẽ có được như sau :

Registers (FPU)	
EAX	6D614E65
ECX	00000004
EDX	7C90EB94 ntd
EBX	7FFDF000
ESP	0013FFC4
EBP	0013FFF0
ESI	00403660 ASCII "ea"
EDI	0040369C CRACKME.0040369C
ETP	00000000 CRACKME.0040100B

Chỉ vào phần tử tiếp theo

Lệnh STOS :

Lệnh này ngược lại với lệnh LODS, có tác dụng chuyển nội dung của thanh EAX đến vị trí đã được định nghĩa bởi ES:EDI. Sau khi thực hiện xong thì EDI tự động tăng/giảm để chỉ vào phần tử tiếp theo tùy theo cờ hướng. Ví dụ :

Address	Hex dump	Disassembly	Comment
00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	
0040100A	AE	STOS DWORD PTR ES:[EDI]	
0040100B	90	NOP	
0040100C	90	NOP	
0040100D	90	NOP	
0040100E	90	NOP	
0040100F	90	NOP	

Trace tới lệnh STOS và quan sát cửa sổ TIP :

EAX=6D614E65
ES:[EDI]=[0040369C]=00000000

Sau khi thực hiện lệnh STOS, kết quả ta có được như sau :

Address	Hex dump	ASCII
0040369C	65 4E 61 6D	eName.....
004036AC	00 00 00 00
004036BC	00 00 00 00
004036CC	00 00 00 00
004036DC	00 00 00 00

Registers (FPU)	
EAX	6D614E65
ECX	00
EDX	7C
EBX	7F
ESP	00
EBP	0013FFF0
ESI	0040365C ASCII "eNameA"
EDI	004036A0 CRACKME.004036A0

Chỉ vào phần tử tiếp theo

Lệnh CPMS :

Cú pháp : CPMS Chuỗi đích, Chuỗi gốc

Lệnh này dùng để so sánh nội dung của ESI với nội dung của EDI. Lệnh này chỉ tạo cờ, không lưu kết quả so sánh, sau khi so sánh các toán hạng không bị thay đổi.

Address	Hex dump	Disassembly	Comment
00401000	BE 5C364000	MOV ESI, CRACKME.0040365C	ASCII "eNameA"
00401005	BF 9C364000	MOV EDI, CRACKME.0040369C	ASCII "eNam"
0040100A	A7	CMPS DWORD PTR DS:[ESI], DWORD PTR ES:[EDI]	
0040100B	90	NOP	
0040100C	90	NOP	
0040100D	90	NOP	
0040100E	90	NOP	

Ta trace tới lệnh CMPS và quan sát :

00401097	FF35 4A204000	PUSH DWORD PTR DS:[4A204000]
ES: [EDI] = [0040369C] = 6D614E65		
DS: [ESI] = [0040365C] = 6D614E65		

Về bản chất của lệnh CPMS là lệnh này trừ dw tại địa chỉ DS:ESI cho dw tại địa chỉ ES:EDI và thiết lập cờ. Nếu kết quả của phép trừ bằng 0x0 thì cờ Z được bật cũng đồng nghĩa hai chuỗi này giống nhau :

Registers (FPU)				
EAX	6D614E65			
ECX	00000004			
EDX	7C90EB94	ntdll.KiFastSystemCall		
EBX	7FFDF000			
ESP	0013FFC4			
EBP	0013FFF0			
ESI	00403660	ASCII "eA"		
EDI	004036A0	CRACKME.004036A0		
EIP	0040100B	CRACKME.0040100B		
C 0	ES 0023	32bit 0(FFFFFFFF)		
P 1	CS 001B	32bit 0(FFFFFFFF)		
A 0	ES 0023	32bit 0(FFFFFFFF)		
Z 1	DS 0023	32bit 0(FFFFFFFF)		
S 0	FS 002E	32bit 0(FFFFFFFF)		

Ok như vậy qua 8 phần có thể nói tôi đã chỉ cho các bạn đã phần nào nắm được các lệnh cơ bản và quan trọng trong quá trình làm việc với Olly. Các phần tiếp theo của loạt tuts này sẽ tập trung vào các vấn đề khác, do đó các bạn tự tìm hiểu thêm về các lệnh trong các tài liệu về lập trình ASM.

Các chế độ địa chỉ

DIRECT (Trực tiếp) :

Trong chế độ địa chỉ trực tiếp, một toán hạng chứa địa chỉ lệch của ô nhớ dùng chứa dữ liệu còn toán hạng kia chỉ có thể là thanh ghi (không được là ô nhớ). Đây là cách chung được sử dụng để tham chiếu tới một địa chỉ trong bộ nhớ, một số ví dụ minh họa :

```
mov dword ptr [00513450], ecx
mov ax, word ptr [00510A25]
mov al, byte ptr [00402811]
CALL 452200
JMP 421000
```

Trong trường hợp này chúng ta sẽ không gặp vấn đề gì trong việc biểu diễn địa chỉ cũng như biết được nơi mà lệnh nhảy sẽ nhảy tới v.v... Chúng ta chỉ việc chỉ rõ các địa chỉ này trong các lệnh mov, lệnh jmp và cả lệnh Call.

INDIRECT (Gián tiếp) :

Thường được gọi là chế độ địa chỉ gián tiếp qua thanh ghi, trong chế độ địa chỉ này một toán hạng là một thanh ghi được sử dụng để chứa địa chỉ lệnh của ô nhớ chứa dữ liệu, còn toán hạng kia chỉ có thể là thanh ghi mà không được là ô nhớ.

```
mov dword ptr [eax], ecx
CALL EAX
JMP [ebx + 4]
```

Như các bạn thấy với cơ chế đánh địa chỉ như trên chúng ta sẽ gặp khó khăn trong việc xác định vị trí, không biết được vị trí mà câu lệnh nhảy sẽ nhảy đến, đoạn routine nào sẽ được gọi bởi lệnh call. Chỉ có sử dụng Olly để debug thì chúng ta mới biết được chính xác thông qua việc xem xét các giá trị của các thanh ghi.

Rất nhiều chương trình sử dụng cơ chế địa chỉ này để ngăn cản hay hạn chế quá trình phân tích chương trình của của cracker/reverser. Lấy một ví dụ để dễ hiểu, khi bạn cho Olly phân tích một chương trình bằng cách load target vào trong Olly, bạn sẽ gặp câu lệnh tương tự như sau :

```
PUSH    DWORD PTR SS:[EBP+8]                ; |hWnd
```

Nếu như bạn không tiến hành debug chương trình, thì thử hỏi khi bạn nhìn vào lệnh trên bạn làm cách nào để biết được nội dung gì sẽ được đẩy lên Stack ?

Câu trả lời như sau, để biết được nội dung gì sẽ được đẩy lên Stack bạn có thể debug trong Olly bằng cách trace từ từ tới câu lệnh trên hoặc đặt một Break point tại câu lệnh này và thực thi chương trình để chương trình dừng tại chỗ mà chúng ta đã set BP. Qua đó chúng ta sẽ quan sát được sự thay đổi và giá trị của thanh ghi EBP, dựa vào thông tin thu được ta sẽ xác định được nội dung của ô nhớ [EBP + 8]. Cụ thể như sau :

Address	Hex dump	Disassembly	Comment
004010E0	. E8 9D030000	CALL <JMP.&USER32.UpdateWindow>	UpdateWindow
004010E5	. 6A 01	PUSH 1	Erase = TRUE
004010E7	. 6A 00	PUSH 0	pRect = NULL
004010E9	. FF75 08	PUSH DWORD PTR SS:[EBP+8]	hWnd
004010EB		USER32.InvalidRect	InvalidRect
004010F1			MsgFilterMax = 0
004010F3			MsgFilterMin = 0
004010F5			hWnd = NULL
004010F7	. 68 48204000	PUSH CRACKME.00402048	pMsg = CRACKME.00402048
004010F9	. E8 D5030000	CALL <JMP.&USER32.GetMessageA>	GetMessageA

Đặt một BP tại lệnh
PUSH

Nhấn F9 để run chương trình, ngay lập tức chúng ta sẽ break tại chỗ mà ta vừa set BP. Quan sát trên cửa sổ Registers, ta thấy được giá trị của EBP là 0x0013FFF0, vậy EBP + 8 sẽ là 0x0013FFF8. Cửa sổ tip sẽ cho ta biết nội dung tại ô nhớ 0x0013FFF8 là gì :

Registers (FPU)	
EAX	00000001
ECX	0013FFA8
EDX	7C90EB94 ntdll.KiFastSystemCallRet
EBX	7FFDF000
ESP	0013FFB8
EBP	0013FFF0
ESI	FFFFFFFF
EDI	7C910738 ntdll.7C910738
EIP	004010E9 CRACKME.004010E9
C 0	ES 0023 32bit 0(FFFFFFFF)
P 0	CS 001B 32bit 0(FFFFFFFF)

00401148	837D 0C 01	CMP	DWORD PTR SS:[EBP+8], 1	
Stack SS:[0013FFF8]=00401000 (CRACKME.<ModuleEntryPoint>)				
CRACKME.<ModuleEntryPoint>+0E9				
Address	Hex dump	ASCII		
00402000	00 00 00 00 E0 04 A6 01 00 00 00 00 00 00 00 00à!.....		
00402010	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00		

Vậy tóm lại lệnh `PUSH [EBP+8]` sẽ PUSH nội dung của `0x0013FFF8` lên Stack. Nếu cẩn thận các bạn có thể vào cửa sổ DUMP, nhấn `Ctrl + G`, nhập địa chỉ trên vào để quan sát và kiểm tra :

Address	Hex dump	ASCII
0013FFF8	00 10 40 00 00 00 00 00	..@.....

Giờ ta nhấn `F7` để thực hiện lệnh Push, đồng thời quan sát cửa sổ Stack để thấy được kết quả :

Address	Value	Comment
0013FFB8	00401000	hWnd = 00401000
0013FFBC	00000000	pRect = NULL
0013FFC0	00000001	Erase = TRUE
0013FFC4	7C816FD7	RETURN to kernel32.7C816FD7
0013FFC8	7C910738	ntdll.7C910738
0013FFCC	FFFFFFFF	
0013FFD0	7FFDF000	kernel32.7FFDF000
0013FFD4	8054A6ED	
0013FFD8	0013FFC8	
0013FFDC	85FE1120	

Okie tôi nghĩ đến đây là đủ cho phần 8, bài viết xin được kết thúc tại đây. Trong phần này tôi có tham khảo thêm một số tài liệu liên quan tới lập trình ASM nhưng có thể không tránh khỏi những sai sót trong quá trình viết, rất mong các bạn góp ý để tôi chỉnh sửa lại. Tôi tin là trải qua 8 bài viết vừa qua tôi đã cung cấp cho các bạn những kiến thức cơ bản nhất để các bạn có thể hiểu và làm việc với Olly. Mặc dù 8 phần này có vẻ hơi nhàm chán với những ai đã có kinh nghiệm trong lập trình ASM tuy nhiên nó lại là những viên gạch đầu tiên cho những người mới làm quen. Những phần tiếp theo của loạt tuts này hứa hẹn sẽ mang đến cho bạn nhiều thông tin bổ ích hơn nữa. Hẹn gặp lại các bạn trong các phần tiếp theo, By3 By3!! ☺

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, dzungltvn, takada, hurt_heart, haule_nth, hytkl 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:

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