

INTRODUCTION TO THE CRACKING WITH OLLYDBG

FROM CRACKLATINOS

([_kienmanowar_](#))



I. Giới thiệu chung

Trong phần 5, tôi đã đề cập tới các câu lệnh liên quan tới việc tính toán cũng như các lệnh logic, nói tóm lại chưa có gì đặc biệt hơn và có thể gây nhầm chán với những anh em nào đã quá Pr0 khả khả ☺. Trong phần 6 của loạt tuts này tôi xin giới thiệu tới các bạn về các câu lệnh so sánh và các lệnh nhảy có điều kiện. Đây là những kiến thức nền tảng cơ bản và quan trọng để có thể đi tiếp các phần tiếp theo. Xin nhắc lại một lần nữa việc cung cấp tất cả các lệnh vượt quá khuôn khổ cho phép của bài viết, cũng như tôi cũng không đủ sức để mà thực hiện điều này. Do đó việc tham khảo thêm các nguồn tài liệu khác để bổ sung thêm kiến thức là điều hết sức cần thiết cho các bạn.

II. Các lệnh so sánh và các lệnh nhảy có điều kiện

Thông thường, khi chúng ta thực hiện việc so sánh là chúng ta so sánh giữa hai đối tượng trở lên và rồi đi đến một quyết định vào đó. Lấy một ví dụ vui : *Người A béo hơn người B do đó suy ra người A ăn nhiều hơn người B* ☺.

Trong Cr@cking, khi chúng ta thực hiện so sánh giữa hai toán hạng thì kết quả của việc so sánh này sẽ quyết định rằng chương trình có thực hiện câu lệnh nhảy bên dưới hay là không. Và đây cũng là những kiến thức cơ bản luôn luôn được đề cập đến trong các bài viết hướng dẫn dành cho Newbie (những người mới bắt đầu làm quen với Crack).

Chúng ta biết rằng khi một chương trình yêu cầu ta phải nhập Serial để đăng kí, thì bản thân chương trình đó sẽ quyết định xem liệu rằng cái dãy serial mà chúng ta nhập vào kia có thỏa mãn (đúng) hay không thỏa mãn thông qua việc tính toán và tiến hành các câu lệnh so sánh. Dựa trên kết quả của việc so sánh này chương trình sẽ đưa ra quyết định có thực hiện lệnh nhảy hay là không, điều này tùy thuộc vào chúng ta có nhập serial không, serial của chúng ta nhập vào đúng hay sai ?

1. Lệnh CMP :

Đây là câu lệnh so sánh rất thường gặp trong quá trình chúng ta phân tích chương trình, các điều kiện nhảy thường được cung cấp bởi lệnh CMP này. Tổng quát câu lệnh CMP có dạng như sau :

CMP Đích, Nguồn

Lệnh này thực hiện việc so sánh toán hạng Đích với toán hạng Nguồn bằng cách lấy toán hạng Đích trừ đi toán hạng Nguồn. Có thể nói lệnh này tương đương với lệnh SUB nhưng nó khác với lệnh SUB ở chỗ kết quả không được lưu lại (tức là toán hạng đích không bị thay đổi). Tác dụng chủ yếu của lệnh CMP là tác động lên các cờ. Một điểm chú ý khác nữa là

các toán hạng của lệnh CMP không thể cùng là các ô nhớ. Vậy tổng kết lại lệnh này thường được dùng để tạo cờ cho các lệnh nhảy có điều kiện.

Dưới đây là bảng trích dẫn các cờ chính theo quan hệ Đích và Nguồn khi so sánh 2 số không dấu :

	CF	ZF	SF
Đích = Nguồn	0	1	0
Đích > Nguồn	0	0	0
Đích < Nguồn	1	0	1

Ta lấy ví dụ trực quan trong Olly :

a) CMP EAX, ECX

Ở đây tôi giả sử giá trị của 2 thanh ghi EAX và ECX là bằng nhau, kết quả so sánh (kết quả của phép trừ) sẽ không làm thay đổi giá trị của EAX mà nó sẽ tác động lên các cờ, trong đó cờ được thiết lập sẽ là cờ Z. Ok , trong Olly chúng ta làm như sau :

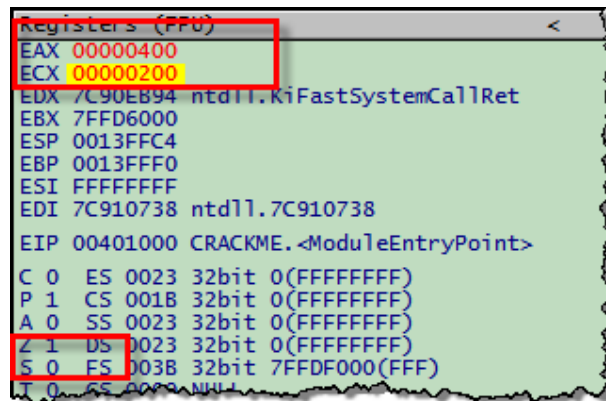
Bây giờ tôi nhấn F8 để trace qua lệnh CMP, quan sát tại cửa sổ Register thì ta thấy rằng giá trị của hai thanh ghi EAX và ECX không hề thay đổi. Tuy nhiên cờ Z được thiết lập.

Mở rộng vấn đề ra một chút, nếu bên dưới của câu lệnh so sánh trên tôi có một lệnh nhảy vậy thì sẽ có hai khả năng xảy ra. Câu lệnh nhảy này sẽ được thực hiện hoặc không thực hiện phụ thuộc vào trạng thái của cờ (mà cụ thể ở đây là cờ Z). Trong trường hợp ví dụ trên tôi giả sử bên dưới là lệnh JZ, lệnh này sẽ thực hiện nhảy nếu như cờ Z được thiết lập

là 1 và không nhảy nếu như cờ Z được thiết lập là 0. Cũng với ví dụ trên tôi giả sử serial mà tôi nhập vào được lưu vào thanh ghi EAX, còn Serial do chương trình tính toán ra được đặt trong thanh ECX. Nếu EAX = ECX, vậy tức là câu lệnh CMP sẽ làm cho cờ Z được bật lên thành 1 và câu lệnh JZ sẽ nhảy tới vùng code dành cho người dùng đã đăng kí hợp pháp, còn ngược lại nếu EAX không bằng ECX thì lúc này lệnh CMP sẽ thiết lập cờ Z là 0 và như vậy chúng ta sẽ bị đưa đến vùng code mà sẽ bắn ra Nag thông báo!! ☹

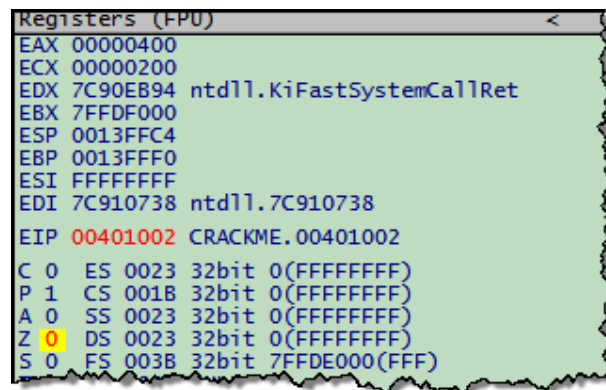
Ngoài việc có tác động lên cờ Z thì lệnh CMP còn tác động lên cả cờ S. Tôi lấy một ví dụ cụ thể như sau :

b) CMP EAX, ECX trong đó EAX lớn hơn ECX



```
Registers (FPU)
EAX 00000400
ECX 00000200
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD6000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 0018 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0038 32bit 7FFDF000(FFF)
T 0 GS 0020 32bit 0(FFFFFFFF)
```

Bây giờ chúng ta nhấn F8 để thực hiện lệnh CMP và quan sát xem giá trị của các cờ thay đổi ra sao.



```
Registers (FPU)
EAX 00000400
ECX 00000200
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 0018 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 0038 32bit 7FFDE000(FFF)
T 0 GS 0020 32bit 0(FFFFFFFF)
```

Ta thấy cờ Z đã được thiết lập là 0 lý do là vì EAX không bằng ECX, đồng thời cờ S cũng được thiết lập là 0 đó là vì EAX lớn hơn ECX (EAX-ECX cho chúng ta kết quả là một số dương). Vậy trong trường hợp EAX nhỏ hơn ECX thì sao, tôi lấy thêm một ví dụ minh họa nữa :

c) CMP EAX, ECX trong đó EAX nhỏ hơn ECX

```

Registers (FPU)
EAX 00000100
ECX 00000200
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDE000(FFF)
T 0 GS 0000 NULL

```

Giờ chúng ta thực hiện lệnh và quan sát sự thay đổi :

```

Registers (FPU)
EAX 00000100
ECX 00000200
EDX 7C90EB94 ntdll.KiFastSystemCall
EBX 7FFDF000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 1 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 7FDE000(FFF)
T 0 GS 0000 NULL

```

Cờ S lúc này được bật lên thành 1 đó là vì kết quả của EAX – ECX sẽ cho chúng ta một số âm.

Vậy chúng ta thấy rằng việc quyết định có thực hiện lệnh nhảy hay không sẽ phụ thuộc vào trạng thái của các cờ. Bên cạnh lệnh so sánh giữa hai thanh ghi, thì CMP còn cho phép chúng ta thực hiện so sánh giữa thanh ghi và một ô nhớ (DWORD, WORD hoặc BYTE). Lấy ví dụ minh họa như sau :

Address	Hex dump	Disassembly	Comment
00401000	3B05 00504000	CMP EAX, DWORD PTR DS:[405000]	
00401005	90	NOB	

```

Registers (FPU)
EAX 00000390
ECX 0013FFB0
EDX 7C90EB94 ntdll.KiFastSystemCall
EBX 7FFD9000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.00401000
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FDE000(FFF)
T 0 GS 0000 NULL

```

DS:[00405000]=00000100
EAX=00000390

Thực hiện lệnh so sánh và quan sát kết quả, ta sẽ thấy rằng lệnh so sánh này sẽ lấy giá trị của EAX trừ đi giá trị tại ô nhớ [405000] là 1000. Kết quả của phép trừ này sẽ là âm và do đó cờ S sẽ được thiết lập là 1.

```

Registers (FPU)
EAX 00000390
ECX 0013FFB0
EDX 7C90EB94 ntdll.KiFast
EBX 7FFD9000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C9107
EIP 00401006 CRACKME.0040
C 1 ES 0023 32bit 0(FFFF
P 1 CS 001B 32bit 0(FFFF
A 0 SS 0023 32bit 0(FFFF
Z 0 DS 0023 32bit 0(FFFF
S 1 FS 003B 32bit 7FFDF0
T 0 GS 0000 NULL

```

2. Lệnh TEST :

Cũng giống như lệnh CMP, lệnh TEST thực hiện thao tác giữa toán hạng Đích và Nguồn mà không làm thay đổi các toán hạng, kết quả cũng không được lưu giữ. Nó khác với lệnh CMP ở chỗ lệnh này sử dụng phép AND, lệnh TEST sẽ tác động tới các cờ **SF, ZF và PF**. Các cờ được tạo ra sẽ được dùng làm điều kiện cho các lệnh nhảy có điều kiện. Tổng quát về lệnh TEST như sau :

TEST Đích, Nguồn

Trong đó toán hạng Đích và Nguồn phải chứa dữ liệu cùng độ dài và không được phép đồng thời là hai ô nhớ.

Lấy ví dụ để minh họa: tôi có lệnh TEST EAX, EAX. Câu lệnh TEST này sẽ kiểm tra xem EAX có bằng không hay không ? Trong Olly chúng ta sửa lại như sau :

Address	Hex dump	Disassembly	Comment
00401000	85 C0	TEST EAX, EAX	
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>	LGetModuleHandleA
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX	
0040100C	6A 00	PUSH 0	SetTitle = NULL

Giả sử rằng giá trị của thanh ghi EAX lúc này đang là 0x0 và cờ Z cũng đang có giá trị 0.

```

Registers (FPU)
EAX 00000000
ECX 0013FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD6000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.<ModuleEntryPoint>
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 0 FS 003B 32bit 7FFDF000(F)
T 0 GS 0000 NULL

```

Trong phần trước tôi đã cung cấp cho các bạn bảng tính toán giữa các bit của phép AND rồi, do đó phần này không nhắc lại nữa. Nhấn F8 để trace qua lệnh TEST và quan sát kết quả :

```

Registers (FPU)
EAX 00000000
ECX 0013FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallRe
EBX 7FFD6000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)

```

Ồ cờ Z đã được bật lên, đó là vì kết quả của phép AND giữa hai toán hạng (ở đây là EAX) có giá trị 0x0 sẽ cho ta là 0x0. Khi kết quả bằng 0x0 thì cờ Z của chúng ta sẽ được bật lên.

Tôi lặp lại ví dụ này, nhưng thay vào đó EAX sẽ mang một giá trị khác 0x0. Ví dụ :

```

Registers (FPU)
EAX 00000390
ECX 0013FFB0
EDX 7C90EB94 ntdll.KiFastSystemCallP
EBX 7FFD6000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
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 0 FS 003B 32bit 7FFDF000(FFF)

```

Ok, nhấn F8 và quan sát cờ Z sẽ thay đổi thế nào :

```

Registers (FPU)
EAX 00000390
ECX 0013FFB0
EDX 7C90EB94 ntdll.KiFastSystemCal
EBX 7FFD6000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
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 0 FS 003B 32bit 7FFDF000(FFF)

```

Cờ Z không được bật lên, điều này là hiển nhiên bởi vì kết quả của phép AND cho chúng ta một giá trị khác 0x0. Chúng ta thử tính toán xem thế nào, ta đổi giá trị 0x390 sang dạng Binary (1110010000) và thực hiện phép AND.

$$\begin{array}{r}
 1110010000 \\
 1110010000 \\
 \hline
 1110010000
 \end{array}$$

Vậy như các bạn thấy ở trên kết quả sau khi tính toán sẽ là khác 0x0 cho nên cờ Z không được bật lên, và một điều đặc biệt ở trên là kết quả sau khi tính toán giống hệt với hai toán hạng của chúng ta.

3. Các lệnh nhảy :

Bảng tổng hợp các lệnh nhảy (chi tiết về các lệnh nhảy sẽ được đề cập ở bên dưới) :

Mnemonic	Meaning	Jump Condition
JA	Jump if Above	CF=0 and ZF=0
JAE	Jump if Above or Equal	CF=0
JB	Jump if Below	CF=1
JBE	Jump if Below or Equal	CF=1 or ZF=1
JC	Jump if Carry	CF=1
JCXZ	Jump if CX Zero	CX=0
JE	Jump if Equal	ZF=1
JG	Jump if Greater (signed)	ZF=0 and SF=OF
JGE	Jump if Greater or Equal (signed)	SF=OF
JL	Jump if Less (signed)	SF != OF
JLE	Jump if Less or Equal (signed)	ZF=1 or SF != OF
JMP	Unconditional Jump	unconditional
JNA	Jump if Not Above	CF=1 or ZF=1
JNAE	Jump if Not Above or Equal	CF=1
JNB	Jump if Not Below	CF=0
JNBE	Jump if Not Below or Equal	CF=0 and ZF=0
JNC	Jump if Not Carry	CF=0
JNE	Jump if Not Equal	ZF=0
JNG	Jump if Not Greater (signed)	ZF=1 or SF != OF
JNGE	Jump if Not Greater or Equal (signed)	SF != OF
JNL	Jump if Not Less (signed)	SF=OF
JNLE	Jump if Not Less or Equal (signed)	ZF=0 and SF=OF
JNO	Jump if Not Overflow (signed)	OF=0
JNP	Jump if No Parity	PF=0
JNS	Jump if Not Signed (signed)	SF=0
JNZ	Jump if Not Zero	ZF=0
JO	Jump if Overflow (signed)	OF=1
JP	Jump if Parity	PF=1
JPE	Jump if Parity Even	PF=1
JPO	Jump if Parity Odd	PF=0
JS	Jump if Signed (signed)	SF=1
JZ	Jump if Zero	ZF=1

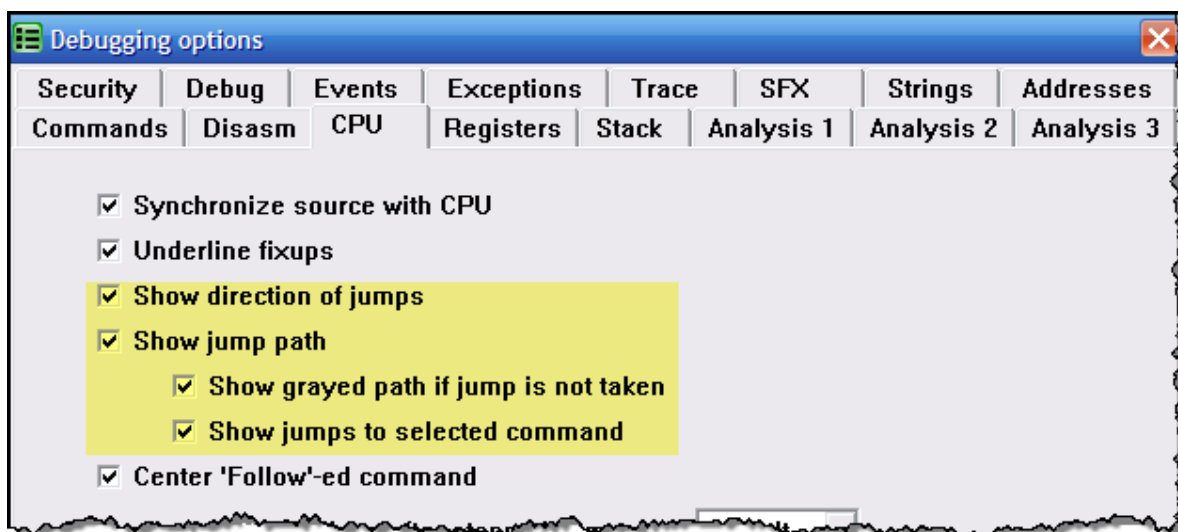
Lệnh JMP

Đây là lệnh nhảy trực tiếp hay nếu dịch sát nghĩa ra thì nó là lệnh nhảy không điều kiện (tức là lệnh nhảy này luôn luôn thực hiện mà không cần bất kì điều kiện nào). Lấy ví dụ trong Olly như sau :

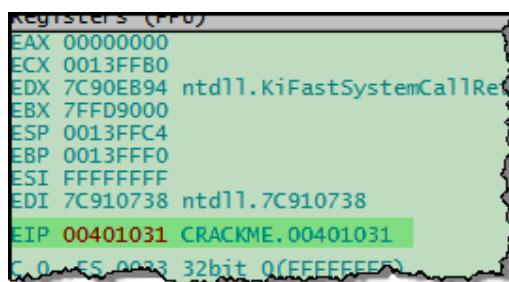
Address	Hex dump	Disassembly	Comment
00401000	EB 2F	JMP SHORT CRACKME.00401031	
00401002	E8 FF040000	CALL <JMP.&KERNEL32.GetModuleHandleA>	GetModuleHandleA
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX	
0040100C	6A 00	PUSH 0	
0040100E	68 F4204000	PUSH CRACKME.004020F4	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	Title = NULL Class = "No need FindWindowA
00401018	0BC0	OR EAX, EAX	
0040101A	74 01	JE SHORT CRACKME.0040101D	
0040101C	C3	RETN	
0040101D	C705 64204000	MOV DWORD PTR DS:[402064], 4003	
00401027	C705 68204000	MOV DWORD PTR DS:[402068], CRACKME.WndProc	
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C], 0	
00401033	C705 70204000	MOV DWORD PTR DS:[402070], 0	

Như quan sát trên hình, khi lệnh JMP thực hiện thì chương trình sẽ chuyển hướng thực thi tới địa chỉ 0x00401031 và thực hiện câu lệnh tiếp theo bắt đầu tại địa chỉ này. Cũng trong

hình minh họa trên, các bạn thấy xuất hiện một mũi tên màu đỏ, mũi tên này chỉ cho ta nơi mà lệnh nhảy sẽ nhảy tới. Để có thể thấy được mũi tên này thì các bạn cấu hình trong Olly như sau :



Bây giờ tôi nhấn F8 để thực thi câu lệnh JMP, lúc này giá trị thanh ghi EIP của tôi sẽ thay đổi thành 0x00401031. Đơn giản là bởi vì thanh ghi EIP luôn luôn trở tới câu lệnh được thực hiện tiếp theo :



Lệnh JE / JZ

Về ý nghĩa thì JE : Nhảy nếu bằng nhau / JZ : Nhảy nếu kết quả bằng không – là hai lệnh nhảy biểu diễn cùng một thao tác nhảy có điều kiện tới một vị trí định trước nếu như cờ ZF = 1. Tôi xét một ví dụ như sau :

Address	Hex dump	Disassembly	Comment
00401000	3BC1	CMP EAX, ECX	
00401002	74 2D	JE SHORT CRACKME.00401031	
00401004	90	NOP	
00401005	90	NOP	
00401006	90	NOP	
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX	
0040100C	6A 00	PUSH 0	
0040100E	68 F4204000	PUSH CRACKME.004020F4	
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>	
00401018	0BC0	OR EAX, EAX	
0040101A	74 01	JE SHORT CRACKME.0040101D	
0040101C	C3	RETN	
0040101D	C705 64204000	MOV DWORD PTR DS:[402064], 4003	
00401027	C705 68204000	MOV DWORD PTR DS:[402068], CRACKME.WndProc	
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C], 0	

Tôi cho giá trị của hai thanh ghi EAX và ECX bằng nhau, và ZF lúc này = 0 :


```

Registers (FPU)
EAX 00002000
ECX 00002000
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFDF000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.<ModuleEntryPoint>
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 0 FS 003B 32bit 7FFDF000(FFF)

```

Do 2 thanh ghi có giá trị bằng nhau nên khi thực hiện lệnh CMP cho ra kết quả bằng 0 nên lúc này cờ ZF sẽ được bật thành 1. Lệnh JE lúc này sẽ dựa vào trạng thái của cờ ZF để quyết định có thực hiện hay không, do lúc này cờ ZF = 1 nên lệnh nhảy này sẽ được thực hiện :

```

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

```

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	74 2D	JE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX, EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064], 4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068], CRACKME.WndProc
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C], 0
0040103E	C705 70204000	MOV DWORD PTR DS:[402070], 0

Lặp lại ví dụ ở trên nhưng lần này giá trị của EAX và ECX sẽ khác nhau, cờ ZF lúc này là 1 :

```

Registers (FPU)
EAX 00002000
ECX 00001000
EDX 7C90EB94 ntdll.KiFastSystemCall
EBX 7FFD6000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDF000(FFF)

```

Nhấn F8 để trace qua lệnh CMP, do thanh ghi EAX có giá trị lớn hơn ECX nên kết quả của việc thực hiện lệnh CMP sẽ cho ta một giá trị khác 0 dẫn đến cờ ZF sẽ được thiết lập lại là 0. Lúc này lệnh JE thấy cờ ZF bị "tắt" cho nên nó không thực hiện lệnh nhảy nữa :

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 0	FS	003B	32bit	7FFDF000(FFF)
T 0	GS	0000	NULL	

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	74 2D	JE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX, EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064], 4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068], CRACKME.WndProc
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C], 0

Khi mà lệnh nhảy JE tại địa chỉ 0x00401002 không được thực hiện thì lệnh tiếp theo sau nó, tức là lệnh NOP tại địa chỉ 0x00401004 sẽ là lệnh tiếp theo được thực thi.

Nhiều lúc trong quá trình làm việc với Olly bạn không muốn can thiệp vào giá trị các thanh ghi để từ đó thay đổi giá trị các cờ. Vậy có cách nào để chúng ta ép chương trình thực hiện lệnh nhảy mà không cần can thiệp vào giá trị thanh ghi để tạo cờ hay không ? Xin thưa, vô cùng đơn giản ☺.

Bạn lập lại ví dụ trên và trace qua lệnh CMP. Ở đây bạn muốn làm sao cho lệnh nhảy dưới lệnh CMP sẽ thực hiện, lẽ ra bạn sẽ phải tính toán và chỉnh lại giá trị các thanh ghi để sao cho khi thực hiện lệnh CMP thì cờ ZF được bật lên thành 1 và lệnh nhảy sẽ được thực hiện. Tuy nhiên ở đây tôi không cần quan tâm các thanh ghi chứa giá trị thế nào, việc đơn giản nhất mà tôi làm là nhấn đúp chuột vào cờ mà tôi muốn **thiết lập / hủy thiết lập**. Cụ thể hơn, các bạn nhìn vào hình trên thì thấy lệnh nhảy sẽ không được thực hiện và cờ Z lúc này đang là 0. Để cho lệnh nhảy sẽ được thực thi tôi nhấn đúp chuột lên cờ ZF, lúc này cờ ZF sẽ được bật lên thành 1 và quan sát cửa sổ CPU các bạn sẽ thấy :

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

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	74 2D	JE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX, EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064], 4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068], CRACKME.WndProc
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C], 0

Lệnh JNE / JNZ

Hai lệnh nhảy này là ngược lại của hai lệnh nhảy JE và JZ, ý nghĩa cụ thể của nó là Nhảy nếu không bằng nhau / Nhảy nếu kết quả không bằng không. Chúng biểu diễn cùng một thao tác nhảy có điều kiện tới một vị trí nếu như cờ ZF = 0. Một ví dụ nhỏ để minh họa :

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	75 2D	JNZ SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP

Registers (FPU)	
EAX	00001234
ECX	00005678
EDX	7C90EB94 ntdll
EBX	7FFD4000
ESP	0013FFC4
EBP	0013FFF0
ESI	FFFFFFFF
EDI	7C910738 ntdll
EIP	00401000 CRACKME
C	0 ES 0023 32bit
P	1 CS 001B 32bit
A	0 SS 0023 32bit
Z	1 DS 0023 32bit
S	0 FS 003B 32bit

Trace qua lệnh CMP, do EAX khác ECX nên cờ ZF được thiết lập lại thành 0. Lệnh JNZ nhận thấy cờ ZF là 0 nên sẽ quyết định thực thi lệnh nhảy.

Registers (FPU)	
EAX	00001234
ECX	00005678
EDX	7C90EB94 n
EBX	7FFD4000
ESP	0013FFC4
EBP	0013FFF0
ESI	FFFFFFFF
EDI	7C910738 n
EIP	00401002 C
C	1 ES 0023 32
P	0 CS 001B 3
A	1 SS 0023 3
Z	0 DS 0023 3

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	75 2D	JNZ SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX
0040100C	6A 00	PUSH 0
0040100E	68 F4204000	PUSH CRACKME.004020F4
00401013	E8 A6040000	CALL <JMP.&USER32.FindWindowA>
00401018	0BC0	OR EAX, EAX
0040101A	74 01	JE SHORT CRACKME.0040101D
0040101C	C3	RETN
0040101D	C705 64204000	MOV DWORD PTR DS:[402064], 4003
00401027	C705 68204000	MOV DWORD PTR DS:[402068], CRACKME.WndProc
00401031	C705 6C204000	MOV DWORD PTR DS:[40206C], 0

Lệnh JS

Đây là lệnh nhảy được thực hiện nếu kết quả âm, tức là lệnh nhảy có điều kiện tới một vị trí định trước nếu SF = 1. Điều này có nghĩa là nếu kết quả âm sau khi thực hiện tính toán các phép toán với các số có dấu thì cờ SF sẽ được thiết lập là 1 và lệnh JS sẽ được thực thi. Ví dụ minh họa :

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	78 2D	JS SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP

```
Registers (FPU)
EAX 00000200
ECX 00000300
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0013FFC4
```

Trace qua lệnh CMP, do $EAX < ECX$ do đó cờ SF sẽ được thiết lập là 1 và lệnh nhảy sẽ được thực thi :

```
C 1 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
```

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	78 2D	JS SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX
0040100C	6A 00	PUSH 0
0040100E	58	PUSH EAX

Lệnh JP / JPE

Đây là hai lệnh nhảy biểu diễn cùng một thao tác nhảy có điều kiện tới một vị trí định trước nếu $PF = 1$ (PF là cờ Parity). Ví dụ minh họa :

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	7A 2D	JPE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP

```
Registers (FPU)
EAX 00000020
ECX 00000018
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.<ModuleEntryPoint>
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 0 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
I 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
```

Giá trị của $EAX = 0x20$, $ECX = 0x18$. Ta trace qua lệnh CMP thì thấy cờ PF vẫn giữ nguyên trạng thái là $PF = 0$.

```
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
I 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
```

Do thành ghi $EAX = 0x20 > ECX = 0x18$, cho nên khi thực hiện lệnh CMP ta sẽ có được kết quả là 0x2. Mà kết quả này nếu đem đi biểu diễn dưới dạng binary là 10. Theo lý thuyết về cờ PF thì tổng số bit 1 trong kết quả này là lẻ cho nên cờ PF không được thiết lập thành 1 và do đó lệnh JPE cũng sẽ không được thực hiện.

Ta cũng lấy ví dụ như trên nhưng lúc này ta cho giá trị của ECX là 0x17. Trace qua lệnh CMP ta có được kết quả như hình minh họa dưới đây :

```

Registers (FPU)
EAX 00000020
ECX 00000017
EDX 7C90EB94 ntdll.KiFastSys
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)

```

Ta thấy cờ PF lúc này đã được bật lên thành 1. Đơn giản là vì kết quả của lệnh CMP là 0x3, biểu diễn dưới dạng nhị phân là 11. Vậy tổng số bit 1 trong kết quả có được là chẵn cho nên cờ PE được bật lên thành 1 và lệnh JPE lúc này sẽ thực thi :

```

00401002 7A 2D JPE SHORT CRACKME.00401031
00401004 90 NOP
00401005 90 NOP
00401006 90 NOP
00401007 A3 CA204000 MOV DWORD PTR DS:[4020CA], EAX
0040100C 6A 00 PUSH 0
0040100E 6A 00 PUSH 0

```

Lệnh JNP / JNPE

Hai lệnh này có điều kiện nhảy ngược với hai lệnh nhảy ở trên, tức là nó chỉ thực hiện khi mà cờ PF mang giá trị 0.

Lệnh JO

Đây là lệnh nhảy có điều kiện tới vị trí định trước nếu OF = 1, tức là xảy ra **tràn** sau khi thực hiện các phép toán với các số có dấu. Ví dụ minh họa :

Address	Hex dump	Disassembly
00401000	03C1	ADD EAX, ECX
00401002	70 2D	JO SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP


```

Registers (FPU)
EAX 7FFFFFFF
ECX 00000001
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.ModuleEntryPoint>

```

Trace qua lệnh ADD và quan sát cửa sổ CPU các bạn sẽ thấy cờ JO được bật lên :

```

Registers (FPU)
EAX 80000000
ECX 00000001
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 0 DS 0023 32bit 0(FFFFFFFF)
S 1 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 1 LastErr ERROR_SUCCESS (00000000)

```

Lệnh JNO

Lệnh này là ngược lại với lệnh JO.

Lệnh JB

Lệnh này sẽ thực hiện nếu cờ CF = 1. Lấy ví dụ :

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	72 2D	JB SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP

```

Registers (FPU)
EAX 00001000
ECX 00002000
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 0 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)

```

Giá trị của EAX ở đây tôi cho nhỏ hơn giá trị của ECX. Trace qua câu lệnh CMP, kết quả là một giá trị < 0 và cờ CF được thiết lập là 1.

```

Registers (FPU)
EAX 00001000
ECX 00002000
EDX 7C90EB94 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 1 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
D 0
O 0 LastErr ERROR_SUCCESS (00000000)

```


Ngược với lệnh JB là lệnh JNB (các bạn tự tìm hiểu thêm về lệnh này).

Lệnh JBE

Nhảy nếu thấp hơn hoặc bằng, lệnh này sẽ được thực hiện nếu cờ CF = 1 hoặc cờ ZF = 1. Tức là nếu ta có EAX = ECX thì lệnh nhảy sẽ được thực hiện hoặc nếu EAX < ECX thì lệnh nhảy cũng sẽ được thực hiện. Ví dụ :

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	76 2D	JBE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP

Nếu ta cho EAX bằng ECX :

Registers (FPU)	
EAX	00001000
ECX	00001000
EDX	7C90EB94 ntdll.KiFastSystemCall
EBX	7FFD7000
ESP	0013FFC4
EBP	0013FFF0
ESI	FFFFFFFF
EDI	7C910738 ntdll.7C910738
EIP	00401000 CRACKME.<ModuleEntry>
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 0	FS 003B 32bit 7FFDF000(FFF)

Trace qua lệnh CMP, kết quả của CMP là 0 cho nên cờ ZF được thiết lập là 1 :

Registers (FPU)	
EIP	00401002 CRACKME.00401002
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)
Z 1	DS 0023 32bit 0(FFFFFFFF)
S 0	FS 003B 32bit 7FFDF000(FFF)
T 0	GS 0000 NULL

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	76 2D	JBE SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP
00401006	90	NOP
00401007	A3 CA204000	MOV DWORD PTR DS:[4020CA], EAX
0040100C	6A 00	PUSH 0

Nếu ta cho EAX < ECX :

Registers (FPU)	
EAX	00000500
ECX	00001000
EDX	7C90EB94 ntdll.KiFastSystemCall
EBX	7FFD7000
ESP	0013FFC4
EBP	0013FFF0
ESI	FFFFFFFF
EDI	7C910738 ntdll.7C910738
EIP	00401002 CRACKME.00401002
C 0	ES 0023 32bit 0(FFFFFFFF)
P 1	CS 001B 32bit 0(FFFFFFFF)
A 0	SS 0023 32bit 0(FFFFFFFF)

Trace qua lệnh CMP, lúc này cờ CF sẽ được bật thành 1 ☺.

```
Registers (FPU)
EAX 00000500
ECX 00001000
EDX 7C90EB94 ntdll.KiFastSystemCa
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401002 CRACKME.00401002
C 1 ES 0023 32bit 0(FFFFFFFF)
P 1 CS 001B 32bit 0(FFFFFFFF)
```

Lệnh JL

Lệnh này thực hiện khi mà cờ SF và cờ OF khác nhau. Một ví dụ minh họa với trường hợp EAX và ECX là hai số dương và EAX lớn hơn ECX.

Address	Hex dump	Disassembly
00401000	3BC1	CMP EAX, ECX
00401002	7C 2D	JL SHORT CRACKME.00401031
00401004	90	NOP
00401005	90	NOP

```
Registers (FPU)
EAX 00002000
ECX 00001000
EDX 7C90EB94 ntdll.KiFastSystemCa
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
EIP 00401000 CRACKME.<ModuleEntry>
```

Trace qua lệnh CMP, quan sát sẽ thấy lệnh Jmp không được thực hiện bởi vì EAX lớn hơn ECX, kết quả của CMP là một số dương do đó các cờ S và O không bị tác động.

```
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 0 FS 003B 32bit 7FFDF000(FFF)
T 0 GS 0000 NULL
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
```

Ngược lại nếu tôi lấy giá trị của EAX < ECX và lặp lại ví dụ trên :

```
Registers (FPU)
EAX 00001000
ECX 00002000
EDX 7C90EB94 ntdll.KiFastSystemCa
EBX 7FFD7000
ESP 0013FFC4
EBP 0013FFF0
ESI FFFFFFFF
EDI 7C910738 ntdll.7C910738
```

Thực hiện CMP , quan sát kết quả ta thấy như sau :

```

C 1 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
D 0
O 0 LastErr ERROR_SUCCESS (00000000)
FFI 00000287 (NO_B, NE, BE, S, PE, I, LE)

```

SF != OF, cho nên lệnh nhảy JL sẽ được thực hiện.

Okie tôi nghĩ đến đây là đã đủ cho phần 6 về Ollydbg , bài viết xin được kết thúc tại đây mặc dù còn một số lệnh nhảy tôi chưa trình bày (cái này để dành cho các bạn tự tìm hiểu). Trong phần này tôi có tham khảo thêm một số tài liệu liên quan tới lập trình ASM. Tôi tin là với những ví dụ minh họa trực quan như ở trên các bạn sẽ nắm được vấn đề nhanh hơn. Tuy nhiên câu lệnh của ASM không phải chỉ có thế, các bạn có thể tham khảo thêm các tài liệu liên quan để có được một cái nhìn sâu hơn. Hẹn gặp lại các bạn trong phần 7 của loạt bài viết về Olly, By3 By3!! ☺

Best Regards

[Kienmanowar]



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

My family, Computer_Angel, Moonbaby , Zombie_Deathman, Littleboy, Benina, QHQCrkcr, 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, dzungltn, 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 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:

kienmanowar[at]reaonline.net