

## Tut3

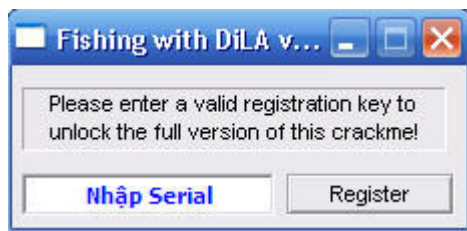
Target	Fwdv4, fwdv5.
Độ khó	Cơ bản
Packed	N/A
Công cụ	Olly Dbg 1.10, PEiD 09.3, Máy tính ☺
Mục tiêu	Tim Serial

Tiếp theo tut2, hôm nay tôi sẽ cùng các bạn thực hành với 2 Crackme tiếp theo, trước khi tiếp tục tôi muốn bạn nhớ lại:

- Các bước cần thiết khi Crack một Target.
  - Cách sử dụng Olly Dbg mức căn bản.
  - Kiến thức về BreakPoint.
  - Ý nghĩa của các câu lệnh câu lệnh JE, CMP, MOV, MOVSX và DEC.
  - Ý nghĩa của thao tác Follow in Dump.
  - Cách thay đổi giá trị trong bộ nhớ và lưu lại file trong Olly.
- ...

## fwdv4 Crackme

Chạy thử Crackme ta thấy như sau:



Giao diện quá quen thuộc ☺

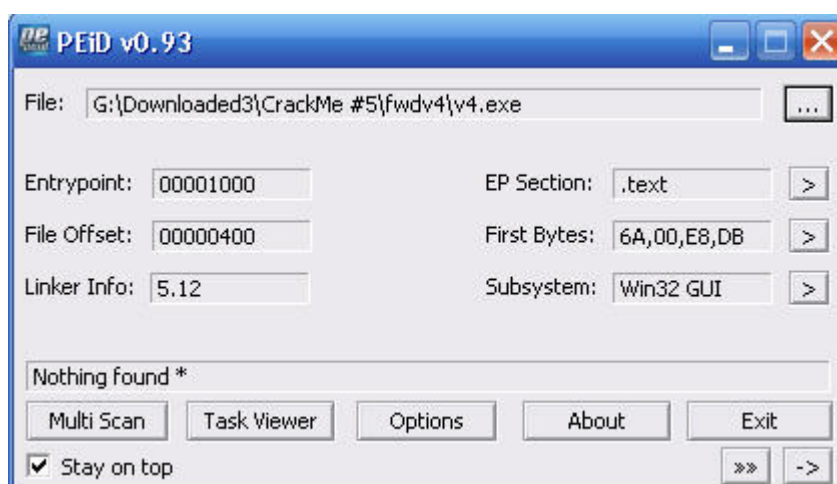
Nhập thử một số bất kỳ vào ô nhập Serial – tôi nhập là **30041991**, sau đó nhấn OK, một cái Nag hiện ra:



Crackme này không có nhiều thông tin để ta tìm hiểu, với một ít thông tin có được, chúng ta bắt tay vào Crack...

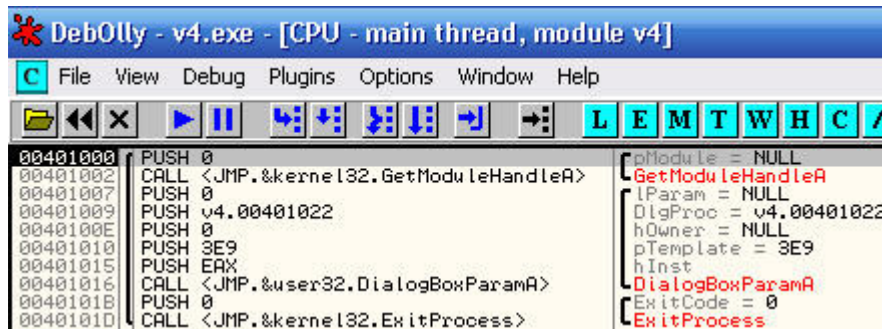
Kiểm tra:

Gần như chắc chắn ta đã biết kết quả khi kiểm tra với PEiD ☺, hãy thử xem sao:

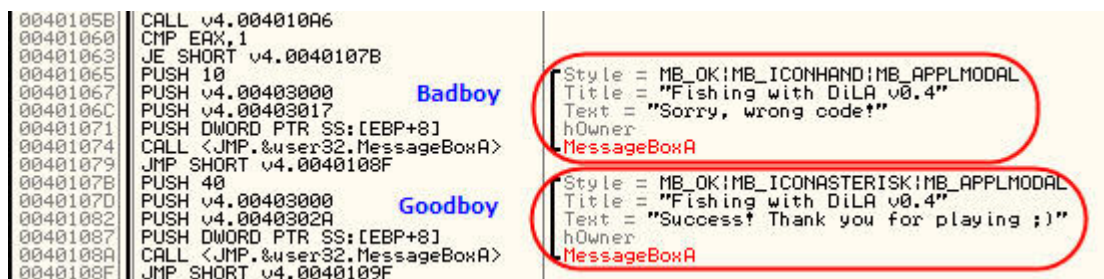


Không sao cả, hãy mở Olly lên ☺.

Nhấn F3 và chọn Crackme, chờ Olly load xong chúng ta ở đây:



Xuống dưới một chút ta thấy đoạn code sau:



Với những kiến thức có được từ hai tut trước, chúng ta thấy một số điều sau:

- Trên Badboy có một lệnh nhảy JE sẽ nhảy đến Goodboy (tại sao ?)
- Lệnh nhảy đó sẽ thực thi nếu EAX thỏa mãn phép sánh CMP EAX, 1 tức là  $EAX = 1$ .

...

Tuy nhiên có thể thấy chúng ta không thấy có câu lệnh nào liên quan đến giá trị của EAX trong đoạn code trên, tuy nhiên điều đó không có nghĩa là chúng ta không thể quyết định được giá trị của EAX, hãy chú ý câu lệnh Call ở trên câu lệnh CMP EAX, 1.

Kiến thức:

-Câu lệnh Call là một câu lệnh gọi đến một đoạn code nào đó, đoạn code đó thực thi một nhiệm vụ nhất định (có thể là chuyển đổi, tính toán một cái gì đấy ☺).

Như vậy dù chúng ta chỉ nhìn thấy một câu lệnh Call nhưng bên trong câu lệnh Call đó có thể có rất nhiều lệnh và biết đâu một

trong số đó lại có ích cho ta ☺, bây giờ cái mà chúng ta cần là vào trong câu lệnh Call đó và xem xét...

Đặt một BP tại câu lệnh Call đó (bạn có hiểu tôi định làm gì không ?):

```
0040105B CALL v4.004010A6
```

sau đó nhấn F9 để Run Target, tôi điền Serial là 30041991, sau đó nhấn nút Register và Oly dừng tại BP:

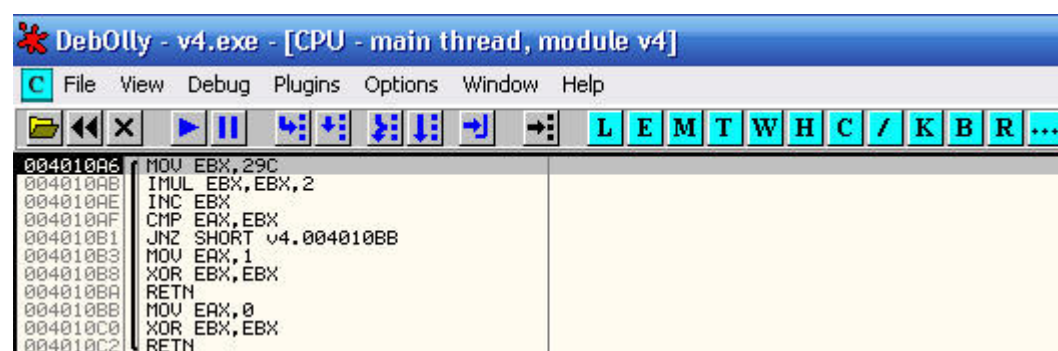
```
0040105B CALL v4.004010A6
```

Bây giờ để vào xem các code trong câu lệnh Call này ta có hai cách:

-Nhấn Enter: Thao tác này giúp ta biết được các code trong hàm Call đó nhưng thực chất thì chúng ta không hề thực thi hàm Call này, bạn hãy hiểu đơn giản là chúng ta chỉ vào để xem xét nhưng quá trình xem xét này không ảnh hưởng đến việc thực thi của Target...

-Nhấn F7 (Step into): Khi bạn nhấn F7 có nghĩa là bạn đã thực thi hàm Call này rồi (tuy nhiên không phải là thực thi hẳn mà bạn chỉ thực thi quá trình gọi đến đoạn code bên trong hàm Call), việc nhấn F7 với Newbie có lợi ích là nếu bạn không hiểu biết rõ về mã ASM dựa vào giá trị của các thanh ghi bạn cũng có thể đoán được phần nào ý nghĩa của một câu lệnh. Việc nhấn F7 thường kèm theo sau đó là những lần nhấn F8 (Step over) qua từng câu lệnh, hãy cùng thực hành để nắm rõ hơn lý thuyết...

Nhấn F7 để vào trong lệnh Call, ta thấy như sau:



```
004010A6 CALL v4.004010A6
004010A8 MOV EBX, 29C
004010AB IMUL EBX, EBX, 2
004010AE INC EBX
004010AF CMP EAX, EBX
004010B1 JNZ SHORT v4.004010BB
004010B3 MOV EAX, 1
004010B5 XOR EBX, EBX
004010B7 RETN
004010B9 MOV EAX, 0
004010BB XOR EBX, EBX
004010BD RETN
```

Kiến thức cần nhớ:

- Câu lệnh **IMUL X, X, Y** → Phép nhân có dấu – sau phép tính này giá trị của **X = X x Y**, ví dụ **X = 3** và **Y = 5** thì sau phép nhân này **X = 3 x 5 = 15**.
- Câu lệnh **INC X** → Trái ngược với câu lệnh **DEC X** (là câu lệnh để làm gì) – Sau phép toán này **X = X +1**, ví dụ **X = 3** thì sau câu lệnh này **X = X +1 = 3 +1 = 4**.
- Câu lệnh **XOR X, X** → Ý nghĩa của câu lệnh này giống như câu lệnh **MOV X, 0** – Có nghĩa là dù **X** đang có giá trị bao nhiêu thì sau câu lệnh này giá trị của **X** cũng = 0.
- Câu lệnh nhảy có điều kiện **JNZ – Jump if not Zero** – Nhảy nếu khác 0 → Trong nhiều trường hợp câu lệnh này lại mang một ý nghĩa khác, ví dụ như có một đoạn code sau:

```
1111  CMP X, Y
2222  JNZ 4444
3333  ...
4444
```

Thì trong đoạn code trên câu lệnh nhảy **JNZ** sẽ thực thi nếu **X** không bằng **Y**.

- Câu lệnh **RET – Return** – Ví dụ bạn có một lệnh **Call** trong đoạn code sau:

```
1111  Call
2222  ....
```

Nếu bạn nhấn **F7** để vào trong hàm **Call** này và đến đoạn code sau:

```
1234  ...
1235  ...
1236  RET
```

Thì sau khi thực thi câu lệnh **RET** bạn sẽ trở về câu lệnh ở địa chỉ **2222**.

Nếu bạn không hiểu lắm cũng không sao, dần dần bạn sẽ hiểu thôi – Hãy tin tôi đi ☺

Với những kiến thức có được cho đến lúc này, bạn hiểu gì về đoạn code trên... Hãy tự suy nghĩ trước khi đọc tiếp...:)

.....

.....

.....

Tiếp tục nhé, trở lại đoạn code và dịch từng câu lệnh theo cách hiểu của chúng ta:

004010A6 /\$>MOV EBX,29C → Gán EBX = 29C  
004010AB |.>IMUL EBX,EBX,2 → EBX = EBX x 2  
004010AE |.>INC EBX → EBX = EBX + 1  
004010AF |.>CMP EAX,EBX → So sánh EAX và EBX  
004010B1 |.>JNZ SHORT v4.004010BB → Nhảy nếu EAX khác EBX  
004010B3 |.>MOV EAX,1 → Gán EAX = 1  
004010B8 |.>XOR EBX,EBX  
004010BA |.>RETN → Thoát khỏi lệnh Call  
004010BB |>>MOV EAX,0 → Câu lệnh JNZ nhảy tới đây, gán EAX = 0  
004010C0 |.>XOR EBX,EBX  
004010C2 \.>RETN → Thoát khỏi lệnh Call

Hãy nhớ lại một chút, điều chúng ta cần là EAX = 1, nhìn vào những câu lệnh trên ta thấy EAX chỉ có thể = 1 khi EAX = EBX (bạn có hiểu vì sao không ?)  
Có thể thấy:

- Ban đầu EBX được gán = 29C (mã HEX)
- Sau đó  $EBX = EBX \times 2 = 29C \times 2 = 538$  (mã HEX – Tính bằng máy tính Windows)
- Cuối cùng  $EBX = EBX + 1 = 538 + 1 = 539$ .
- Lấy EBX so sánh với EAX, nếu bằng thì ta thành công ☺



Tuy nhiên ta cần biết EAX đang bằng bao nhiêu tại câu lệnh **CMP EAX, EBX**...

Bây giờ chúng ta hãy nhấn F8 để xem xét đoạn code:

Kiến thức:

-Việc nhấn F8 (Step over) thực chất là ta set BP tại câu lệnh ngay bên dưới câu lệnh đang xét và thực thi việc nhấn F9, như vậy mỗi lần nhấn F8 ta thấy là ta đi qua một câu lệnh một, giai đoạn Step over qua từng câu lệnh giúp ta phân tích rõ câu lệnh đang xét đồng thời có thể thấy được giá trị của các thanh ghi tại câu lệnh đó...

Quá trình nhấn F8:

Nhấn F8 lần 1:

004010AB | IMUL EBX,EBX,2 | EBX 0000029C

Nhấn F8 lần 2:

004010AE | INC EBX | EBX 00000538

Nhấn F8 lần 3:

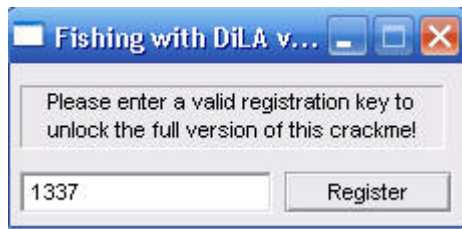
004010AF | CMP EAX,EBX | EBX 00000539 | EAX 01CA6787

Như vậy tại câu lệnh **CMP EAX, EBX** thì **EBX = 539** (đúng như ta đã tính) và **EAX = 1CA6787** (có lẽ không cần nói nhiều về giá trị này nữa – Đây là mã HEX của 30041991)

Như vậy chỉ cần **EAX = 539** là ta sẽ có Goodboy, dùng máy tính Windows để chuyển đổi ta được Serial thực là :

The image shows two screenshots of the Windows calculator. The top screenshot shows the number 539 in the decimal input field, with the 'Hex' radio button selected, resulting in the hex value 00000217. The bottom screenshot shows the number 1337 in the decimal input field, with the 'Hex' radio button selected, resulting in the hex value 0539.

Mở Crackme lên và điền vào **1337**:



Nhấn nút Register và...:

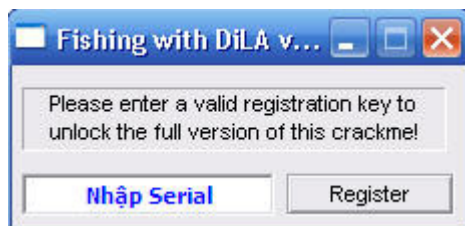


Vậy là xong...

Hãy nghỉ ngơi một chút trước khi đến với Crackme tiếp theo 😊.....  
.....  
.....  
.....

fwdv5 Crackme

Chạy thử Crackme ta thấy như sau:



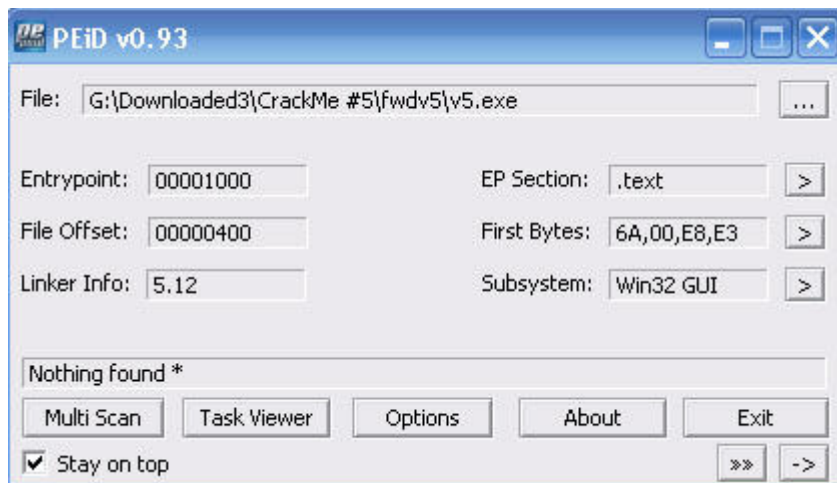
Giao diện quá quen thuộc 😊



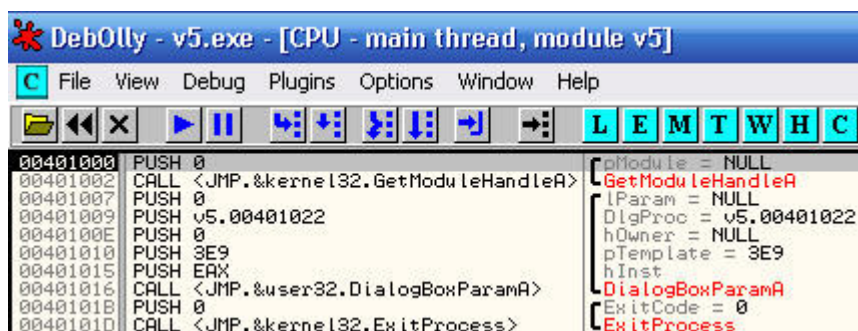
Nhập thử một số bất kỳ vào ô nhập Serial – tôi nhập là 30041991, sau đó nhấn OK, một cái Nag hiện ra:



Vẫn là cái nag đó, kiểm tra bằng PEiD (tôi luôn nhắc bạn hãy nhớ thao tác này, dù nó có nhàm chán thì bạn vẫn cần làm vì nó sẽ rất có ích cho bạn trong việc định hướng quá trình Crack của mình):



Mở Olly lên, nhấn F3 chọn Crackme, chờ Olly load xong chúng ta ở đây:



Xuống dưới một chút và ta thấy đoạn code sau:

```
0040104B PUSH EAX
0040104C PUSH 05.004010BF
00401051 CALL 05.0040109E
00401056 CMP EAX,1
00401059 JE SHORT 05.00401071
0040105B PUSH 10
0040105D PUSH 05.00403000 Badboy
00401062 PUSH 05.00403017
00401067 PUSH DWORD PTR SS:[EBP+8]
0040106A CALL <JMP.&user32.MessageBoxA>
0040106F JMP SHORT 05.00401085
00401071 PUSH 40
00401073 PUSH 05.00403000 Goodboy
00401078 PUSH 05.0040302A
0040107D PUSH DWORD PTR SS:[EBP+8]
00401080 CALL <JMP.&user32.MessageBoxA>
```

Style = MB\_OK!MB\_ICONHAND!MB\_APPLMODAL  
Title = "Fishing with DiLA v0.5"  
Text = "Sorry, wrong code!"  
hOwner  
MessageBoxA

Style = MB\_OK!MB\_ICONASTERISK!MB\_APPLMODAL  
Title = "Fishing with DiLA v0.5"  
Text = "Success! Thank you for playing ;)"  
hOwner  
MessageBoxA

Tương tự như Target trước, chúng ta thấy một số điều sau:

- Trên Badboy có một lệnh nhảy JE sẽ nhảy đến Goodboy.
- Lệnh nhảy đó sẽ thực thi nếu EAX thỏa mãn phép sánh CMP EAX, 1 tức là  $EAX = 1$ .
- Trên câu lệnh CMP EAX, 1 có một lệnh Call ?!?!?
- Trên câu lệnh Call có một lệnh PUSH EAX → Đây là câu lệnh lưu giá trị hiện tại của EAX vào bộ nhớ...

...

Đặt một BP tại lệnh Call ở địa chỉ 00401051 (để làm gì ?), sau đó nhấn F9 để chạy Target, tôi điền là 30041991 sau đó nhấn nút Register, Olly **dừng** tại BP:

```
00401051 CALL 05.0040109E
```

Nhấn F7 để Step into hàm Call này, chúng ta đến đoạn code sau:

```
0040109E ADD ESP,10
004010A1 SUB ESP,0C
004010A4 XOR AX,0DEAF
004010A8 ROL EAX,10
004010AB MOV EBX,3ADAFCCF
004010B0 CMP EAX,EBX
004010B2 JNZ SHORT 05.004010BC
004010B4 MOV EAX,1
004010B9 XOR EBX,EBX
004010BB RETN
004010BC XOR EBX,EBX
004010BE RETN
004010BF POP EAX
004010C0 ADD AH,20
004010C3 NEG EAX
004010C5 PUSH 05.004010A1
004010CA RETN
004010CB INT3
```

Để hiểu đoạn code trên, tôi muốn bạn biết thêm một số lệnh trong ngôn ngữ ASM:

- **SUB X, Y** → Phép toán  $X = X - Y$ , ví dụ  $X = 3$ ,  $Y = 1$  thì sau phép toán này  $X = X - Y = 3 - 1 = 2$ .


- **ADD X, Y** → Phép toán  $X = X + Y$ , ví dụ  $X = 1, Y = 2$  thì sau câu lệnh này  $X = X + Y = 1 + 2 = 3$ .
- **NEG X** → Phép đảo dấu giá trị  $X$ , ví dụ như  $X = 1$  thì sau câu lệnh này  $X = -1$ .
- **XOR X, Y** → Phép XOR hai giá trị  $X$  và  $Y$  (có thể dùng máy tính Windows để tính. Lưu ý nếu  $X \text{ XOR } Y = Z$  thì  $X = Y \text{ XOR } Z$  hoặc  $Y = X \text{ XOR } Z$  ...)
- **ROL X, Y** → Phép xoay bit sang trái (ở hệ nhị phân, hiển thị bởi các ký tự 0 và 1).
- **AX** (tương tự cho BX, CX...) – Lấy 4 bit của EAX – Ví dụ nếu EAX là 123456 thì AX là 1234...
- **AH** (tương tự cho BH, CH...) – Lấy 2 bit của EAX tại vị trí 5 & 6. Ví dụ EAX = 12345678 thì AH = 56.

Trong các lệnh trên, trừ lệnh **ROL** ra, thì các câu lệnh còn lại đều dễ dàng tính được dựa vào máy tính Windows, tuy nhiên nếu không biết một câu lệnh thì ta cũng không thể tìm Serial – có lẽ bạn nghĩ vậy. Thực sự tôi không biết rõ thuật toán của câu lệnh **ROL**, nhưng tôi có thể đoán được dựa vào giá trị các thanh ghi qua từng câu lệnh – đó chính là một lợi thế của Olly, hãy thử xem chúng ta Fishing Serial như thế nào qua đoạn code trên.

Nhấn F8 hai lần (hãy luôn nhớ cái ta cần quan tâm lúc này là giá trị của EAX thay đổi thế nào):

004010A4 | . 66:35 AFDE | XOR AX,0DEAF | EAX 01CA6787

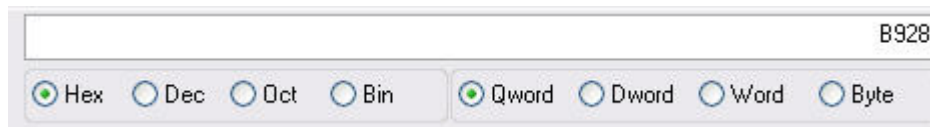
EAX lúc này = **1CA6787** → AX = **6787**, dùng máy tính Windows cho phép XOR AX, 0DEAF :



Nhấn nút XOR và điền tiếp:



Nhấn Enter:



Trở lại Olly và nhấn tiếp F8 một lần nữa:

```
004010A8 | . C1C0 10 | ROL EAX,10 | EAX 01CAB928
```

Như vậy là chúng ta đã tính đúng ( $EAX = 1CAB928 \rightarrow AX = B928$ )

Với câu lệnh ROL EAX, 10 lúc này thực sự là tôi không biết thuật toán là gì cả, vì vậy hãy thử nhấn F8 một lần nữa để xem giá trị của EAX thay đổi ra sao:

Sau khi nhấn tiếp F8:

```
004010AB | . BB CFFFD93A | MOV EBX,3ADAFFCF | EAX B92801CA
```

Bạn thấy sao:

-Ban đầu  $EAX = 01CAB928$

-Sau câu lệnh ROL EAX, 10 giá trị  $EAX = B92801CA$ , như vậy có nghĩa là từ chuỗi  $EAX = 01CAB928$  ta cắt lấy chuỗi B928 và gán lên đầu chuỗi còn lại (lúc này là 01CA).

Như vậy ta có thể đoán phần nào ý nghĩa của câu lệnh ROL EAX, 10 này ☺

Tiếp theo, câu lệnh này:

```
004010AB | . BB CFFFD93A | MOV EBX,3ADAFFCF
```

Sẽ gán giá trị của  $EBX = 3ADAFFCF$

Nhấn F8 thêm lần nữa:

```
004010B0 | . 3BC3 | CMP EAX,EBX | EAX B92801CA | EBX 3ADAFFCF
```

Ta thấy hiển nhiên là EAX không bằng EBX, tiếp tục nhấn F8 ta thấy như sau:

004010B2	75 08	JNZ SHORT v5.004010BC
004010B4	B8 01000000	MOV EAX,1
004010B9	33DB	XOR EBX,EBX
004010BB	C3	RETN
004010BC	33DB	XOR EBX,EBX
004010BE	C3	RETN

Vì EAX không bằng EBX nên câu lệnh JNZ được thực thi, ta thấy rằng nếu câu lệnh nhảy này không nhảy thì ta sẽ đến câu lệnh MOV EAX, 1 sau đó đến một câu lệnh RET → Chúng ta cần xem lệnh RET đó đưa chúng ta trở về đâu...

Rõ ràng là lệnh nhảy JNZ đã thực thi nhưng ta vẫn có cách để nó không thực thi, bạn làm như sau:

-Nhìn sang bên cửa sổ Registers (FPU) ta thấy:

```

EIP 004010B2 v5.004010B2
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)
S 0 FS 003B 32bit 7FFDD000(FFF)
T 0 GS 0000 NULL
D 0
O 1 LastErr ERROR_SUCCESS (00000000)

```

Bạn hãy chú ý giá trị của chữ Z (thực sự ta phải gọi là cờ Z) lúc này = 0, bạn cần biết rằng giá trị của cờ Z trong các lệnh nhảy như JE, JNE, JNZ ... hay là cờ S với các lệnh nhảy như JG, JB... sẽ quyết định xem lệnh nhảy đó có thực thi hay không ( ví dụ như thực thi nếu Z = 1 hoặc ngược lại). Như vậy trong trường hợp này nếu không muốn lệnh nhảy JNZ thực thi ta hãy thay Z = 1, để làm điều đó, ta nhấn đúp chuột vào số 0 sau chữ Z, sau đó ta thấy như sau:

```

C 0 ES 0023 32bit 0(FFFFFFFF)
P 0 CS 001B 32bit 0(FFFFFFFF)
A 1 SS 0023 32bit 0(FFFFFFFF)
Z 1 DS 0023 32bit 0(FFFFFFFF)
S 0 FS 003B 32bit 7FFDD000(FFF)
T 0 GS 0000 NULL
D 0
O 1 LastErr ERROR_SUCCESS (00000000)

```

004010B2	75 08	JNZ SHORT v5.004010BC
004010B4	B8 01000000	MOV EAX,1
004010B9	33DB	XOR EBX,EBX
004010BB	C3	RETN
004010BC	33DB	XOR EBX,EBX
004010BE	C3	RETN

Vậy là lệnh nhảy JNZ đã không thực thi, hãy F8 tiếp tục đến câu lệnh RET rồi nhấn F8 thêm một lần nữa ta đến đoạn code sau:

004010BF	58	POP EAX
004010C0	80C4 20	ADD AH,20
004010C3	F7D8	NEG EAX
004010C5	68 A1104000	PUSH v5.004010A1
004010CA	C3	RETN
004010CB	CC	INT3

Dịch nghĩa các câu lệnh



**POP EAX** → Gán trở lại EAX = 1CA6787 (Giá trị đã lưu từ câu lệnh PUSH EAX)

**ADD AH, 20** → AH = AH + 20

**NEG EAX** → EAX = -EAX

Từ những nhận xét trên ta thấy đoạn code ban đầu không làm thay đổi giá trị của EAX vì suy cho cùng rồi EAX cũng nhận lại giá trị ban đầu của nó là mã HEX của Serial ta nhập vào...

Nhấn F8 cho đến câu lệnh RET và nhấn thêm một lần F8 nữa ta đến:

004010A1	. 83EC 0C	SUB ESP,0C
004010A4	. 66:35 AFDE	XOR AX,0DEAF
004010A8	. C1C0 10	ROL EAX,10
004010AB	. BB CFFFD3A	MOV EBX,3ADAFFCF
004010B0	. 3BC3	CMP EAX,EBX
004010B2	. 75 08	JNZ SHORT 05.004010BC
004010B4	. B8 01000000	MOV EAX,1
004010B9	. 33DB	XOR EBX,EBX
004010BB	. C3	RET
004010BC	. 33DB	XOR EBX,EBX
004010BE	. C3	RET

Đây là một đoạn code quen thuộc, vì vậy hãy nhấn F8 cho đến câu lệnh JNZ:

004010B2	. 75 08	JNZ SHORT 05.004010BC
004010B4	. B8 01000000	MOV EAX,1
004010B9	. 33DB	XOR EBX,EBX
004010BB	. C3	RET
004010BC	. 33DB	XOR EBX,EBX
004010BE	. C3	RET

Đổi giá trị cờ Z để câu lệnh nhảy này không thực thi, sau đó tiếp tục nhấn F8 cho qua câu lệnh RET (bạn hãy hiểu rằng tôi đang muốn biết chúng ta sẽ phải lặp với đoạn code này bao nhiêu lần), sau khi qua câu lệnh RET chúng ta trở lại:

00401056	. 83F8 01	CMP EAX,1
00401059	. 74 16	JE SHORT 05.00401071

Như vậy là chúng ta đã thoát ra ngoài câu lệnh Call → chúng ta sẽ lặp hai lần tại đoạn code này:

004010A1	. 83EC 0C	SUB ESP,0C
004010A4	. 66:35 AFDE	XOR AX,0DEAF
004010A8	. C1C0 10	ROL EAX,10
004010AB	. BB CFFFD3A	MOV EBX,3ADAFFCF
004010B0	. 3BC3	CMP EAX,EBX
004010B2	. 75 08	JNZ SHORT 05.004010BC
004010B4	. B8 01000000	MOV EAX,1
004010B9	. 33DB	XOR EBX,EBX
004010BB	. C3	RET
004010BC	. 33DB	XOR EBX,EBX
004010BE	. C3	RET



Nhiệm vụ của chúng ta bây giờ là phải tính toán ngược lại để tìm “Serial thực”...

Như vậy ta cần : Tại lần lặp thứ 2:

→ Sau câu lệnh ROL EAX, 10 thì EAX phải = 3ADA FFCF (để EAX = EBX)

→ Trước câu lệnh ROL EAX, 10 thì EAX phải = FFCF3ADA (theo ta đoán từ sự thay đổi giá trị của EAX hồi này – không chắc sẽ đúng nhưng cứ thử xem sao) → AX = 3ADA.

→ Như vậy trước câu lệnh XOR EAX, 0DEAF thì giá trị của AX = DEAF XOR 3ADA (kiến thức cơ bản về câu lệnh XOR)

Dùng máy tính Windows ta tính được giá trị EAX = E495 (mã HEX) → sau một lần qua đoạn code thì EAX phải = FFCFE475.

Trước đó EAX phải qua đoạn code này:

004010BF	58	POP EAX
004010C0	80C4 20	ADD AH,20
004010C3	F7D8	NEG EAX
004010C5	68 A1104000	PUSH v5.004010A1
004010CA	C3	RET
004010CB	CC	INT3

Như vậy để tính ngược thì ta phải theo thứ tự sau:

1. EAX = -EAX
2. EAX = EAX – 20 (mã HEX).

Dùng máy tính Windows, ta được kết quả của EAX = - EAX = FFFFFFFF00301B8B → Nhưng vì mỗi thanh ghi chỉ có thể lưu giá trị tối đa là 8 bit → EAX = 00301B8B

AH = AH – 20 (nhớ chọn mã HEX) → kết quả AH = FB → EAX = 0030FB8B

Vì ta biết rằng chỉ tại lần lặp thứ 2 của đoạn code giá trị của EAX mới thực sự được quyết định → nếu ta suy luận đúng thì  $EAX = 30FB8B$  chuyển sang hệ  $DEC = 3210123...$

Thử với Target:



Cuối cùng cũng xong...

Còn tiếp...  
**Còn tiếp...**

**Xin gửi lời Cảm ơn đến:**

NhatPhuongLe, 3rr0r, HTS, moth, Benina, hacnho, Unregistered  
!, Merc, Rongchaua, TQN, Why not Bar, TrickyBoy, **ZOMBIE**  
nhc1987, ::Xcross87::..., lena151, Registered, sonlata,  
[www.VnCeRt.info](http://www.VnCeRt.info), [www.REAOnline.net](http://www.REAOnline.net),

**Và những ai đọc tut này...**

====P.E Onimusha====  
07/09/2007

