ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 3

MÔN: KIẾN TRÚC MÁY TÍNH & HỢP NGỮ

CRACKING

THỰC HIỆN:

18120195 – Vương Thị Ngọc Linh

18120247 – Phạm Hồ Ngọc Trâm

18120261 – Phạm Hoàng Việt

18120284 – Lê Trọng Bằng

18120304 – Võ Văn Hoàng Danh

Thành phố Hồ Chí Minh – Tháng 6/2020

MỤC LỤC

CHƯƠNG 1: GIỚI THIỆU

- Trong phần đồ án này ta sẽ sử dụng những kiến thức về hợp ngữ x86 để crack phần mềm (vì mục đích học tập).
- Đồ án gồm :
  + Báo cáo mô tả các bước cracking câu 1, 2, 3.
  + File .cpp và .exe keygen câu 2, 3.
- [MSSV các thành viên +] mod 3 + 1 = 1 = 24 mod 3 = 0 + 1 = Đề 1.

CHƯƠNG 2: NHỮNG PHẦN ĐÃ HOÀN THÀNH VÀ CHƯA HOÀN THÀNH

Công việc	Mức độ hoàn thành	Ghi chú
Crack câu 1	100%	
Crack câu 2	100%	
Crack câu 3	100%	
Keygen câu 2	100%	
Keygen câu 3	100%	

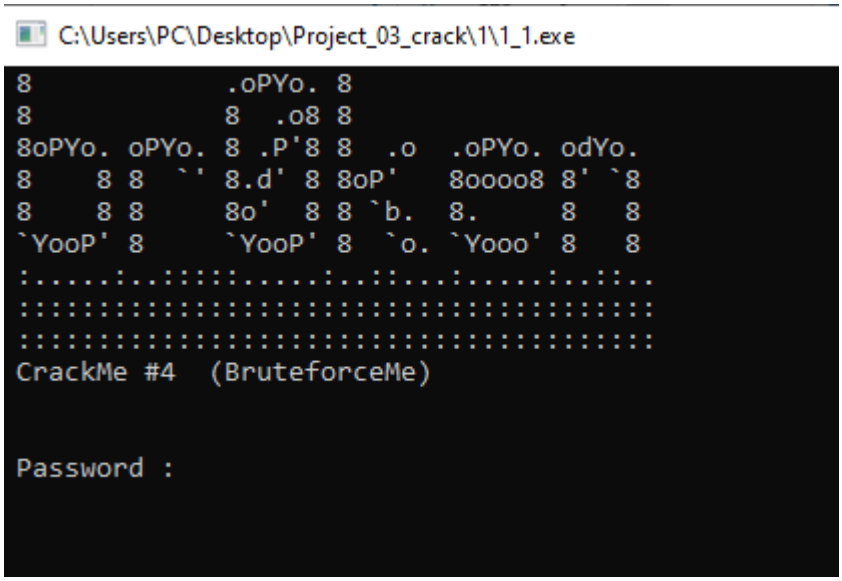
CHƯƠNG 3: PHÂN CHIA CÔNG VIỆC

Công việc	Người phụ trách	Ghi chú
Crack câu 1	Bằng + Danh	
Crack câu 2	Việt + Danh	
Crack câu 3	Linh	
Keygen câu 2	Việt	
Keygen câu 3	Trâm	
Báo cáo	Linh + Trâm	

CHƯƠNG 4: CÂU 1

4.1 Thông tin cơ bản:

- Khởi động chương trình.



- Password là 1 chuỗi kí tự (chưa biết có bao nhiêu kí tự).

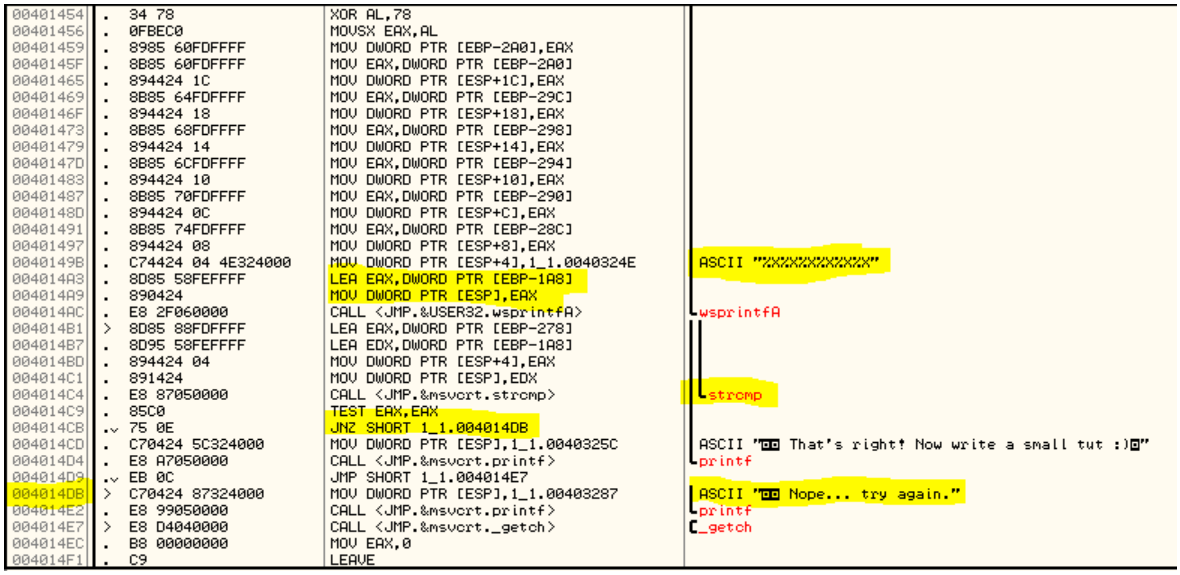
- Khi nhập password sai, chương trình sẽ thông báo "**Nope… try again**."

- Để tìm được password, ta cần tìm tất cả text string. Click chuột phải

-> chọn Search for -> All references text strings.

- Ta tìm thấy thông báo khi nhập sai password, click vào địa chỉ của thông báo này.

- Kiểm tra chương trình, ta thấy thông báo khi nhập đúng password là "**That's right! Now write a small tut :)**.



- Ta thấy trong chương trình, ở địa chỉ **004014CB**, có lệnh JNZ SHORT 1_1.004014DB. Địa chỉ này chính là địa chỉ xuất ra thông báo khi ta nhập sai password.

- Ngay trên lệnh JNZ đó, ta có lời gọi hàm CALL strcmp( ), nghĩa là chương trình sẽ thực hiện so sánh 2 chuỗi nếu trùng khớp sẽ xuất thông báo password đúng và ngược lại.

- Trên lời gọi hàm strcmp() chương trình có thêm vào stack 2 dữ liệu được lấy từ 2 địa chỉ là [EBP-278] và [EBP-1A8]

- > Ta cần xác định đâu là dữ liệu cố định của chương trình, đâu là dữ liệu mà người dùng nhập vào.

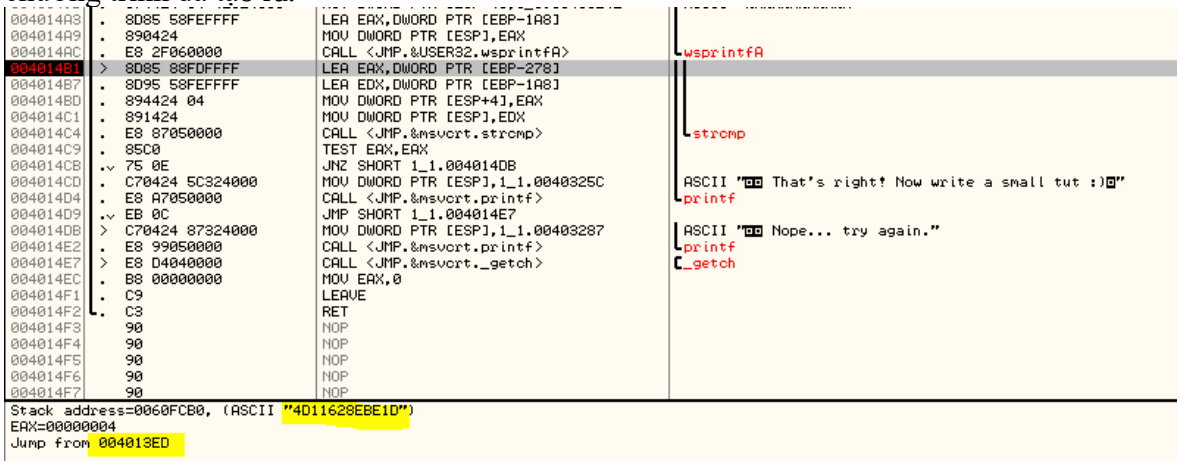
- Tại địa chỉ **004014A3**, chương trình thự hiện dòng lệnh lưu địa chỉ của [EBP-1A8] vào thanh ghi EAX.

- Sau đó đưa địa chỉ này vào đầu stack [ESP] rồi gọi hàm wsprint à địa chỉ [EBP-1A8] ẽ lưu 1 chuỗi sau khi đã được thự hiện 1 số bước xử lí ở các dòng lệnh phía trên.

-> Ta có thể đoán được dữ liệu tại địa chỉ **[EBP-1A8]** là dữ liệu mà người dùng nhập vào và dữ liệu **[EBP-278]** là dữ liệu cố định trong chương trình.

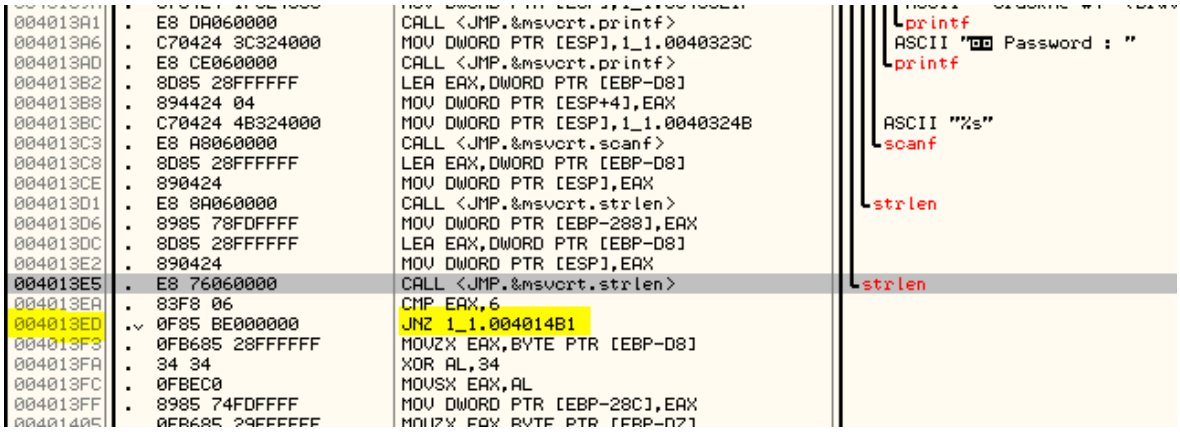
4.2 Debug bằng OllyDgb:

- Ta đặt Break point tại **địa chỉ 004014B1** và tiến hành debug để tìm ra giá trị cố định mà chương trình đã tạo ra.



- Xác định được giá trị tại địa chỉ **[EBP-278]** là "**4D11628EBE1D"**

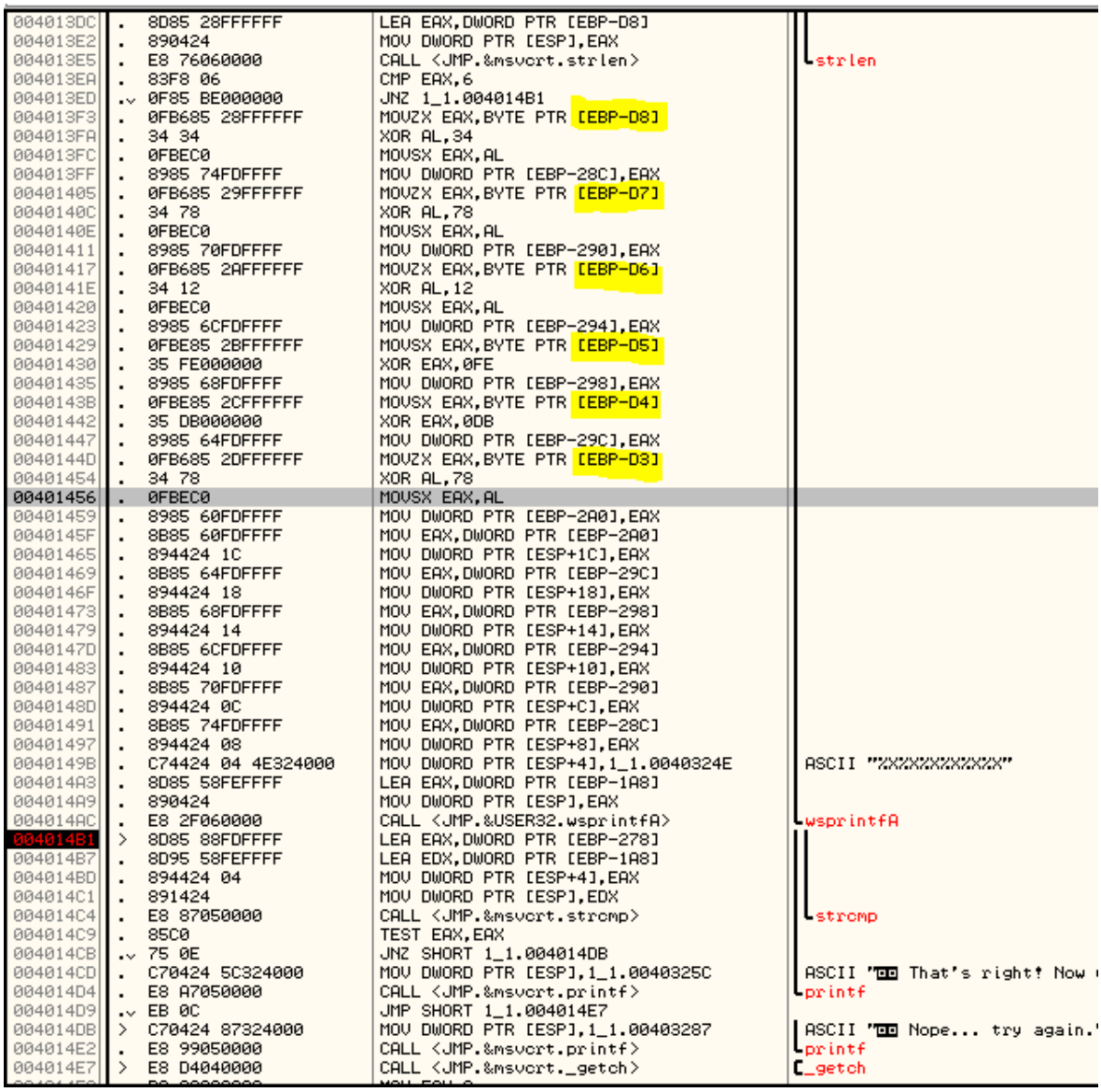
- Ngoài ra, ta thấy dòng "Jump from **004013ED**" nghĩa là dòng này được nhảy đến từ dòng **004013ED** ta tìm đến địa chỉ này.



- Ta thấy, lệnh “CMP EAX,6”. Mặc khác, sau khi thực hiện lời gọi hàm “scanf”, password người dùng nhập vào sẽ được lưu tại địa chỉ **[EBP-D8],** tiếp theo chương trình thực hiện lời gọi hàm strlen để đếm độ dài của chuỗi nhập vào rồi lưu độ dài vào thanh ghi EAX.

-> Password là 1 chuỗi có độ dài là 6.

- Lúc này chương trình sẽ so sánh độ dài chuỗi người dùng nhập vào với 6, nếu khác nhau sẽ thông báo password sai, ngược lại thì sẽ tiếp tục xử lí chuỗi này.



- Sau khi kiểm tra độ dài chuỗi đã thỏa mãn điều kiện, chương trình tiếp tục thực hiện 1 số lệnh bên dưới.

- Tại địa chỉ **004013F3**, chương trình lấy 1 byte tại địa chỉ **[EBP-D8],** tức là lấy kí tự đầu tiên của chuỗi người dùng nhập vào lưu vào 8 bit thấp (AL) của thanh ghi EAX, đồng thời các bit trái còn lại được điền là những số 0.

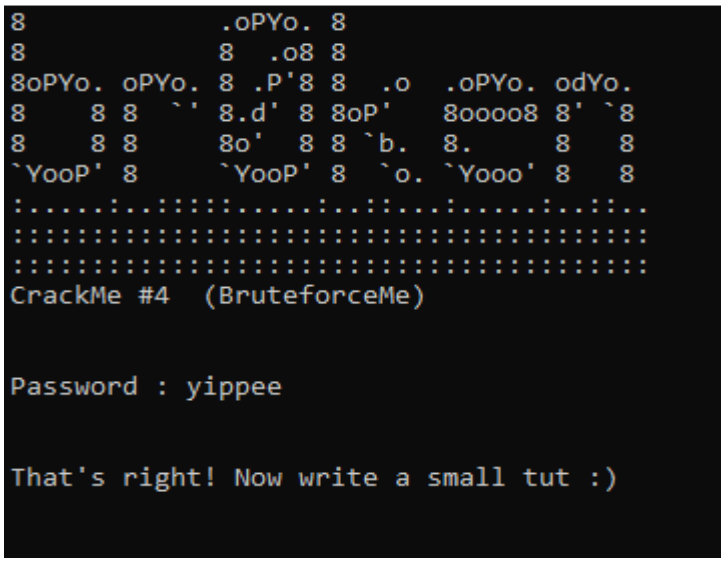
- Theo quy tắc xor, **A xor B = C -> A = C xor B.**

- A là password cần tìm để nhập đúng nên để tìm A ta lấy C (các giá trị lưu trong AL) xor lần lượt với 34, 78, 12, 0FE, 0DB, 78

-Sau khi xor, chương trình lưu giá trị này tại địa chỉ [EBP-28C].

	Hash[0]	Hash[1]	Hash[2]	Hash[3]	Hash[4]	Hash[5]
RES	4D	11	62	8E	BE	1D
XOR	34	78	12	0FE	0DB	78
HEX	79	69	70	70	65	65
ASCII	y	i	p	p	e	e

-> **Password là yippee**

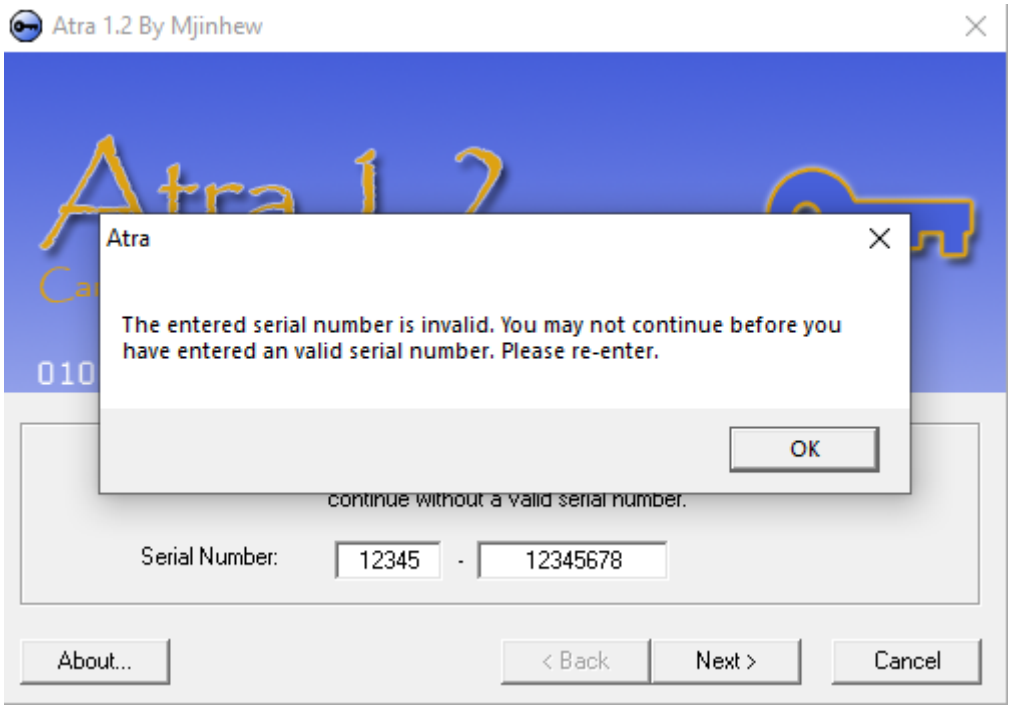


CHƯƠNG 5: CÂU 2

5.1 Thông tin cơ bản:

Người dùng nhập vào một chuỗi Serial gồm 2 phần, phần 1 – 5 ký tự, phần 2 – 8 ký tự (như hình dưới)

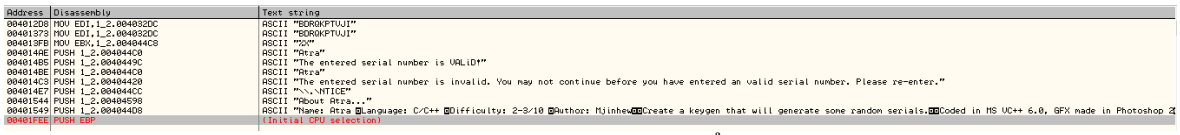
Nếu nhập sai thì sẽ hiện ra thông báo: "**The entered serial number is invalid. You may not continue before you have entered an valid serial number. Please re-enter.**"



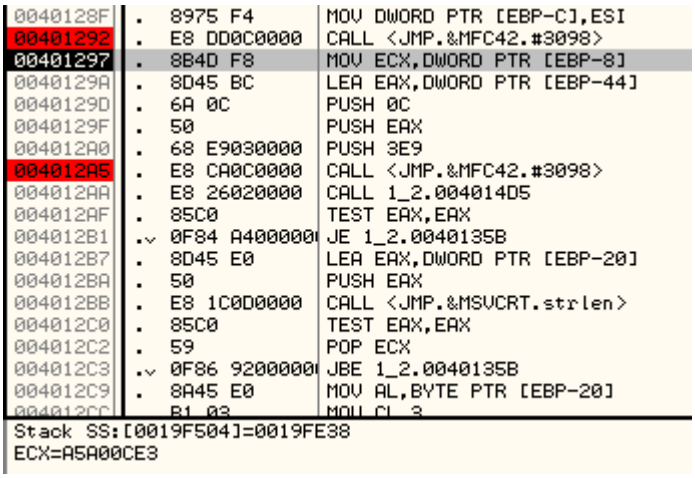
5.2 Debug chương trình bằng OllyDbg:

Tìm các text strings trong chương trình (chuột phải → Search for → All reference text strings)

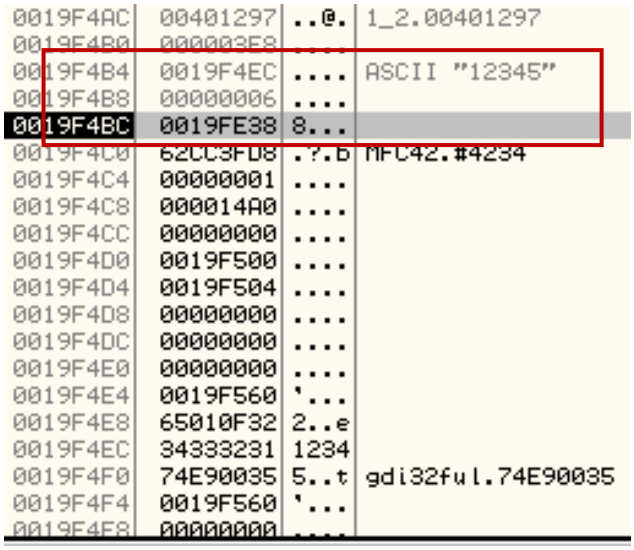
Ta thấy thông báo nhập đúng là **"The entered serial bumber is VALiD!"**



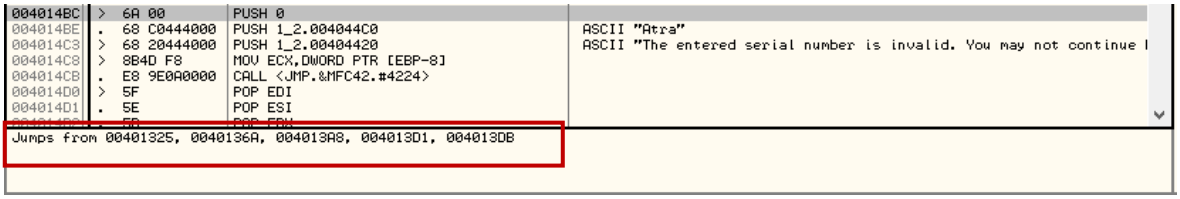
Đặt break-point tại dòng **00401292** và **004012A5**. F9 để Run -> F8 :



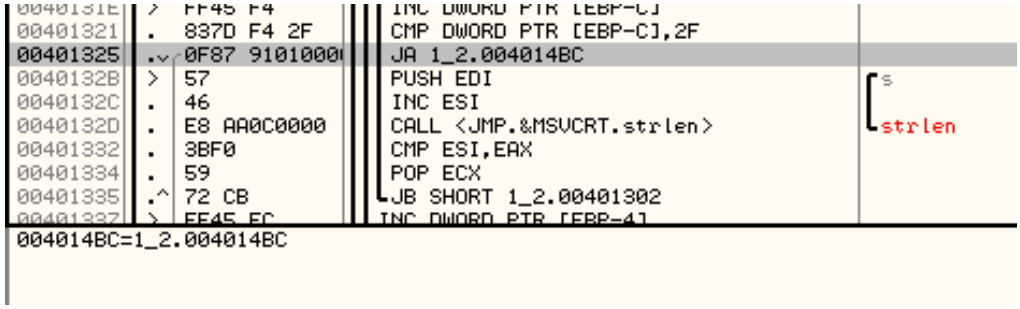
Chuỗi nhập vào được lưu tại:



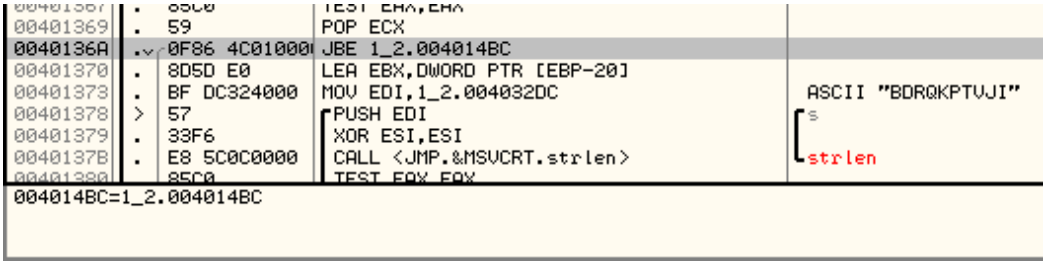
Tại vị trí in ra "badboy", ta thấy nó được jump từ các dòng: **00401325, 0040136A, 004013A8, 004013D1, 004013DB.**



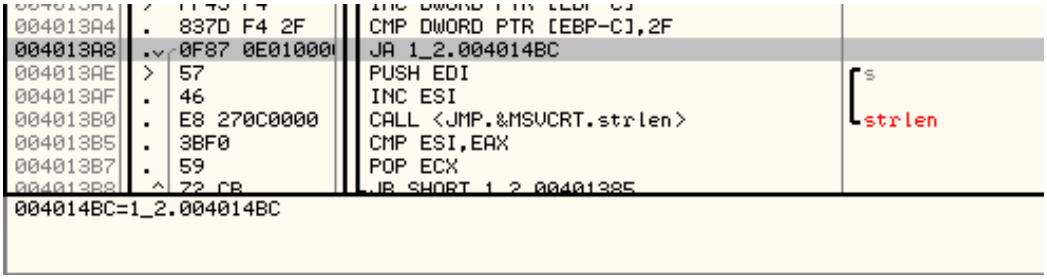
- Dòng **00401325:**



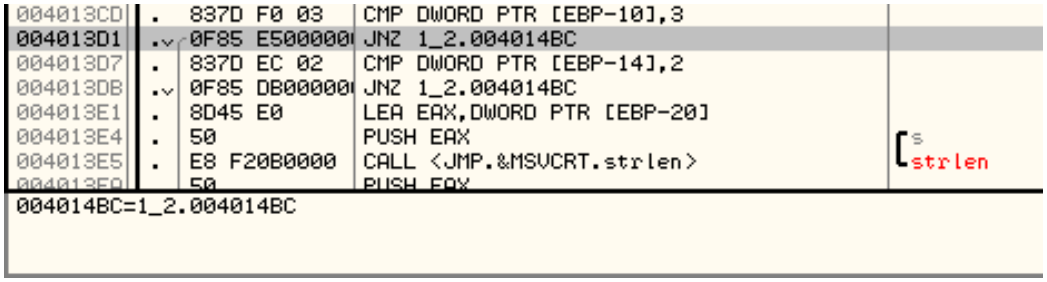
- Dòng **0040136A:**

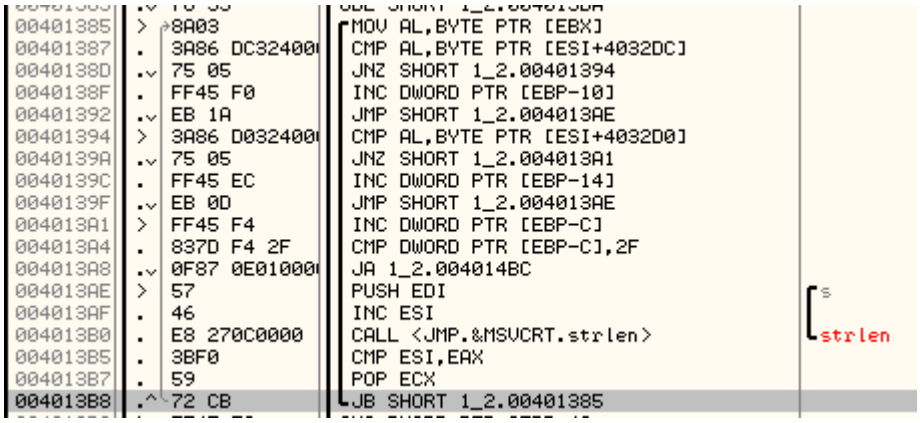


- Dòng **004013A8:**



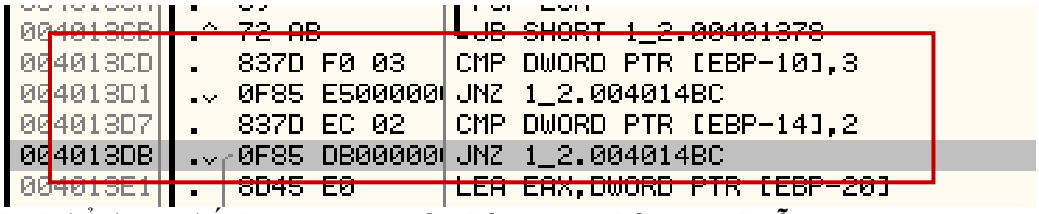
- Dòng **004013D1:**





Tại dòng **00401373** ta thấy có một chuỗi ASCII được quy định sẵn là "**BDRQKPTVJI**". Đoạn code từ dòng **00401385** đến dòng **004013B8** đếm số ký tự trong chuỗi part 1 mà người dùng nhập vào mà có xuất hiện trong chuỗi "**BDRQKPTVJI**" và lưu vào *[EBP-10]*, sau đó kiểm tra số ký tự trong chuỗi part 1 có xuất hiệnt trong chuỗi "**0123456789**" và lưu vào *[EBP-14]*.

Đoạn code tại dòng **004013CD** đếm xem *[EBP-10]* có bằng 3 không, nếu không nhảy đến "badboy". Tương tự code tại dòng **004013D7** kiểm tra *[EBP-14]* có bằng 2 không, nếu không thì nhảy đến "badboy".



Từ đó có thể đưa ra kết luận: **Part 1 có 5 kí tự với 3 kí tự từ chuỗi "BDRQKPTVJI" và 2 kí tự từ chuỗi "0123456789".**

5.2.1 Xử lý part 1:

Sau khi thực hiện việc so sánh part 1 với các chuỗi ở *[EBP-10]* và *[EBP-14]*, và đều đúng thì chương trình tiếp tục thực hiện cho đến dòng **004013EF** và jump đến **00401D5E.**

```
00401D5E  r$  55            PUSH EBP
00401D5F  .   8BEC          MOV EBP,ESP
00401D61  .   83EC 68       SUB ESP,68
00401D64  .   8D45 98       LEA EAX,DWORD PTR [EBP-68]
00401D67  .   50            PUSH EAX
00401D68  .   E8 3AF8FFFF   CALL 1_2.004015A7
00401D6D  .   FF75 0C       PUSH DWORD PTR [EBP+C]
00401D70  .   8D45 98       LEA EAX,DWORD PTR [EBP-68]
00401D73  .   FF75 08       PUSH DWORD PTR [EBP+8]
00401D76  .   50            PUSH EAX
00401D77  .   E8 53F8FFFF   CALL 1_2.004015CF
00401D7C  .   8D45 F0       LEA EAX,DWORD PTR [EBP-10]
00401D7F  .   50            PUSH EAX
00401D80  .   8D45 98       LEA EAX,DWORD PTR [EBP-68]
00401D83  .   50            PUSH EAX
00401D84  .   E8 2CFFFFFF   CALL 1_2.00401CB5
00401D89  .   8B45 FC       MOV EAX,DWORD PTR [EBP-4]
00401D8C  .   83C4 18       ADD ESP,18
00401D8F  .   3345 F8       XOR EAX,DWORD PTR [EBP-8]
00401D92  .   3345 F4       XOR EAX,DWORD PTR [EBP-C]
00401D95  .   3345 F0       XOR EAX,DWORD PTR [EBP-10]
00401D98  .   C9            LEAVE
00401D99  L.  C3            RET
```

Sau khi thực hiện các lệnh trong đoạn từ **00401D5E** đến **00401D99**, **RET** quay về dòng **004013FF**.

Đoạn code từ **004013F4** đến **00401405** là tiến hành chuyển kết quả thu được thành hệ hexa ở dạng string.

```
004013F4  .  8B3D B031400(  MOV EDI,DWORD PTR [<&MSVCRT.sprintf>]   MSVCRT.sprintf
004013FA  .  50             PUSH EAX                                <%X>
004013FB  .  BB C8444000    MOV EBX,1_2.004044C8                    ASCII "%X"
00401400  .  8D45 D4        LEA EAX,DWORD PTR [EBP-2C]
00401403  .  53             PUSH EBX                                format => "%X"
00401404  .  50             PUSH EAX                                s
00401405  .  FFD7           CALL EDI                                sprintf
00401407  .  8365 FC 00     AND DWORD PTR [EBP-4],0
0040140B  .  8D45 D4        LEA EAX,DWORD PTR [EBP-2C]
0040140E  .  50             PUSH EAX                                s
0040140F  .  E8 C80B0000    CALL <JMP.&MSVCRT.strlen>               strlen
00401414  .  83C4 1C        ADD ESP,1C
00401417  .  85C0           TEST EAX,EAX
00401419  .v 76 29          JBE SHORT 1_2.00401444
```

Dòng **00401419**, jump đến dòng **00401444** nếu *below or equal*.

Đoạn code sau khi jump đến dòng **00401444:**

```
00401444  > 8365 FC 00      AND DWORD PTR [EBP-4],0
00401448  .  8D45 D4        LEA EAX,DWORD PTR [EBP-2C]
0040144B  .  50             PUSH EAX                                s
0040144C  .  E8 8B0B0000    CALL <JMP.&MSVCRT.strlen>               strlen
00401451  .  85C0           TEST EAX,EAX
00401453  .  59             POP ECX
00401454  .v 76 29          JBE SHORT 1_2.0040147F
```

Nếu không thì tiếp tục thực hiện đoạn code từ **0040141B** đến **00401442**. Chú ý, ở dòng **00401429**, có lệnh **CALL 1_2.00401247** → Đi đến dòng **00401247**:

```
00401419  .v 76 29          JBE SHORT 1_2.00401444
0040141B  > 8B45 FC         MOV EAX,DWORD PTR [EBP-4]
0040141E  .  50             PUSH EAX
0040141F  .  8D7405 D4      LEA ESI,DWORD PTR [EBP+EAX-2C]
00401423  .  0FBE4405 D4    MOVSX EAX,BYTE PTR [EBP+EAX-2C]
00401428  .  50             PUSH EAX
00401429  .  E8 19FEFFFF    CALL 1_2.00401247
0040142E  .  FF45 FC        INC DWORD PTR [EBP-4]
00401431  .  8806           MOV BYTE PTR [ESI],AL
00401433  .  8D45 D4        LEA EAX,DWORD PTR [EBP-2C]
00401436  .  50             PUSH EAX                                s
00401437  .  E8 A00B0000    CALL <JMP.&MSVCRT.strlen>               strlen
0040143C  .  83C4 0C        ADD ESP,0C
0040143F  .  3945 FC        CMP DWORD PTR [EBP-4],EAX
00401442  .^ 72 D7          JB SHORT 1_2.0040141B
```

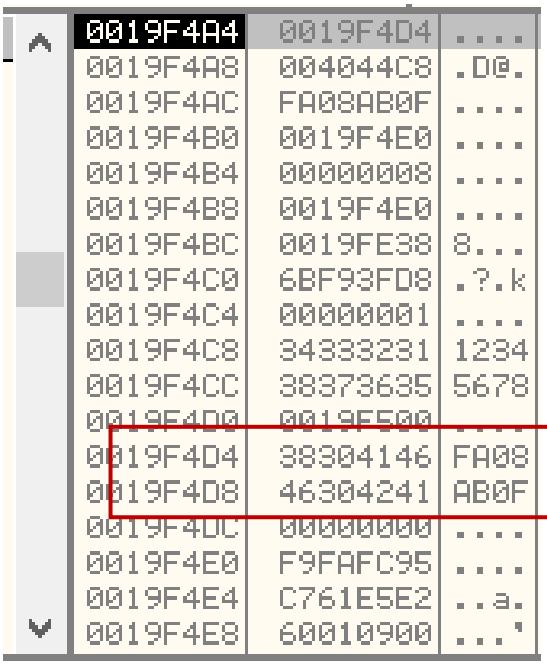
- Dòng **00401247**: **"MOV EAX, DWORD PTR [ESP + 8]" → EAX** = 32 bit trên cùng của stack.
- Dòng **0040124B**: **"MOV ECX, EAX" →** copy giá trị từ thanh ghi *EAX* vào thanh ghi *ECX***.**
- Dòng **0040124D**: "**SHL ECX, 2" →** dịch trái 2 lần dãy bit trong **ECX,** tương đương với ECX_{16} x 16^2.
- Dòng **00401250**: **"MOV EAX, DWORD PTR [ECX + 4032E8]" →** lưu giá trị từ thanh ghi có địa chỉ *ECX+4032E8* vào thanh ghi *EAX*.

```
00401247  r$  8B4424 08      MOV EAX,DWORD PTR [ESP+8]
0040124B  .   8BC8           MOV ECX,EAX
0040124D  .   C1E1 02        SHL ECX,2
00401250  .   8B81 E8324000  MOV EAX,DWORD PTR [ECX+4032E8]
00401256  .   2B81 20404000  SUB EAX,DWORD PTR [ECX+404020]
0040125C  .   334424 04      XOR EAX,DWORD PTR [ESP+4]
00401260  L.  C3             RET
```

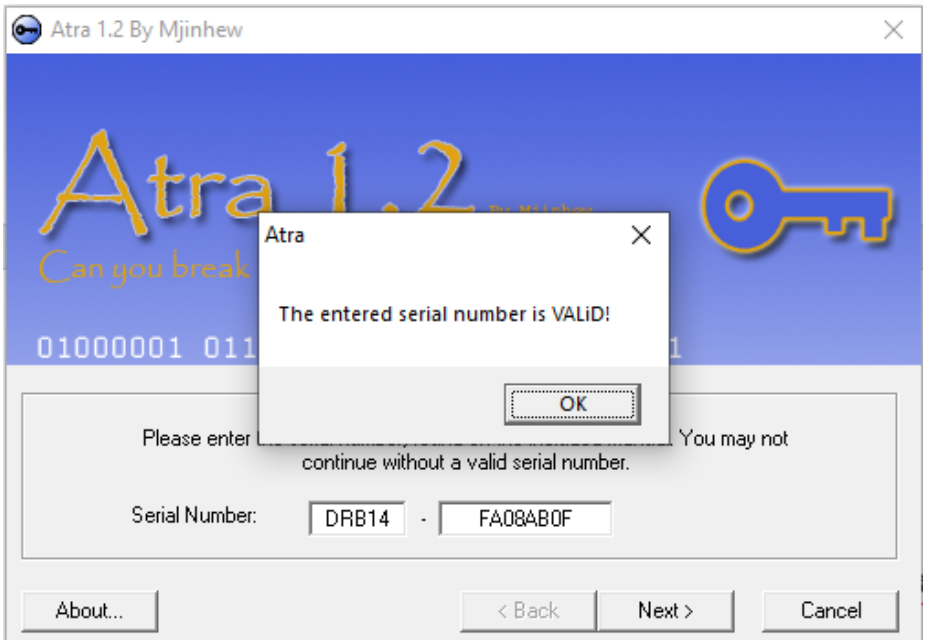
5.2.2 Xử lý part 2:

Với part 1 đã có định dạng, ta tiến hành nhập bất kì 1 part 1 nào đó, ở đây chúng ta nhập: **"DRB14"** và part 2 là **"12345678".**

Đặt Break point ở dòng **0040149A**, kết quả thu được là:



Ta có thể thấy ở dòng số **0019F4D4:** xuất hiện string **FA08AB0F**. Thử lại string này với vai trò là part 2, part 1 giữ nguyên.



Vậy đây là part 2 cần tìm, đoạn code phía trên dòng đặt break-point đã biến đổi chuỗi part 1 thành part 2 bằng 1 thuật toán nào đó.

Từ dòng **004015CF** đến **0040166C**: Xử lí chuỗi part 1 thành 1 chuỗi mới có 16bit hex.

Hình [1]

```
004015CF  $  55            PUSH EBP
004015D0  .  8BEC          MOV EBP,ESP
004015D2  .  53            PUSH EBX
004015D3  .  56            PUSH ESI
004015D4  .  8B75 08       MOV ESI,DWORD PTR [EBP+8]
004015D7  .  57            PUSH EDI
004015D8  .  8B7D 10       MOV EDI,DWORD PTR [EBP+10]
004015DB  .  8B4E 10       MOV ECX,DWORD PTR [ESI+10]
004015DE  .  8BD7          MOV EDX,EDI
004015E0  .  8BC1          MOV EAX,ECX
004015E2  .  C1E8 03       SHR EAX,3
004015E5  .  8D0CF9        LEA ECX,DWORD PTR [ECX+EDI*8]
004015E8  .  83E0 3F       AND EAX,3F
004015EB  .  C1E2 03       SHL EDX,3
004015EE  .  3BCA          CMP ECX,EDX
004015F0  .  894E 10       MOV DWORD PTR [ESI+10],ECX
004015F3  .v 73 03         JNB SHORT 1_2.004015F8
004015F5  .  FF46 14       INC DWORD PTR [ESI+14]
004015F8  >  6A 40         PUSH 40
004015FA  .  8BCF          MOV ECX,EDI
004015FC  .  5B            POP EBX
004015FD  .  C1E9 1D       SHR ECX,1D
00401600  .  014E 14       ADD DWORD PTR [ESI+14],ECX
00401603  .  2BD8          SUB EBX,EAX
00401605  .  3BFB          CMP EDI,EBX
00401607  .v 72 42         JB SHORT 1_2.0040164B
00401609  .  53            PUSH EBX                          n
0040160A  .  8D4430 18     LEA EAX,DWORD PTR [EAX+ESI+18]
0040160E  .  FF75 0C       PUSH DWORD PTR [EBP+C]            src
00401611  .  50            PUSH EAX                          dest
00401612  .  E8 CB090000   CALL <JMP.&MSVCRT.memcpy>         memcpy
00401617  .  8D46 18       LEA EAX,DWORD PTR [ESI+18]
0040161A  .  50            PUSH EAX
0040161B  .  56            PUSH ESI
0040161C  .  E8 4C000000   CALL 1_2.0040166D
00401621  .  83C4 14       ADD ESP,14
00401624  .  895D 08       MOV DWORD PTR [EBP+8],EBX
00401627  .  83C3 3F       ADD EBX,3F
0040162A  >  3BDF          CMP EBX,EDI
0040162C  .v 73 19         JNB SHORT 1_2.00401647
0040162E  .  8B45 0C       MOV EAX,DWORD PTR [EBP+C]
00401631  .  8D4418 C1     LEA EAX,DWORD PTR [EAX+EBX-3F]
00401635  .  50            PUSH EAX
00401636  .  56            PUSH ESI
00401637  .  E8 31000000   CALL 1_2.0040166D
0040163C  .  8345 08 40    ADD DWORD PTR [EBP+8],40
00401640  .  59            POP ECX
00401641  .  59            POP ECX
00401642  .  83C3 40       ADD EBX,40
00401645  .^ EB E3         JMP SHORT 1_2.0040162A
00401647  >  33C0          XOR EAX,EAX
00401649  .v EB 04         JMP SHORT 1_2.0040164F
0040164B  >  8365 08 00    AND DWORD PTR [EBP+8],0
0040164F  >  8B4D 08       MOV ECX,DWORD PTR [EBP+8]
00401652  .  8B55 0C       MOV EDX,DWORD PTR [EBP+C]
00401655  .  2BF9          SUB EDI,ECX
00401657  .  03CA          ADD ECX,EDX
00401659  .  57            PUSH EDI                          n
0040165A  .  8D4430 18     LEA EAX,DWORD PTR [EAX+ESI+18]
0040165E  .  51            PUSH ECX                          src
0040165F  .  50            PUSH EAX                          dest
00401660  .  E8 7D090000   CALL <JMP.&MSVCRT.memcpy>         memcpy
00401665  .  83C4 0C       ADD ESP,0C
00401668  .  5F            POP EDI
00401669  .  5E            POP ESI
0040166A  .  5B            POP EBX
0040166B  .  5D            POP EBP
0040166C  L. C3            RET
```

Các bước xử lí như sau:

- Ban đầu có part 1 là string, tạm gọi là s1. Từ s1 này, qua thuật toán ở *hình[1]* thì s1 sẽ được chuyển sang dạng hex và gắn vào đầu của 1 chuỗi hex mới (s có 64 bit) với dạng.

+ 5 phần tử đầu của ***s*** là s1, phần tử thứ 6 có mã là 0x80, phần tử thứ 57 có mã là 0x28.

+ Thực hiện biến chuỗi 64 phần tử thành chuỗi có 16 phần tử nhưng kiểu dữ liệu tăng từ 8bit lên 32bit.

```
00401419  .v 76 29          JBE SHORT 1_2.00401444
0040141B  > 8B45 FC         MOV EAX,DWORD PTR [EBP-4]
0040141E  . 50              PUSH EAX
0040141F  . 8D7405 D4       LEA ESI,DWORD PTR [EBP+EAX-2C]
00401423  . 0FBE4405 D4     MOVSX EAX,BYTE PTR [EBP+EAX-2C]
00401428  . 50              PUSH EAX
00401429  . E8 19FEFFFF     CALL 1_2.00401247
0040142E  . FF45 FC         INC DWORD PTR [EBP-4]
00401431  . 8806            MOV BYTE PTR [ESI],AL
00401433  . 8D45 D4         LEA EAX,DWORD PTR [EBP-2C]
00401436  . 50              PUSH EAX                          s
00401437  . E8 A00B0000     CALL <JMP.&MSVCRT.strlen>         strlen
0040143C  . 83C4 0C         ADD ESP,0C
0040143F  . 3945 FC         CMP DWORD PTR [EBP-4],EAX
00401442  .^ 72 D7          JB SHORT 1_2.0040141B
```

```
00401247  $ 8B4424 08       MOV EAX,DWORD PTR [ESP+8]
0040124B  . 8BC8            MOV ECX,EAX
0040124D  . C1E1 02         SHL ECX,2
00401250  . 8B81 E8324000   MOV EAX,DWORD PTR [ECX+4032E8]
00401256  . 2B81 20404000   SUB EAX,DWORD PTR [ECX+404020]
0040125C  . 334424 04       XOR EAX,DWORD PTR [ESP+4]
00401260  . C3              RET
```

+ Hai đoạn mã ở 2 hình trên sẽ thực hiện chuyển ***s*** về 1 chuỗi khác, như sau:

- Gọi i là index của giá trị ascii trong s.
- Chuỗi s được lưu ở thanh ghi ESI.
- EAX = i = ECX
- ESP = s[i]
- ECX = ECX << i

EAX = giá trị tại thanh ghi có địa chỉ [ECX + 4032E8]
EAX = EAX – giá trị tại thanh ghi có địa chỉ [ECX + 404020]
EAX = EAX xor ESP

- Lấy 2 bit cuối của EAX rồi lưu vào s[i]
- Nếu EAX < 8 thì thực hiện lại.

```
00401261  $ 8B4424 04       MOV EAX,DWORD PTR [ESP+4]
00401265  . 8B4C24 08       MOV ECX,DWORD PTR [ESP+8]
00401269  . D3E0            SHL EAX,CL
0040126B  . 0B4424 04       OR EAX,DWORD PTR [ESP+4]
0040126F  . C3              RET
```

+ Tiếp theo, đoạn mã 8 kí tự đó sẽ được thực hiện theo nguyên tắc như hình trên:

- Gọi p là mảng kí tự, I là vị trí của kí tự trong mảng
- i = 0
- p[i] = (p[i] << i) | p[i]
- i++, nếu i = 8 thì thoát

+ Tiếp đó, đoạn mã từ dòng 004010C0 đến 004010FA:

```
004010C0  $  56            PUSH ESI
004010C1  .  33D2          XOR EDX,EDX
004010C3  .  83CE FF       OR ESI,FFFFFFFF
004010C6  .  395424 0C     CMP DWORD PTR [ESP+C],EDX
004010CA  .v 7E 29         JLE SHORT 1_2.004010F5
004010CC  .  57            PUSH EDI
004010CD  .  B9 FF000000   MOV ECX,0FF
004010D2  >  8B4424 0C     MOV EAX,DWORD PTR [ESP+C]
004010D6  .  8BFE          MOV EDI,ESI
004010D8  .  23F9          AND EDI,ECX
004010DA  .  8A0402        MOV AL,BYTE PTR [EDX+EAX]
004010DD  .  23C1          AND EAX,ECX
004010DF  .  33C7          XOR EAX,EDI
004010E1  .  C1EE 08       SHR ESI,8
004010E4  .  8B0485 20404(  MOV EAX,DWORD PTR [EAX*4+404020]
004010EB  .  33F0          XOR ESI,EAX
004010ED  .  42            INC EDX
004010EE  .  3B5424 10     CMP EDX,DWORD PTR [ESP+10]
004010F2  .^ 7C DE         JL SHORT 1_2.004010D2
004010F4  .  5F            POP EDI
004010F5  >  8BC6          MOV EAX,ESI
004010F7  .  5E            POP ESI
004010F8  .  F7D0          NOT EAX
004010FA  L. C3            RET
```

- Thực hiện chuyển đổi lần cuối, với nhiều thao tác
- 8 lần thực hiện

+ Sau khi thực hiện xong thì ta sẽ nhận được kết quả là 1 string 8byte.

- Chương trình sẽ so sánh chuỗi **s** này với part 2 do người dùng nhập. Nếu đúng thì sẽ hiện ra thông báo đúng hoặc ngược lại.

5.3 Key gen 1_2:

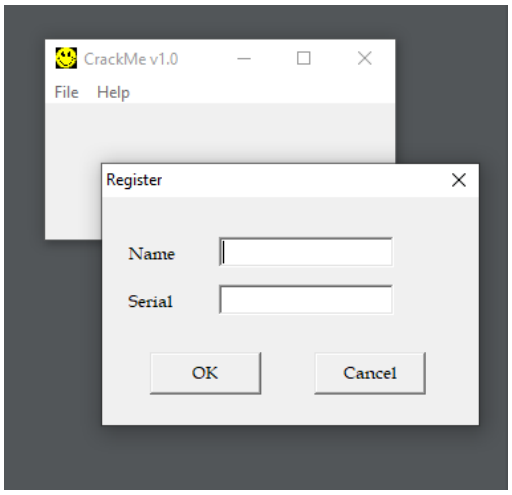
Nhận ra các phép biến đổi chuỗi để tạo thành key như được nhắc đến phía trên, ta viết được chương trình keygen, chương trình này thực hiện 3 thao tác:

- Đọc dữ liệu từ vùng nhớ
- Yêu cầu người dùng nhập serial number 1 thoả yêu cầu
- Thực hiện biến đổi và xuất full key

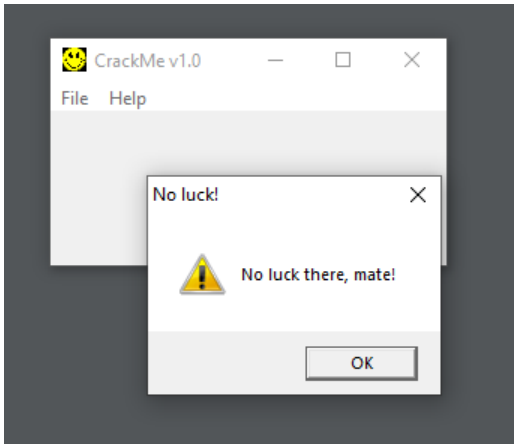
CHƯƠNG 6: CÂU 3

6.1 Thông tin cơ bản:

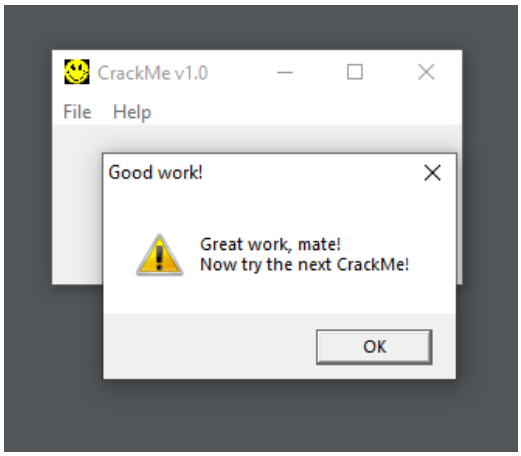
-Giao diện: Giao diện chương trình có nơi nhập Name và Serial đi kèm trong phần Register.



- Nếu nhập sai.



- Nếu nhập đúng.

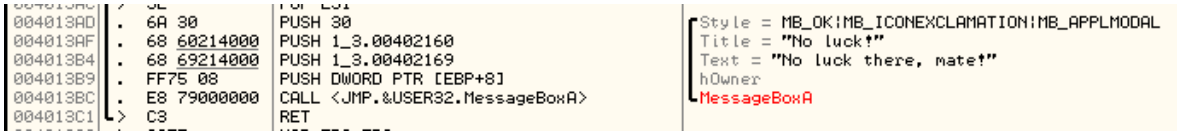


6.2 Debug bằng OllyDbg:

- Xét đoạn code in ra badboy:

```
00401369  .  6A 30        PUSH 30                                  Style = MB_OK|MB_ICONEXCLAMATION|MB_APPLMODAL
0040136B  .  68 60214000  PUSH 1_3.00402160                        Title = "No luck!"
00401370  .  68 69214000  PUSH 1_3.00402169                        Text = "No luck there, mate!"
00401375  .  FF75 08      PUSH DWORD PTR [EBP+8]                   hOwner
00401378  .  E8 BD000000  CALL <JMP.&USER32.MessageBoxA>           MessageBoxA
```

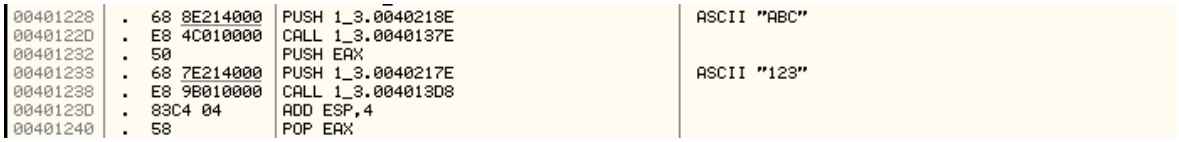
Và



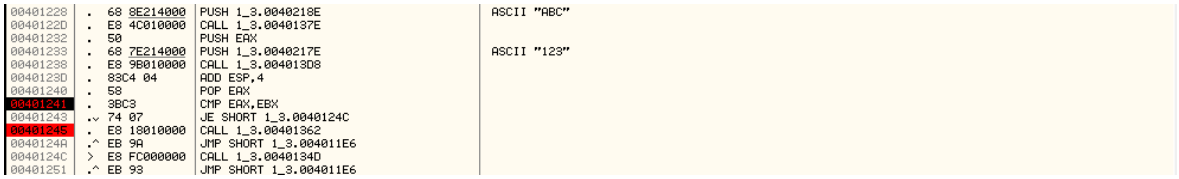
-Kiểm tra vị trí của badboy, ta thấy dòng code print badboy window ở vị trí **00401362** được gọi từ dòng **00401245** và vị trí **004013AC** được jump từ dòng **0040138B.**

-Ta tiến hành đặt breakpoint ở trước các vị trí call goodboy và badboy ở điều kiện đủ là dòng **00401245, 0040138B**.

-Thử debug với username là abc và password là 123:

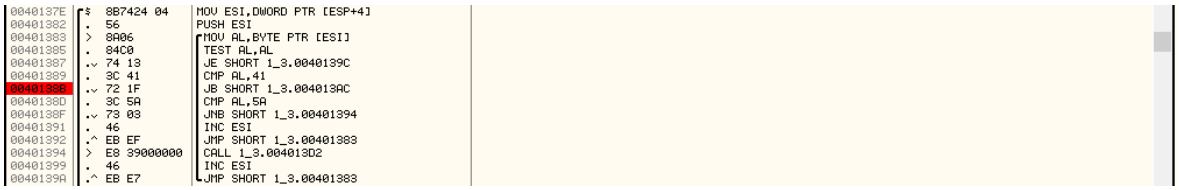


-> Name input được lưu ở 0040218E, Serial input được lưu ở 0040217E và dòng 0040137E là vị trí xử lý Name, dòng 004013D8 là vị trí xử lý Serial.



-> Sau khi xử lí, chương trình so sánh 2 chuỗi vừa thua được. Nếu giống nhau thì *jump* đến dòng **0040124C -> 0040134D** (in ra goodboy). Nếu khác nhau thì *jump* đến dòng **00401362** để in ra badboy…

- Truy vết username tại dòng **0040137E**:



- Ta thấy từ **00401383** đến **0040139A**: tiến hành kiểm tra các ký tự nhập vào có mã ASCII bé hơn 65 hay không. Nếu có thì nhảy đến badboy thứ 2 **004013AC**

- Đoạn code ở dòng **004013C2** tiến hành cộng tất cả các ký tự đã được kiểm tra và upcase hoặc trừ đi 20 trong mã ASCII của nó, ở phía trước lại với nhau.

```
33FF        XOR EDI,EDI
33DB        XOR EBX,EBX
8A1E        MOV BL,BYTE PTR [ESI]
84DB        TEST BL,BL
74 05       JE SHORT 1_3.004013D1
03FB        ADD EDI,EBX
46          INC ESI
EB F5       JMP SHORT 1_3.004013C6
C3          RET
```

- Cuối cùng tiến hành XOR biến vừa thu được với 0x5678 để thu được kết quả sau cùng và trở về vị trí trước đó để xử lý serial key ở dòng **004013A2**

- **Xử lí Serial:**

Ta tiến hành kiểm tra tại vị trí 004013D8 và thấy được đoạn code xử lý serial như sau:

```
004013D7  C3          RET
004013D8  33C0        XOR EAX,EAX
004013DA  33FF        XOR EDI,EDI
004013DC  33DB        XOR EBX,EBX
004013DE  8B7424 04   MOV ESI,DWORD PTR [ESP+4]
004013E2  B0 0A       MOV AL,0A
004013E4  8A1E        MOV BL,BYTE PTR [ESI]
004013E6  84DB        TEST BL,BL
004013E8  74 0B       JE SHORT 1_3.004013F5
004013EA  80EB 30     SUB BL,30
004013ED  0FAFF8      IMUL EDI,EAX
004013F0  03FB        ADD EDI,EBX
004013F2  46          INC ESI
004013F3  EB ED       JMP SHORT 1_3.004013E2
004013F5  81F7 3412000 XOR EDI,1234
004013FB  8BDF        MOV EBX,EDI
004013FD  C3          RET
```

➔ Trong đó đoạn code từ 004013E2 đến 004013F5 thực hiện convert chuỗi serial key thành số, sau đó XOR với **0x1234** ở dòng **004013F5.**

6.3 Viết keygen:

Ta rõ ràng có thể thấy tương ứng 1 name chỉ có 1 serial và name phải là chữ cái. Và serial có thể được generate từ name bằng cách lợi dụng tính chất của phép XOR. Thay vì đem serial đem xor với 0x1234. Ta tiến hành như sau:

• Kiểm tra các kí tự của name có mã ASCII < 41_{16} hay không, nếu có thì báo lỗi và yêu cầu nhập lại. Nếu đúng thì lấy mã ASCII của kí tự đó từ đi 20.

• Đem cộng giá trị ascii các ký tự của name lại với nhau

• Đem xor kết quả với 0x5678 .

• Đem xor kết quả tiếp với 0x1234 .

• Kết quả thu được chính là serial ở dạng số.

CHƯƠNG 7: THAM KHẢO

Slide bài giảng của thầy Viet Long về các cracking tutorial.

Tham khảo thêm kiến thức và fix bug ở các trang

www.stackoverflow.com

https://daynhauhoc.com

http://www.cs.virginia.edu/

https://www.eecg.utoronto.ca/

https://doc.dpdk.org/