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 HỆ THỐNG MÁY TÍNH

| Đề tài |

REVERSE ENGINEERING

| Sinh viên thực hiện |

Ngô Bá Sỹ Nguyên	**23120020**
Nguyễn Thái Bảo	**23120023**
Nguyễn Phú Dinh	**23120031**

| Giáo viên hướng dẫn |

Thầy Lê Viết Long

Môn học: Hệ thống máy tính

Thành phố Hồ Chí Minh - 2025

MỤC LỤC

1. THÔNG TIN SINH VIÊN

	Sinh viên 1	Sinh viên 2	Sinh viên 3
Họ và tên	Ngô Bá Sỹ Nguyên	Nguyễn Thái Bảo	Nguyễn Phú Dinh
Mã số sinh viên	23120020	23120023	23120031
Lớp	23CTT1		

2. ĐÁNH GIÁ

Bài	Người phụ trách chính	Ghi chú	Đánh giá mức độ hoàn thành
1	Nguyễn Phú Dinh - 23120031	**Hoàn thành đầy đủ yêu cầu:** - Cơ bản: Chỉ ra đoạn phát sinh key, giải thích ý nghĩa và đưa ra một key tương ứng với username minh họa - Nâng cao: Viết chương trình keygen bằng ngôn ngữ Python để phát sinh các khóa hợp lệ cho chương trình CrackMe1, đồng thời thử các trường hợp đặc biệt giả định mỗi số là một mã ASCII	100%
2	Nguyễn Thái Bảo - 23120023	**Hoàn thành đầy đủ yêu cầu:** - Cơ bản: Chỉ ra đoạn phát sinh key, giải thích ý nghĩa và đưa ra một key tương ứng với username minh họa - Nâng cao: Viết chương trình keygen bằng ngôn ngữ Python để phát sinh khóa từ username người dùng nhập vào	100%
3	Ngô Bá Sỹ Nguyên - 23120020	**Hoàn thành đầy đủ yêu cầu:** - Cơ bản: Chỉ ra đoạn phát sinh key, giải thích ý nghĩa và đưa ra một key tương ứng với username minh họa - Nâng cao: Viết chương trình keygen bằng ngôn ngữ Python để phát sinh khóa từ username người dùng nhập vào	100%

Đánh giá tổng thể mức độ hoàn thành đồ án: 100%

3. KẾT QUẢ BÀI LÀM

BÀI 1.

Đoạn phát sinh key:

- **Hàm _calc:**

```
text:004012E8                   public _calc
text:004012E8 _calc             proc near
text:004012E8
text:004012E8 var_20            = dword ptr -20h
text:004012E8 var_1C            = dword ptr -1Ch
text:004012E8 var_18            = dword ptr -18h
text:004012E8 var_14            = dword ptr -14h
text:004012E8 var_10            = dword ptr -10h
text:004012E8 var_C             = dword ptr -0Ch
text:004012E8 var_8             = dword ptr -8
text:004012E8 var_4             = dword ptr -4
text:004012E8 arg_0             = dword ptr  8
text:004012E8 arg_4             = dword ptr  0Ch
text:004012E8 arg_8             = dword ptr  10h
text:004012E8 arg_C             = dword ptr  14h
text:004012E8 arg_10            = dword ptr  18h
text:004012E8 arg_14            = dword ptr  1Ch
text:004012E8 arg_18            = dword ptr  20h
text:004012E8 arg_1C            = dword ptr  24h
text:004012E8 arg_20            = dword ptr  28h
```

Lần lượt các biến và tham số:

- Các biến:
  - var_20 (check3), var_1C (check2), var_18 (check3) dùng để kiểm tra lúc sau.
  - var_14, var_10, var_C, var_8, var_4 dùng cho các tính toán.
- Các tham số: arg0 → arg20 (lần lượt là các số được nhập vào từ input, tạm gọi number1 → number9).

Tính giá trị var_4:

```
.text:004012F6    mov     eax, [ebp+arg_10]
.text:004012F9    mov     ebx, [ebp+arg_0]
.text:004012FC    cdq
.text:004012FD    idiv    ecx
.text:004012FF    mov     eax, ebx
.text:00401301    mov     ecx, edx
.text:00401303    cdq
.text:00401304    mov     edi, [ebp+arg_8]
.text:00401307    mov     esi, [ebp+arg_4]
.text:0040130A    push    ecx
.text:0040130B    mov     ecx, 0Ah
.text:00401310    idiv    ecx
.text:00401312    pop     ecx
.text:00401313    mov     [ebp+var_4], edx
```

Các bước:

- mov eax, [ebp + arg_0]
- mov ecx, 0Ah
- cdq
- idiv ecx
- mov [ebp + var_4], edx

➔ Chia dư number1 cho 10. Vậy **var_4 = number1 % 10**.

Tính giá trị var_8:

```
text:00401316        mov     eax, [ebp+arg_14]
text:00401319        push    ecx
text:0040131A        cdq
text:0040131B        mov     ecx, 0Ah
text:00401320        idiv    ecx
text:00401322        pop     ecx
text:00401323        mov     [ebp+var_8], edx
```

Các bước tính var_8:

- mov eax, [ebp + arg_14]
- cdq
- mov ecx, 0Ah
- idiv ecx
- mov [ebp + var_8], edx

➔ Chia dư number6 cho 10. Vậy **var_8 = number6 % 10**.

Tính giá trị var_C:

```
.text:00401326        mov     eax, esi
.text:00401328        push    ecx
.text:00401329        cdq
.text:0040132A        mov     ecx, 0Ah
.text:0040132F        idiv    ecx
.text:00401331        pop     ecx
.text:00401332        add     edx, 3
.text:00401335        mov     [ebp+var_C], edx
```

Các bước tính var_C:

- mov eax, esi //esi = arg_4
- cdq
- mov ecx, 0Ah
- edv ecx
- add edx, 3
- mov [ebp + var_C], edx

➔ Chia dư number2 cho 10 và cộng 3. Vậy **var_C = number2 % 10 + 3**.

Tính var_10:

```
ext:00401338    mov     eax, edi
ext:0040133A    cdq
ext:0040133B    push    ecx
ext:0040133C    mov     ecx, 0Ah
ext:00401341    idiv    ecx
ext:00401343    cdq
ext:00401344    pop     ecx
ext:00401345    push    ecx
ext:00401346    mov     ecx, 0Ah
ext:0040134B    idiv    ecx
ext:0040134D    pop     ecx
ext:0040134E    add     edx, 5
ext:00401351    mov     [ebp+var_10], edx
```

Các bước tính var_10:

- Mov eax, edi (edi = number3)
- Cdq
- Mov ecx, 0Ah
- Idiv ecx
- Cdq
- Mov ecx, 0Ah
- Idiv ecx
- Add edx, 5
- Mov [ebp + var_10], edx

➔ Chia number3 cho 10, tiếp tục chia dư cho 10 và cộng thêm 5. Vậy var_10 = (number3/ **10) % 10 + 5**.

Tính giá trị của var_14:

```
.text:00401354    mov     eax, [ebp+arg_20]
.text:00401357    push    ecx
.text:00401358    cdq
.text:00401359    mov     ecx, 0Ah
.text:0040135E    idiv    ecx
.text:00401360    cdq
.text:00401361    pop     ecx
.text:00401362    push    ecx
.text:00401363    mov     ecx, 0Ah
.text:00401368    idiv    ecx
.text:0040136A    pop     ecx
.text:0040136B    mov     [ebp+var_14], edx
.text:0040136E    mov     eax, [ebp+arg_18]
```

Các bước tính var_14:

- Mov eax, [ebp + arg_20]
- Mov ecx, 0Ah

- Idiv ecx
- mov ecx, 0Ah
- idiv ecx
- Mov [ebp + var_14], edx

➔ Lần lượt chia nguyên và chia dư number9(arg_18) cho 10 và lưu vào var_14. Vậy **var14 = (number9 / 10) % 10.**

Tính giá trị check1(var_18):

```
text:0040136E        mov     eax, [ebp+arg_18]
text:00401371        push    ecx
text:00401372        cdq
text:00401373        mov     ecx, 0Ah
text:00401378        idiv    ecx
text:0040137A        pop     ecx
text:0040137B        mov     eax, edx
text:0040137D        mov     edx, ecx
text:0040137F        add     eax, 2
text:00401382        add     edx, edx
text:00401384        lea     edx, [edx+edx*4]
text:00401387        add     edx, [ebp+var_4]
text:0040138A        add     edx, edx
text:0040138C        lea     edx, [edx+edx*4]
text:0040138F        add     edx, [ebp+var_10]
text:00401392        add     edx, edx
text:00401394        lea     edx, [edx+edx*4]
text:00401397        add     edx, [ebp+var_14]
text:0040139A        add     edx, edx
text:0040139C        lea     edx, [edx+edx*4]
text:0040139F        add     edx, eax
text:004013A1        mov     [ebp+var_18], edx
```

Check1 = 10000 * (number5 % 10) + 1000 * (number1 % 10) + 100 * (int (number3 / 10) % 10 + 5) + 10 * (int (number9 / 10) % 10) + (number7 % 10) + 2

Tính giá trị check2 (var_1C):

```
.text:004013A4        mov     edx, ecx
.text:004013A6        add     edx, edx
.text:004013A8        add     ecx, ecx
.text:004013AA        lea     edx, [edx+edx*4]
.text:004013AD        lea     ecx, [ecx+ecx*4]
.text:004013B0        add     edx, [ebp+var_8]
.text:004013B3        add     edx, edx
.text:004013B5        lea     edx, [edx+edx*4]
.text:004013B8        add     edx, [ebp+var_10]
.text:004013BB        add     edx, edx
.text:004013BD        lea     edx, [edx+edx*4]
.text:004013C0        add     edx, [ebp+var_14]
.text:004013C3        add     edx, edx
.text:004013C5        lea     edx, [edx+edx*4]
.text:004013C8        add     edx, eax
.text:004013CA        mov     [ebp+var_1C], edx
```

Check2 = 10000 * (number5 % 10) + 1000 * (number6 % 10) + 100 * (int (number3 / 10) % 10 + 5) + 10 * (int (number9 / 10) % 10) + (number7 % 10) + 2

Tính giá trị check3(var_20):

```
text:004013CD    add     ecx, [ebp+var_C]
text:004013D0    mov     edx, ecx
text:004013D2    add     edx, edx
text:004013D4    lea     edx, [edx+edx*4]
text:004013D7    add     edx, [ebp+var_10]
text:004013DA    mov     ecx, edx
text:004013DC    add     ecx, ecx
text:004013DE    lea     ecx, [ecx+ecx*4]
text:004013E1    add     ecx, [ebp+var_14]
text:004013E4    mov     edx, ecx
text:004013E6    add     edx, edx
text:004013E8    lea     edx, [edx+edx*4]
text:004013EB    add     eax, edx
text:004013ED    mov     [ebp+var_20], eax
```

Check3 = 10000 * (number5 % 10) + 1000 * ((number2 % 10) + 3) + 100 * (int(number3 / 10) % 10 + 5) + 10 * (int(number9 / 10) % 10) + (number7 % 10) + 2

Gọi hàm sub_401478 để so sánh với các check1, check2, check3:

```
.text:004013F0    mov     eax, [ebp+arg_20]
.text:004013F3    push    eax
.text:004013F4    mov     ecx, [ebp+arg_1C]
.text:004013F7    push    ecx
.text:004013F8    mov     eax, [ebp+arg_18]
.text:004013FB    push    eax
.text:004013FC    mov     edx, [ebp+arg_14]
.text:004013FF    push    edx
.text:00401400    mov     ecx, [ebp+arg_10]
.text:00401403    push    ecx
.text:00401404    mov     eax, [ebp+arg_C]
.text:00401407    push    eax
.text:00401408    push    edi
.text:00401409    push    esi
.text:0040140A    push    ebx
.text:0040140B    call    sub_401478
.text:00401410    add     esp, 24h
.text:00401413    cmp     eax, [ebp+var_18]
.text:00401416    jnz     short loc_401468

text:00401418     mov     edx, [ebp+arg_20]
text:0040141B     push    edx
text:0040141C     mov     ecx, [ebp+arg_1C]
text:0040141F     push    ecx
text:00401420     mov     eax, [ebp+arg_18]
text:00401423     push    eax
text:00401424     mov     edx, [ebp+arg_14]
text:00401427     push    edx
text:00401428     mov     ecx, [ebp+arg_10]
text:0040142B     push    ecx
text:0040142C     mov     eax, [ebp+arg_C]
text:0040142F     push    eax
text:00401430     push    edi
text:00401431     push    esi
text:00401432     push    ebx
text:00401433     call    sub_401478
text:00401438     add     esp, 24h
text:0040143B     cmp     eax, [ebp+var_1C]
text:0040143E     jnz     short loc_401468
```

```
.text:00401440                 mov     edx, [ebp+arg_20]
.text:00401443                 push    edx
.text:00401444                 mov     ecx, [ebp+arg_1C]
.text:00401447                 push    ecx
.text:00401448                 mov     eax, [ebp+arg_18]
.text:0040144B                 push    eax
.text:0040144C                 mov     edx, [ebp+arg_14]
.text:0040144F                 push    edx
.text:00401450                 mov     ecx, [ebp+arg_10]
.text:00401453                 push    ecx
.text:00401454                 mov     eax, [ebp+arg_C]
.text:00401457                 push    eax
.text:00401458                 push    edi
.text:00401459                 push    esi
.text:0040145A                 push    ebx
.text:0040145B                 call    sub_401478
.text:00401460                 add     esp, 24h
.text:00401463                 cmp     eax, [ebp+var_20]
.text:00401466                 jz      short loc_40146C

.text:00401468 loc_401468:                             ; CODE XREF: _calc+12E↑j
.text:00401468                                         ; _calc+156↑j
.text:00401468                 xor     eax, eax
.text:0040146A                 jmp     short loc_401471
.text:0040146C loc_40146C:                             ; CODE XREF: _calc+17E↑j
.text:0040146C                 mov     eax, 1
.text:00401471
.text:00401471 loc_401471:                             ; CODE XREF: _calc+182↑j
.text:00401471                 pop     edi
.text:00401472                 pop     esi
.text:00401473                 pop     ebx
.text:00401474                 mov     esp, ebp
.text:00401476                 pop     ebp
.text:00401477                 retn
.text:00401477 _calc           endp
```

Nạp lần lượt các tham số: number9, number8, number7, number6, number5, number4, number3(edi), number2 (esi), number1(ebx) vào stack. Sau đó gọi hàm sub_401478 và dọn dẹp stack và cuối cùng so sánh với check1, nếu bằng nhau thì chương trình sẽ tiếp tục thực thi, còn không thì sẽ nhảy đến loc_401468 và trả về kết quả fail. Tương tự cho check2 và check3. Sau khi so sánh với check3 nếu bằng thì nhảy đến loc_40146C và trả về kết quả true.

Hàm sub_401478:

```
.text:00401478 ; ||||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:00401478
.text:00401478 ; Attributes: bp-based frame
.text:00401478
.text:00401478 sub_401478      proc near               ; CODE XREF: _calc+123↑p
.text:00401478                                         ; _calc+14B↑p ...
.text:00401478
.text:00401478 arg_0           = dword ptr  8
.text:00401478 arg_4           = dword ptr  0Ch
.text:00401478 arg_8           = dword ptr  10h
.text:00401478 arg_C           = dword ptr  14h
.text:00401478 arg_10          = dword ptr  18h
.text:00401478 arg_14          = dword ptr  1Ch
.text:00401478 arg_18          = dword ptr  20h
.text:00401478 arg_1C          = dword ptr  24h
.text:00401478 arg_20          = dword ptr  28h
.text:00401478
.text:00401478                 push    ebp
.text:00401479                 mov     ebp, esp
.text:0040147B                 push    ebx
.text:0040147C                 push    esi
.text:0040147D                 mov     eax, [ebp+arg_0]
.text:00401480                 mov     esi, [ebp+arg_C]
.text:00401483                 mov     ebx, eax
.text:00401485                 mov     ecx, [ebp+arg_8]
.text:00401488                 imul    ebx, [ebp+arg_10]
.text:0040148C                 imul    esi, [ebp+arg_1C]
.text:00401490                 imul    esi, ecx
```

```
.text:00401493                  imul    ebx, [ebp+arg_20]
.text:00401497                  add     ebx, esi
.text:00401499                  mov     edx, [ebp+arg_4]
.text:0040149C                  mov     esi, [ebp+arg_18]
.text:0040149F                  imul    esi, edx
.text:004014A2                  imul    esi, [ebp+arg_14]
.text:004014A6                  add     ebx, esi
.text:004014A8                  mov     esi, [ebp+arg_14]
.text:004014AB                  imul    esi, [ebp+arg_1C]
.text:004014AF                  imul    esi, eax
.text:004014B2                  imul    ecx, [ebp+arg_10]
.text:004014B6                  mov     eax, [ebp+arg_20]
.text:004014B9                  imul    ecx, [ebp+arg_18]
.text:004014BD                  imul    edx
.text:004014BF                  imul    [ebp+arg_C]
.text:004014C2                  add     ecx, esi
.text:004014C4                  add     ecx, eax
.text:004014C6                  sub     ebx, ecx
.text:004014C8                  mov     eax, ebx
.text:004014CA                  pop     esi
.text:004014CB                  pop     ebx
.text:004014CC                  pop     ebp
.text:004014CD                  retn
.text:004014CD sub_401478       endp
.text:004014CD
.text:004014CD ; ---------------------------------------------------------------------------
.text:004014CE                  align 10h
```

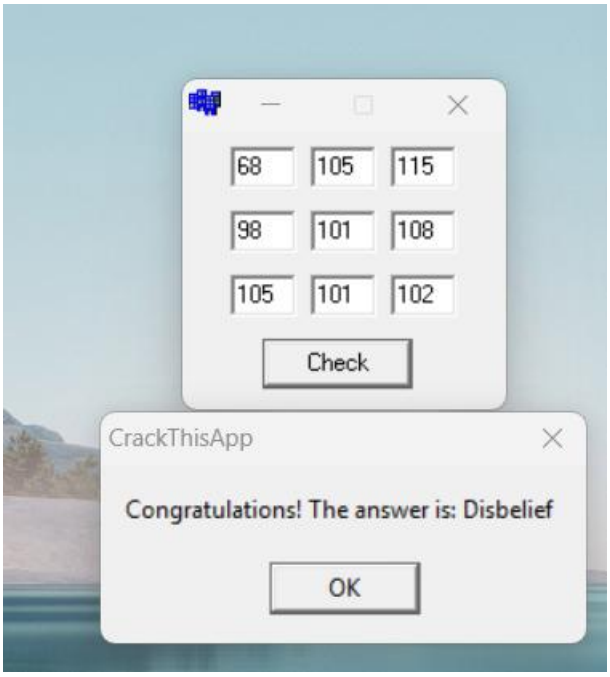
Hàm nhận vào 9 tham số (number1 → number9).
Giá trị trả về của hàm: number6 * number2 * number7 + number3 * number8 * number4 + number9 * number5 * number1 - (number4 * number2 * number9 + number1 * number8 * number6 + number7 * number5 * number3)

Như vậy, bất kỳ tập hợp 9 số nào thỏa điều kiện làm cho check1 = check2 = check3 = giá trị trả về của hàm sub_401478 sẽ là một dãy thỏa mãn, chương trình sẽ báo thành công (Congratulations). Tức là, có vô số trường hợp thỏa mãn.

Ta thử tìm thêm các trường hợp thỏa mãn đặc biệt. Cho rằng mỗi số này là mã ASCII của một kí tự, 9 số sẽ tạo thành 1 từ có 9 chữ cái. Ta thử duyệt qua danh sách các từ tiếng Anh có 9 chữ cái và kiểm tra điều kiện, kết quả là không có trường hợp nào thỏa mãn. Ta nghi vấn thêm trường hợp từ này có viết hoa chữ cái đầu và duyệt lại một lần nữa. Kết quả nhận được là dãy các số điền từ trái qua phải, từ trên xuống dưới lần lượt là: **68, 105, 115, 98, 101, 108, 105, 101, 102**. Quy đổi ra kí tự, ta được từ **"Disbelief"** là một từ tiếng Anh có nghĩa.

Ghi chú: Dãy **[68, 105, 115, 98, 101, 108, 105, 101, 102] "Disbelief"** chỉ là một trường hợp đặc biệt thỏa khi ta giả định mỗi số tượng trưng cho một mã ASCII. Ngoài ra, các dãy 9 số khác thỏa điều kiện đã nêu đều được coi là một đáp án của chương trình, tức là thỏa mãn và chương trình in ra thông báo thành công.

Ví dụ minh họa: [68, 105, 115, 98, 101, 108, 105, 101, 102]



Kiểm tra:

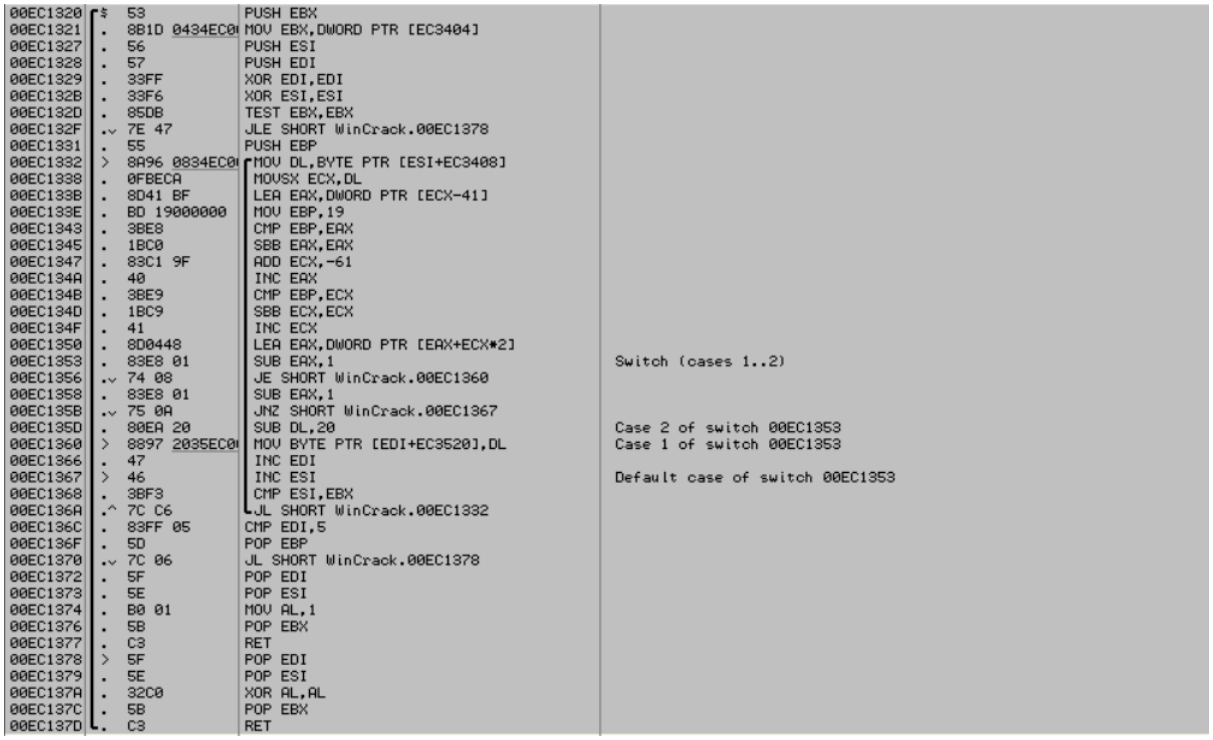
- Giá trị **check1** = 10000 * (101 % 10) + 1000 * (68 % 10) + 100 * (int (115 / 10) % 10 + 5) + 10 * (int (102 / 10) % 10) + (105 % 10) + 2 = 18607
- + Giá trị **check2** = 10000 * (101 % 10) + 1000 * (108 % 10) + 100 * (int (115 / 10) % 10 + 5) + 10 * (int (102 / 10) % 10) + (105 % 10) + 2 = 18607
- + Giá trị **check3** = 10000 * (101 % 10) + 1000 * ((105 % 10) + 3) + 100 * (int (115 / 10) % 10 + 5) + 10 * (int (102 / 10) % 10) + (105 % 10) + 2 = 18067
- + **Hàm sub_401478 =** 108 * 105 * 105 + 115 * 101 * 98 + 102 * 101 * 68 - (98 * 105 * 102 + 68 * 101 * 108 + 105 * 101 * 115) = 18067

Vậy **check1 = check2 = check3** và **bằng với giá trị trả về của hàm sub_401478**.

Do đó, **[68, 105, 115, 98, 101, 108, 105, 101, 102]** là một dãy thỏa mãn.

BÀI 2.

Đoạn mã xử lý chuỗi username:



```
00EC1320 r$  53          PUSH EBX
00EC1321 .   8B1D 0434EC0( MOV EBX,DWORD PTR [EC3404]
00EC1327 .   56          PUSH ESI
00EC1328 .   57          PUSH EDI
00EC1329 .   33FF        XOR EDI,EDI
00EC132B .   33F6        XOR ESI,ESI
00EC132D .   85DB        TEST EBX,EBX
00EC132F .v  7E 47       JLE SHORT WinCrack.00EC1378
00EC1331 .   55          PUSH EBP
00EC1332 >   8A96 0834EC0( MOV DL,BYTE PTR [ESI+EC3408]
00EC1338 .   0FBECA      MOVSX ECX,DL
00EC133B .   8D41 BF     LEA EAX,DWORD PTR [ECX-41]
00EC133E .   BD 19000000 MOV EBP,19
00EC1343 .   3BE8        CMP EBP,EAX
00EC1345 .   1BC0        SBB EAX,EAX
00EC1347 .   83C1 9F     ADD ECX,-61
00EC134A .   40          INC EAX
00EC134B .   3BE9        CMP EBP,ECX
00EC134D .   1BC9        SBB ECX,ECX
00EC134F .   41          INC ECX
00EC1350 .   8D0448      LEA EAX,DWORD PTR [EAX+ECX*2]
00EC1353 .   83E8 01     SUB EAX,1                    Switch (cases 1..2)
00EC1356 .v  74 08       JE SHORT WinCrack.00EC1360
00EC1358 .   83E8 01     SUB EAX,1
00EC135B .v  75 0A       JNZ SHORT WinCrack.00EC1367
00EC135D .   80EA 20     SUB DL,20                    Case 2 of switch 00EC1353
00EC1360 >   8897 2035EC0( MOV BYTE PTR [EDI+EC3520],DL Case 1 of switch 00EC1353
00EC1366 .   47          INC EDI
00EC1367 >   46          INC ESI                      Default case of switch 00EC1353
00EC1368 .   3BF3        CMP ESI,EBX
00EC136A .^  7C C6       JL SHORT WinCrack.00EC1332
00EC136C .   83FF 05     CMP EDI,5
00EC136F .   5D          POP EBP
00EC1370 .v  7C 06       JL SHORT WinCrack.00EC1378
00EC1372 .   5F          POP EDI
00EC1373 .   5E          POP ESI
00EC1374 .   B0 01       MOV AL,1
00EC1376 .   5B          POP EBX
00EC1377 .   C3          RET
00EC1378 >   5F          POP EDI
00EC1379 .   5E          POP ESI
00EC137A .   32C0        XOR AL,AL
00EC137C .   5B          POP EBX
00EC137D L.  C3          RET
```

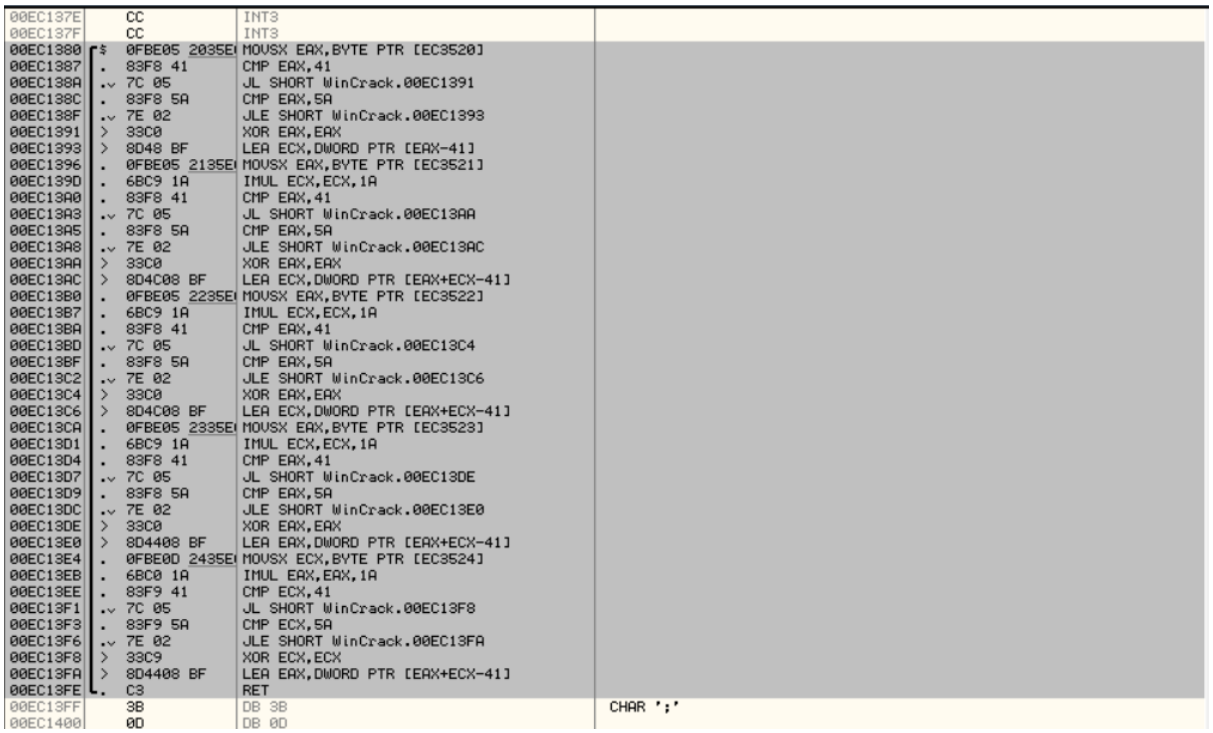
Đoạn mã trên lọc các chữ cái từ chuỗi đầu vào (username) và giữ lại chỉ các chữ cái, sau đó lưu lại tối đa 5 ký tự.

Có thể hình dung thuật toán như sau:

1. Lấy độ dài chuỗi đầu vào từ [EC3404] vào EBX
2. Khởi tạo ESI (vị trí đọc) và EDI (vị trí ghi) bằng 0
3. Kiểm tra nếu độ dài chuỗi là 0, nhảy đến kết thúc
4. Dùng vòng lặp để xử lý từng ký tự trong chuỗi đầu vào:
    - Đọc ký tự tại vị trí [ESI+EC3408]
    - Kiểm tra xem ký tự đó là chữ cái in hoa (A-Z) hay in thường (a-z)
    - Nếu là chữ cái in thường, chuyển thành in hoa (bằng cách trừ 20h)
    - Nếu là chữ cái, lưu vào vùng đệm kết quả tại [EDI+EC3520] và tăng EDI
    - Tăng ESI để xử lý ký tự tiếp theo
5. Sau khi xử lý toàn bộ chuỗi, kiểm tra xem đã có ít nhất 5 ký tự hợp lệ chưa
    - Nếu có ít nhất 5 ký tự (EDI >= 5), trả về AL = 1 (*thành công*)
    - Nếu có ít hơn 5 ký tự, trả về AL = 0 (*thất bại*)

Sau khi đoạn mã thực hiện, ta có 5 chữ cái in hoa được lấy từ 5 chữ cái đầu tiên xuất hiện trong username được nhập. 5 chữ cái này được sử dụng để tạo ra 1 hash value sẽ sử dụng để so sánh với serial.

Đoạn mã tạo ra hash value từ 5 chữ cái được lấy ra từ đoạn code trên:



Đoạn mã này đọc 5 byte liên tiếp từ các địa chỉ bộ nhớ (EC3520 đến EC3524), kiểm tra xem mỗi ký tự có phải là chữ cái in hoa (từ 'A' đến 'Z') không, và thực hiện tính tích lũy để tạo ra giá trị hash value.

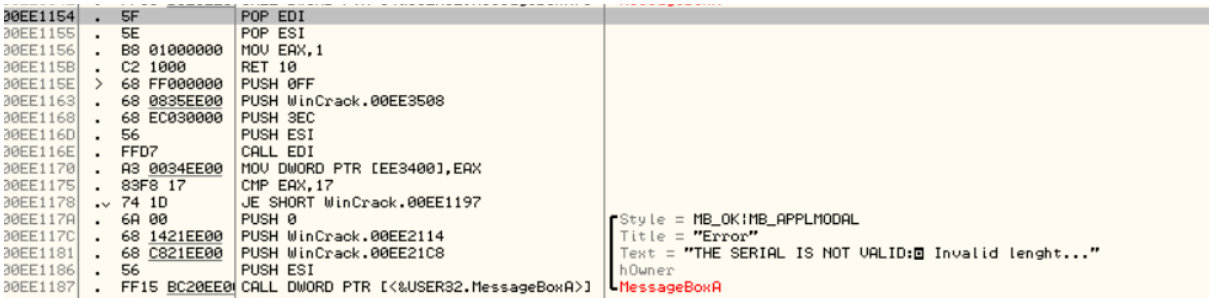
Có thể hình dung thuật toán như sau:

1. Đọc ký tự đầu tiên từ địa chỉ [EC3520]
2. Kiểm tra xem ký tự có nằm trong khoảng 'A' (41h) đến 'Z' (5Ah)
3. Nếu không, gán EAX = 0
4. Tính ECX = (ký tự - 'A')
5. Lặp lại quá trình với 4 ký tự tiếp theo
6. Với mỗi ký tự mới, áp dụng công thức: result = result * 26 + (ký tự - 'A')

Thuật toán chuyển đổi chuỗi 5 ký tự in hoa bên trên thành một giá trị số nguyên – hash value cần dùng, sử dụng hệ cơ số 26 (ứng với số lượng chữ cái trong bảng chữ cái tiếng Anh). Mỗi ký tự không phải là chữ cái in hoa sẽ được tính là 0.

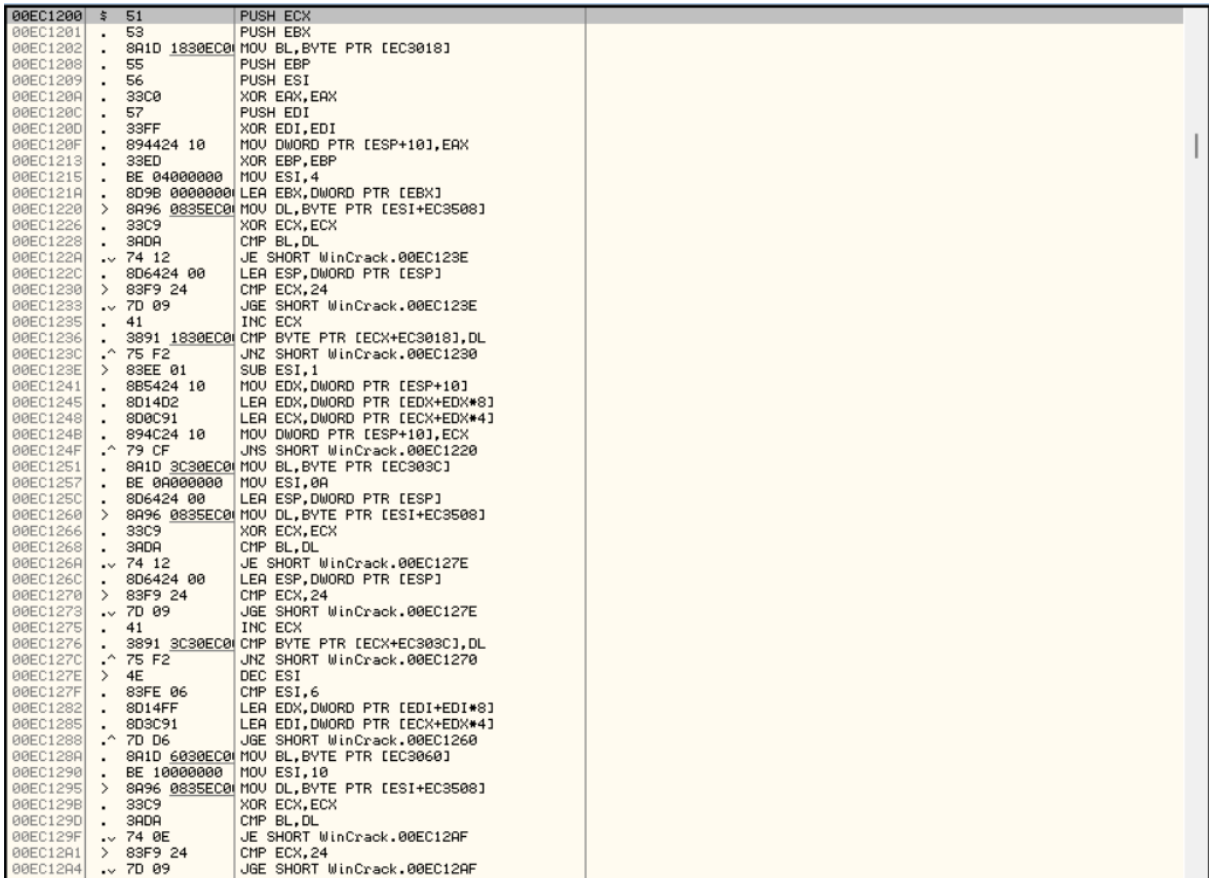
Nếu giá trị hash value bé hơn 24 trong hệ cơ số 16 thì chương trình báo lỗi tên không hợp lệ (“This name is a joke”).

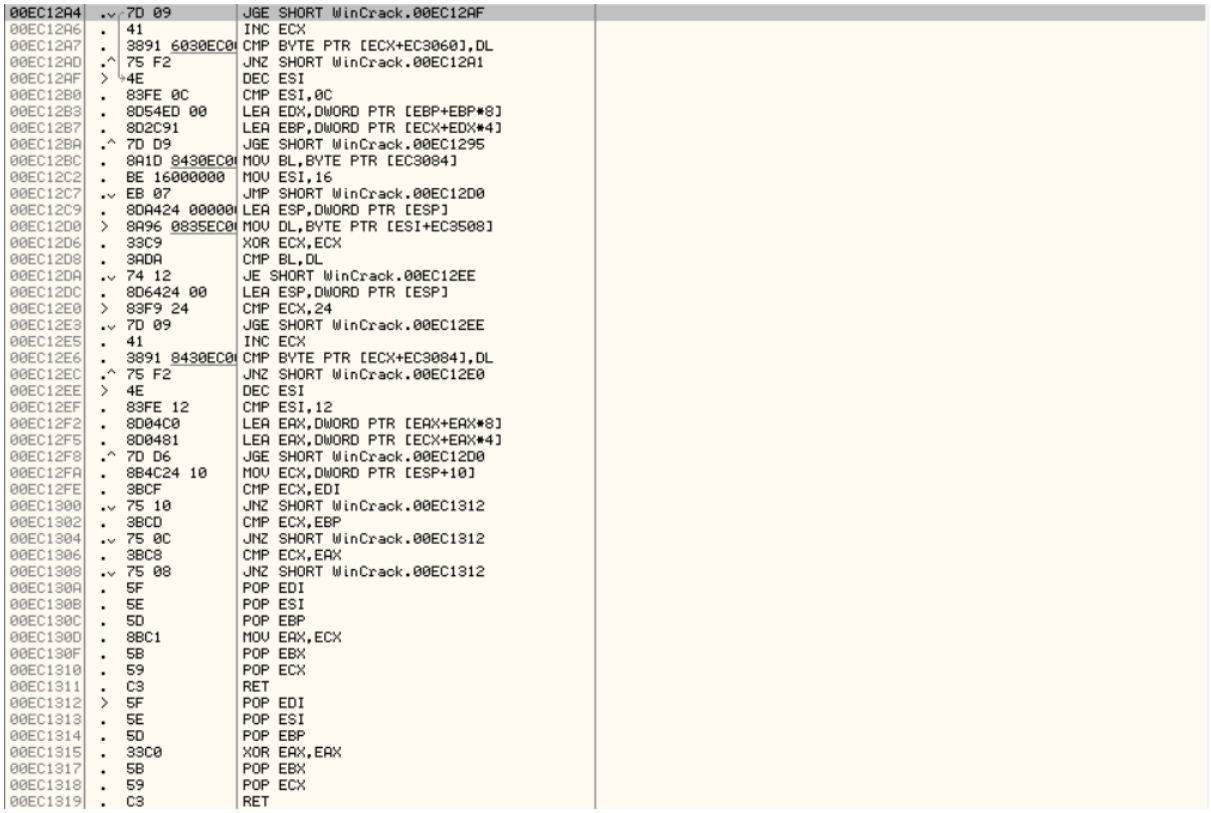
Kiểm tra độ dài serial:



Nếu độ dài serial bằng 17 hệ cơ số 16, tức là 23 trong cơ số 10, thì serial hợp lệ và tiếp tục chương trình. Ngược lại, chương trình báo lỗi.

Đoạn mã kiểm tra tính hợp lệ của serial:





```
00EC12A4  .v 7D 09        JGE SHORT WinCrack.00EC12AF
00EC12A6  .  41           INC ECX
00EC12A7  .  3891 6030EC0 CMP BYTE PTR [ECX+EC3060],DL
00EC12AD  .^ 75 F2        JNZ SHORT WinCrack.00EC12A1
00EC12AF  >  4E           DEC ESI
00EC12B0  .  83FE 0C      CMP ESI,0C
00EC12B3  .  8D54ED 00    LEA EDX,DWORD PTR [EBP+EBP*8]
00EC12B7  .  8D2C91       LEA EBP,DWORD PTR [ECX+EDX*4]
00EC12BA  .^ 7D D9        JGE SHORT WinCrack.00EC1295
00EC12BC  .  8A1D 8430EC0 MOV BL,BYTE PTR [EC3084]
00EC12C2  .  BE 16000000  MOV ESI,16
00EC12C7  .v EB 07        JMP SHORT WinCrack.00EC12D0
00EC12C9  .  8DA424 00000 LEA ESP,DWORD PTR [ESP]
00EC12D0  >  8A96 0835EC0 MOV DL,BYTE PTR [ESI+EC3508]
00EC12D6  .  33C9         XOR ECX,ECX
00EC12D8  .  3ADA         CMP BL,DL
00EC12DA  .v 74 12        JE SHORT WinCrack.00EC12EE
00EC12DC  .  8D6424 00    LEA ESP,DWORD PTR [ESP]
00EC12E0  >  83F9 24      CMP ECX,24
00EC12E3  .v 7D 09        JGE SHORT WinCrack.00EC12EE
00EC12E5  .  41           INC ECX
00EC12E6  .  3891 8430EC0 CMP BYTE PTR [ECX+EC3084],DL
00EC12EC  .^ 75 F2        JNZ SHORT WinCrack.00EC12E0
00EC12EE  >  4E           DEC ESI
00EC12EF  .  83FE 12      CMP ESI,12
00EC12F2  .  8D04C0       LEA EAX,DWORD PTR [EAX+EAX*8]
00EC12F5  .  8D0481       LEA EAX,DWORD PTR [ECX+EAX*4]
00EC12F8  .^ 7D D6        JGE SHORT WinCrack.00EC12D0
00EC12FA  .  8B4C24 10    MOV ECX,DWORD PTR [ESP+10]
00EC12FE  .  3BCF         CMP ECX,EDI
00EC1300  .v 75 10        JNZ SHORT WinCrack.00EC1312
00EC1302  .  3BCD         CMP ECX,EBP
00EC1304  .v 75 0C        JNZ SHORT WinCrack.00EC1312
00EC1306  .  3BC8         CMP ECX,EAX
00EC1308  .v 75 08        JNZ SHORT WinCrack.00EC1312
00EC130A  .  5F           POP EDI
00EC130B  .  5E           POP ESI
00EC130C  .  5D           POP EBP
00EC130D  .  8BC1         MOV EAX,ECX
00EC130F  .  5B           POP EBX
00EC1310  .  59           POP ECX
00EC1311  .  C3           RET
00EC1312  >  5F           POP EDI
00EC1313  .  5E           POP ESI
00EC1314  .  5D           POP EBP
00EC1315  .  33C0         XOR EAX,EAX
00EC1317  .  5B           POP EBX
00EC1318  .  59           POP ECX
00EC1319  .  C3           RET
```

Đoạn mã serial được nhập từ người dùng sẽ chia làm 4 phần, mỗi phần gồm 5 kí tự:

- Phần 1: 0 – 4 (tính index bắt đầu từ 0)
- Phần 2: 6 – 10
- Phần 3: 12 – 16
- Phần 4: 18 – 22

Ghi chú: các kí tự ở index 5, 11, 17 coi như các kí tự nối giữa các phần.

Đoạn mã sẽ thực hiện 4 vòng lặp khá tương tự nhau cho 4 phần, mỗi phần sẽ trả về 1 giá trị. Nếu cả 4 phần cùng trả về 1 giá trị thì đoạn mã (hàm) trả về giá trị đó; ngược lại, trả về 0.

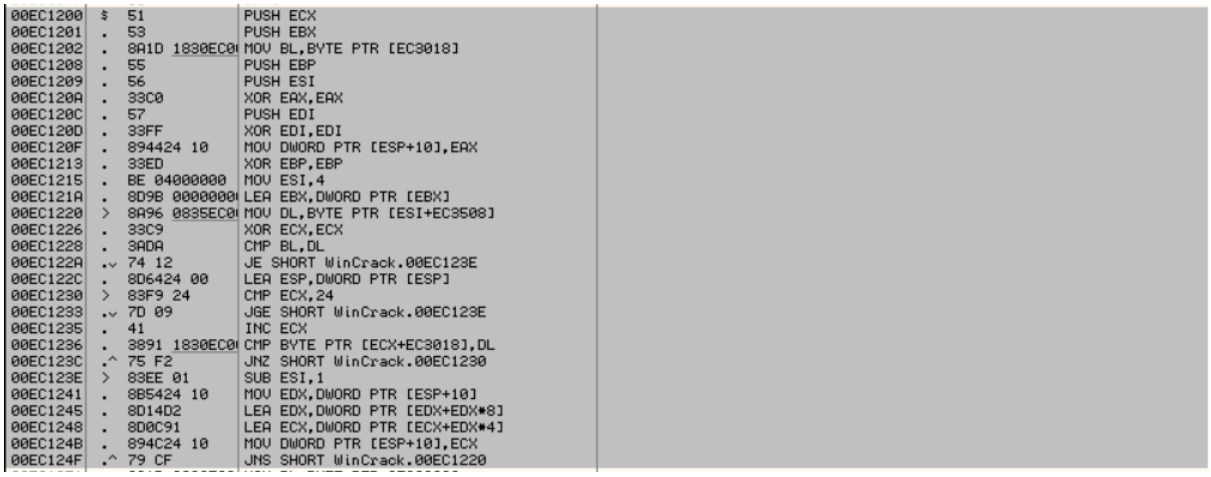
Trước tiên, ta sẽ chú ý **chuỗi "thần kỳ"** sau:

AGMSY4 BHNTZ5 CIOU06 DJPV17 EKQW28 FLRX39

Có thể hiểu quy luật của chuỗi: có 36 ký tự gồm 26 chữ cái trong bảng chữ cái tiếng anh và 10 chữ số từ 0 đến 9, lần lượt chèn các ký tự vào 6 "chiếc hộp" rồi quay lại hộp đầu tiên, lặp lại cho đến khi hết 36 ký tự.

Ta phân tích từng vòng lặp.

Vòng lặp 1:



```
00EC1200  $  51           PUSH ECX
00EC1201  .  53           PUSH EBX
00EC1202  .  8A1D 1830EC0( MOV BL,BYTE PTR [EC3018]
00EC1208  .  55           PUSH EBP
00EC1209  .  56           PUSH ESI
00EC120A  .  33C0         XOR EAX,EAX
00EC120C  .  57           PUSH EDI
00EC120D  .  33FF         XOR EDI,EDI
00EC120F  .  894424 10    MOV DWORD PTR [ESP+10],EAX
00EC1213  .  33ED         XOR EBP,EBP
00EC1215  .  BE 04000000  MOV ESI,4
00EC121A  .  8D9B 0000000( LEA EBX,DWORD PTR [EBX]
00EC1220  >  8A96 0835EC0( MOV DL,BYTE PTR [ESI+EC3508]
00EC1226  .  33C9         XOR ECX,ECX
00EC1228  .  3ADA         CMP BL,DL
00EC122A  .v 74 12        JE SHORT WinCrack.00EC123E
00EC122C  .  8D6424 00    LEA ESP,DWORD PTR [ESP]
00EC1230  >  83F9 24      CMP ECX,24
00EC1233  .v 7D 09        JGE SHORT WinCrack.00EC123E
00EC1235  .  41           INC ECX
00EC1236  .  3891 1830EC0( CMP BYTE PTR [ECX+EC3018],DL
00EC123C  .^ 75 F2        JNZ SHORT WinCrack.00EC1230
00EC123E  >  83EE 01      SUB ESI,1
00EC1241  .  8B5424 10    MOV EDX,DWORD PTR [ESP+10]
00EC1245  .  8D14D2       LEA EDX,DWORD PTR [EDX+EDX*8]
00EC1248  .  8D0C91       LEA ECX,DWORD PTR [ECX+EDX*4]
00EC124B  .  894C24 10    MOV DWORD PTR [ESP+10],ECX
00EC124F  .^ 79 CF        JNS SHORT WinCrack.00EC1220
```

Phần khởi tạo (00EC1200-00EC1215):

- **PUSH ECX, PUSH EBX, PUSH EBP, PUSH ESI, PUSH EDI**: Đẩy các thanh ghi vào stack để lưu trữ giá trị
- **MOV BL, BYTE PTR [EC3018]**: Lấy một byte từ địa chỉ bộ nhớ [EC3018] và lưu vào thanh ghi BL. Cụ thể, đó chính là ký tự đầu tiên của chuỗi "thần kỳ" phía trên (ký tự 'A').
- **XOR EAX, EAX, XOR EDI, EDI, XOR EBP, EBP**: Đặt các thanh ghi EAX, EDI, EBP về 0
- **MOV DWORD PTR [ESP+10], EAX**: Lưu giá trị của EAX (bằng 0) vào địa chỉ bộ nhớ [ESP+10]
- **MOV ESI, 4**: Đặt thanh ghi ESI bằng 4

Vòng lặp chính (00EC1220-00EC124F):

- **MOV DL, BYTE PTR [ESI+EC3508]**: Lấy một byte từ địa chỉ [ESI+EC3508] vào thanh ghi DL. Ở đây, địa chỉ [EC3508] chính là địa chỉ lưu kí tự đầu tiên của serial mà người dùng đã nhập.
- **XOR ECX, ECX**: Đặt thanh ghi ECX về 0
- **CMP BL, DL**: So sánh giá trị trong BL và DL
- **JE SHORT WinCrack.00EC123E**: Nếu BL bằng DL, nhảy đến địa chỉ 00EC123E → Tính toán

Vòng lặp con (00EC1230-00EC123C):

- **CMP ECX,24**: So sánh ECX với giá trị 24 (36 trong hệ thập phân - ứng với 36 kí tự trong chuỗi thần kỳ)
- **JGE SHORT WinCrack.00EC123E**: Nếu ECX >= 24, nhảy đến địa chỉ 00EC123E → tính toán
- **INC ECX**: Tăng ECX lên 1

- **CMP BYTE PTR [ECX+EC3018], DL**: So sánh byte tại địa chỉ [ECX+EC3018] với giá trị trong DL
- **JNZ SHORT WinCrack.00EC1230**: Nếu không bằng nhau, quay lại địa chỉ 00EC1230

Phần tính toán (00EC123E-00EC124F):

- **SUB ESI,1**: Giảm ESI đi 1
- **MOV EDX, DWORD PTR [ESP+10]**: Lấy giá trị từ [ESP+10] vào thanh ghi EDX
- **LEA EDX, DWORD PTR [EDX+EDX*8]**: Tính EDX = EDX + EDX * *8 (tức là EDX = EDX* * 9)
- **LEA ECX, DWORD PTR [ECX+EDX*4]**: Tính ECX = ECX + EDX * 4
- **MOV DWORD PTR [ESP+10], ECX**: Lưu giá trị mới của ECX vào [ESP+10]
- **JNS SHORT WinCrack.00EC1220**: Nếu kết quả không âm (bit dấu = 0), quay lại địa chỉ 00EC1220

Như vậy, ta có thể hiểu thuật toán hoạt động như sau:

1. Chạy từ phần tử phải cùng của phần (part), biết mỗi phần có 5 kí tự.
2. ECX = 0. Thực hiện vòng lặp con duyệt qua 36 kí tự trong chuỗi thần kỳ đến khi kí tự đó trong chuỗi = kí tự đang xét của phần đó. Mỗi lần duyệt qua một kí tự thì ECX tăng thêm 1. Nếu đã duyệt hết 36 kí tự mà vẫn không thấy thì thoát vòng lặp con.
3. Lưu giá trị trong stack [ESP + 10] vào EDX (ban đầu = 0)
4. Tính toán:
   EDX = EDX + EDX * *8* = EDX * 9
   ECX = ECX + EDX * 4 = ECX + EDX * 36
5. Lưu giá trị ECX vừa tính được vào lại stack [ESP + 10], rồi quay về 1 cho đến khi duyệt qua hết 5 kí tự của phần.

Sau khi vòng lặp 1 này kết thúc, ta rút ra được các phép tính toán đã được thực hiện như sau:

$$\begin{cases} EDX_0 = 0 \\ ECX_0 = 0 \\ EDX_1 = ECX_1 + EDX_0 * 36 \\ EDX_2 = ECX_2 + EDX_1 * 36 \\ EDX_3 = ECX_3 + EDX_2 * 36 \\ EDX_4 = ECX_4 + EDX_3 * 36 \\ result = ECX_5 + EDX_4 * 36 \end{cases}$$

Giải hệ trên bằng cách thế lần lượt các phương trình từ trên xuống, cuối cùng ta được:

$$result = ECX_1 * 36^4 + ECX_2 * 36^3 + ECX_3 * 36^2 + ECX_4 * 36 + ECX_5$$

Đây là giá trị sẽ được sử dụng để so sánh với 3 vòng lặp còn lại.

Về 3 vòng lặp còn lại:

Về cơ bản, 3 vòng lặp còn lại thực hiện thuật toán gần như tương tự với vòng lặp 1, chỉ có điểm khác biệt là:

- Ta vẫn so sánh với chuỗi thần kỳ đã cho, khác biệt là ở index bắt đầu không còn là từ ký tự 'A', mà lần lượt là 'B', 'C', 'D' tương ứng với 3 vòng lặp thứ 2, 3, 4, duyệt vòng tròn đến trước khi quay về kí tự bắt đầu. Tức là chuỗi thần kỳ tương ứng cho ba vòng lặp tương ứng như sau:
  - Vòng 2: **BHNTZ5 CIOU06 DJPV17 EKQW28 FLRX39 AGMSY4**
  - Vòng 3: **CIOU06 DJPV17 EKQW28 FLRX39 AGMSY4 BHNTZ5**
  - Vòng 4: **DJPV17 EKQW28 FLRX39 AGMSY4 BHNTZ5 CIOU06**
- Các thanh ghi lưu kết quả cho 3 vòng lặp lần lượt là: EDI, EBP, EAX

Kết thúc 4 vòng lặp, tiến hành so sánh 4 giá trị đã được tính toán. Giá trị của vòng lặp 1 được lưu từ stack vào ECX. Nếu cả 4 giá trị đều bằng nhau, khôi phục các thanh ghi, gán EAX = giá trị đó, kết thúc hàm. Ngược lại, khôi phục các thanh ghi, gán EAX = 0, kết thúc hàm.

Sau cùng, kiểm tra hash value từ username và giá trị trả về từ việc phân tích serial do người dùng nhập:

```
00EC1197  >  E8 64000000   CALL WinCrack.00EC1200
00EC119C  .  6A 00         PUSH 0                                   Style = MB_OK|MB_APPLMODAL
00EC119E  .  3B05 3836EC0( CMP EAX,DWORD PTR [EC3638]
00EC11A4  .v 74 1B         JE SHORT WinCrack.00EC11C1
00EC11A6  .  68 1421EC00   PUSH WinCrack.00EC2114                    Title = "Error"
00EC11AB  .  68 F421EC00   PUSH WinCrack.00EC21F4                    Text = "THE SERIAL IS NOT VALID:■ Check Name and Serial ..."
00EC11B0  .  56            PUSH ESI                                 hOwner
00EC11B1  .  FF15 BC20EC0( CALL DWORD PTR [<&USER32.MessageBoxA>]   MessageBoxA
00EC11B7  .  5F            POP EDI
00EC11B8  .  5E            POP ESI
00EC11B9  .  B8 01000000   MOV EAX,1
00EC11BE  .  C2 1000       RET 10
00EC11C1  >  68 2822EC00   PUSH WinCrack.00EC2228                    Title = "Good boy !"
00EC11C6  .  68 3422EC00   PUSH WinCrack.00EC2234                    Text = "SERIAL IS OK:■ Write a KEYGEN now ..."
00EC11CB  .  56            PUSH ESI                                 hOwner
00EC11CC  .  FF15 BC20EC0( CALL DWORD PTR [<&USER32.MessageBoxA>]   MessageBoxA
```

Nếu hai giá trị bằng nhau thì nhảy tới địa chỉ 00EC11C1 để hiển thị thông báo thành công. Ngược lại, hiển thị thông báo rằng username và serial chưa khớp với nhau.

Ví dụ minh họa:

Nhập vào username "ABCDE".

Ta phân tích hash value:

- res = 0 * 26 + 0 = 0
- res = 0 * 26 + 1 = 1
- res = 1 * 26 + 2 = 28
- res = 28 * 26 + 3 = 731
- res = 731 * 26 + 4 = 19010

Vậy hash value có được từ username là 19010.

Để serial hợp lệ thì giá trị trả về của mỗi trong 4 phần trích ra từ serial cũng phải bằng 19010.

Ta quay lại công thức:

$$res = ECX_1 * 36^4 + ECX_2 * 36^3 + ECX_3 * 36^2 + ECX_4 * 36 + ECX_5$$

Cho result = 19010, ta thực hiện chia lấy dư cho 36 để tìm ra các hệ số từ ECX_5 đến ECX_1. Kết quả trả về lần lượt là: 2, 24, 14, 0, 0.

Đây sẽ là các bước nhảy từ chuỗi thần kỳ ứng với từng phần để tìm ra giá trị của mỗi phần. ECX_5 sẽ là index của kí tự cuối cùng được duyệt, là kí tự trái cùng, tương tự cho đến ECX_1 sẽ là index của kí tự đầu tiên được duyệt, là kí tự phải cùng.

Ví dụ, với vòng lặp 1, đặt phần 1 có dạng abcde:

- a = index 2 trong chuỗi thần kỳ = 'M'
- b = index 24 trong chuỗi thần kỳ = 'E'
- c = index 14 trong chuỗi thần kỳ = 'O'
- d = index 0 trong chuỗi thần kỳ = 'A'
- e = index 0 trong chuỗi thần kỳ = 'A'

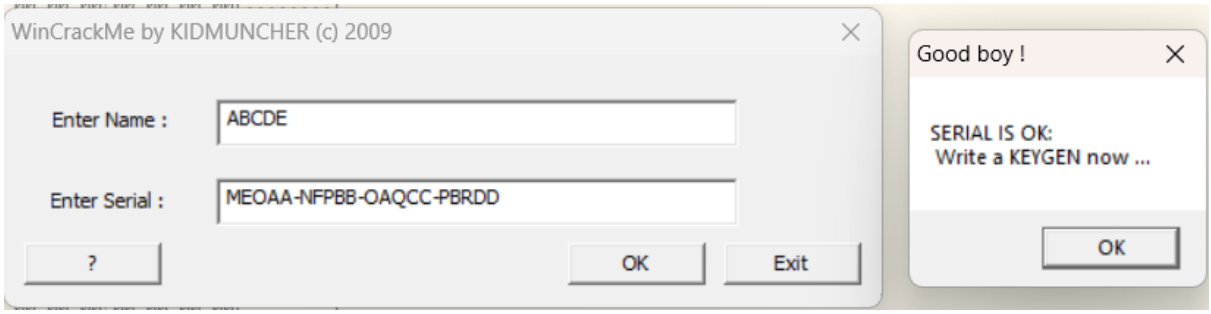
Vậy phần 1 có dạng: MEOAA. Tương tự, ta tìm được các phần sau:

- Phần 2: NFPBB
- Phần 3: OAQCC
- Phần 4: PBRDD

Giữa các phần nối bằng kí tự bất kỳ. Để dễ dàng, ta nối bằng dấu gạch ngang '-'.

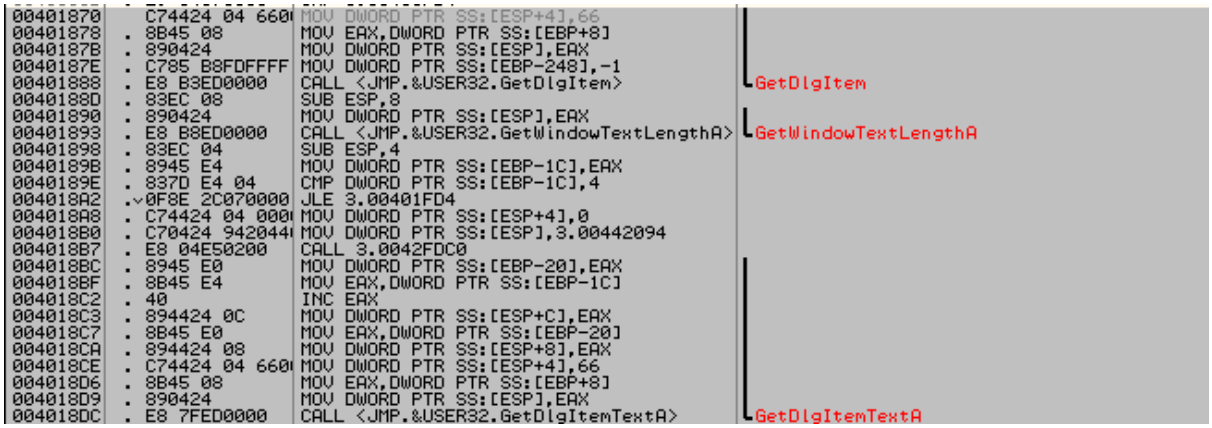
Vậy serial hợp lệ sẽ là: MEOAA-NFPBB-OAQCC-PBRDD.

Kiểm tra:



BÀI 3.

Đoạn mã xử lý chuỗi username:

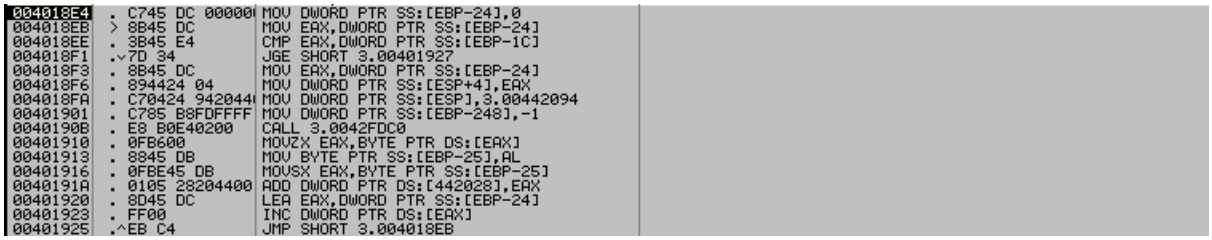


Chương trình thực hiện thu thập dữ liệu được nhập từ người dùng qua các bước sau:

1. Gọi hàm GetDlgItem để cho phép người dùng nhập chữ vào ô trống.
2. Gọi hàm GetWindowTextLengthA để đếm số lượng kí tự trong chuỗi username, kết quả được lưu trong thanh ghi EAX. Sau đó độ dài chuỗi được lưu ở địa chỉ EBP – 1C.
3. Sau khi đã có được độ dài chuỗi, so sánh với 4. Nếu chuỗi có độ dài lớn hơn 4 thì chuỗi là hợp lệ và tiếp tục. Ngược lại, nhảy tới địa chỉ 0x00401FD4 và thực hiện dừng chương trình.
4. Gọi hàm GetDlgItemTextA để lấy vào chuỗi username. Được lưu ở địa chỉ 0x00442094.

Sau khi đã có được username, chương trình bắt đầu tính toán các con số được hash từ chuỗi ban đầu. Được thực hiện bằng thuật toán trình bày sau đây:

Đầu tiên, cộng tất cả giá trị theo bảng mã ASCII của username, gọi giá trị này là R1, điều này được thực hiện thông qua vòng lặp:



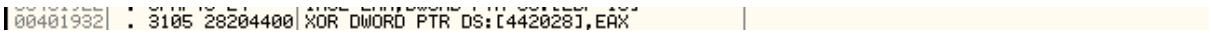
- Vòng lặp này chạy qua các kí tự thứ i chuỗi rồi cộng vào một vùng nhớ chứa tổng. Số chỉ index được lưu ở địa chỉ EBP – 24. Ở dòng đầu tiên trong ảnh thì tại đó có lệnh **MOV DWORD PTR SS:[EBP – 24], 0**, tại đây khởi tạo biến index có giá trị 0.
- Có thể thấy giá trị index này sẽ được tăng dần và sẽ thoát vòng lặp khi nó lớn hơn hoặc bằng độ dài username. Điều này được thể hiện ở 2 câu lệnh:
  - **CMP EAX, DWORD PTR SS:[EBP – 1C]**
  - **JGE SHORT 3.00401927**

- Với mỗi lần lặp, ta gán con trỏ chỉ vào chuỗi username bắt đầu từ vị trí thứ i vào thanh ghi EAX, thông qua lệnh **CALL 3.0042FDC0**. Sau đó, tách kí tự đầu tiên của chuỗi đó vào EAX, thông qua lệnh **MOVZX EAX, BYTE PTR DS:[EAX]**, câu lệnh này tách 1 byte của chuỗi nguồn và gán vào đích (ngoài ra các bit đầu tiên của thanh ghi không chứa giá trị sẽ được chỉnh thành 0). Sau đó, lưu kí tự vừa tách được vào stack, cụ thể ở vị trí có địa chỉ EBP – 25, qua câu lệnh **MOV BYTE PTR SS:[EBP – 25], AL** (Vì 1 kí tự có kích thước 1 byte, nên ở đây lấy nguồn là AL – 16 bit thấp của thanh ghi, là đủ).
- Tương tự với vòng lặp tính tổng trong các ngôn ngữ lập trình bậc cao, ta cần một nơi để chứa tổng hiện có để tính toán qua mỗi lần lặp. Vùng nhớ đó chính là 0x00442028, ban đầu được khởi tạo bằng 0, và cũng chính là vùng nhớ để lưu giá trị R1 ta cần tìm. Vùng nhớ này được đề cập trong lệnh **ADD DWORD PTR DS:[442028], EAX** để cộng các kí tự được lưu ở thanh ghi EAX ở bước trên trong mỗi lần lặp.
- Cuối cùng, ta tăng index lên 1 bằng lệnh **INC** lưu tại địa chỉ EBP – 24 nói trên. Nhưng vì không thể thực hiện **INC** là phép cộng trực tiếp trên stack, nên ta tính toán thông qua EAX bằng lệnh **LEA –** dùng để cộng trừ trực tiếp tại vùng nhớ được chỉ định.
- Và sau đó, cứ mỗi lần lặp được thực hiện, lệnh **CALL 3.0042FDC0** đề cập ở trên sẽ gán vào EAX chuỗi bắt đầu từ kí tự thứ i. Giá trị i này trước khi gọi phải được lưu vào vùng nhớ ESP + 4. Tương tự như con trỏ chuỗi trong C/C++ vậy, giả sử con trỏ chỉ vào chuỗi là ptr, vậy thì việc **CALL 3.0042FDC0** sẽ trả về ptr + i, lưu chuỗi đó vào EAX.

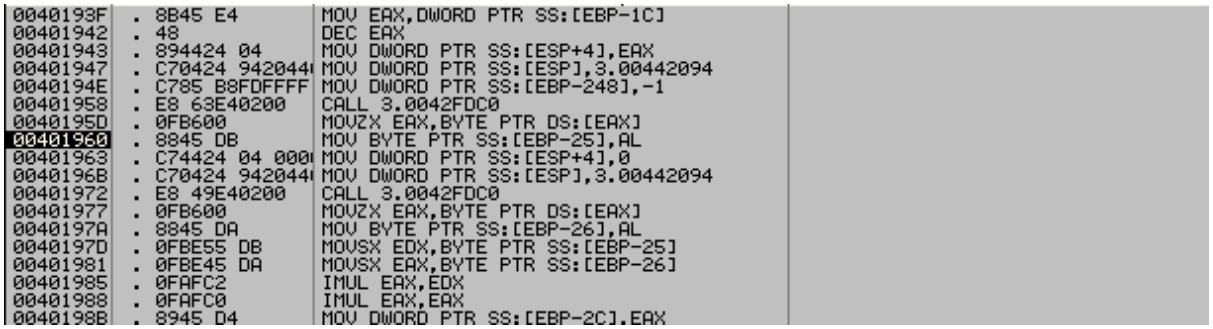
Tiếp theo, lấy độ dài chuỗi mũ 3, được thể hiện qua câu lệnh:

- Chương trình gán giá trị tại địa chỉ EBP – 1C (nơi chứa độ dài chuỗi username) vào EAX. Sau đó gọi **IMUL EAX, DWORD PTR SS:[EBP – 1C]** 2 lần, tức là lấy độ dài chuỗi nhân với chính nó thêm 2 lần, tương đương mũ 3.

Tiếp theo, lấy kết quả vừa tìm được XOR với R1 (tức giá trị ở 0x00442028), và gán ngược lại vào R1, ta được R1 mới.

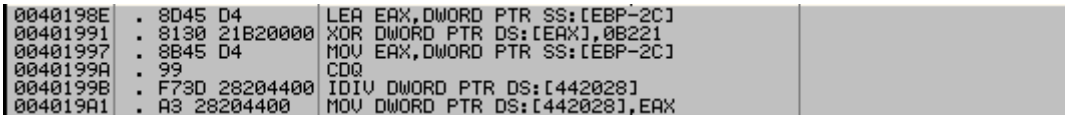


Sau đó, ta lấy giá trị ASCII kí tự đầu tiên của chuỗi nhân với ASCII kí tự cuối cùng của chuỗi, rồi lấy kết quả vừa tìm được bình phương, gọi giá trị này là R2. Được thể hiện qua đoạn mã sau:



- Lấy index kí tự cuối cùng bằng cách lấy số lượng kí tự - 1, thông qua:
  - **MOV EAX, DWORD PTR SS:[EBP – 1C]**
  - **DEC EAX**
- Sau đó, gọi **CALL 3.0042FDC0** và **MOVZX EAX, BYTE PTR DS:[EAX]** để lấy kí tự thứ cuối cùng và lưu vào EBP – 25. Sau đó, thực hiện tương tự để lấy kí tự đầu tiên trong chuỗi, bằng cách điều chỉnh ESP + 4 thành 0 và lưu nó vào EBP – 26.
- Cuối cùng ta gán lần lượt giá trị tại EBP – 25, EBP – 26 vào thanh ghi EDX và EAX, thông qua 2 lệnh **MOVSX** ở gần cuối và sau đó nhân chúng với nhau, rồi bình phương. **IMUL EAX, EDX** tức là ASCII kí tự đầu X ASCII kí tự cuối và gán ngược lại EAX. **IMUL EAX, EAX** nghĩa là bình phương kết quả vừa tìm được. Gán giá kết quả R1 vào EBP – 2C.

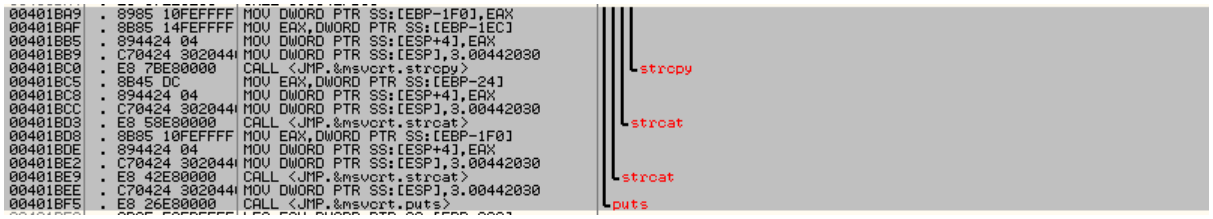
Lấy giá trị R2 XOR với hằng số 0x0B221 ta được R2 mới. Sau đó lấy R2 vừa tìm được chia R1, ta được R1 mới.



- **LEA EAX, DWORD PTR SS:[EBP – 2C]** cho phép chỉnh sửa trực tiếp trên EBP – 2, tiếp theo đó là phép **XOR DWORD PTR DS:[EAX], 0B221**, kết quả ngay lập tức được lưu vào EBP – 2C.
- Lấy kết quả vừa tìm được chia cho R1, thông qua lệnh **IDIV DWORD PTR:[442028]**, IDIV lấy giá trị được lưu tại EAX chia với một số nguyên, ở đây là tại địa chỉ 0x00442028 tức là R1. Sau đó lưu kết quả vừa tìm được vào 0x00442028, ta được R1 mới.

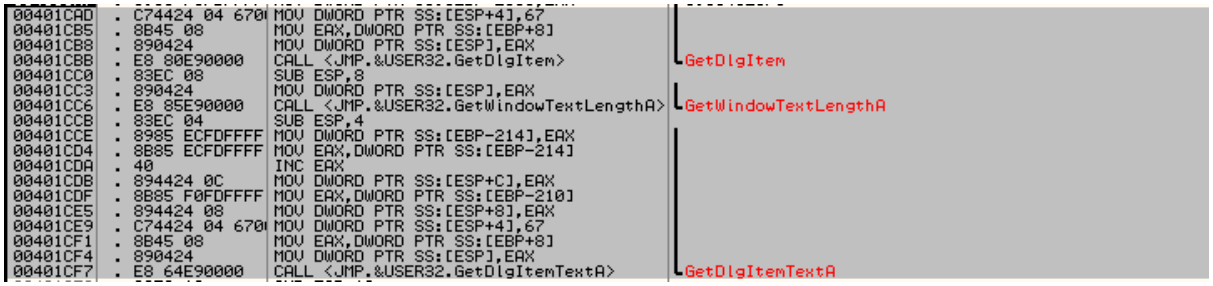
Và ta đã hoàn thành thuật toán tìm Key. Key là một chuỗi gồm 2 số được nối với nhau bằng dấu gạch ngang "-". **Key chính là "<R1>-<R2>"**.

Từ đoạn 0x004019A6 đến 0x00401BA4 là quá trình biến đổi 2 số R1, R2 tìm được từ dạng số nguyên sạng kiểu chuỗi, cùng với một số thao tác vùng nhớ. Ta phân tích đoạn tiếp theo, nơi dùng để tạo ra chuỗi key kết quả, bằng phép strcat và puts:

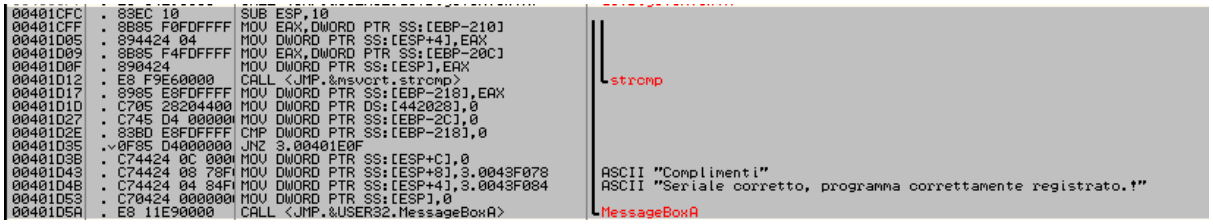


- Kết quả đã chuyển sang kiểu chuỗi kí tự của B được lưu ở vùng nhớ có địa chỉ EBP – 1F0, tương tự với của R1 được lưu ở EBP – 1EC.
- Cách làm là ta cần một nơi để lưu chuỗi kết quả, và đó là ở 0x00442030. Đầu tiên ta cần copy chuỗi R1 vào 0x00442030, sau đó lần lượt strcat() dấu gạch ngang "-" và chuỗi R2 vào đó. 3 câu lệnh sau để gọi hàm strcpy() chuỗi R1 vào 0x00442030:
    - **MOV EAX, DWORD PTR SS:[EBP – 1EC]**: Gán chuỗi R1 vào EAX.
    - **MOV DWORD PTR SS:[ESP + 4], EAX**: Gán vào phần tử thứ 2 trong stack để làm tham số thứ 2 cho hàm strcpy(), cho biết nguồn là EAX.
    - **MOV DWORD PTR SS:[ESP], 3.00442030**: Tương tự, cho biết đích là vùng nhớ 0x00442030.
    - **CALL <JMP.&msvcrt.strcpy>**: Gọi hàm.
- Gọi hàm strcat để cat chuỗi "-" vào 0x00442030:
    - **MOV EAX, DWORD PTR SS:[EBP – 24]**: Gán chuỗi "-" vào EAX. Chuỗi này được tạo ra ở những lệnh trước phần này, được lưu ở EBP – 24.
    - **MOV DWORD PTR SS:[ESP + 4], EAX**: Gán vào phần tử thứ 2 trong stack để làm tham số thứ 2 cho hàm strcpy(), cho biết nguồn là EAX.
    - **MOV DWORD PTR SS:[ESP], 3.00442030**: Cho biết đích là vùng nhớ 0x00442030.
    - **CALL <JMP.&msvcrt.strcat>**: Gọi hàm. Kết quả sẽ là chuỗi "<R1>-".
- Gọi thêm hàm strcat để cat chuỗi R2 vào 0x00442030:
    - **MOV EAX, DWORD PTR SS:[EBP – 1F0]**: Gán chuỗi "<R2>" vào EAX.
    - **MOV DWORD PTR SS:[ESP + 4], EAX**: Gán vào phần tử thứ 2 trong stack để làm tham số thứ 2 cho hàm strcpy(), cho biết nguồn là EAX.
    - **MOV DWORD PTR SS:[ESP], 3.00442030**: Cho biết đích là vùng nhớ 0x00442030.
    - **CALL <JMP.&msvcrt.strcat>**: Gọi hàm. Kết quả sẽ là chuỗi "<R1>-<R2>".

Sau đó, chương trình cũng lấy chuỗi key nhập vào tương tự như username.



Cuối cùng, điều cần làm là so sánh chuỗi mà người dùng nhập với key được tạo, bằng hàm strcmp():



- Strcmp() sẽ trả về giá trị 0 nếu đúng và ngược lại là khác 0. Dòng **CMP DWORD PTR SS:[EBP – 218], 0** quyết định kết quả của chương trình là có in dòng thông báo thành công hay kết thúc.

Ví dụ minh họa:

Nhập vào username “ABCDE”.

1. Chuỗi có độ dài là 5 > 4. Thỏa yêu cầu độ dài tối thiểu.
2. Theo thuật toán:
3. Tổng các giá trị ASCII: R1 = ‘A’ (65) + ‘B’ (66) + ‘C’ (67) + ‘D’ (68) + ‘E’ (69) = 335.
4. Độ dài chuỗi mũ 3 = 5 * 5 * 5 = 125.
5. XOR 2 giá trị trên: R1 = R1 XOR 125 = 335 XOR 125 = 306
6. Kí tự đầu nhân kí tự cuối: R2 = ‘A’ (65) * ‘E’ (69) = 4485.
7. R2 mũ 2: R2 = R2 * R2 = 4485 * 4485 = 20115225.
8. R2 XOR 0x0B221: R2 = R2 XOR 0x0B221 = 20115225 XOR 45601 = 20077880.
9. R1 = R2 // R1 = 2077880 // 306 = 65613.

Vậy Key là: 65613-2007880

Kết quả: