

**ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH**  
**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN**



**ĐỒ ÁN HỆ THỐNG MÁY TÍNH**  
**Project: Cracking phần mềm**

**Môn học:** Hệ thống máy tính

**Giảng viên hướng dẫn:** Lê Viết Long

Phạm Tuấn Sơn

**Sinh viên thực hiện:** Phan Thị Phương Chi

Hồ Thùy Hương

Nguyễn Thị Ánh Ngọc

**Thành phố Hồ Chí Minh – 2025**

# MỤC LỤC

<b>1. Thông tin sinh viên và mức độ hoàn thiện .....</b>	<b>3</b>
1.1. Thông tin sinh viên .....	3
1.2. Mức độ hoàn thiện .....	3
1.3. Bảng phân công công việc .....	3
<b>2. Nội dung .....</b>	<b>4</b>
1.1. File 1.exe .....	4
1.1.1. Mạnh mỗi tìm ra đoạn phát sinh key .....	4
1.1.2. Phân tích đoạn phát sinh key .....	7
1.1.3. Kết luận và ví dụ minh họa.....	12
1.1.4. Chương trình keygen .....	14
1.2. File 2.exe .....	16
1.2.1. Phân tích đoạn phát sinh key .....	16
1.2.2. Kết luận và ví dụ minh họa.....	27
1.2.3. Chương trình keygen .....	30
1.3. File 3.exe .....	32
1.3.1. Phân tích đoạn phát sinh key .....	32
1.3.2. Kết luận và ví dụ minh họa.....	40
1.3.3. Chương trình keygen .....	42
<b>3. Tài liệu tham khảo.....</b>	<b>43</b>

## 1. Thông tin sinh viên và mức độ hoàn thiện

### 1.1. Thông tin sinh viên

STT	Họ và tên	MSSV
1	Phan Thị Phương Chi	23120025
2	Hồ Thùy Hương	23120046
3	Nguyễn Thị Ánh Ngọc	23120061

### 1.2. Mức độ hoàn thiện

STT	Yêu cầu	Tỉ lệ hoàn thành
1	Tải và sử dụng các ứng dụng hỗ trợ trong thư mục đồ án (Ollydbg, Ida)	100%
2	Phân công công việc cụ thể cho từng thành viên.	100%
3	Với mỗi crackme, chỉ ra đoạn phát sinh key, giải thích ý nghĩa và đưa ra một key tương ứng với username minh họa.	100%
4	Viết chương trình keygen để phát sinh khóa từ username người dùng nhập vào.	100%
5	Viết báo cáo tổng hợp các kết quả đã làm.	100%

### 1.3. Bảng phân công công việc

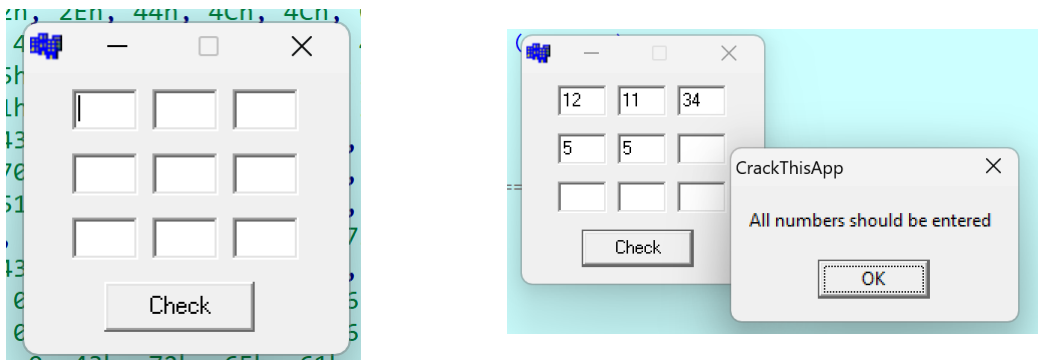
STT	Người thực hiện	Công việc được phân
1	Hồ Thùy Hương	1. Làm bài 1 2. Kiểm tra lại bài 2 3. Viết báo cáo cho bài 1
2	Phan Thị Phương Chi	1. Làm bài 2 2. Kiểm tra lại bài 3 3. Viết báo cáo cho bài 2
3	Nguyễn Thị Ánh Ngọc	1. Làm bài 3 2. Kiểm tra lại bài 1 3. Viết báo cáo cho bài 3

## 2. Nội dung

### 1.1. File 1.exe

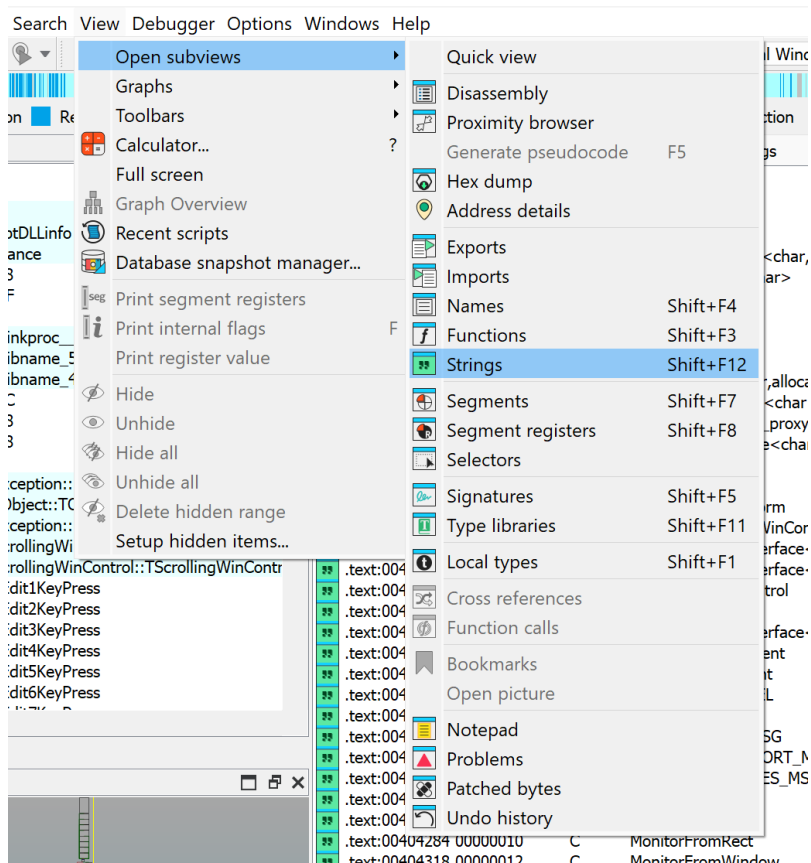
#### 1.1.1. Mạnh mẽ tìm ra đoạn phát sinh key

- Ban đầu, chạy thử file .exe để biết được key cần gì. Không nhập được kí tự nào khác ngoài chữ số. Sau đó, thử nhập không đủ 9 ô, chương trình hiển thị như bên dưới.

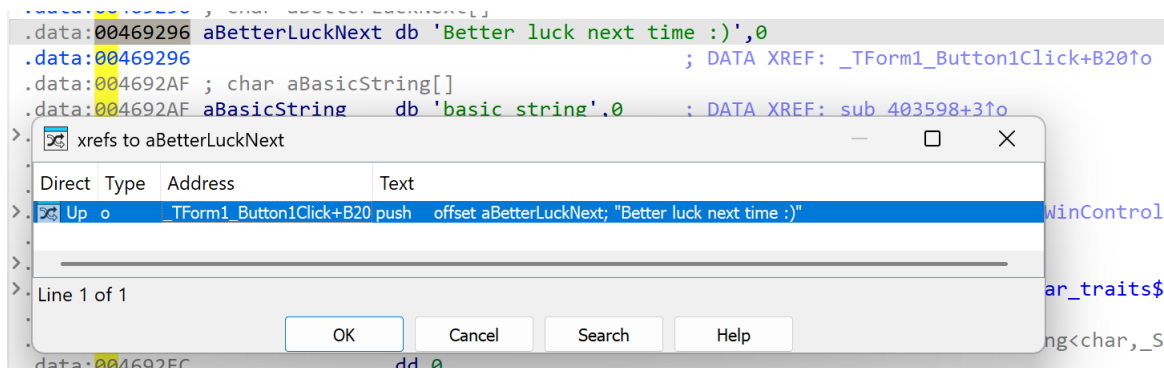


Như vậy, xác định được chương trình yêu cầu **key phải có đủ 9 số**.

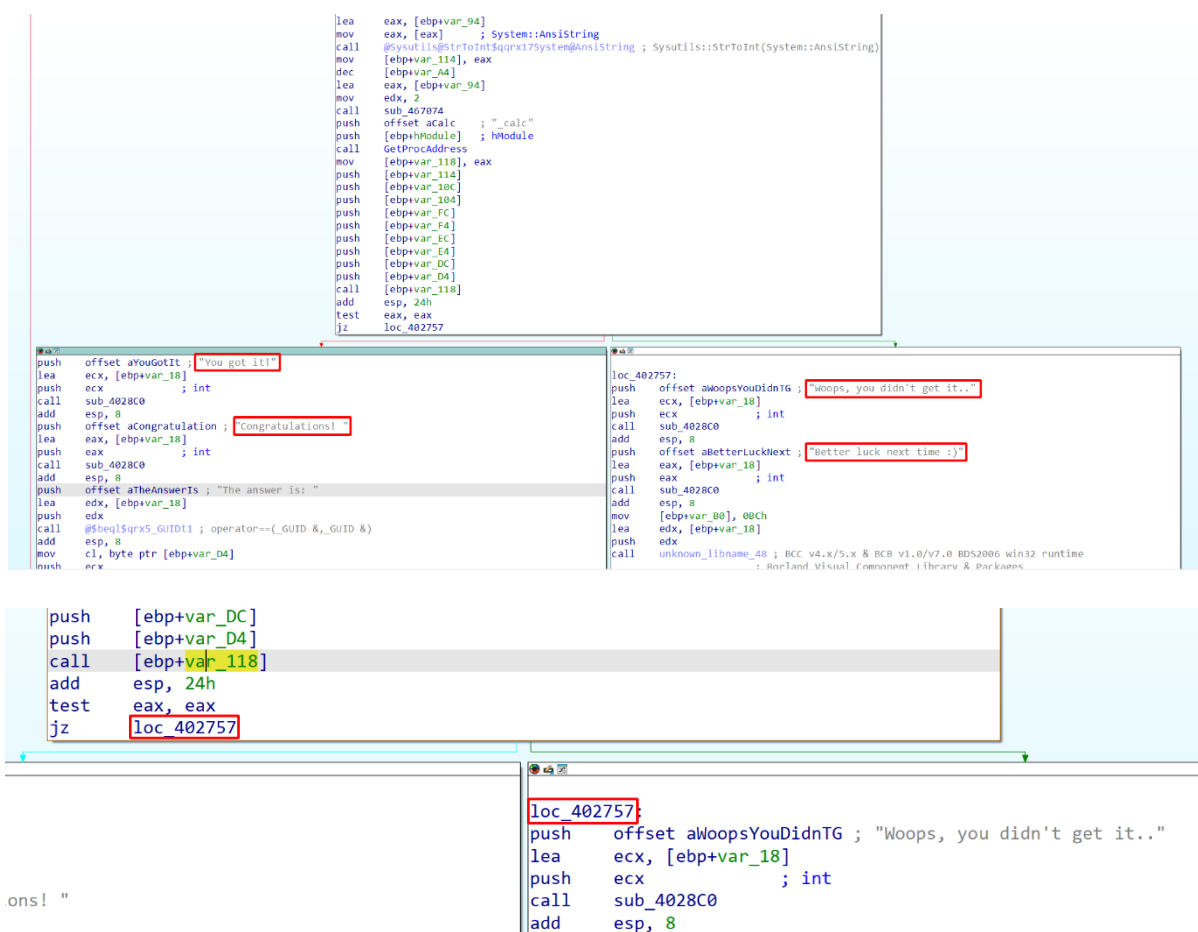
- Trong Ida, tìm chuỗi “Better luck next time :)” và tìm tham chiếu đến hàm đang dùng chuỗi này. Sau đó, đi đến hàm và mở Graph view để xem được một cách tổng quát các nhánh xử lý của chương trình.



.data:00469...	00000006	C	_calc
.data:00469...	0000000C	C	You got it!
.data:00469...	00000012	C	Congratulations!
.data:00469...	00000010	C	The answer is:
.data:00469...	0000001B	C	Woops, you didn't get it..
.data:00469...	00000019	C	Better luck next time :)
.data:00469...	0000000D	C	basic_string



- Sau các bước trên, nhận được Graph view như hình bên dưới, nhận biết được nhánh thất bại và nhánh thành công từ các thông báo nó in ra: **nhánh thành công** (“You got it!”, “Congratulations!”) và **nhánh thất bại** (“Woops, you didn’t get it...”, “Better luck next time”).



Trước khi chuyển đến các nhánh, chú ý các câu lệnh sau:

**jz loc\_402757:** Nhảy đến địa chỉ 402757 nếu ZF (Zero Flag) = 1. Nhận thấy địa chỉ 402757 là phần xử lý của nhánh thất bại. Ta tiếp tục tìm phép so sánh nào set ZF.

**test eax, eax:** Thực hiện phép AND giữa eax với chính nó để thiết lập cờ (flags), nhưng không thay đổi giá trị eax.

Cụ thể câu lệnh kiểm tra xem eax có bằng 0 hay không:

- Nếu  $eax == 0$  thì  $ZF = 1$ .
- Nếu  $eax != 0$  thì  $ZF = 0$ .

**call [ebp+var\_118]:** Gọi hàm được lưu tại địa chỉ [ebp+var\_118].

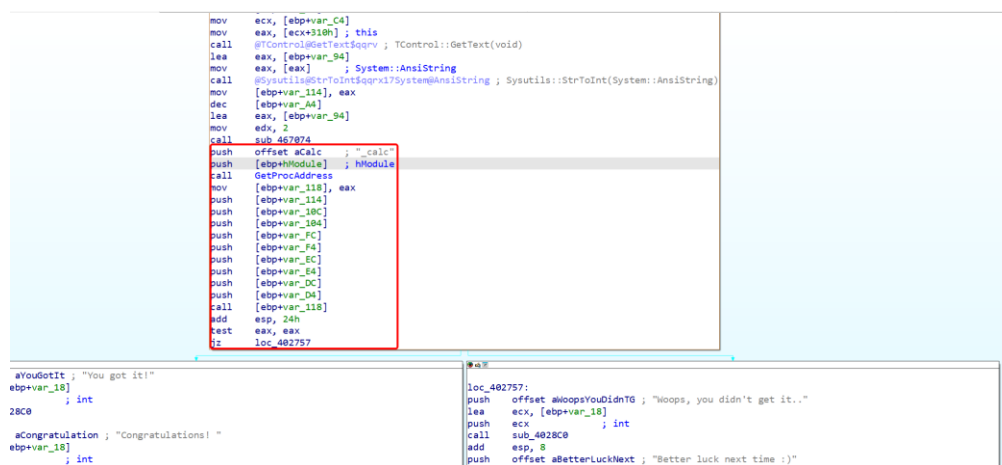
Cụ thể:

- var\_118 là một biến cục bộ hoặc tham chiếu cách vị trí ebp một khoảng - 0x118.
- [ebp+var\_118] chứa một địa chỉ hàm → lệnh call sẽ nhảy đến địa chỉ đó và thực thi.

Như vậy, dự đoán được lời gọi hàm ở được lưu tại địa [ebp+var\_118] thực hiện thao tác gì đó, sau đó kiểm tra giá trị eax có bằng 0 hay không. Nếu bằng 0 thì set  $ZF = 1$  và trả ra nhánh thất bại, ngược lại thì đoạn mã tiếp tục, tức là xử lý theo nhánh thành công.

Và eax có thể lưu trữ kết quả của hàm nào đó được lưu tại địa chỉ [ebp+var\_118].

- Nhìn tiếp đoạn mã bên trên nằm trong khối xử lý trước khi chuyển đến các nhánh, chú ý các lệnh sau:



```

mov     ecx, [ebp+var_C4]
mov     eax, [ecx+310h]; this
call    @?Control@GetText$qqv; TControl::GetText(void)
lea     eax, [ebp+var_94]
mov     eax, [eax]; System::AnsiString
call    @Sysutils@StrToInt$qqx17System@AnsiString; Sysutils::StrToInt(System::AnsiString)
mov     [ebp+var_114], eax
dec     [ebp+var_1A4]
lea     eax, [ebp+var_94]
mov     edx, 2
call    sub_407074
push    offset aCalc; "_calc"
push    [ebp+Module]; hModule
call    GetProcAddress
mov     [ebp+var_118], eax
push    [ebp+var_114]
push    [ebp+var_10C]
push    [ebp+var_104]
push    [ebp+var_FC]
push    [ebp+var_F4]
push    [ebp+var_EC]
push    [ebp+var_E4]
push    [ebp+var_DC]
push    [ebp+var_D4]
call    [ebp+var_118]
add     esp, 24h
test    eax, eax
jz      loc_402757

aYouGotIt; "You got it!"
ebp+var_18; int
28C0
aCongratulation; "Congratulations!"
ebp+var_18; int
~~~~

loc_402757:
push    offset aOopsYouDnTG; "Oops, you didn't get it..."
lea     ecx, [ebp+var_18]
push    ecx; int
call    sub_4028C0
add     esp, 8
push    offset aBetterLuckNext; "Better luck next time :)"

```

**push offset aCalc; "\_calc":** Push địa chỉ chuỗi "\_calc" lên stack (chuỗi tên hàm cần tìm).

**push [ebp+hModule]; hModule:** Push hModule (handle module DLL hoặc EXE đã load) lên stack.

**call GetProcAddress:** Gọi GetProcAddress(hModule, "\_calc"), tìm địa chỉ hàm \_calc trong module.

**mov [ebp+var\_118], eax:** Lưu địa chỉ hàm \_calc vào biến var\_118.

**push [ebp+var\_114]:** Push tham số thứ 9.

**push [ebp+var\_10C]:** Push tham số thứ 8.

**push [ebp+var\_104]:** Push tham số thứ 7.

**push [ebp+var\_FC]:** Push tham số thứ 6.

**push [ebp+var\_F4]:** Push tham số thứ 5.

**push [ebp+var\_EC]:** Push tham số thứ 4.

**push [ebp+var\_E4]:** Push tham số thứ 3.

**push [ebp+var\_DC]:** Push tham số thứ 2.

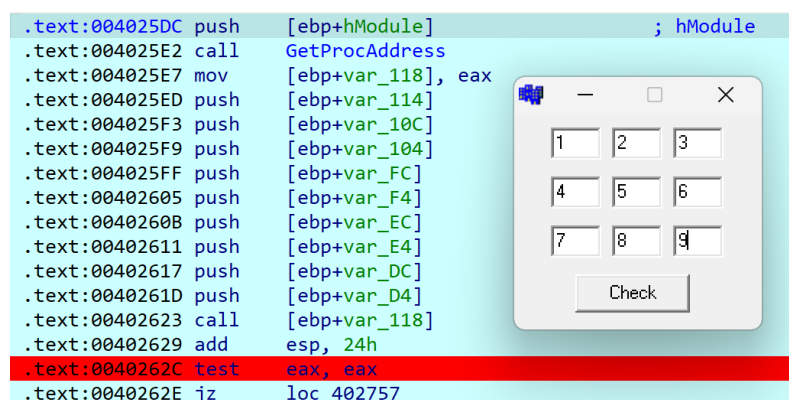
**push [ebp+var\_D4]:** Push tham số thứ 1.

**call [ebp+var\_118]:** Gọi hàm \_calc với 9 tham số vừa được push.

**add esp, 24h:** Sau khi gọi hàm, dọn dẹp  $9 \times 4 = 36$  bytes (0x24h) khỏi stack (cleanup stack sau call).

Như vậy, hàm \_calc chứa logic xử lý key và 9 tham số có thể là 9 số mà chương trình yêu cầu nhập vào ban đầu. Tiếp tục, cần kiểm tra stack tại 9 địa chỉ trên sau khi nhập vào để xem có phải là 9 tham số nhập vào hay không và kiểm tra hàm \_calc trong file .dll.

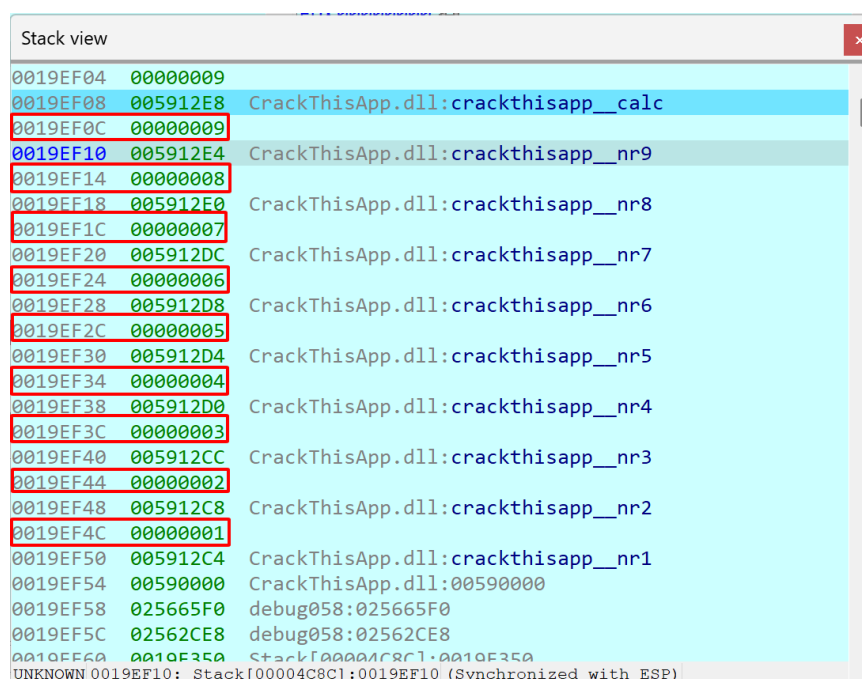
- Đặt breakpoint tại lệnh sau các lệnh push như ảnh và nhập vào 9 số, sau đó kiểm tra địa chỉ vùng nhớ của các tham số đó và kiểm tra nó trong stack trong Stack view.



Trở đến từng địa chỉ của tham số, ví dụ [ebp+var\_114] ta được địa chỉ vùng nhớ của tham số thứ 9. Dùng địa chỉ đó và kiểm tra nội dung được chứa trong nó trong

stack. Lần lượt thực hiện với 9 tham số, ta được giá trị của 9 tham số từ 1 đến 9 như 9 tham số nhập vào.

```
.text:004025E7 mov     [ebp+var_118], eax
.text:004025ED push    [ebp+var_114]
.text:004025F3 push    [ebp+var_114]
.text:004025F9 push    [ebp+var_114]=[Stack[00004C8C]:0019EF0C]
.text:004025FF push    [ebp+var_114] db 9
.text:00402605 push    [ebp+var_114] db 0
.text:0040260B push    [ebp+var_114] db 0
.text:00402611 push    [ebp+var_114] db 0
.text:00402617 push    [ebp+var_114] db 0E4h
.text:0040261D push    [ebp+var_114] db 12h
.text:00402623 call    [ebp+var_114] db 59h ; Y
.text:00402629 add     esp, 24h db 0
.text:0040262C test    eax, eax db 8
.text:0040262E jz      loc_40275 db 0
```



Xác định được 9 tham số đó lấy từ 9 tham số nhập vào ban đầu khi chương trình yêu cầu. Tiếp tục, kiểm tra hàm `_calc` trong file `.dll`.

- Truy cập file `.dll`, vào phần Exports để tìm hàm `_calc`. Sau đó double-click vào để xem hàm `_calc`.

Hàm `calc` nhận 9 tham số nguyên (a1 đến a9). Tính ra 3 giá trị (v9, v12, v11) từ một vài chữ số cụ thể của các tham số. Tính lại giá trị từ 9 tham số bằng hàm `sub_401478`. Cuối cùng, so sánh giá trị từ `sub_401478` với 3 giá trị (v9, v12, v11). Trải qua 4 vòng check. Nếu cả 4 điều kiện đều đúng, trả về `true`; ngược lại `false`. Như vậy, cần tìm 9 số mà nhập vào thỏa được điều kiện của hàm `_calc`.

- Tiếp tục quan sát đoạn code phía sau phần in ra thông báo “Congratulations!” ở bên nhánh thành công, chú ý một số câu lệnh:



**push offset aYouGotIt ; “You got it!”:** Đẩy địa chỉ chuỗi “You got it!” vào stack.

**lea ecx, [ebp+var\_18]:** Load địa chỉ vùng nhớ tạm [ebp+var\_18].

**push ecx; int:** Đẩy địa chỉ vùng nhớ vào stack (nơi sẽ lưu ghép kết quả).

**call sub\_4028C0:** Gọi hàm sub\_4028C0 để nối chuỗi "You got it!" vào vùng nhớ [ebp+var\_18].

(Lặp lại tương tự cho "Congratulations!" và "The answer is:")

**mov cl, byte ptr [ebp+var\_D4]**

**push ecx**

**lea eax, [ebp+var\_18]**

**push eax**

**call sub\_4028F4**

→ Lấy từng byte (ký tự) từ biến lưu giá trị. Gọi hàm sub\_4028F4 để nối giá trị đó vào chuỗi được lưu ở vùng nhớ [ebp+var\_18].

**(Lặp lại cho các biến var\_DC, var\_E4,...var\_114)**

→ Ghi đủ 9 giá trị ghép thành 1 chuỗi kết quả.

**mov [ebp+var\_B0], 0B0h:** Gán giá trị 0xB0 vào var\_B0 (có thể là kích thước hoặc tham số cần thiết cho tiếp theo).

**lea ecx, [ebp+var\_18]**

**push ecx**

**call unknown\_libname\_48**

**pop ecx**

**mov edx, eax**

→ Xử lý hoàn thiện chuỗi, biến các giá trị được lưu tại vùng nhớ [ebp+var\_18] thành string chuẩn và lưu vào eax, truyền vào tiếp theo để dùng cho GUI.

**lea eax, [ebp+var\_98]**

**call sub\_467000:** Khởi tạo biến kiểu struct hoặc class

**inc [ebp+var\_A4]**

**mov eax, [eax]**

**call @Dialogs@ShowMessage\$qqrx17System@AnsiString**

→ Gọi hàm sub\_467000 thiết lập var\_98. Sau đó gọi hàm ShowMessage để hiện hộp thoại MessageBox hiển thị nội dung vừa ghép.

```
dec [ebp+var_A4]
```

```
lea eax, [ebp+var_98]
```

```
mov edx, 2
```

```
call sub_467074
```

→ Gọi sub\_467074(obj, 2) để **giải phóng bộ nhớ hoặc handle** đã dùng để tạo chuỗi AnsiString. Chức năng của đoạn mã trên là dọn dẹp tài nguyên.

**jmp short loc\_4027BC:** Xử lý kết thúc và nhảy về cuối nhánh (tiếp tục chương trình).

### Tóm gọn lại:

Ghép thông báo “You got it! Congratulations! The answer is:” + 9 số/ byte thành một chuỗi hoàn chỉnh và hiển thị chuỗi đó bằng hộp thoại MessageBox.

- Vấn đề còn lại là tìm 9 số nguyên sao cho hàm \_calc trả về 1 (true). Phần còn lại là chương trình tự mã hóa thành chuỗi và in ra kết quả.

### 1.1.2. Phân tích đoạn phát sinh key

#### Vòng check thứ nhất

.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

**Giải thích:**  $[var\_18] = 10000 \cdot (n5 \% 10) + 10000 \cdot (n1 \% 10) + 100 \cdot [(n3 / 10 \% 10) + 5] + 10 \cdot (n9 / 10 \% 10) + (n7 \% 10) + 2$

```
mov eax, edx
```

```
add eax, 2
```

→  $eax = (n7 \% 10) + 2$

```
mov edx, ecx: edx = n5 % 10
```

```
add edx, edx: edx = 2 * (n5 % 10)
```

```
lea edx, [edx+edx*4]: edx = 10 * (n5 % 10)
```

```
add edx, [ebp+var_4]: [ebp+var_4] = n1 % 10 → edx = 10 * (n5 % 10) + (n1 % 10)
```

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4]:**  $edx *= 5 \rightarrow edx = 100*(n5\%10) + 10*(n1\%10)$

**add edx, [ebp+var\_10]:**  $[ebp+var\_10] = (n3/10\%10)+5$

$\rightarrow edx = 100*(n5\%10) + 10*(n1\%10) + [(n3/10\%10)+5]$

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4]:**  $edx *= 5 \rightarrow edx = 1000*(n5\%10) + 100*(n1\%10) + 10*[(n3/10\%10)+5]$

**add edx, [ebp+var\_14]:**  $[ebp+var\_14] = (n9/10\%10) \rightarrow edx = 1000*(n5\%10) + 100*(n1\%10) + 10*[(n3/10\%10)+5] + (n9/10\%10)$

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4];**  $edx *= 5 \rightarrow edx = 10000*(n5\%10) + 1000*(n1\%10) + 100*[(n3/10\%10)+5] + 10*(n9/10\%10)$

**add edx, eax:**  $eax = (n7\%10)+2 \rightarrow$  Có được công thức cho vòng check đầu tiên.

**mov [ebp+var\_18], edx:** Lưu kết quả vòng check đầu tiên.

## Vòng check thứ 2

```
.text:004013A4      mov     ecx, 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
```

**Giải thích:**  $[var\_1C] = 10000*(n5\%10) + 1000*(n6\%10) + 100*[(n3/10\%10)+5] + 10*(n9/10\%10) + (n7\%10)+2$

**mov edx, ecx:**  $edx = n5 \% 10$

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4]:**  $edx *= 5 \rightarrow edx = 10*(n5\%10)$

**add edx, [ebp+var\_8]:**  $[ebp+var\_8] = n6\%10 \rightarrow edx = 10*(n5\%10) + (n6\%10)$

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4];**  $edx *= 5 \rightarrow edx = 100*(n5\%10) + 10*(n6\%10)$

**add edx, [ebp+var\_10]:**  $[ebp+var\_10] = [(n3/10\%10)+5] \rightarrow edx = 100*(n5\%10) + 10*(n6\%10) + [(n3/10\%10)+5]$

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4]:**  $edx *= 5 \rightarrow edx = 1000*(n5\%10) + 100*(n6\%10) + 10*[(n3/10\%10)+5]$

**add edx, [ebp+var\_14]:**  $ebp+var\_14 = (n9/10\%10)$

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4]:**  $edx *= 5 \rightarrow edx = 10000*(n5\%10) + 1000*(n6\%10) + 100*[(n3/10\%10)+5] + 10*(n9/10\%10)$

**add edx, eax:**  $eax = (n7\%10)+2 \rightarrow$  Có được công thức cho vòng check thứ 2.

**mov [ebp+var\_1C], edx:** Lưu kết quả vào vòng check thứ hai.

### Vòng check thứ 3

```
.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
```

**Giải thích:**  $[var\_20] = 10000*(n5\%10) + 1000*[(n2\%10)+3] + *[(n3/10\%10)+5] + 10*(n9/10\%10) + (n7\%10)+2$

**add ecx, ecx:**  $ecx = 2*(n5\%10)$

**lea ecx, [ecx+ecx\*4]:**  $ecx *= 5 \rightarrow ecx = 10*(n5\%10)$

**add ecx, [ebp+var\_C]:**  $[ebp+var\_C] = [(n2\%10)+3] \rightarrow ecx = 10*(n5\%10) + [(n2\%10)+3]$

**mov edx, ecx:**  $edx = ecx$

**add edx, edx:**  $edx *= 2$

**lea edx, [edx+edx\*4]:**  $edx *= 5 \rightarrow edx = 100*(n5\%10) + 10*[(n2\%10)+3]$

**add edx, [ebp+var\_10]:**  $[ebp+var\_10] = [(n3/10\%10)+5]$

$\rightarrow edx = 100*(n5\%10) + 10*[(n2\%10)+3] + [(n3/10\%10)+5]$

**mov ecx, edx:**  $ecx = edx$

**add ecx, ecx:**  $ecx *= 2$

**lea ecx, [ecx+ecx\*4]:** ecx \*= 5

→ ecx = 1000\*(n5%10) + 100\*[(n2%10)+3] + 10\*[(n3/10%10)+5]

**add ecx, [ebp+var\_14]:** [ebp+var\_14] = (n9/10%10)

→ ecx = 1000\*(n5%10) + 100\*[(n2%10)+3] + 10\*[(n3/10%10)+5] + (n9/10%10)

**mov edx, ecx:** edx = ecx

**add edx, edx:** edx \*= 2

**lea edx, [edx+edx\*4]:** edx \*= 5 → edx = 10000\*(n5%10) + 1000\*[(n2%10)+3] + 100\*[(n3/10%10)+5] + 10\*(n9/10%10)

**add eax, edx:** edx = (n7%10)+2 → Có được công thức cho vòng check thứ 3.

**mov [ebp+var\_20], eax:** Lưu kết quả vào vòng check thứ 3.

### Vòng check thứ 4

```
.text:00401478
.text:00401478 ; Attributes: bp-based frame
.text:00401478
.text:00401478 ; int __cdecl sub_401478(int, int, int, int, int, int, int, int, int, int)
.text:00401478 sub_401478      proc near          ; CODE XREF: _calc+1231p
.text:00401478                                     ; _calc+1481p ...
.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
.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: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
```

**Giải thích:** Hàm `sub_00401478` =  $(n2 * n6 * n7 + n3 * n4 * n8 + n1 * n5 * n9) - (n2 * n4 * n9 + n1 * n6 * n8 + n3 * n5 * n7)$

**arg\_0 đến arg\_20** tương ứng với các đối số được truyền vào hàm, lần lượt là `n1` đến `n9`.

**Các lệnh `mov` ban đầu** di chuyển các đối số vào các thanh ghi để thực hiện phép nhân:

`eax = n1 ([ebp+arg_0])`

`esi = n4 ([ebp+arg_C])`

`ebx = n1 (eax)`

`ecx = n3 ([ebp+arg_8])`

`edx = n2 ([ebp+arg_4])`

**Các lệnh `imul` (integer multiply) thực hiện các phép nhân:**

`imul ebx, [ebp+arg_10] → ebx = n1 * n6`

`imul esi, [ebp+arg_1C] → esi = n4 * n8`

`imul esi, ecx → esi = n4 * n8 * n3`

`imul ebx, [ebp+arg_20] → ebx = n1 * n6 * n9`

**`add ebx, esi:`** `ebx += esi`

→ **`ebx`** =  $n1 * n6 * n9 + n3 * n4 * n8$

**`mov edx, [ebp+arg_4]:`** `edx = n2`

**`mov esi, [ebp+arg_18]:`** `esi = n7`

**`imul esi, edx:`** `esi = esi * edx`

→ `esi = n7 * n2 = n2 * n7` .text:004014A2

**`imul esi, [ebp+arg_14]:`** `esi = esi * [ebp+arg_14]`

→ `esi = (n2 * n7) * n5 = n2 * n5 * n7`

**`add ebx, esi:`** `ebx = ebx + esi`

→ `ebx = (n1 * n6 * n9 + n3 * n4 * n8) + (n2 * n5 * n7)`

**`mov esi, [ebp+arg_14]:`** `esi = n5`

**`imul esi, [ebp+arg_1C]:`** `esi *= [ebp+arg_1C] → esi = n5 * n8`

**`imul esi, eax:`** `esi *= eax → esi = (n5 * n8) * n1 = n1 * n5 * n8`

**`imul ecx, [ebp+arg_10]:`** `ecx *= [ebp+arg_10] → ecx = n3 * n6`

**`mov eax, [ebp+arg_20]:`** `eax = n9`

**`imul ecx, [ebp+arg_18]:`** `ecx *= [ebp+arg_18]`

→  $ecx = (n3 * n6) * n7 = n3 * n6 * n7$

**imul edx:**  $edx *= eax \rightarrow edx = n2 * n9$

**imul [ebp+arg\_C]:**  $[ebp+arg_C] *= ecx$

→  $[ebp+arg_C] = n4 * (n3 * n6 * n7) = n3 * n4 * n6 * n7$

**add ecx, esi:**  $ecx += esi \rightarrow ecx = (n3 * n6 * n7) + (n1 * n5 * n8)$

**add ecx, eax:**  $ecx += eax \rightarrow ecx = (n3 * n6 * n7 + n1 * n5 * n8) + n9$

**sub ebx, ecx:**  $ebx -= ecx$

→  $ebx = (n1 * n6 * n9 + n3 * n4 * n8 + n2 * n5 * n7) - (n3 * n6 * n7 + n1 * n5 * n8 + n9)$

**mov eax, ebx:**  $eax = ebx$  (giá trị trả về).

### 1.1.3. Kết luận và ví dụ minh họa

#### a) Kết luận

##### Tóm gọn lại, thuật toán sinh key

**Mục tiêu:** Tìm ra một bộ 9 số ( $n1$  đến  $n9$ ) thỏa mãn một số điều kiện nhất định.

**Các bước thực hiện:**

#### 1. Thiết lập mối quan hệ cho $n1$ , $n6$ , và $n2$ :

- Đầu tiên, có một ràng buộc quan trọng:
  - Chữ số cuối cùng của  $n1$  ( $n1 \% 10$ ) phải bằng với chữ số cuối cùng của  $n6$  ( $n6 \% 10$ ).
  - Đồng thời, chữ số cuối cùng của  $n6$  phải bằng với chữ số cuối cùng của  $n2$  cộng thêm 3 ( $(n2 \% 10) + 3$ ).
- Để bắt đầu, chúng ta sẽ **chọn một số ngẫu nhiên** cho  $n2$ .
- Sau khi chọn  $n2$ , chúng ta sẽ tính  $n1$  và  $n6$  dựa trên  $n2$  theo quy tắc sau:  $n1$  và  $n6$  sẽ có cùng chữ số cuối cùng, và chữ số cuối cùng này phải bằng chữ số cuối cùng của  $n2$  cộng thêm 3.
  - Ví dụ: Nếu  $n2$  là 15 (chữ số cuối cùng là 5), thì  $(n2 \% 10) + 3 = 5 + 3 = 8$ . Vậy, chúng ta có thể chọn  $n1$  là 8 hoặc 18 hoặc 28,... và  $n6$  cũng là 8 hoặc 18 hoặc 28,... (chỉ cần chữ số cuối cùng là 8).

#### 2. Gán giá trị ngẫu nhiên cho các số còn lại:

- Chúng ta sẽ **chọn các giá trị ngẫu nhiên** cho  $n3$ ,  $n4$ ,  $n5$ ,  $n7$ , và  $n8$ . Hiện tại không có ràng buộc cụ thể nào được đặt ra cho các số này.

### 3. Tìm giá trị phù hợp cho n9:

- Đây là bước quan trọng nhất. Chúng ta cần **tìm một giá trị** cho n9 sao cho giá trị của một biến tên là check4 (được tính toán dựa trên cả 9 số) phải **khớp với** giá trị của ba biến khác là check1, check2, và check3.
- Điều này có nghĩa là giá trị của fourthcheck phải **bằng đồng thời** giá trị của check1, check2, và check3.
- check1, check2, và check3 đã được tính toán ở các bước trước dựa trên các giá trị đã gán cho n1 đến n8.

### 4. Thử lại nếu không tìm được n9:

- Nếu sau khi thử nhiều giá trị khác nhau cho n9, chúng ta **không thể tìm được** một giá trị nào khiến check4 khớp với cả ba giá trị check1, check2, và check3, thì chúng ta cần **quay lại bước 1**.
- Ở bước 1, chúng ta sẽ **chọn một giá trị ngẫu nhiên mới** cho n2 và lặp lại toàn bộ quá trình (tính lại n1 và n6, giữ nguyên hoặc chọn lại giá trị ngẫu nhiên cho các số khác, và cố gắng tìm n9 mới).

#### b) Ví dụ minh họa

Chọn n2 = 6, theo ràng buộc:  $(n2 \% 10) + 3 = 6 + 3 = 9$

Do đó,  $n1 \% 10 = n6 \% 10 = 9$ . Chọn n1 = 9 và n6 = 9

Chọn ngẫu nhiên n3 = n4 = n5 = n7 = n8 = 51

Tính toán các giá trị:

$$n5 \% 10 = 51 \% 10 = 1$$

$$n1 \% 10 = 9$$

$$n6 \% 10 = 9 \quad (n2 \% 10) + 3 = 9$$

$$\text{int}(n3 / 10) \% 10 = \text{int}(51 / 10) \% 10 = 5 \% 10 = 5$$

$$\text{int}(n9 / 10) \% 10 = \text{int}(140 / 10) \% 10 = 14 \% 10 = 4$$

$$n7 \% 10 = 51 \% 10 = 1$$

#### Vòng check đầu tiên

$$= 10000 * (n5 \% 10) + 1000 * (n1 \% 10) + 100 * (\text{int}(n3 / 10) \% 10 + 5) + 10 * (\text{int}(n9 / 10) \% 10) + (n7 \% 10 + 2)$$

$$= 10000 * 1 + 1000 * 9 + 100 * (5 + 5) + 10 * 4 + (1 + 2) = 10000 + 9000 + 1000 + 40 + 3$$

$$= 20043$$



### Vòng check thứ 2

$$\begin{aligned} &= 10000 * (n5 \% 10) + 1000 * (n6 \% 10) + 100 * (\text{int}(n3 / 10) \% 10 + 5) + 10 * (\text{int}(n9 / 10) \% 10) + (n7 \% 10 + 2) \\ &= 10000 * 1 + 1000 * 9 + 100 * (5 + 5) + 10 * 4 + (1 + 2) \\ &= 10000 + 9000 + 1000 + 40 + 3 \\ &= 20043 \end{aligned}$$

### Vòng check thứ 3

$$\begin{aligned} &= 10000 * (n5 \% 10) + 1000 * ((n2 \% 10) + 3) + 100 * (\text{int}(n3 / 10) \% 10 + 5) + 10 * (\text{int}(n9 / 10) \% 10) + (n7 \% 10 + 2) \\ &= 10000 * 1 + 1000 * 9 + 100 * (5 + 5) + 10 * 4 + (1 + 2) = 10000 + 9000 + 1000 + 40 + 3 \\ &= 20043 \end{aligned}$$

### Vòng check thứ 4

$$\begin{aligned} &= (n2 * n6 * n7 + n3 * n4 * n8 + n1 * n5 * n9) - (n2 * n4 * n9 + n1 * n6 * n8 + n3 * n5 * n7) \\ &= (6 * 9 * 51 + 51 * 51 * 51 + 9 * 51 * n9) - (6 * 51 * n9 + 9 * 9 * 51 + 51 * 51 * 51) \end{aligned}$$

Vì cả 3 vòng check đều cho kết quả = 20043 nên  $n9=140$

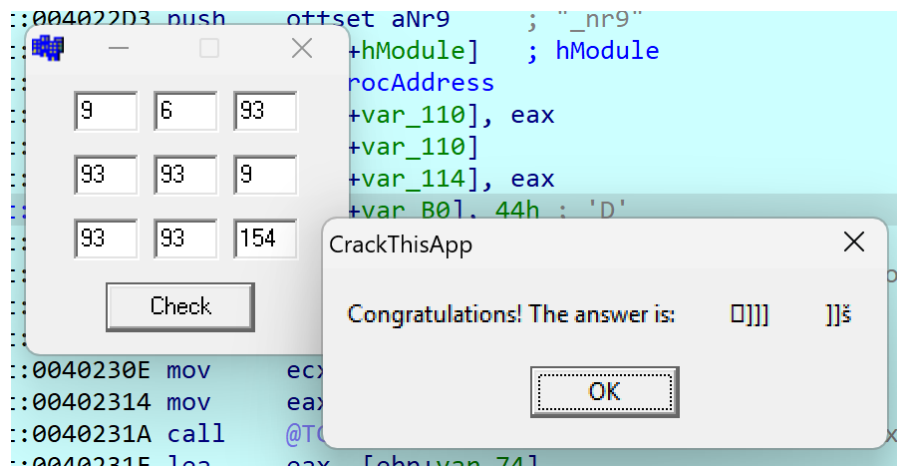
Vậy bộ số thỏa mãn là [9, 6, 51, 51, 51, 9, 51, 51, 140]

#### 1.1.4. Chương trình keygen

Từ thuật toán xác định được ở trên, test thử key sinh ra tvới chương trình.

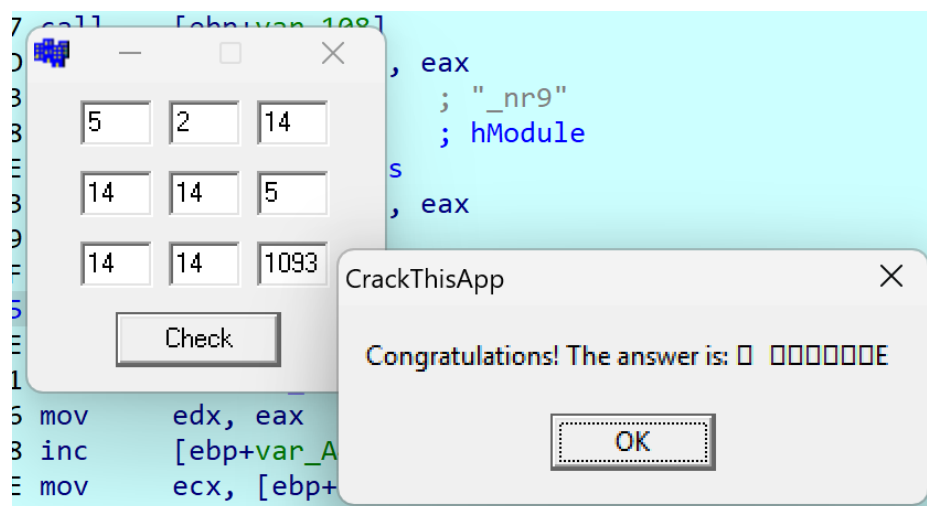
#### Lần test 1:

```
PS C:\Users\Admin\source\repos\output> & .\'keygen.exe'  
Keygen successful!  
Number1: 9  
Number2: 6  
Number3: 93  
Number4: 93  
Number5: 93  
Number6: 9  
Number7: 93  
Number8: 93  
Number9: 154  
Check Value: 40455  
PS C:\Users\Admin\source\repos\output> |
```



### Lần test 2:

```
PS C:\Users\Admin\source\repos\output> & .\'keygen.exe'
Keygen successful!
Number1: 5
Number2: 2
Number3: 14
Number4: 14
Number5: 14
Number6: 5
Number7: 14
Number8: 14
Number9: 1093
Check Value: 45696
PS C:\Users\Admin\source\repos\output>
```



## 1.2. File 2.exe

File 2.exe là một chương trình dạng Crackme yêu cầu người dùng nhập một chuỗi vào ô nhập name. Chuỗi đó sẽ được xử lý qua một thuật toán mã hóa đặc biệt. Nếu nhập đúng ở serial key, sẽ hiển thị thông báo đúng.

### 1.2.1. Phân tích đoạn phát sinh key

#### \* Username

00B110CF	. 8B3D CC20B10	MOV EDI,DWORD PTR [<&USER32.GetDlgItemTextA	USER32.GetDlgItemTextA
00B110D5	. 68 FF000000	PUSH 0FF	Count = FF (255.)
00B110DA	. 68 0834B100	PUSH WinCrack.00B13408	Buffer = WinCrack.00B13408
00B110DF	. 68 EB030000	PUSH 3EB	ControlID = 3EB (1003.)
00B110E4	. 56	PUSH ESI	hWnd
00B110E5	. FFD7	CALL EDI	GetDlgItemTextA

Đầu tiên, gọi hàm GetDlgItemTextA để lấy username từ ô nhập liệu có ID 1003 (0x3EB).

00B110EC	. 85C0	TEST EAX,EAX	
00B110EE	. 75 1C	JNZ SHORT WinCrack.00B1110C	
00B110F0	. 50	PUSH EAX	
00B110F1	. 68 1421B100	PUSH WinCrack.00B12114	
00B110F6	. 68 1C21B100	PUSH WinCrack.00B1211C	
00B110FB	. 56	PUSH ESI	
00B110FC	. FF15 BC20B10	CALL DWORD PTR [<&USER32.MessageBoxA>]	MessageBoxA
00B11102	. 5F	POP EDI	
00B11103	. 5E	POP ESI	
00B11104	. B8 01000000	MOV EAX,1	
00B11109	. C2 1000	RET 10	
00B1110C	. E8 0F020000	CALL WinCrack.00B11320	
00B11111	. 84C0	TEST AL,AL	
00B11113	. 75 1D	JNZ SHORT WinCrack.00B11132	
00B11115	. 6A 00	PUSH 0	
00B11117	. 68 1421B100	PUSH WinCrack.00B12114	
00B1111C	. 68 4021B100	PUSH WinCrack.00B12140	
00B11121	. 56	PUSH ESI	
00B11122	. FF15 BC20B10	CALL DWORD PTR [<&USER32.MessageBoxA>]	MessageBoxA
00B11128	. 5F	POP EDI	
00B11129	. 5E	POP ESI	
00B1112A	. B8 01000000	MOV EAX,1	
00B1112F	. C2 1000	RET 10	
00B11132	. E8 49020000	CALL WinCrack.00B11380	
00B11137	. 83F8 24	CMP EAX,24	
00B1113A	. A3 3836B100	MOV DWORD PTR [B13638],EAX	
00B1113F	. 7D 1D	JGE SHORT WinCrack.00B1115E	
00B11141	. 6A 00	PUSH 0	
00B11143	. 68 1421B100	PUSH WinCrack.00B12114	
00B11148	. 68 9021B100	PUSH WinCrack.00B12190	
00B1114D	. 56	PUSH ESI	
00B1114E	. FF15 BC20B10	CALL DWORD PTR [<&USER32.MessageBoxA>]	MessageBoxA

Phần này kiểm tra các điều kiện của username:

- Kiểm tra độ dài username (0x00B110E – 0x00B1115): kiểm tra username có rỗng không. Nếu rỗng (EAX = 0), hiển thị thông báo lỗi.
- Kiểm tra tính hợp lệ của username (0x00B110C – 0x00B111F): gọi hàm tại địa chỉ 0x00B11320 để kiểm tra tính hợp lệ của username. Hàm này trả về AL = 1 nếu username hợp lệ, ngược lại AL = 0. Nếu username ít hơn 5 ký tự thì AL = 0 và hiển thị thông báo lỗi username không đủ số ký tự hợp lệ tối thiểu.
- Kiểm tra username có đủ giá trị hợp lệ không:

```

00B11380  0FB05 2035B  MOVSX EAX,BYTE PTR [B13520]
00B11387  . 83F8 41      CMP EAX,41
00B1138A  .v 7C 05      JL SHORT WinCrack.00B11391
00B1138C  . 83F8 5A      CMP EAX,5A
00B1138F  .v 7E 02      JLE SHORT WinCrack.00B11393
00B11391  > 33C0      XOR EAX,EAX
00B11393  > 8D48 BF      LEA ECX,DWORD PTR [EAX-41]
00B11396  . 0FB05 2135B  MOVSX EAX,BYTE PTR [B13521]
00B1139D  . 6BC9 1A      IMUL ECX,ECX,1A
00B113A0  . 83F8 41      CMP EAX,41
00B113A3  .v 7C 05      JL SHORT WinCrack.00B113AA
00B113A5  . 83F8 5A      CMP EAX,5A
00B113A8  .v 7E 02      JLE SHORT WinCrack.00B113AC
00B113AA  > 33C0      XOR EAX,EAX
00B113AC  > 8D4C08 BF    LEA ECX,DWORD PTR [EAX+ECX-41]
00B113B0  . 0FB05 2235B  MOVSX EAX,BYTE PTR [B13522]
00B113B7  . 6BC9 1A      IMUL ECX,ECX,1A
00B113BA  . 83F8 41      CMP EAX,41
00B113BD  .v 7C 05      JL SHORT WinCrack.00B113C4
00B113BF  . 83F8 5A      CMP EAX,5A
00B113C2  .v 7E 02      JLE SHORT WinCrack.00B113C6
00B113C4  > 33C0      XOR EAX,EAX
00B113C6  > 8D4C08 BF    LEA ECX,DWORD PTR [EAX+ECX-41]
00B113CA  . 0FB05 2335B  MOVSX EAX,BYTE PTR [B13523]
00B113D1  . 6BC9 1A      IMUL ECX,ECX,1A
00B113D4  . 83F8 41      CMP EAX,41
00B113D7  .v 7C 05      JL SHORT WinCrack.00B113DE
00B113D9  . 83F8 5A      CMP EAX,5A
00B113DC  .v 7E 02      JLE SHORT WinCrack.00B113E0
00B113DE  > 33C0      XOR EAX,EAX
00B113E0  > 8D4408 BF    LEA EAX,DWORD PTR [EAX+ECX-41]
00B113E4  . 0FB0D 2435B  MOVSX ECX,BYTE PTR [B13524]
00B113EB  . 6BC0 1A      IMUL EAX,EAX,1A
00B113EE  . 83F9 41      CMP ECX,41
00B113F1  .v 7C 05      JL SHORT WinCrack.00B113F8
00B113F3  . 83F9 5A      CMP ECX,5A
00B113F6  .v 7E 02      JLE SHORT WinCrack.00B113FA
00B113F8  > 33C9      XOR ECX,ECX
00B113FA  > 8D4408 BF    LEA EAX,DWORD PTR [EAX+ECX-41]
00B113FE  . C3          RET

```

+ Kiểm tra phần đầu tiên (0x00B1380 – 0x00B139D): Xử lý ký tự đầu tiên của username đã được xử lý (từ vùng nhớ B13520).

- Nếu ký tự nằm trong khoảng 'A'-'Z', ECX = mã ASCII - 'A' (để A=0, B=1, ...)
- Nếu không phải chữ cái in hoa, EAX = 0 và ECX = -65

+ Xử lý ký tự thứ hai:

- Đầu tiên nhân ECX (giá trị từ ký tự đầu) với 26 (IMUL ECX,ECX,1A)
- Sau đó kiểm tra ký tự thứ hai có phải chữ cái in hoa không
- Cộng thêm (ký tự thứ hai - 'A') vào ECX: ECX = ECX + (EAX - 'A')

=> Đây chính là phép tính:  $ECX = ECX * 26 + (ký\_tự\_2 - 'A')$

+ Xử lý các kí tự còn lại, với mỗi kí tự:

- Nhân EAX với 26
- Cộng thêm (ký tự tại vị trí đó - 'A') vào EAX
- Trả về giá trị cuối cùng trong EAX

=> Đoạn code này thực hiện việc tính toán một giá trị (checksum) từ 5 ký tự của username đã được xử lý. Công thức tính toán là:  $checksum = (((char1 - 'A') * 26 + (char2 - 'A')) * 26 + (char3 - 'A')) * 26 + (char4 - 'A')) * 26 + (char5 - 'A')$

Kết quả trả về lưu trong thanh ghi EAX, nếu giá trị  $\geq 36$ , username được chấp nhận; nếu không sẽ hiển thị hộp thoại “Error” với nội dung "THE ENTERED NAME IS NOT VALID: This name is a joke..." khi giá trị tính từ username  $< 36$ .

=> Kết quả checksum hợp lệ này sẽ được lưu vào ô có địa chỉ [00B13638], đây là giá trị quan trọng, khởi đầu để tìm ra serial key cho từng username khác nhau.

### \* Serial Key

<pre> 00B11163 . 68 0835B100 PUSH WinCrack.00B13508 00B11168 . 68 EC030000 PUSH 3EC 00B1116D . 56          PUSH ESI 00B1116E . FFD7       CALL EDI 00B11170 . A3 0034B100 MOV DWORD PTR [B13400],EAX 00B11175 . 83F8 17    CMP EAX,17 00B11178 ~ 74 1D     JE SHORT WinCrack.00B11197 00B1117A . 6A 00     PUSH 0 00B1117C . 68 1421B100 PUSH WinCrack.00B12114 00B11181 . 68 C821B100 PUSH WinCrack.00B121C8 00B11186 . 56          PUSH ESI 00B11187 . FF15 BC20B10 CALL DWORD PTR [&lt;&amp;USER32.MessageBoxA&gt;] 00B1118D . 5F          POP EDI 00B1118E . 5E          POP ESI 00B1118F . B8 01000000 MOV EAX,1 00B11194 . C2 1000    RET 10 00B11197 &gt; E8 64000000 CALL WinCrack.00B11200 00B1119C . 6A 00     PUSH 0 00B1119E . 3B05 3836B10 CMP EAX,DWORD PTR [B13638] 00B111A4 ~ 74 1B     JE SHORT WinCrack.00B111C1 00B111A6 . 68 1421B100 PUSH WinCrack.00B12114 00B111AB . 68 F421B100 PUSH WinCrack.00B121F4 00B111B0 . 56          PUSH ESI 00B111B1 . FF15 BC20B10 CALL DWORD PTR [&lt;&amp;USER32.MessageBoxA&gt;] 00B111B7 . 5F          POP EDI 00B111B8 . 5E          POP ESI 00B111B9 . B8 01000000 MOV EAX,1 00B111BE . C2 1000    RET 10 00B111C1 &gt; 68 2822B100 PUSH WinCrack.00B12228 00B111C6 . 68 3422B100 PUSH WinCrack.00B12234 00B111CB . 56          PUSH ESI 00B111CC . FF15 BC20B10 CALL DWORD PTR [&lt;&amp;USER32.MessageBoxA&gt;] </pre>	<pre> Style = MB_OK!MB_APPLMODAL Title = "Error" Text = "THE SERIAL IS NOT VALID: Invalid lenght..." hOwner MessageBoxA  Style = MB_OK!MB_APPLMODAL Title = "Error" Text = "THE SERIAL IS NOT VALID: Check Name and Serial ..." hOwner MessageBoxA  Title = "Good boy !" Text = "SERIAL IS OK: Write a KEYGEN now ..." hOwner MessageBoxA </pre>
--	--

- Phần 1: Kiểm tra độ dài Serial Key:

**PUSH WinCrack.00B13508:** Đẩy địa chỉ chứa serial key lên stack

**PUSH 0x3EC:** ID của ô nhập serial key (1004)

**PUSH ESI:** Handle của cửa sổ

**CALL EDI:** Gọi GetDlgItemTextA để lấy serial key

**MOV [00813400],EAX:** Lưu độ dài serial key vào biến

**CMP EAX,17:** So sánh độ dài với 23 (0x17)

**JE SHORT 0x00B11197:** Nếu độ dài = 23, nhảy tới kiểm tra tiếp theo.

Đoạn này lấy serial key từ ô nhập liệu, sau đó kiểm tra độ dài. Serial key phải có **đúng 23** ký tự (bao gồm cả dấu gạch ngang). Nếu không đúng 23 ký tự, hiển thị thông báo lỗi “THE SERIAL ID NOT VALID: Invalid length...”

- Phần 2: Kiểm tra giá trị của Serial Key (0x00B11197 – 0x00B111C1):

+ Gọi hàm tại địa chỉ 0x00811200 để tính toán giá trị từ serial key, hàm này thực hiện việc:

1. Chia serial key thành 4 phần (dựa trên dấu gạch ngang)
2. Tính toán giá trị từ mỗi phần dựa trên vị trí của ký tự trong bảng charset tương ứng
3. Kiểm tra xem 4 giá trị có bằng nhau không
4. Trả về giá trị đó trong thanh ghi EAX

Sau đó, so sánh giá trị này với giá trị đã tính từ username (lưu tại địa chỉ 0x00B13638).

**Khởi tạo:**

00B11200	\$	51	PUSH ECX
00B11201	.	53	PUSH EBX
00B11202	.	8A1D 1830B10	MOV BL,BYTE PTR [B13018]
00B11208	.	55	PUSH EBP
00B11209	.	56	PUSH ESI
00B1120A	.	33C0	XOR EAX,EAX
00B1120C	.	57	PUSH EDI
00B1120D	.	33FF	XOR EDI,EDI
00B1120F	.	894424 10	MOV DWORD PTR [ESP+10],EAX
00B11213	.	33ED	XOR EBP,EBP
00B11215	.	BE 04000000	MOV ESI,4
00B1121A	.	8D9B 00000000	LEA EBX,DWORD PTR [EBX]

**PUSH ECX:** Lưu thanh ghi ECX vào stack

**PUSH EBX:** Lưu thanh ghi EBX vào stack

**MOV BL,BYTE PTR [B13018]:** Lấy byte đầu tiên từ charset1 vào BL

**PUSH EBP:** Lưu thanh ghi EBP vào stack

**PUSH ESI:** Lưu thanh ghi ESI vào stack

**XOR EAX,EAX:** Khởi tạo EAX = 0 (cho phần 4 của serial)



**PUSH EDI:** Lưu thanh ghi EDI vào stack

**XOR EDI,EDI:** Khởi tạo EDI = 0 (cho phần 2 của serial)

**MOV DWORD PTR [ESP+10],EAX :** [ESP+10] = 0 (cho phần 1 của serial)

**XOR EBP,EBP:** Khởi tạo EBP = 0 (cho phần 3 của serial)

**MOV ESI,4:** ESI = 4 (index của ký tự cuối của phần 1)

**LEA EBX,DWORD PTR [EBX]:** NOP (không thao tác)

Tính giá trị từ phần 1 của serial (0x00B11220 – 0x00B1124F)

00B11220	>	8A96 0835B10	MOV DL,BYTE PTR [ESI+B13508]
00B11226	.	33C9	XOR ECX,ECX
00B11228	.	3ADA	CMP BL,DL
00B1122A	✓	74 12	JE SHORT WinCrack.00B1123E
00B1122C	.	8D6424 00	LEA ESP,DWORD PTR [ESP]
00B11230	>	83F9 24	CMP ECX,24
00B11233	✓	7D 09	JGE SHORT WinCrack.00B1123E
00B11235	.	41	INC ECX
00B11236	.	3891 1830B10	CMP BYTE PTR [ECX+B13018],DL
00B1123C	^	75 F2	JNZ SHORT WinCrack.00B11230
00B1123E	>	83EE 01	SUB ESI,1
00B11241	.	8B5424 10	MOV EDX,DWORD PTR [ESP+10]
00B11245	.	8D14D2	LEA EDX,DWORD PTR [EDX+EDX*8]
00B11248	.	8D0C91	LEA ECX,DWORD PTR [ECX+EDX*4]
00B1124B	.	894C24 10	MOV DWORD PTR [ESP+10],ECX
00B1124F	^	79 CF	JNS SHORT WinCrack.00B11220

**MOV DL,BYTE PTR [ESI+B13508]:** Lấy ký tự tại vị trí ESI+B13508 (từ serial)

**XOR ECX,ECX:** Khởi tạo ECX = 0 (vị trí trong charset)

**CMP BL,DL:** So sánh ký tự đầu tiên của charset1 với ký tự của serial

**JE SHORT 0x00B1123E:** Nếu bằng nhau, nhảy đến 0x00B1123E

; Vòng lặp tìm vị trí của ký tự trong charset1

**LEA ESP,DWORD PTR [ESP]:** NOP (không thao tác)

**CMP ECX,24:** So sánh ECX với 0x24 (36 decimal)

**JGE SHORT 0x00B1123E:** Nếu ECX >= 36, nhảy đến 0x00B1123E

**INC ECX:** ECX++ (tăng vị trí tìm kiếm)

**CMP BYTE PTR [ECX+B13018],DL:** So sánh ký tự tại vị trí ECX trong charset1 với ký tự của serial

**JNZ SHORT 0x00B11230:** Nếu không bằng nhau, lặp lại

; Tính giá trị cho phần 1

**SUB ESI,1:** ESI-- (chuyển đến ký tự trước đó)

**MOV EDX,DWORD PTR [ESP+10]:** Lấy giá trị tích lũy hiện tại

**LEA EDX,DWORD PTR [EDX+EDX\*8]:**  $EDX = EDX * 9$

**LEA ECX,DWORD PTR [ECX+EDX\*4]:**  $ECX = ECX + EDX * 4 = ECX + EDX * 9 * 4$   
 $= ECX + EDX * 36$

**MOV DWORD PTR [ESP+10]:** Cập nhật giá trị tích lũy

**JNS SHORT 0x00B11220:** Nếu  $ESI \geq 0$ , lặp lại (còn ký tự để xử lý)

=> Đây là đoạn code xử lý phần đầu tiên của serial key (5 ký tự đầu, trước dấu gạch ngang đầu tiên):

1. Bắt đầu từ ký tự thứ 5 ( $ESI = 4$ ) và đi ngược lên ký tự đầu tiên ( $ESI$  giảm dần đến 0)
2. Với mỗi ký tự, tìm vị trí của nó trong bảng charset1
3. Tính giá trị tích lũy theo công thức:  $value = value * 36 + position$
4. Lưu giá trị cuối cùng vào  $[ESP+10]$

Tính giá trị từ phần 2 của serial ( $0x00B11251 - 0x00B11288$ )

00B11251	. 8A1D 3C30B10	MOV BL,BYTE PTR [B1303C]
00B11257	. BE 0A000000	MOV ESI,0A
00B1125C	. 8D6424 00	LEA ESP,DWORD PTR [ESP]
00B11260	> 8A96 0835B10	MOV DL,BYTE PTR [ESI+B13508]
00B11266	. 33C9	XOR ECX,ECX
00B11268	. 3ADA	CMP BL,DL
00B1126A	.. 74 12	JE SHORT WinCrack.00B1127E
00B1126C	. 8D6424 00	LEA ESP,DWORD PTR [ESP]
00B11270	> 83F9 24	CMP ECX,24
00B11273	.. 7D 09	JGE SHORT WinCrack.00B1127E
00B11275	. 41	INC ECX
00B11276	. 3891 3C30B10	CMP BYTE PTR [ECX+B1303C],DL
00B1127C	.. 75 F2	JNZ SHORT WinCrack.00B11270
00B1127E	> 4E	DEC ESI
00B1127F	. 83FE 06	CMP ESI,6
00B11282	. 8D14FF	LEA EDX,DWORD PTR [EDI+EDI*8]
00B11285	. 8D3C91	LEA EDI,DWORD PTR [ECX+EDX*4]
00B11288	.. 7D D6	JGE SHORT WinCrack.00B11260

**MOV BL,BYTE PTR [B1303C]:** Lấy byte đầu tiên từ charset2 vào BL

**MOV ESI,0A:**  $ESI = 10$  ( $0x0A$ ) (index của ký tự cuối của phần 2)

**LEA ESP,DWORD PTR [ESP]:** NOP (không thao tác)



; Tương tự như xử lý phần 1, nhưng sử dụng charset2 và vị trí từ 6-10

**MOV DL,BYTE PTR [ESI+B13508]:** Lấy ký tự tại vị trí ESI+B13508

**XOR ECX,ECX:** Khởi tạo ECX = 0

**CMP BL,DL:** So sánh với ký tự đầu tiên của charset2

**JE SHORT 0x00B1127E:** Nếu bằng nhau, nhảy đến 0x00B1127E

; Vòng lặp tìm vị trí trong charset2

**LEA ESP,DWORD PTR [ESP]:** NOP

**CMP ECX,24:** So sánh ECX với 36

**JGE SHORT 0x00B1127E:** Nếu ECX >= 36, nhảy đến 0x00B1127E

**INC ECX:** ECX++

**CMP BYTE PTR [ECX+B1303C],DL:** So sánh ký tự tại vị trí ECX trong charset2

**JNZ SHORT 0x00B11270:** Nếu không bằng, lặp lại

; Tính giá trị cho phần 2

**DEC ESI:** ESI-- (chuyển đến ký tự trước đó)

**CMP ESI,6:** So sánh ESI với 6 (bắt đầu của phần 2)

**LEA EDX,DWORD PTR [EDI+EDI\*8]:** EDX = EDI\*9

**LEA EDI,DWORD PTR [ECX+EDX\*4]:** EDI = ECX + EDX\*4 = ECX + EDI\*36

**JGE SHORT 0x00B11260:** Nếu ESI >= 6, lặp lại

=> Đoạn này xử lý phần thứ hai của serial key (5 ký tự sau dấu gạch ngang đầu tiên):

1. Bắt đầu từ ký tự thứ 11 (ESI = 10) và đi ngược lên ký tự thứ 7 (ESI = 6)
2. Với mỗi ký tự, tìm vị trí của nó trong bảng charset2
3. Tính giá trị tích lũy theo công thức: value = value \* 36 + position
4. Lưu giá trị cuối cùng vào EDI

Tính giá trị từ phần 3 của serial (0x00B1128A - 0x00B112BA)

```

00B1128A . 8A1D 6030B10 MOV BL,BYTE PTR [B13060]
00B11290 . BE 10000000 MOV ESI,10
00B11295 > 8A96 0835B10 MOV DL,BYTE PTR [ESI+B13508]
00B1129B . 33C9 XOR ECX,ECX
00B1129D . 3ADA CMP BL,DL
00B1129F . 74 0E JE SHORT WinCrack.00B112AF
00B112A1 > 83F9 24 CMP ECX,24
00B112A4 . 7D 09 JGE SHORT WinCrack.00B112AF
00B112A6 . 41 INC ECX
00B112A7 . 3891 6030B10 CMP BYTE PTR [ECX+B13060],DL
00B112AD . 75 F2 JNZ SHORT WinCrack.00B112A1
00B112AF > 4E DEC ESI
00B112B0 . 83FE 0C CMP ESI,0C
00B112B3 . 8D54ED 00 LEA EDX,DWORD PTR [EBP+EBP*8]
00B112B7 . 8D2C91 LEA EBP,DWORD PTR [ECX+EDX*4]
00B112BA . 7D D9 JGE SHORT WinCrack.00B11295

```

**MOV BL,BYTE PTR [B13060]:** Lấy byte đầu tiên từ charset3 vào BL

**MOV ESI,10:** ESI = 16 (0x10) (index của ký tự cuối của phần 3)

; Xử lý tương tự, nhưng sử dụng charset3 và vị trí từ 12-16

**MOV DL,BYTE PTR [ESI+B13508]:** Lấy ký tự tại vị trí ESI+B13508

**XOR ECX,ECX:** ECX = 0

**CMP BL,DL:** So sánh với ký tự đầu tiên của charset3

**JE SHORT 0x00B112AF:** Nếu bằng nhau, nhảy đến 0x00B112AF

; Vòng lặp tìm vị trí trong charset3

**CMP ECX,24:** So sánh ECX với 36

**JGE SHORT 0x00B112AF:** Nếu ECX >= 36, nhảy đến 0x00B112AF

**INC ECX:** ECX++

**CMP BYTE PTR [ECX+B13060],DL:** So sánh ký tự tại vị trí ECX trong charset3

**JNZ SHORT 0x00B112A1:** Nếu không bằng, lặp lại

; Tính giá trị cho phần 3

**DEC ESI:** ESI--

**CMP ESI,0C:** So sánh ESI với 12 (bắt đầu của phần 3)

**LEA EDX,DWORD PTR [EBP+EBP\*8]:** EDX = EBP\*9

**LEA EBP,DWORD PTR [ECX+EDX\*4]:** EBP = ECX + EDX\*4 = ECX + EBP\*36

**JGE SHORT 0x00B11295:** Nếu ESI >= 12, lặp lại

=> Đoạn này xử lý phần thứ ba của serial key (5 ký tự sau dấu gạch ngang thứ hai):

1. Bắt đầu từ ký tự thứ 17 (ESI = 16) và đi ngược lên ký tự thứ 13 (ESI = 12)
2. Với mỗi ký tự, tìm vị trí của nó trong bảng charset3
3. Tính giá trị tích lũy theo công thức:  $value = value * 36 + position$
4. Lưu giá trị cuối cùng vào EBP

Tính giá trị từ phần 4 của serial (0x00B112BC - 0x00B112F8)

00B112BC	. 8A1D 8430B10	MOV BL, BYTE PTR [B13084]
00B112C2	. BE 16000000	MOV ESI, 16
00B112C7	.v EB 07	JMP SHORT WinCrack.00B112D0
00B112C9	. 8DA424 00000	LEA ESP, DWORD PTR [ESP]
00B112D0	> 8A96 0835B10	MOV DL, BYTE PTR [ESI+B13508]
00B112D6	. 33C9	XOR ECX, ECX
00B112D8	. 3ADA	CMP BL, DL
00B112DA	.v 74 12	JE SHORT WinCrack.00B112EE
00B112DC	. 8D6424 00	LEA ESP, DWORD PTR [ESP]
00B112E0	> 83F9 24	CMP ECX, 24
00B112E3	.v 7D 09	JGE SHORT WinCrack.00B112EE
00B112E5	. 41	INC ECX
00B112E6	. 3891 8430B10	CMP BYTE PTR [ECX+B13084], DL
00B112EC	.^ 75 F2	JNZ SHORT WinCrack.00B112E0
00B112EE	> 4E	DEC ESI
00B112EF	. 83FE 12	CMP ESI, 12
00B112F2	. 8D04C0	LEA EAX, DWORD PTR [EAX+EAX*8]
00B112F5	. 8D0481	LEA EAX, DWORD PTR [ECX+EAX*4]
00B112F8	.^ 7D D6	JGE SHORT WinCrack.00B112D0

**MOV BL, BYTE PTR [B13084]:** Lấy byte đầu tiên từ charset4 vào BL

**MOV ESI, 16:** ESI = 22 (0x16) (index của ký tự cuối của phần 4)

**JMP SHORT 0x00B112D0:** Nhảy đến 0x00B112D0

**LEA ESP, DWORD PTR [ESP]:** NOP

; Xử lý tương tự, nhưng sử dụng charset4 và vị trí từ 18-22

**MOV DL, BYTE PTR [ESI+B13508]:** Lấy ký tự tại vị trí ESI+B13508

**XOR ECX, ECX:** ECX = 0

**CMP BL, DL:** So sánh với ký tự đầu tiên của charset4

**JE SHORT 0x00B112EE:** Nếu bằng nhau, nhảy đến 0x00B112EE

; Vòng lặp tìm vị trí trong charset4

**LEA ESP, DWORD PTR [ESP]:** NOP

**CMP ECX, 24:** So sánh ECX với 36

**JGE SHORT 0x00B112EE:** Nếu ECX >= 36, nhảy đến 0x00B112EE

**INC ECX:** ECX++

**CMP BYTE PTR [ECX+B13084],DL:** So sánh ký tự tại vị trí ECX trong charset4

**JNZ SHORT 0x00B112E0:** Nếu không bằng, lặp lại

; Tính giá trị cho phần 4

**DEC ESI:** ESI--

**CMP ESI,12:** So sánh ESI với 18 (bắt đầu của phần 4)

**LEA EAX,DWORD PTR [EAX+EAX\*8]:** EAX = EAX\*9

**LEA EAX,DWORD PTR [ECX+EAX\*4]:** EAX = ECX + EAX\*4 = ECX + EAX\*36

**JGE SHORT 0x00B112D0:** Nếu ESI >= 18, lặp lại

=> Đoạn này xử lý phần thứ tư của serial key (5 ký tự cuối):

1. Bắt đầu từ ký tự thứ 23 (ESI = 22) và đi ngược lên ký tự thứ 19 (ESI = 18)
2. Với mỗi ký tự, tìm vị trí của nó trong bảng charset4
3. Tính giá trị tích lũy theo công thức: value = value \* 36 + position
4. Lưu giá trị cuối cùng vào EAX

Kiểm tra và trả về kết quả (0x00B1119E - 0x00B11319)

00B112FA	. 8B4C24 10	MOV ECX,DWORD PTR [ESP+10]
00B112FE	. 3BCF	CMP ECX,EDI
00B11300	.v 75 10	JNZ SHORT WinCrack.00B11312
00B11302	. 3BCD	CMP ECX,EBP
00B11304	.v 75 0C	JNZ SHORT WinCrack.00B11312
00B11306	. 3BC8	CMP ECX,EAX
00B11308	.v 75 08	JNZ SHORT WinCrack.00B11312
00B1130A	. 5F	POP EDI
00B1130B	. 5E	POP ESI
00B1130C	. 5D	POP EBP
00B1130D	. 8BC1	MOV EAX,ECX
00B1130F	. 5B	POP EBX
00B11310	. 59	POP ECX
00B11311	. C3	RET
00B11312	> 5F	POP EDI
00B11313	. 5E	POP ESI
00B11314	. 5D	POP EBP
00B11315	. 33C0	XOR EAX,EAX
00B11317	. 5B	POP EBX
00B11318	. 59	POP ECX
00B11319	. C3	RET

**MOV ECX,DWORD PTR [ESP+10]:** Lấy giá trị phần 1 vào ECX

**CMP ECX,EDI:** So sánh với giá trị phần 2 (EDI)

**JNZ SHORT 0x00B11312:** Nếu khác nhau, nhảy đến 0x00B11312 (trả về 0)

**CMP ECX,EBP:** So sánh với giá trị phần 3 (EBP)

**JNZ SHORT 0x00B11312:** Nếu khác nhau, nhảy đến 0x00B11312

**CMP ECX,EAX:** So sánh với giá trị phần 4 (EAX)

**JNZ SHORT 0x00B11312:** Nếu khác nhau, nhảy đến 0x00B11312

; Trả về giá trị thành công (nếu tất cả đều bằng nhau)

**POP EDI:** Khôi phục EDI

**POP ESI:** Khôi phục ESI

**POP EBP:** Khôi phục EBP

**MOV EAX,ECX:**  $EAX = ECX$  (giá trị tính được)

**POP EBX:** Khôi phục EBX

**POP ECX:** Khôi phục ECX

**RET:** Trả về với  $EAX =$  giá trị tính được

; Trả về giá trị  $EAX = 0$  (nếu có bất kì 1 giá trị khác)

**POP EDI:** Khôi phục EDI

**POP ESI:** Khôi phục ESI

**POP EBP:** Khôi phục EBP

**XOR EAX,EAX:**  $EAX = 0$

**POP EBX:** Khôi phục EBX

**POP ECX:** Khôi phục ECX

**RET:** Trả về với  $EAX =$  giá trị tính được

=> Đoạn cuối cùng này:

1. So sánh giá trị tính từ 4 phần của serial key
2. Nếu tất cả đều bằng nhau, trả về giá trị đó trong EAX

3. Nếu có bất kỳ sự khác biệt nào, trả về 0 (tại 0x00B11312)

Sau đó so sánh, giá trị này với giá trị đã tính từ username (lưu tại địa chỉ 0x00B13638)

+ Nếu giá trị từ serial key không bằng giá trị từ username, hiển thị thông báo lỗi "THE SERIAL IS NOT VALID: Check Name and Serial ..."

+ Nếu bằng nhau hiển thị thông báo thành công (0x00B111C1 – 0x00B111CB)

### 1.2.2. Kết luận và ví dụ minh họa

#### a) Kết luận

- Tính L = Độ dài của chuỗi username (không tính ký tự kết thúc \0):

+ Nếu chuỗi username  $\leq 4$ : Không phát sinh key.

+ Nếu chuỗi username  $> 4$ :

- Tính Checksum =  $(((((\text{char1} - 'A') * 26 + (\text{char2} - 'A')) * 26 + (\text{char3} - 'A')) * 26 + (\text{char4} - 'A')) * 26 + (\text{char5} - 'A'))$ .

Trong đó:

- char1 đến char5 là 5 ký tự đầu tiên của username (chữ hoa)
- Mỗi ký tự được chuyển đổi thành số từ 0-25 (A=0, B=1, ..., Z=25)

- Tính giá trị từ một phần của serial key: Value =  $(((((\text{char5\_pos} * 36 + \text{char4\_pos}) * 36 + \text{char3\_pos}) * 36 + \text{char2\_pos}) * 36 + \text{char1\_pos})$

Trong đó:

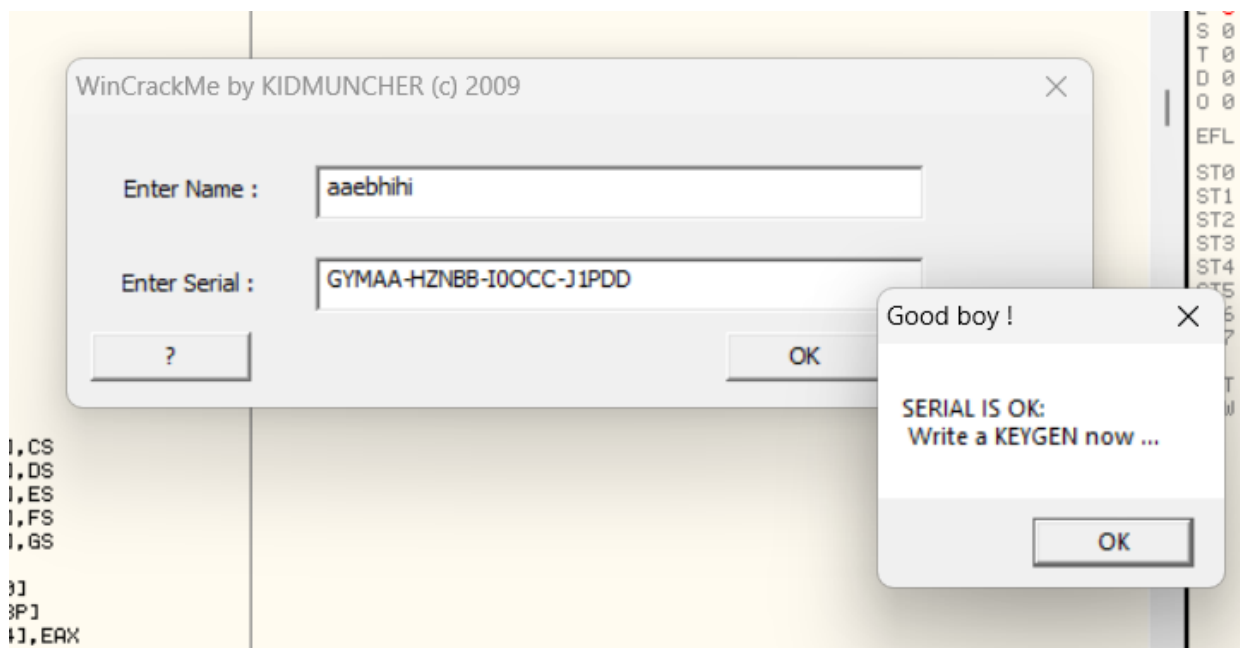
- charN\_pos là vị trí của ký tự thứ N trong bảng charset tương ứng
- Ký tự được xử lý từ phải sang trái (ký tự cuối cùng trước)

#### b) Ví dụ minh họa

##### Ví dụ 1:

Username: auebhihi

Serial key: GYMAA-HZNBB-I0OCC-J1PDD



-  $L = 8$

- Checksum:  $(((((A - A) * 26 + (A - A)) * 26 + (E - A)) * 26 + (B - A)) * 26 + (H - A) = 2737(\text{dec})$

- Value:

+ Value1 =  $(((((0 * 36 + 0) * 36 + 2) * 36 + 4) * 36 + 1) = 2737(\text{dec})$

+ Value2 =  $(((((0 * 36 + 0) * 36 + 2) * 36 + 4) * 36 + 1) = 2737(\text{dec})$

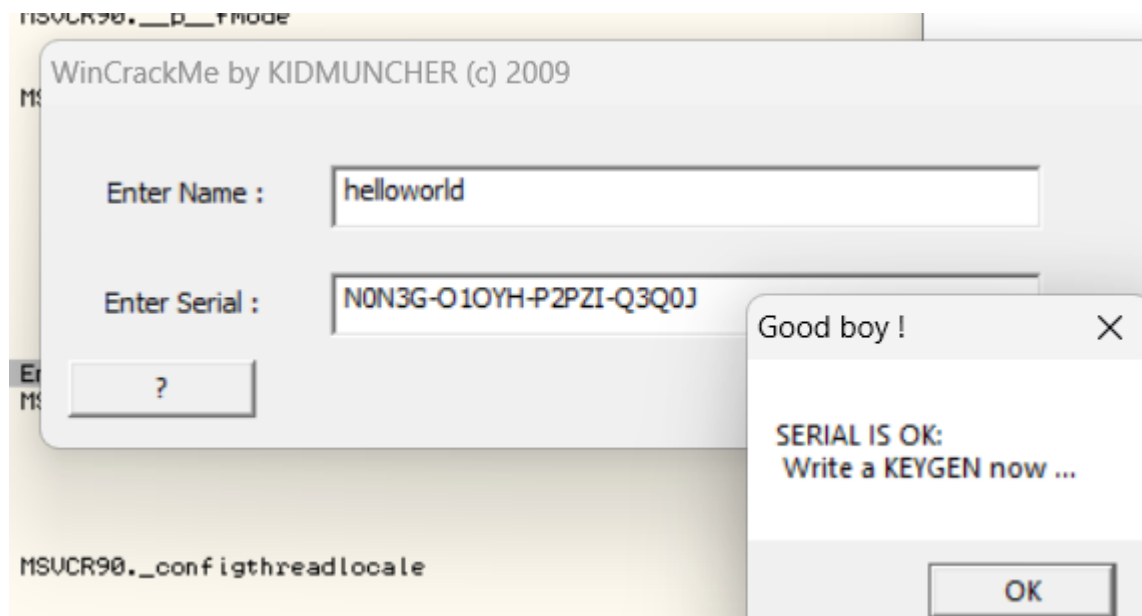
+ Value3 =  $(((((0 * 36 + 0) * 36 + 2) * 36 + 4) * 36 + 1) = 2737(\text{dec})$

+ Value4 =  $(((((0 * 36 + 0) * 36 + 2) * 36 + 4) * 36 + 1) = 2737(\text{dec})$

**Ví dụ 2:**

Username: helloworld

Serial key: N0N3G-O1OYH-P2PZI-Q3Q0J



-  $L = 10$

- Checksum:  $(((((('H' - 'A') * 26 + ('E' - 'A')) * 26 + ('L' - 'A')) * 26 + ('L' - 'A')) * 26 + ('O' - 'A') = 3276872(\text{dec})$

- Value:

+  $\text{Value1} = (((((1 * 36 + 34) * 36 + 8) * 36 + 16) * 36 + 8) = 3276872(\text{dec})$

+  $\text{Value2} = (((((1 * 36 + 34) * 36 + 8) * 36 + 16) * 36 + 8) = 3276872(\text{dec})$

+  $\text{Value3} = (((((1 * 36 + 34) * 36 + 8) * 36 + 16) * 36 + 8) = 3276872(\text{dec})$

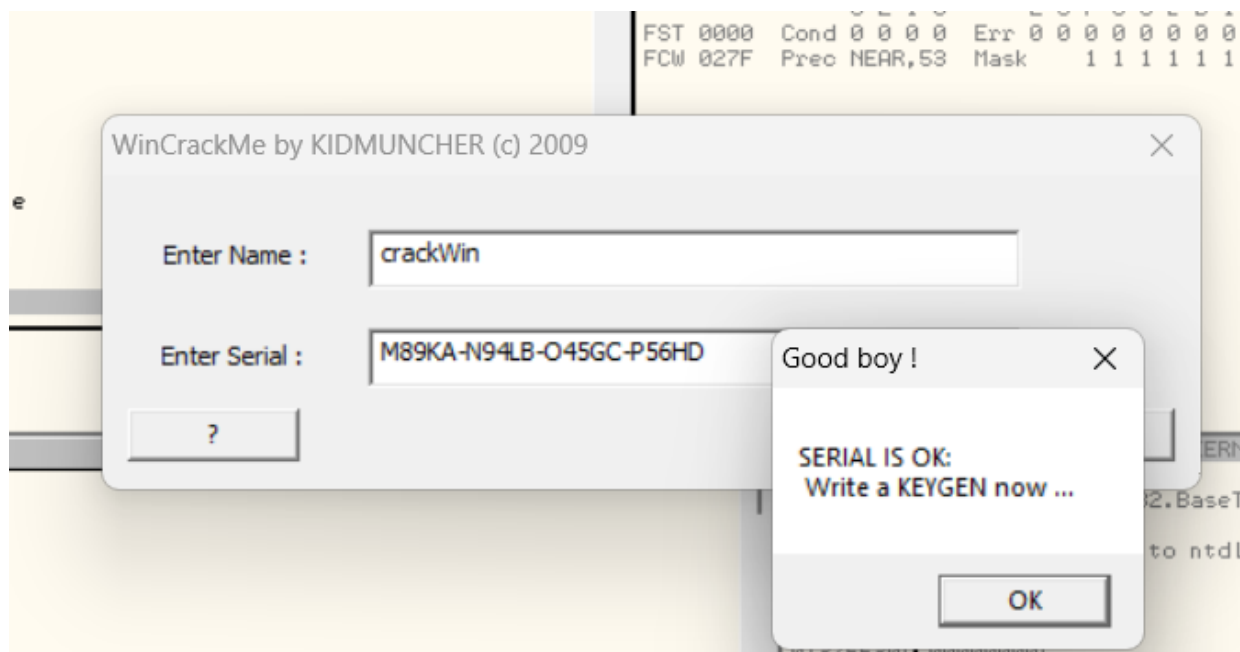
+  $\text{Value4} = (((((1 * 36 + 34) * 36 + 8) * 36 + 16) * 36 + 8) = 3276872(\text{dec})$

**Ví dụ 3:**

Username: crackWin

Serial key: M89KA-N94LB-O45GC-P56HD





-  $L = 8$

- Checksum:  $(((((('C' - 'A') * 26 + ('R' - 'A')) * 26 + ('A' - 'A')) * 26 + ('C' - 'A')) * 26 + ('K' - 'A')) = 1212806(\text{dec})$

- Value:

+ Value1 =  $(((((0 * 36 + 25) * 36 + 35) * 36 + 29) * 36 + 2) = 1212806(\text{dec})$

+ Value2 =  $(((((0 * 36 + 25) * 36 + 35) * 36 + 29) * 36 + 2) = 1212806(\text{dec})$

+ Value3 =  $(((((0 * 36 + 25) * 36 + 35) * 36 + 29) * 36 + 2) = 1212806(\text{dec})$

+ Value4 =  $(((((0 * 36 + 25) * 36 + 35) * 36 + 29) * 36 + 2) = 1212806(\text{dec})$

### 1.2.3. Chương trình keygen

Chương trình keygen được viết bằng ngôn ngữ C/C++ với các bước phát sinh thuật toán key như đã phân tích ở trên. Thuật toán:

- Tìm checksum của username.
- Dùng thuật toán lần ngược để tìm ra vị trí của từng kí tự trong mỗi phần.
- Tính Value từng phần và kiểm tra chúng có bằng nhau và bằng checksum không.
- Nếu thỏa thì in ra serial key đầy đủ, ngược lại báo rằng username không thể tạo serial key.

- Đối với chuỗi username không đủ kí tự:

```
===== Keygen từ Username =====  
Tính checksum và tạo serial key từ username  
  
Nhập username (ít nhất 5 ký tự chữ cái): hi  
Username phải có ít nhất 5 ký tự chữ cái!
```

- Đối với chuỗi username bình thường (ví dụ 2):

```
===== Keygen từ Username =====  
Tính checksum và tạo serial key từ username  
  
Nhập username (ít nhất 5 ký tự chữ cái): helloworld  
Username đã xử lý: HELLO  
Ký tự 1: H (idx=7) -> 7 (0x00000007)  
Ký tự 2: E (idx=4) -> 186 (0x000000BA)  
Ký tự 3: L (idx=11) -> 4847 (0x000012EF)  
Ký tự 4: L (idx=11) -> 126033 (0x0001EC51)  
Ký tự 5: O (idx=14) -> 3276872 (0x00320048)  
  
Checksum tính từ username: 3276872 (0x00320048)  
✓ Có thể tạo serial key từ username này!  
  
Đang tạo serial key...  
  
Xác minh các giá trị:  
Phần 1 (N0N3G): 0x00320048  
Phần 2 (010YH): 0x00320048  
Phần 3 (P2PZI): 0x00320048  
Phần 4 (Q3Q0J): 0x00320048  
Checksum: 0x00320048  
  
Serial key đã được xác minh thành công!  
  
Serial key: N0N3G-010YH-P2PZI-Q3Q0J
```

- Đối với chuỗi username không thể tạo serial key:

```
===== Keygen từ Username =====  
Tính checksum và tạo serial key từ username  
  
Nhập username (ít nhất 5 ký tự chữ cái): abcdefg  
Username đã xử lý: ABCDE  
Ký tự 1: A (idx=0) -> 0 (0x00000000)  
Ký tự 2: B (idx=1) -> 1 (0x00000001)  
Ký tự 3: C (idx=2) -> 28 (0x0000001C)  
Ký tự 4: D (idx=3) -> 731 (0x000002DB)  
Ký tự 5: E (idx=4) -> 19010 (0x00004A42)  
  
Checksum tính từ username: 19010 (0x00004A42)  
X Không thể tạo serial key từ username này!
```

### 1.3. File 3.exe

File 3.exe là một chương trình dạng Crackme yêu cầu người dùng nhập một chuỗi vào ô nhập Name. Chuỗi đó sẽ được xử lý qua một thuật toán mã hóa đặc biệt. Nếu nhập đúng ở Key, sẽ hiển thị thông báo đúng.

#### 1.3.1. Phân tích đoạn phát sinh key

00401870	>	C74424 04 66	MOV DWORD PTR [ESP+4],66	
00401878	.	8B45 08	MOV EAX,DWORD PTR [EBP+8]	
0040187B	.	890424	MOV DWORD PTR [ESP],EAX	
0040187E	.	C785 B8FDFF	MOV DWORD PTR [EBP-248],-1	
00401888	.	E8 B3ED0000	CALL <JMP.&USER32.GetDlgItem>	GetDlgItem
0040188D	.	83EC 08	SUB ESP,8	
00401890	.	890424	MOV DWORD PTR [ESP],EAX	
00401893	.	E8 B8ED0000	CALL <JMP.&USER32.GetWindowTextLengthA>	GetWindowTextLengthA
00401898	.	83EC 04	SUB ESP,4	
0040189B	.	8945 E4	MOV DWORD PTR [EBP-1C],EAX	
0040189E	.	837D E4 04	CMP DWORD PTR [EBP-1C],4	
004018A2	✓	0F8E 2C070000	JLE 3.00401FD4	
004018A8	.	C74424 04 00	MOV DWORD PTR [ESP+4],0	
004018B0	.	C70424 94204	MOV DWORD PTR [ESP],3.00442094	
004018B7	.	E8 04E50200	CALL 3.0042FDC0	
004018BC	.	8945 E0	MOV DWORD PTR [EBP-20],EAX	
004018BF	.	8B45 E4	MOV EAX,DWORD PTR [EBP-1C]	
004018C2	.	40	INC EAX	
004018C3	.	894424 0C	MOV DWORD PTR [ESP+C],EAX	
004018C7	.	8B45 E0	MOV EAX,DWORD PTR [EBP-20]	
004018CA	.	894424 08	MOV DWORD PTR [ESP+8],EAX	
004018CE	.	C74424 04 66	MOV DWORD PTR [ESP+4],66	
004018D6	.	8B45 08	MOV EAX,DWORD PTR [EBP+8]	
004018D9	.	890424	MOV DWORD PTR [ESP],EAX	
004018DC	.	E8 7FED0000	CALL <JMP.&USER32.GetDlgItemTextA>	GetDlgItemTextA

Đoạn code trên lấy Handle (định danh) của Control có ID 66 (ô nhập “Name”) bằng GetDlgItem. Sau đó gọi GetWindowTextLengthA để lấy độ dài chuỗi nhập. Nếu độ dài ≤ 4, chương trình không sinh key và thoát. Chương trình chỉ tiếp tục thực thi các lệnh tiếp theo nếu username có độ dài lớn hơn 4 ký tự. Đây là một điều kiện quan trọng.

004018A8	.	C74424 04 00	MOV DWORD PTR [ESP+4],0	
004018B0	.	C70424 94204	MOV DWORD PTR [ESP],3.00442094	
004018B7	.	E8 04E50200	CALL 3.0042FDC0	
004018BC	.	8945 E0	MOV DWORD PTR [EBP-20],EAX	
004018BF	.	8B45 E4	MOV EAX,DWORD PTR [EBP-1C]	
004018C2	.	40	INC EAX	
004018C3	.	894424 0C	MOV DWORD PTR [ESP+C],EAX	
004018C7	.	8B45 E0	MOV EAX,DWORD PTR [EBP-20]	
004018CA	.	894424 08	MOV DWORD PTR [ESP+8],EAX	
004018CE	.	C74424 04 66	MOV DWORD PTR [ESP+4],66	
004018D6	.	8B45 08	MOV EAX,DWORD PTR [EBP+8]	
004018D9	.	890424	MOV DWORD PTR [ESP],EAX	
004018DC	.	E8 7FED0000	CALL <JMP.&USER32.GetDlgItemTextA>	GetDlgItemTextA

Đoạn code trên gọi hàm 0042FDC0 để lấy buffer chứa username, sau đó gọi GetDlgItemTextA để sao chép chuỗi từ ô nhập vào buffer này.

⇒ **Kết quả:** [EBP-20] chứa chuỗi username mà người dùng nhập vào. Đây là bước khởi đầu thực sự cho việc xử lý/sinh key dựa trên nội dung chuỗi.

```

004018E1 . 83EC 10 SUB ESP,10
004018E4 . C745 DC 0000 MOV DWORD PTR [EBP-24],0
004018EB > 8B45 DC MOV EAX,DWORD PTR [EBP-24]
004018EE . 3B45 E4 CMP EAX,DWORD PTR [EBP-1C]
004018F1 . 7D 34 JGE SHORT 3.00401927
004018F3 . 8B45 DC MOV EAX,DWORD PTR [EBP-24]
004018F6 . 894424 04 MOV DWORD PTR [ESP+4],EAX
004018FA . C70424 94204 MOV DWORD PTR [ESP],3.00442094
00401901 . C785 B8FDFFF MOV DWORD PTR [EBP-248],-1
00401908 . E8 B0E40200 CALL 3.0042FDC0
00401910 . 0FB600 MOVZX EAX,BYTE PTR [EAX]
00401913 . 8B45 DB MOV BYTE PTR [EBP-25],AL
00401916 . 0FBE45 DB MOVSX EAX,BYTE PTR [EBP-25]
0040191A . 0105 2820440 ADD DWORD PTR [442028],EAX
00401920 . 8D45 DC LEA EAX,DWORD PTR [EBP-24]
00401923 . FF00 INC DWORD PTR [EAX]
00401925 . ^ EB C4 JMP SHORT 3.004018EB

```

Đoạn code trên bắt đầu một vòng lặp tính tổng mã ASCII của từng ký tự trong chuỗi username nhập vào, có xét dấu (signed char). Cụ thể:

- **SUB ESP, 10:** Cấp phát thêm 16 byte trên Stack.
- **MOV DWORD PTR [EBP-24], 0:** Đặt giá trị 0 vào vị trí [EBP-24] (hay biến đếm  $i = 0$ ). Biến này là chỉ số duyệt từng ký tự trong username.
- **MOV EAX, DWORD PTR [EBP-24]:** Đặt giá trị [EBP-24] (biến đếm vòng lặp) vào EAX.
- **CMP EAX, DWORD PTR [EBP-1C]:** So sánh giá trị EAX (biến đếm  $i$ ) với giá trị DWORD tại [EBP-1C] (length - độ dài username).
- **JGE SHORT 3.00401927:** Nếu  $i \geq \text{length}$ , thoát khỏi vòng lặp và chuyển đến vị trí 00401927.

Trong vòng lặp, ta có:

- **MOV EAX, DWORD PTR [EBP-24]:** Lấy giá trị [EBP-24] (biến đếm  $i$ ) vào EAX.
- **MOV DWORD PTR [ESP+4], EAX:** Đặt giá trị EAX (biến đếm  $i$ ) vào [ESP+4] – đây là tham số thứ hai cho hàm sắp gọi tới.
- **MOV DWORD PTR [ESP], 3.00442094:** Đặt giá trị hằng số 00442094 (con trỏ tới chuỗi username) vào [ESP] – đây là tham số đầu tiên cho hàm sắp gọi tới.
- **MOV DWORD PTR [EBP-248], -1:** Khởi tạo biến cục bộ tại [EBP-248] với giá trị -1. Đây có thể là một kích thước, cờ, hoặc chỉ số.

- **CALL 3.0042FDC0**: Gọi hàm 0042FDC0 với hai tham số: Con trỏ chuỗi (00442094) ([ESP]) và biến đếm i ([ESP+4]). Hàm này trả về địa chỉ của ký tự tại chỉ số đó trong chuỗi (&username[i]) và lưu vào EAX.
- **MOVZX EAX, BYTE PTR [EAX]**: Đọc giá trị byte (mã ASCII) tại địa chỉ EAX trỏ tới (&username[i]) và mở rộng giá trị đó thành một DWORD trong EAX bằng cách điền các bit 0 vào phía trước. EAX bây giờ chứa mã ASCII của ký tự đó.
- **MOV BYTE PTR [EBP-25], AL**: Đặt byte thấp của EAX (chứa mã ASCII) vào [EBP-25].
- **MOVSX EAX, BYTE PTR [EBP-25]**: Đọc lại mã ASCII từ [EBP-25] và mở rộng giá trị đó thành một DWORD trong EAX bằng cách sao chép bit dấu (bit 7) vào các bit phía trước (Sign-Extend). Nếu bit 7 là 1, các bit phía trước là 1 (biểu thị số âm); nếu bit 7 là 0, các bit phía trước là 0 (biểu thị số dương). Đây là bước lấy giá trị mã ASCII có dấu.
- **ADD DWORD PTR [442028], EAX**: Cộng giá trị EAX (mã ASCII có dấu) vào giá trị DWORD tại địa chỉ [442028].  
Cuối thân vòng lặp, ta có:
- **LEA EAX, DWORD PTR [EBP-24]**: Tính toán địa chỉ của [EBP-24] (biến đếm vòng lặp) và đưa vào thanh ghi EAX.
- **INC DWORD PTR [EAX]**: Tăng giá trị DWORD tại địa chỉ EAX trỏ tới hay tức là tăng biến đếm i (i++).
- **JMP SHORT 3.004018EB**: Nhảy không điều kiện về đầu vòng lặp (tại 04018EB) để kiểm tra  $i < \text{length}$  cho lần lặp tiếp theo.

⇒ **Kết quả**: Biến toàn cục [442028] chứa tổng các giá trị mã ASCII có dấu (signed 8-bit) của từng ký tự trong username (sum ASCII).

00401927	>	8B45 E4	MOV EAX,DWORD PTR [EBP-1C]
0040192A	.	0FAF45 E4	IMUL EAX,DWORD PTR [EBP-1C]
0040192E	.	0FAF45 E4	IMUL EAX,DWORD PTR [EBP-1C]
00401932	.	3105 2820440	XOR DWORD PTR [442028],EAX

Đoạn code trên kết hợp độ dài chuỗi với tổng ASCII để tạo ra giá trị hỗn hợp đặc trưng cho chuỗi đầu vào. Cụ thể:

- **MOV EAX, DWORD PTR [EBP-1C]**: Đặt giá trị DWORD tại [EBP-1C] (length – độ dài chuỗi username) vào EAX.

- **IMUL EAX, DWORD PTR [EBP-1C]:** Nhân EAX với length  $\rightarrow$  EAX = length \* length.
- **IMUL EAX, DWORD PTR [EBP-1C]:** Tiếp tục nhân EAX với length  $\rightarrow$  EAX = length \* length \* length.
- **XOR DWORD PTR [442028], EAX:** Thực hiện phép XOR giữa biến toàn cục [442028] (sum ASCII – tổng ASCII của username) với giá trị trong EAX (length<sup>3</sup>). Kết quả được lưu trở lại địa chỉ [442028]. Đây là một bước quan trọng trong việc tạo ra một giá trị phụ thuộc vào cả nội dung và độ dài của username.

⇒ **Kết quả:** Biến toàn cục [442028] chứa giá trị sum ASCII XOR length<sup>3</sup>.

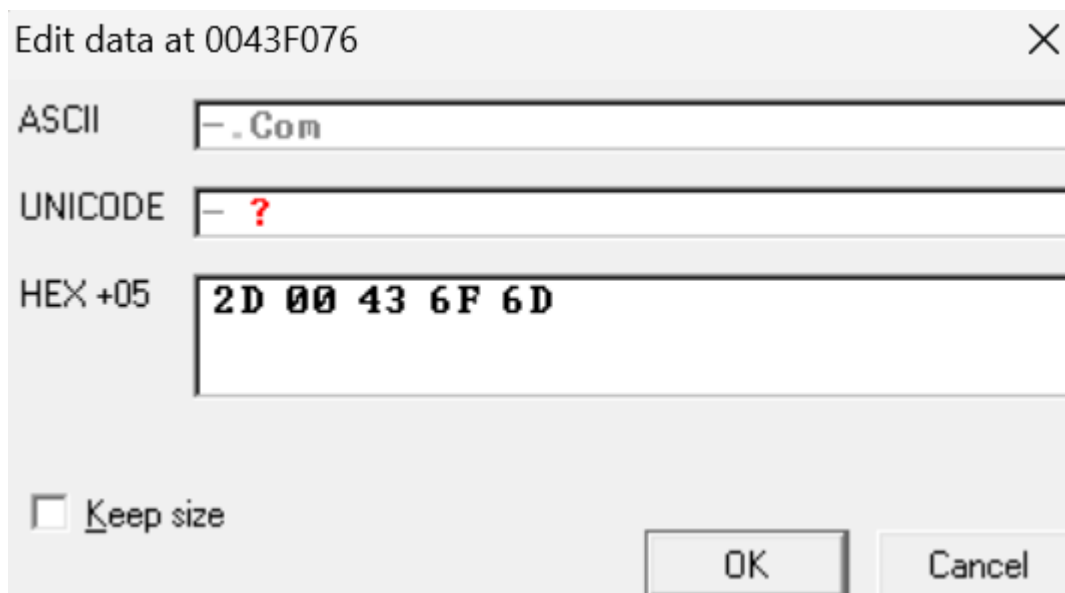
```
00401938 | . C745 DC 76F0 | MOV DWORD PTR [EBP-24], 3.0043F076
```

Đoạn code trên thực hiện bước gán dữ liệu chuỗi cho [EBP-24]. Cụ thể:

- **MOV DWORD PTR [EBP-24], 3.0043F076:** Đặt địa chỉ hằng số 0x0043F076 vào vị trí [EBP-24]. Giá trị được lưu trong 0043F076 cụ thể như sau:

```
0043F076 | 2D 00436F6D | SUB EAX, 6D6F4300
```

Ở đây, OllyDbg hiển thị lệnh như hình. Tuy nhiên, đây là sự giải đoán nhầm vì địa chỉ này thuộc vùng .rdata, tức là vùng dữ liệu chỉ đọc chứa các chuỗi hằng số, chứ không phải mã lệnh thực thi. Kiểm tra lại dữ liệu ở địa chỉ 0043F07:



So sánh kết quả bên Ida, ta thấy như sau:

```
.rdata:0043F076 asc_43F076 db '-',0 ; DATA XREF: _Z15WindowProcedureP6HWND__j1(x,x,x,x)+15A7o
```

⇒ **Kết quả:** [EBP-24] trỏ đến chuỗi '-.'.



```

0040193F . 8B45 E4      MOV EAX,DWORD PTR [EBP-1C]
00401942 . 48          DEC EAX
00401943 . 894424 04    MOV DWORD PTR [ESP+4],EAX
00401947 . C70424 94204 MOV DWORD PTR [ESP],3.00442094
0040194E . C785 B8FDFFFF MOV DWORD PTR [EBP-248],-1
00401958 . E8 63E40200 CALL 3.0042FDC0
0040195D . 0FB600      MOVZX EAX,BYTE PTR [EAX]
00401960 . 8845 DB      MOV BYTE PTR [EBP-25],AL
00401963 . C74424 04 00 MOV DWORD PTR [ESP+4],0
00401968 . C70424 94204 MOV DWORD PTR [ESP],3.00442094
00401972 . E8 49E40200 CALL 3.0042FDC0
00401977 . 0FB600      MOVZX EAX,BYTE PTR [EAX]
0040197A . 8845 DA      MOV BYTE PTR [EBP-26],AL
0040197D . 0FBE55 DB    MOVSX EDX,BYTE PTR [EBP-25]
00401981 . 0FBE45 DA    MOVSX EAX,BYTE PTR [EBP-26]
00401985 . 0FAFC2      IMUL EAX,EDX
00401988 . 0FAFC0      IMUL EAX,EAX
0040198B . 8945 D4      MOV DWORD PTR [EBP-2C],EAX
0040198E . 8D45 D4      LEA EAX,DWORD PTR [EBP-2C]
00401991 . 8130 21B2000 XOR DWORD PTR [EAX],0B221

```

Đoạn code trên xử lý thêm 2 ký tự đầu và cuối trong chuỗi username. Cụ thể:

- **MOV EAX, DWORD PTR [EBP-1C]:** Đặt giá trị [EBP-1C] (length – độ dài của username) vào EAX.
- **DEC EAX:** Giảm giá trị EAX đi 1. Bây giờ EAX chứa giá trị length - 1, là index của ký tự cuối cùng trong chuỗi username.
- **MOV DWORD PTR [ESP+4], EAX:** Đặt giá trị EAX (length – 1) vào [ESP+4] – đây là tham số thứ hai cho hàm sắp gọi tới.
- **MOV DWORD PTR [ESP], 3.00442094:** Đặt địa chỉ 00442094 (địa chỉ trỏ đến chuỗi username) vào [ESP] – đây là tham số đầu tiên cho hàm sắp gọi tới.
- **MOV DWORD PTR [EBP-248], -1:** Khởi tạo biến cục bộ tại [EBP-248] với giá trị -1. Đây có thể là một kích thước, cờ, hoặc chỉ số.
- **CALL 0042FDC0:** Gọi hàm 0042FDC0 với hai tham số: địa chỉ của chuỗi username ([ESP]) và length – 1 ([ESP+4]). Hàm này sẽ trả về địa chỉ của ký tự cuối cùng trong username (&username[length – 1]) và lưu vào EAX.
- **MOVZX EAX, BYTE PTR [EAX]:** Đọc byte tại EAX (mã ASCII của ký tự cuối) và mở rộng thành một DWORD với các byte cao bằng 0 vào EAX. Bây giờ EAX chứa mã ASCII của ký tự cuối cùng trong username.
- **MOV BYTE PTR [EBP-25], AL:** Lưu byte thấp của EAX (chứa mã ASCII của ký tự cuối cùng) vào [EBP-25].

- **Từ 00401963 – 0040197A:** Sau đoạn xử lý ký tự cuối, chương trình thực hiện tương tự với ký tự đầu tiên trong chuỗi username. Kết quả là [EBP-26] chứa mã ASCII của ký tự đầu tiên.
- **MOVSX EDX, BYTE PTR [EBP-25]:** Đọc byte tại [EBP-25] (mã ASCII của ký tự cuối cùng) và mở rộng dấu của nó thành một DWORD, lưu vào EDX.
- **MOVSX EAX, BYTE PTR [EBP-26]:** Đọc byte tại [EBP-26] (mã ASCII của ký tự đầu tiên) và mở rộng dấu của nó thành một DWORD, lưu vào EAX.
- **IMUL EAX, EDX:** Thực hiện phép nhân có dấu giữa giá trị trong EAX (mã ASCII ký tự đầu tiên) và giá trị trong EDX (mã ASCII ký tự cuối cùng). Kết quả được lưu vào EAX.
- **IMUL EAX, EAX:** Bình phương kết quả (tích của mã ASCII ký tự đầu và cuối) trong EAX và lưu vào EAX.
- **MOV DWORD PTR [EBP-2C], EAX:** Đặt giá trị EAX (bình phương tích của mã ASCII ký tự đầu và cuối) vào [EBP-2C].
- **LEA EAX, DWORD PTR [EBP-2C]:** Đặt giá trị [EBP-2C] (bình phương tích của mã ASCII ký tự đầu và cuối) vào EAX.
- **XOR DWORD PTR [EAX], 0B221:** Thực hiện phép XOR giữa giá trị DWORD tại EAX trở tới (tức là giá trị tại [EBP-2C]) với hằng số 0xB221. Kết quả được lưu vào [EBP-2C].

⇒ **Kết quả:** Biến cục bộ [EBP-2C] chứa giá trị  $(\text{username}[0] * \text{username}[\text{length} - 1])^2 \text{ XOR } 0xB221$ .

00401997	· 8B45 D4	MOV EAX, DWORD PTR [EBP-2C]
0040199A	· 99	CDQ
0040199B	· F73D 28204400	IDIV DWORD PTR [442028]
004019A1	· A3 28204400	MOV DWORD PTR [442028], EAX

Đoạn code trên thực hiện phép chia giữa hai biểu thức đã tính, nhằm tạo ra một giá trị quan trọng trong quá trình sinh key. Cụ thể:

- **MOV EAX, DWORD PTR [EBP-2C]:** Đặt giá trị [EBP-2C]  $((\text{username}[0] * \text{username}[\text{length} - 1])^2 \text{ XOR } 0xB221)$  vào EAX. Đây sẽ là số bị chia.
- **CDQ:** Chuyển đổi giá trị DWORD trong EAX thành một QWORD (64-bit) trong cặp thanh ghi EDX:EAX bằng cách mở rộng dấu – chuẩn bị cho phép chia.



- **IDIV DWORD PTR [442028]:** Thực hiện phép chia số nguyên có dấu của QWORD EDX:EAX cho giá trị DWORD tại địa chỉ [442028] (sum ASCII). Thương của phép chia được lưu vào EAX, và số dư được lưu vào EDX.
- **MOV DWORD PTR [442028], EAX:** Đặt giá trị EAX (thương của phép chia) vào địa chỉ [442028].

⇒ **Kết quả:** Biến toàn cục [442028] chứa giá trị thương của phép tính  $((\text{username}[0] * \text{username}[\text{length} - 1])^2 \text{ XOR } 0xB221) / (\text{sum ASCII XOR length}^3)$ .

```

004019A6 . 8D45 B8 LEA EAX,DWORD PTR [EBP-48]
004019A9 . 890424 MOV DWORD PTR [ESP],EAX
004019AC . E8 2FDD0200 CALL 3.0042F6E0
004019B1 . C74424 04 08 MOV DWORD PTR [ESP+4],8
004019B9 . C70424 100000 MOV DWORD PTR [ESP],10
004019C0 . E8 83A30300 CALL 3.0043BD48
004019C5 . 894424 04 MOV DWORD PTR [ESP+4],EAX
004019C9 . 8D85 F8FEFFFF LEA EAX,DWORD PTR [EBP-108]
004019CF . 890424 MOV DWORD PTR [ESP],EAX
004019D2 . C785 B8FDFFFF MOV DWORD PTR [EBP-248],8
004019DC . E8 5F480300 CALL 3.00436240
004019E1 . A1 28204400 MOV EAX,DWORD PTR [442028]
004019E6 . 894424 04 MOV DWORD PTR [ESP+4],EAX
004019EA . 8D85 F8FEFFFF LEA EAX,DWORD PTR [EBP-108]
004019F0 . 83C0 08 ADD EAX,8
004019F3 . 890424 MOV DWORD PTR [ESP],EAX
004019F6 . C785 B8FDFFFF MOV DWORD PTR [EBP-248],7
00401A00 . E8 8BA40200 CALL 3.0042BE90
00401A05 . 8D95 E8FEFFFF LEA EDX,DWORD PTR [EBP-118]
00401A0B . 8D85 F8FEFFFF LEA EAX,DWORD PTR [EBP-108]
00401A11 . 894424 04 MOV DWORD PTR [ESP+4],EAX
00401A15 . 891424 MOV DWORD PTR [ESP],EDX
00401A18 . E8 B3240100 CALL 3.00413ED0
00401A1D . 83EC 04 SUB ESP,4
00401A20 . 8D85 E8FEFFFF LEA EAX,DWORD PTR [EBP-118]
00401A26 . 894424 04 MOV DWORD PTR [ESP+4],EAX
00401A2A . 8D45 B8 LEA EAX,DWORD PTR [EBP-48]
00401A2D . 890424 MOV DWORD PTR [ESP],EAX
00401A30 . C785 B8FDFFFF MOV DWORD PTR [EBP-248],6
00401A3A . E8 01E30200 CALL 3.0042FD40
00401A3F . 74 35 JMP SHORT 3.00401A76
00401A41 . 8B95 A8FDFFFF MOV EDX,DWORD PTR [EBP-258]
00401A47 . 8995 ACFDFFFF MOV DWORD PTR [EBP-254],EDX
00401A4D . 8D85 E8FEFFFF LEA EAX,DWORD PTR [EBP-118]
00401A53 . 890424 MOV DWORD PTR [ESP],EAX
00401A56 . C785 B8FDFFFF MOV DWORD PTR [EBP-248],0
00401A60 . E8 3BE10200 CALL 3.0042FBA0
00401A65 . 8B85 ACFDFFFF MOV EAX,DWORD PTR [EBP-254]
00401A6B . 8985 A8FDFFFF MOV DWORD PTR [EBP-258],EAX
00401A71 . 74 1C JMP 3.00401EC1
00401A76 . 8D85 E8FEFFFF LEA EAX,DWORD PTR [EBP-118]
00401A7C . 890424 MOV DWORD PTR [ESP],EAX
00401A7F . C785 B8FDFFFF MOV DWORD PTR [EBP-248],7
00401A89 . E8 12E10200 CALL 3.0042FBA0

```

```

00401A8E . 8D85 E8FEFFFF LEA EAX,DWORD PTR [EBP-118]
00401A94 . 890424 MOV DWORD PTR [ESP],EAX
00401A97 . E8 44DC0200 CALL 3.0042F6E0
00401A9C . C74424 04 08 MOV DWORD PTR [ESP+4],8
00401AA4 . C70424 100000 MOV DWORD PTR [ESP],10
00401AAB . E8 98A20300 CALL 3.0043BD48
00401AB0 . 894424 04 MOV DWORD PTR [ESP+4],EAX
00401AB4 . 8D85 28FEFFFF LEA EAX,DWORD PTR [EBP-1D8]
00401ABA . 890424 MOV DWORD PTR [ESP],EAX
00401ABD . C785 B8FDFFFF MOV DWORD PTR [EBP-248],5
00401AC7 . E8 74470300 CALL 3.00436240
00401ACC . 8B45 D4 MOV EAX,DWORD PTR [EBP-2C]
00401ACF . 894424 04 MOV DWORD PTR [ESP+4],EAX
00401AD3 . 8D85 28FEFFFF LEA EAX,DWORD PTR [EBP-1D8]
00401AD9 . 83C0 08 ADD EAX,8
00401ADC . 890424 MOV DWORD PTR [ESP],EAX
00401ADF . C785 B8FDFFFF MOV DWORD PTR [EBP-248],4
00401AE9 . E8 F2A30200 CALL 3.0042BEE0
00401AEE . 8D95 18FEFFFF LEA EDX,DWORD PTR [EBP-1E8]
00401AF4 . 8D85 28FEFFFF LEA EAX,DWORD PTR [EBP-1D8]
00401AFA . 894424 04 MOV DWORD PTR [ESP+4],EAX
00401AFE . 891424 MOV DWORD PTR [ESP],EDX
00401B01 . E8 CA230100 CALL 3.00413ED0
00401B06 . 83EC 04 SUB ESP,4
00401B09 . 8D85 18FEFFFF LEA EAX,DWORD PTR [EBP-1E8]
00401B0F . 894424 04 MOV DWORD PTR [ESP+4],EAX
00401B13 . 8D85 E8FEFFFF LEA EAX,DWORD PTR [EBP-118]
00401B19 . 890424 MOV DWORD PTR [ESP],EAX
00401B1C . C785 B8FDFFFF MOV DWORD PTR [EBP-248],3
00401B26 . E8 15E20200 CALL 3.0042FD40
00401B2B . < EB 35 JMP SHORT 3.00401B62
00401B2D > 8B95 A8FDFFFF MOV EDX,DWORD PTR [EBP-258]
00401B33 . 8995 A4FDFFFF MOV DWORD PTR [EBP-25C],EDX
00401B39 . 8D85 18FEFFFF LEA EAX,DWORD PTR [EBP-1E8]
00401B3F . 890424 MOV DWORD PTR [ESP],EAX
00401B42 . C785 B8FDFFFF MOV DWORD PTR [EBP-248],0
00401B4C . E8 4FE00200 CALL 3.0042FBA0
00401B51 . 8B85 A4FDFFFF MOV EAX,DWORD PTR [EBP-25C]
00401B57 . 8985 A8FDFFFF MOV DWORD PTR [EBP-258],EAX
00401B5D . < E9 C7020000 JMP 3.00401E29
00401B62 > 8D85 18FEFFFF LEA EAX,DWORD PTR [EBP-1E8]
00401B68 . 890424 MOV DWORD PTR [ESP],EAX
00401B6B . C785 B8FDFFFF MOV DWORD PTR [EBP-248],4
00401B75 . E8 26E00200 CALL 3.0042FBA0

```

Đoạn code trên thực hiện chức năng tương tự nhau, đều khởi tạo buffer và stringstream, sau đó ghi giá trị thương của phép chia trước đó vào đối tượng std::stringstream. Cuối cùng, sao chép nó ra một buffer tạm để sử dụng tiếp và giải phóng bộ nhớ cho đối tượng std::stringstream.

⇒ **Kết quả:** [EBP-48] chứa chuỗi kết quả thương của phép tính  $((\text{username}[0] * \text{username}[\text{length} - 1])^2 \text{ XOR } 0xB221) / (\text{sum ASCII XOR length}^3)$ . [EBP-118] chứa chuỗi của giá trị  $(\text{username}[0] \times \text{username}[\text{length} - 1])^2 \text{ XOR } 0xB221$ .

00401B7A	. C74424 04 0000	MOV DWORD PTR [ESP+4],0	
00401B82	. 8D45 B8	LEA EAX,DWORD PTR [EBP-48]	
00401B85	. 890424	MOV DWORD PTR [ESP],EAX	
00401B88	. E8 33E20200	CALL 3.0042FDC0	
00401B8D	. 8985 14FEFFFF	MOV DWORD PTR [EBP-1EC],EAX	
00401B93	. C74424 04 0000	MOV DWORD PTR [ESP+4],0	
00401B9B	. 8D85 E8FEFFFF	LEA EAX,DWORD PTR [EBP-118]	
00401BA1	. 890424	MOV DWORD PTR [ESP],EAX	
00401BA4	. E8 17E20200	CALL 3.0042FDC0	
00401BA9	. 8985 10FEFFFF	MOV DWORD PTR [EBP-1F0],EAX	
00401BAF	. 8B85 14FEFFFF	MOV EAX,DWORD PTR [EBP-1EC]	
00401BB5	. 894424 04	MOV DWORD PTR [ESP+4],EAX	
00401BB9	. C70424 3020440	MOV DWORD PTR [ESP],3.00442030	
00401BC0	. E8 7BE80000	CALL <JMP.&msvort.strocpy>	strocpy
00401BC5	. 8B45 DC	MOV EAX,DWORD PTR [EBP-24]	
00401BC8	. 894424 04	MOV DWORD PTR [ESP+4],EAX	
00401BCC	. C70424 3020440	MOV DWORD PTR [ESP],3.00442030	
00401BD3	. E8 58E80000	CALL <JMP.&msvort.stroat>	stroat
00401BD8	. 8B85 10FEFFFF	MOV EAX,DWORD PTR [EBP-1F0]	
00401BDE	. 894424 04	MOV DWORD PTR [ESP+4],EAX	
00401BE2	. C70424 3020440	MOV DWORD PTR [ESP],3.00442030	
00401BE9	. E8 42E80000	CALL <JMP.&msvort.stroat>	stroat

Đoạn code trên nối các chuỗi phép tính lại – đây là giai đoạn cuối cùng trong quá trình tạo chuỗi key hoàn chỉnh. Đầu tiên, chương trình sẽ sao chép chuỗi trong [EBP-48] vào 00442030. Sau đó, nối 00442030 với [EBP-24] và [EBP-118].

⇒ **Kết quả:** Giá trị hằng số 00442030 chứa chuỗi kết quả thương của phép tính  $((\text{username}[0] * \text{username}[\text{length} - 1])^2 \text{ XOR } 0xB221) / (\text{sum ASCII XOR length}^3)$  nối với chuỗi thứ hai (-), nối tiếp với chuỗi thứ ba (chuỗi của giá trị  $(\text{username}[0] * \text{username}[\text{length} - 1])^2 \text{ XOR } 0xB221)$ .

### 1.3.2. Kết luận và ví dụ minh họa

#### a) Kết luận

- Tính L = Độ dài của chuỗi username (không tính kí tự kết thúc \0):

+ Nếu chuỗi username  $\leq 4$ : Không phát sinh key.

+ Nếu chuỗi username  $> 4$ :

- Tính S = Tổng các ký tự ASCII có dấu trong username.
- Tính  $P = S \text{ XOR } L^3$
- Tính  $Q = (\text{username}[0] * \text{username}[L - 1])^2 \text{ XOR } 0xB221$
- Tính  $R = Q / P$  với R là phần nguyên của phép chia.
- Chuyển R, Q thành chuỗi.
- Key = R-Q

#### b) Ví dụ minh họa

**Ví dụ 1:** Chuỗi là “hi” → Không sinh ra key nào cả.

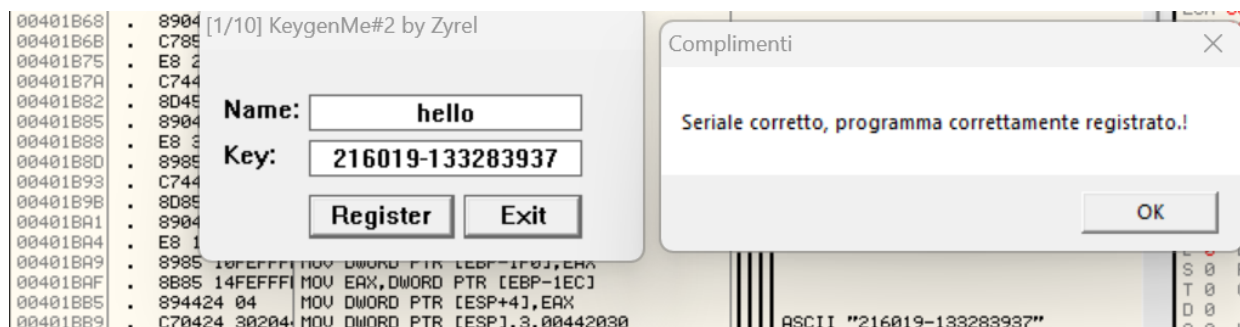
Kiểm tra lại kết quả bằng chương trình:



**Ví dụ 2:** Chuỗi là “hello”

- $L = 5$
- $S = 104 + 101 + 108 + 108 + 111 = 532$ .
- $P = S \text{ XOR } L^3 = 532 \text{ XOR } 5^3 = 532 \text{ XOR } 125 = 617$
- $Q = (\text{username}[0] * \text{username}[L - 1])^2 \text{ XOR } 0xB221 = (104 * 111)^2 \text{ XOR } 45601 = 133263936 \text{ XOR } 45601 = 133283937$
- $R = Q / P = 133283937 / 617 = 216019$
- $\text{Key} = R - Q = 216019 - 133283937$

Kiểm tra lại kết quả bằng chương trình:



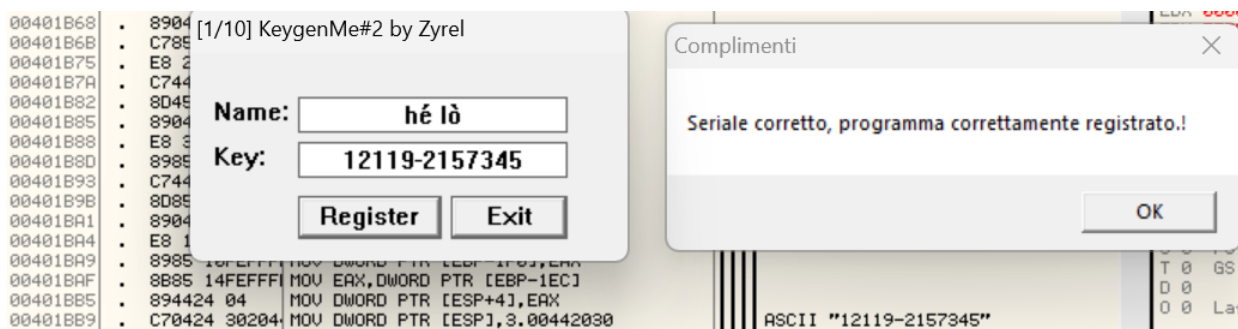
**Ví dụ 3:** Chuỗi là “hé lò”

- $L = 5$
- $S = 104 + (233 - 256) + 32 + 108 + (242 - 256) = 207$
- $P = S \text{ XOR } L^3 = 207 \text{ XOR } 5^3 = 207 \text{ XOR } 125 = 178$
- $Q = (\text{username}[0] * \text{username}[L - 1])^2 \text{ XOR } 0xB221 = [104 * (242 - 256)]^2 \text{ XOR } 45601 = 2119936 \text{ XOR } 45601 = 2157345$
- $R = Q / P = 2157345 / 178 = 12119$



- Key = R-Q = 12119-2157345

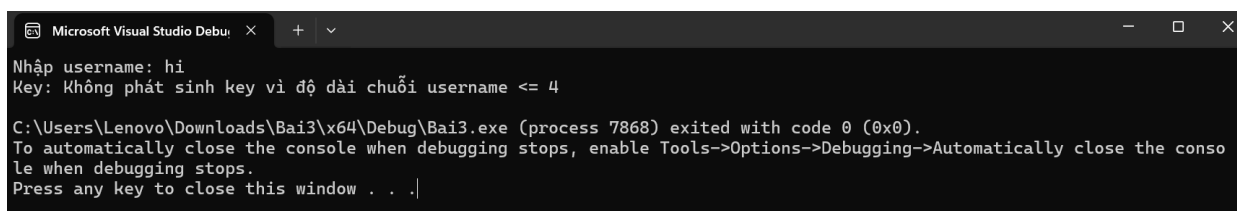
Kiểm tra lại kết quả bằng chương trình:



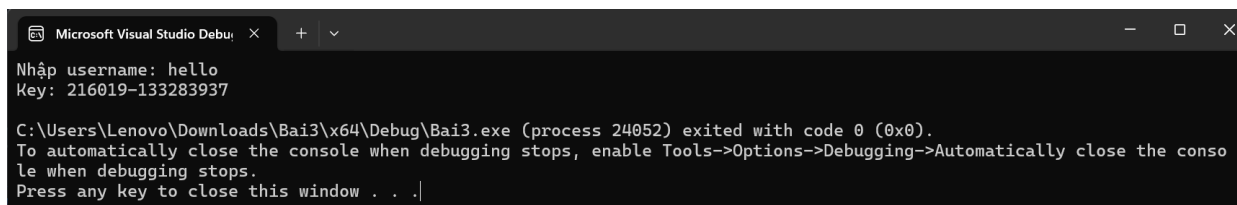
### 1.3.3. Chương trình keygen

Chương trình keygen được viết bằng ngôn ngữ C/C++ với các bước phát sinh thuật toán key như đã phân tích ở trên. Dưới đây là một vài trường hợp chạy thử xem chương trình có giống với ví dụ đã phân tích ở trên không:

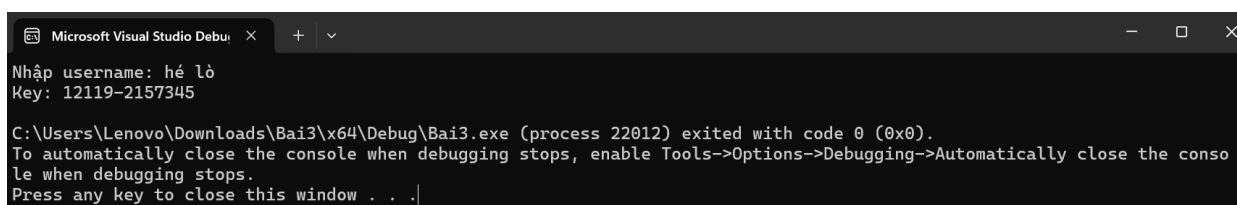
- Đối với chuỗi username ngắn:



- Đối với chuỗi username bình thường:



- Đối với chuỗi username có dấu:



⇒ Chương trình keygen đã phát sinh key của từng username đúng so với chương trình gốc 3.exe.

### 3. Tài liệu tham khảo

- [1] <https://en.wikipedia.org/wiki/Crackme>
- [2] <https://0xk4n3ki.github.io/posts/Heavens-Gate-Technique/>