

ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN



**ĐỒ ÁN CRACK PHẦN MỀM – ĐỀ 2
KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ**



♦ SINH VIÊN THỰC HIỆN ♦

Nguyễn Hữu Gia Trí	-	1712254
Huỳnh Thái Anh	-	1712272
Đỗ Quang Vinh	-	1712207

MỤC LỤC

THÔNG TIN THÀNH VIÊN VÀ PHÂN CÔNG	3
BÀI 2.1	4
Bài 2.2.....	6
Bài 2.3.....	9
Bài 2.4.....	14
Bài 2.5.....	19
MỨC ĐỘ HOÀN THIỆN	26
TÀI LIỆU THAM KHẢO.....	26

THÔNG TIN THÀNH VIÊN VÀ PHÂN CÔNG

1. Thông tin thành viên

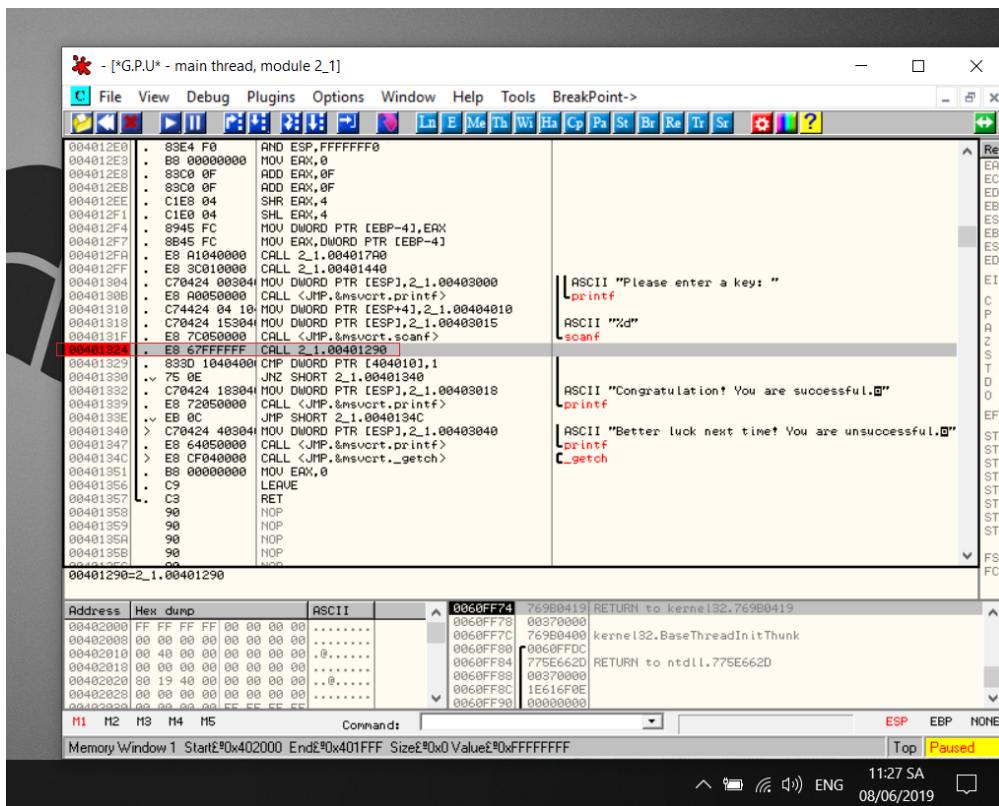
Tên	MSSV	Email
Nguyễn Hữu Gia Trí	1712254	1712254@student.hcmus.edu.vn
Huỳnh Thái Anh	1712272	1712272@student.hcmus.edu.vn
Đỗ Quang Vinh	1712207	1712207@student.hcmus.edu.vn

2. Phân công

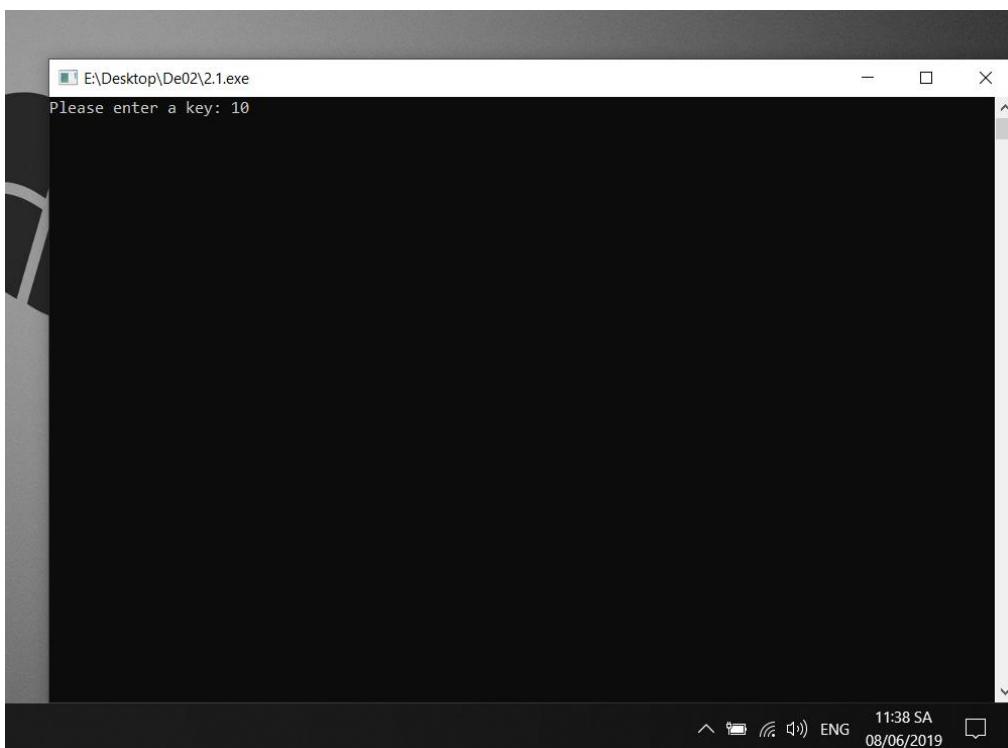
Tên	Công việc	Mức độ hoàn thành
Nguyễn Hữu Gia Trí	<ul style="list-style-type: none">Crack bài 2.1 + 2.2 + 2.4Viết báo cáo	100%
Huỳnh Thái Anh	<ul style="list-style-type: none">Crack bài 2.5Hỗ trợ viết báo cáo	100%
Đỗ Quang Vinh	<ul style="list-style-type: none">Crack bài 2.3Hỗ trợ viết báo cáo	100%

BÀI 2.1

- Sau khi tìm đến **badboy** của crackme, ta nhận thấy ở dòng **40401324** có gọi đến một hàm nào đó, ta nghi ngờ đây chính là hàm xử lý key → Đặt break point tại đây.



- Nhập thử key là **10**.

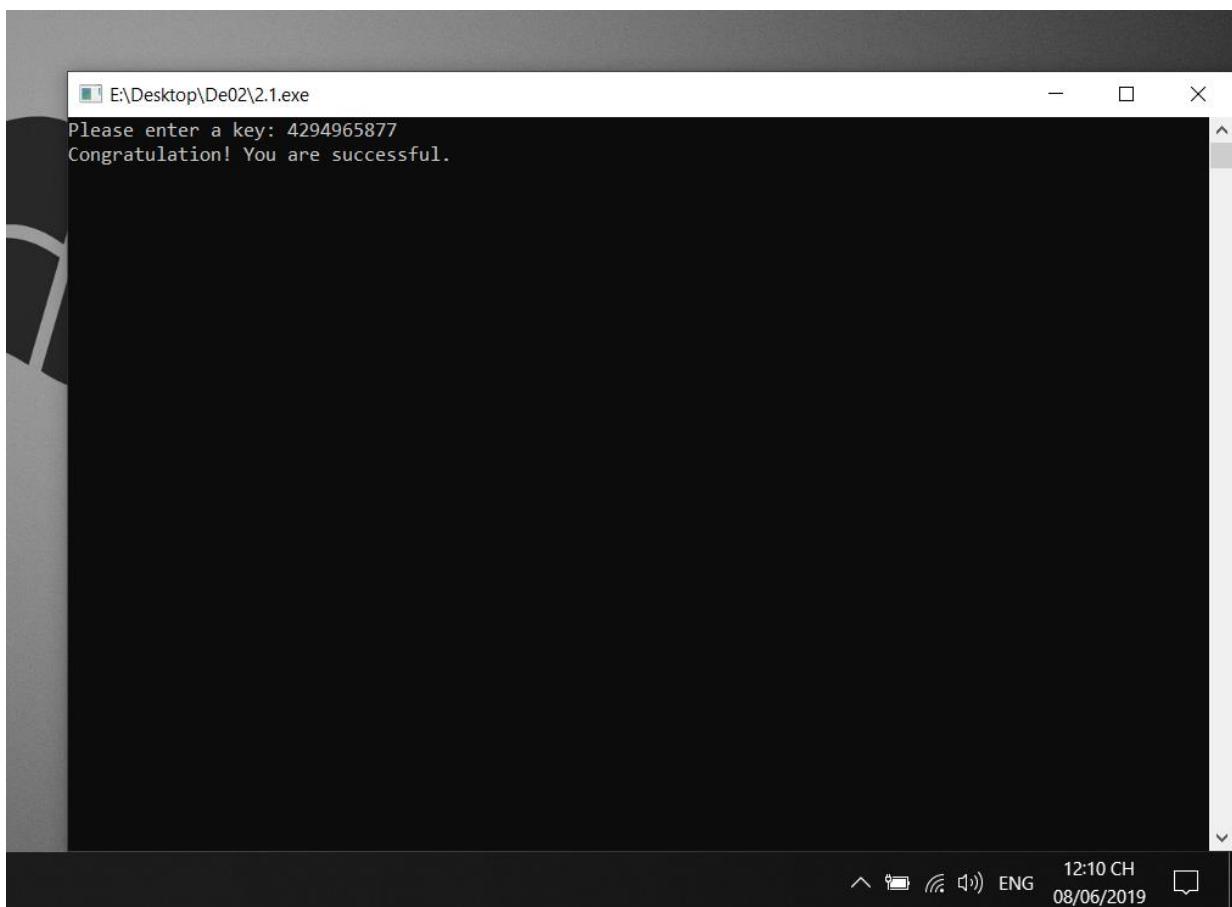


⊕ Đi vào hàm xử lý key

```
00401290 PUSH EBP          //Lưu lại giá trị của EBP = 0060FF28
00401291 MOV EBP,ESP       //Gán EBP = ESP = 0060FEF8
00401293 SUB ESP,4         //ESP = ESP-4 = 0060FEF4
00401296 MOV DWORD PTR [EBP-4],96 // [ESP] = 96
0040129D MOV EDX,DWORD PTR [EBP-4] //EDX = [ESP] = 96
004012A0 MOV EAX,EDX        //EAX = EDX = 96
004012A2 SHL EAX,3         //EAX dịch trái EAX 3 bits = 4B0
004012A5 SUB EAX,EDX        //EAX = EAX-EDX = 4B0 - 96 = 41A
004012A7 MOV DWORD PTR [EBP-4],EAX // [ESP] = EAX = 41A
004012AA LEA EAX,DWORD PTR [EBP-4] //EAX = EBP-4 = 0060FEF4
004012AD NOT DWORD PTR [EAX]    //not [EAX] -> ESP = FFFFFBE5
004012AF LEA EAX,DWORD PTR [EBP-4] //EAX = EBP - 4 = 0060FEF4
004012B2 XOR DWORD PTR [EAX],190 //XOR ESP với 190 -> ESP = FFFFA75
004012B8 MOV EAX,DWORD PTR [404010] //Gán EAX = [404010] = 0000000A
004012BD CMP EAX,DWORD PTR [EBP-4] //So sánh EAX và ESP
```

- ⊕ Hãy chú ý vào dòng **004012BD**, ta thấy hàm đang so sánh **EAX** đang mang giá trị **0000000A** với **ESP** tức có nghĩa đang so sánh key ta nhập vào ở dạng hexa với **ESP** cũng đang ở dạng hexa.
→ **ESP** ở dạng dec chính là key của chúng ta. Đổi ra ta có key = **FFFFFA75₁₆** = **4294965877₁₀**.

- ⊕ Test lại lần nữa, ta có kết quả:

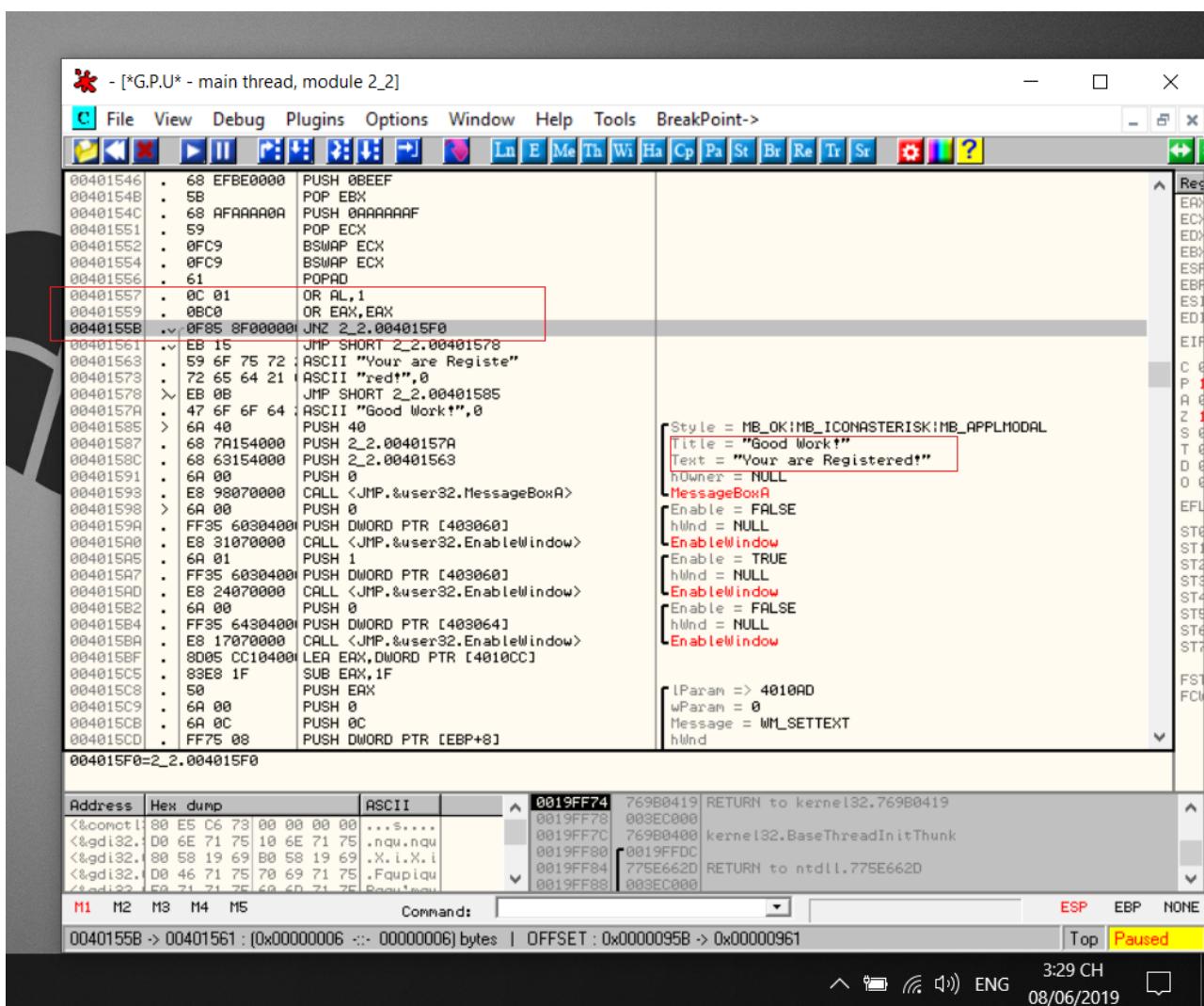


Bài 2.2

-  Ta thấy ở dòng **00401557**, sau khi chương trình thực hiện lệnh **OR AL, 1** (Thanh ghi **AL** mang giá trị 8 bits cuối của **EAX**) thì **EAX = 00000001**.

Đến dòng tiếp theo $EAX = EAX OR EAX = 1 OR 1 = 1$, mà chương trình chỉ nhảy đến *goodboy* khi $EAX = 0$ nên chương trình sẽ luôn luôn bỏ qua *goodboy*.

→ Như vậy message box “*Your are Registered*” không phải là *goodboy* thực sự.



- ✚ Xét đoạn lệnh từ dòng **401598** đến **4015EB**, chương trình thực hiện thay đổi nội dung của hộp tin (*Unregistered*) thành (*Registered*), *post your solution please*. Vậy đoạn lệnh này được gọi ở đâu ?

The screenshot shows the Immunity Debugger interface. The assembly pane displays the following sequence of instructions:

```

00401598 > 6A 00    PUSH 0
0040159A . FF35 60304000 PUSH DWORD PTR [403060]
004015A0 . E8 31070000 CALL <JMP.&user32.EnableWindow>
004015A5 . 6A 01    PUSH 1
004015A7 . FF35 60304000 PUSH DWORD PTR [403060]
004015AD . E8 24070000 CALL <JMP.&user32.EnableWindow>
004015B2 . 6A 00    PUSH 0
004015B4 . FF35 64304000 PUSH DWORD PTR [403064]
004015BA . E8 17070000 CALL <JMP.&user32.EnableWindow>
004015BF . 8005 CC104000 LEA EAX, DWORD PTR [4010CC]
004015C5 . 83E8 1F    SUB EAX, 1F
004015C8 . 50        PUSH EAX
004015C9 . 6A 00    PUSH 0
004015CB . 6A 0C    PUSH 0C
004015CD . FF75 08    PUSH DWORD PTR [EBP+8]
004015D0 . E8 67070000 CALL <JMP.&user32.SendMessageA>
004015D5 . 8005 AE324000 LEA EAX, DWORD PTR [4032AE]
004015DE . 83C0 0F    ADD EAX, 0F
004015DF . 50        PUSH EAX
004015E0 . 6A 00    PUSH 0
004015E1 . 6A 0C    PUSH 0C
004015E3 . 68 CA000000 PUSH 0CA
004015E8 . FF75 08    PUSH DWORD PTR [EBP+8]
004015EB . E8 46070000 CALL <JMP.&user32.SendDlgItemMessageA>
004015F0 > 8170 10 2F01 CMP DWORD PTR [EBP+10], 12F

```

The memory dump pane shows the following ASCII dump:

Address	Hex dump	ASCII
004032AE	28 55 6E 72 65 67 69 73	(Unregis
004032B6	74 65 72 65 64 29 00 28	tered). (
004032BE	52 65 67 69 73 74 65 72	Register
004032C6	65 64 29 2C 20 70 6F 73	ed), pos
004032CE	74 20 79 6F 75 72 20 73	your s
004032D6	6F 6C 75 74 69 6F 6E 20	olution.
004032DE	70 6C 65 61 73 65 2E 00	please..
004032E6	00 00 00 00 00 00 00 00
004032EF	00 00 00 00 00 00 00 00

A red box highlights the word "Register" in the memory dump. An arrow points from this box to the assembly instruction at address 0x4015E8, which corresponds to the `PUSH DWORD PTR [EBP+8]` instruction.

✚ Xét tiếp đoạn lệnh sau:

00401692 MOV EAX,DWORD PTR [403330] //EAX = 4 ký tự cuối theo thứ tự từ phải qua trái
00401697 XOR EBX, EBX
00401699 MOV BL,BYTE PTR [403331] //BL = ký tự cuối

0040169F CMP BL,2D //So sánh BL với '-'
004016A2 JNZ SHORT 2_2.004016C7 //Nếu không bằng thì nhảy

004016AD BSWAP EAX //Đảo ngược EAX
004016AF CMP AL, 53h //So sánh ký tự thứ 8 với 'S'
004016B1 JE SHORT 2_2.004016B8 //Đúng thì nhảy
004016B3 CMP AH,38 //So sánh ký tự thứ 7 với '8'

✚ Vậy ta có 2 dạng key:

Key = *****-*S***

Key = *****-8****

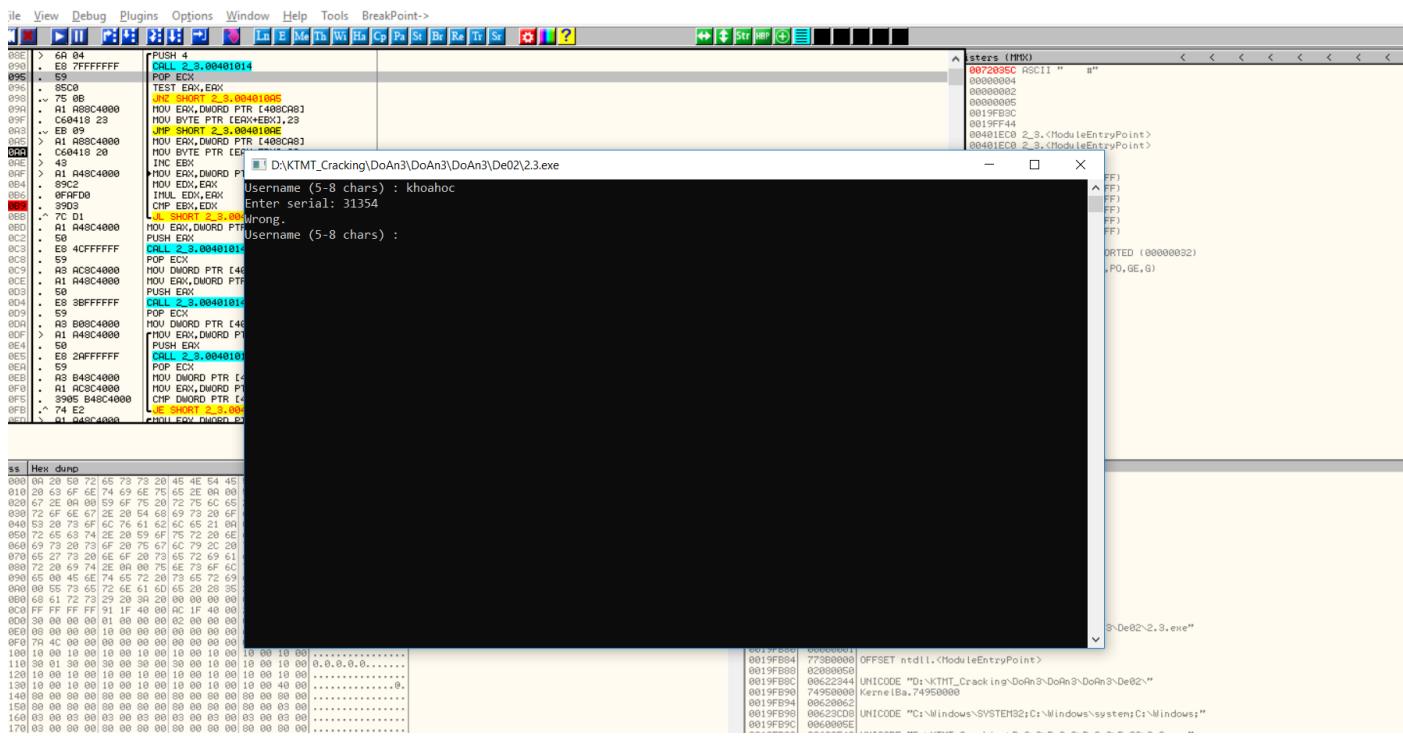
(Với * là bất kì ký tự nào)

✚ Test lại lần nữa, ta có kết quả:



Bài 2.3

Đầu tiên ta chạy thử chương trình:

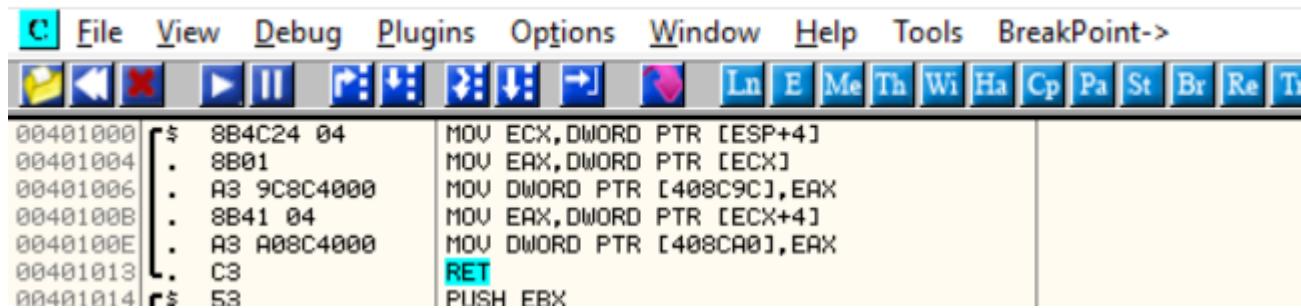


Chúng ta có hai thứ cần nhập là Username và Serial.

Sau khi đọc qua và phân tích đoạn code assembly của qua chương trình OllyDbg ta thấy được đây là chương trình phát Key(Serial) theo Username nhập vào.

Ta bắt đầu với đoạn code:

 - [*G.P.U* - main thread, module 2_3]



Đây là đoạn code hàm khởi tạo giá trị đầu tiên t1 và t2: 00401000 -> 00401013

t1: là 4 ký tự đầu của chuỗi Username

t2: là những ký tự còn lại.

Code trong keygen:

Solution '2.3' (1 project)

Solution

- References
- External Dependencies
- Header Files
- Resource Files
- Source Files**
 - * Source.cpp

```

9     unsigned int x,
10    unsigned int y;
11 };
12
13 unsigned int t1 = 0x00, t2 = 0x00;
14 string Name, serial;
15 unsigned int Namelen, SizeMaze;
16 string Maze[];
17 Point start, finish;
18 string key;
19
20 void Init()//Khởi tạo t1 và t2
21 {
22     for (int i = 3; i >= 0; i--)
23     {
24         char s = Name[i] & 0xff;
25         t1 = t1 << 8;
26         t1 = t1 + int(s);
27     }
28     Namelen = Name.length();
29     for (int i = Namelen - 1; i >= 4; i--)
30     {
31         char s = Name[i] & 0xff;
32         t2 = t2 << 8;
33         t2 = t2 + int(s);
34     }
}

```

Tiếp theo là tạo mê cung: Trong phần tạo mê cung ta có 2 hàm là hàm lấy số ngẫu nhiên và tạo ra mê cung.

+ Hàm lấy 1 số ngẫu nhiên:

<pre> 00401014 \$ 53 PUSH EBX 00401015 . A1 9C8C4000 MOV EAX,DWORD PTR [408C9C] 0040101A . BA C15D0000 MOV EDX,5DC1 0040101F . F7E2 MUL EDX 00401021 . B9 0B560000 MOV ECX,560B 00401026 . 31D2 XOR EDX,EDX 00401028 . F7F1 DIV ECX 0040102A . 8915 9C8C4000 MOV DWORD PTR [408C9C],EDX 00401030 . A1 A08C4000 MOV EAX,DWORD PTR [408CA0] 00401035 . BA 9D540000 MOV EDX,549D 0040103A . F7E2 MUL EDX 0040103C . B9 A1510000 MOV ECX,51A1 00401041 . 31D2 XOR EDX,EDX 00401043 . F7F1 DIV ECX 00401045 . 8915 A08C4000 MOV DWORD PTR [408CA0],EDX 0040104B . A1 9C8C4000 MOV EAX,DWORD PTR [408C9C] 00401050 . 8B15 A08C4000 MOV EDX,DWORD PTR [408CA0] 00401056 . 01D0 ADD EAX,EDX 00401058 . 884C24 08 MOV ECX,DWORD PTR [ESP+8] 0040105C . 31D2 XOR EDX,EDX 0040105E . F7F1 DIV ECX 00401060 . 89D3 MOV EBX,EDX 00401062 . 8908 MOV EAX,EBX 00401064 . 5B POP EBX 00401065 . C3 RET 00401066 r\$ 53 PUSH EBX </pre>	<pre> 36 int Getrandom(unsigned int num)// lấy 1 số ngẫu nhiên 37 { 38 unsigned int a = t1; 39 unsigned int d = 0x5dc1; 40 a = a * d; 41 unsigned int c = 0x560b; 42 d = 0; 43 t1 = a % c; 44 a = t2; 45 d = 0x549d; 46 a = a * d; 47 c = 0x51a1; 48 d = 0; 49 t2 = a % c; 50 a = t1; 51 d = t2; 52 a = a + d; 53 c = num; 54 unsigned int b = a % c; 55 a = b; 56 return a; 57 } 58 </pre>
--	---

Bên cạnh là hàm trong keygen.

Số sẽ được trả lại trong hàm tạo mê cung.

+Hàm tạo mê cung: từ 00401066 ->0040115c

Kích thước của mê cung= giá trị trả về từ hàm Getrandom cộng với 14

Nằm ở dòng 00401069 ->00401072

00401066	\$ 53	PUSH EBX
00401067	. 6A 14	PUSH 14
00401069	. E8 A6FFFFFF	CALL 2_3.00401014
0040106E	. 59	POP ECX
0040106F	. 83C0 14	ADD EAX,14
00401072	. A3 A48C4000	MOV DWORD PTR [408CA4],EAX
00401073	. 89D2	MOV EDX,EAX
00401079	. 0FAFD0	IMUL EDX,EAX
0040107C	. 89D0	MOV EAX,EDX
0040107E	. 50	PUSH EAX
0040107F	. E8 2C0B0000	CALL 2_3.00401BB0
00401084	. 59	POP ECX
00401085	. A3 A88C4000	MOV DWORD PTR [408CA8],EAX
0040108A	. 31DB	XOR EBX,EBX
0040108C	.^ EB 21	JMP SHORT 2_3.004010AF
0040108E	> 6A 04	PUSH 4
00401090	. E8 7FFFFFFF	CALL 2_3.00401014
00401095	. 59	POP ECX
00401096	. 85C0	TEST EAX,EAX
00401098	.^ 75 0B	JNZ SHORT 2_3.004010A5
0040109A	. A1 A88C4000	MOV EAX,DWORD PTR [408CA8]
0040109F	. C60418 23	MOV BYTE PTR [EAX+EBX],23
004010A3	.^ EB 09	JMP SHORT 2_3.004010AE
004010A5	. A1 A88C4000	MOV EAX,DWORD PTR [408CA8]
004010AA	. C60418 20	MOV BYTE PTR [EAX+EBX],20
004010AE	. 43	INC EBX
004010AF	. A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
004010B4	. 89C2	MOV EDX,EAX
004010B6	. 0FAFD0	IMUL EDX,EAX
004010B9	. 39D3	CMP EBX,EDX
004010BB	.^ 7C D1	JL SHORT 2_3.0040108E
004010BD	. A1 A48C4000	MOV EAX,DWORD PTR [408CA4]
004010C2	. 50	PUSH EAX
004010C3	. E8 4CEFFFFFFF	CALL 2_3.00401014

Local call from 004014D7

Hàm tạo mê cung được minh họa bằng code C++:

```

59  void Createmaze()// tạo mê cung
60  {
61      unsigned int num = 0x14;
62      unsigned int a = Getrandom(num);
63      a += 0x14;
64      SizeMaze = a;
65      unsigned int b = 0;
66      while (b < SizeMaze*SizeMaze)
67      {
68          num = 0x4;
69          a = Getrandom(num);
70          if (a == 0)
71              Maze += "=";
72          else
73              Maze += ".";
74          b++;
75      }
76
77      num = SizeMaze;
78      start.x = Getrandom(num);
79      start.y = Getrandom(num);
80
81      do
82      {
83          num = SizeMaze;
84          finish.x = Getrandom(num);
85      } while (start.x == finish.x);
86
87      do
88      {
89          num = SizeMaze;
90          finish.y = Getrandom(num);
91      } while (start.y == finish.y);
92
93      a = start.x;
94      unsigned int d = SizeMaze;
95      a = a * d;
96      d = start.y;
97      a += d;
98      Maze[a] = 'S';
99
100     a = finish.x;
101     d = SizeMaze;
102     a = a * d;
103     d = finish.y;
104     a += d;
105     Maze[a] = 'F';
106
107 }
```

Tiếp theo là hàm kiểm tra xem có đi đúng đường hay không(Kiểm tra key ta nhập vào có chính xác hay không): 40115d->40123b



Tiếp theo là keygen tìm ra đường đi đúng trong mê cung:

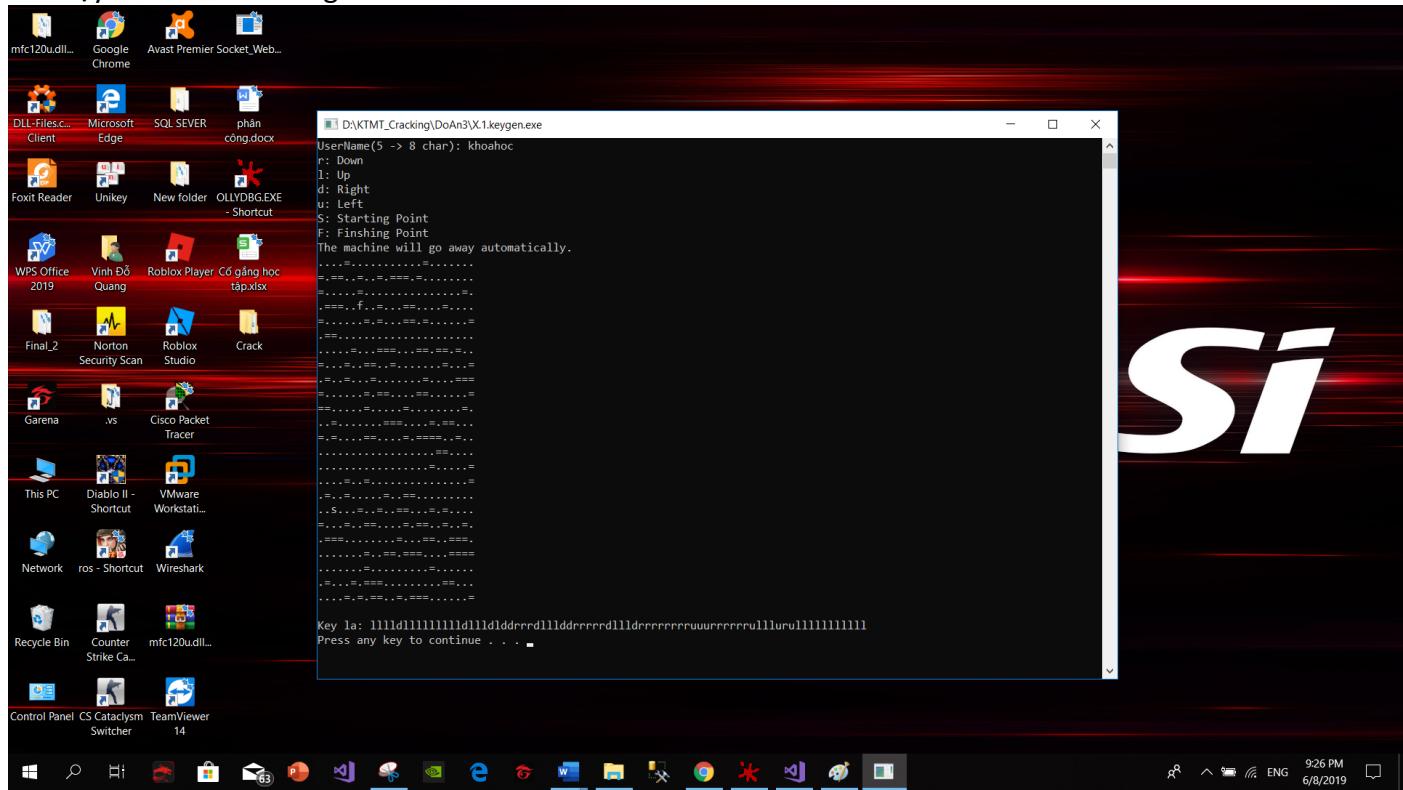
```

108     bool SolveMaze(int x, int y)
109    {
110        if (x > SizeMaze - 1 || x < 0 || y > SizeMaze - 1 || y < 0)
111            return false;
112        if (Maze[x*SizeMaze + y] == 'S')
113            return true;
114        if (Maze[x*SizeMaze + y] != '.' & Maze[x*SizeMaze + y] != 'S')
115            return false;
116        Maze[x*SizeMaze + y] = '+';
117
118        if (SolveMaze(x - 1, y) == true)
119        {
120            key += 'l';
121            return true;
122        }
123        if (SolveMaze(x + 1, y) == true)
124        {
125            key += 'r';
126            return true;
127        }
128        if (SolveMaze(x, y + 1) == true)
129        {
130            key += 'd';
131            return true;
132        }
133        if (SolveMaze(x, y - 1) == true)
134        {
135            key += 'u';
136            return true;
137        }
138        Maze[x*SizeMaze + y] = '.';
139    }
140
141    string GetKey() //hàm trả ra key
142    {
143        if (SolveMaze(start.x, start.y))
144        {
145            string temp = "";
146            for (int i = key.length() - 1; i >= 0; i--)
147            {
148                temp += key[i];
149            }
150            return temp;
151        }
152        else
153        {
154            return "Unsolvable";
155        }
156    }
157
158

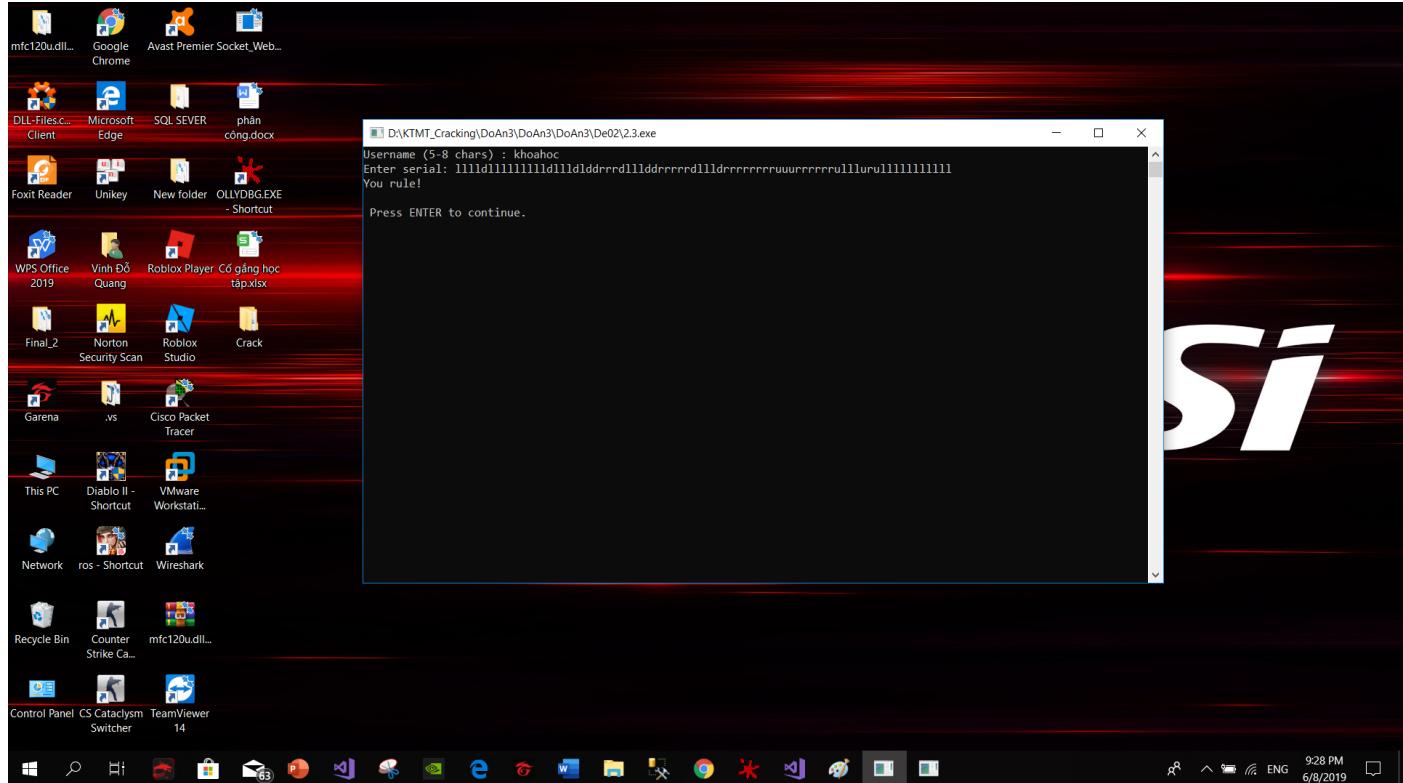
```

Như vậy key sẽ được trả về là key được giải mã từ mã cung được tạo ra từ Username

Ta chạy kiểm thử chương trình:



Ta sẽ nhận được key: ||||ld|||||d|||ld|||ldddrrrd|||ddrrrrrd|||drrrrrrruuurrurrrrullurull|||||



Bài 2.4

- + Đầu tiên, chương trình kiểm tra key nhập vào có đủ 12 kí tự hay chưa. Chúng ta không cần quan tâm đến đoạn chương trình này.
- + Nếu key nhập vào có đủ 12 kí tự, chương trình tiếp tục xử lý 4 kí tự đầu tiên thành chuỗi 8 bytes.

The screenshot shows the Immunity Debugger interface with the following details:

- Registers (FPU):**
 - EAX 0019FFCC
 - ECX 0040A270 2_4.<ModuleEntryPoint>
 - EDX 0040A270 2_4.<ModuleEntryPoint>
 - EBX 003C2000
 - ESP 0019FF74
 - EBP 0019FF80
 - ESI 0040A270 2_4.<ModuleEntryPoint>
 - EDI 0040A270 2_4.<ModuleEntryPoint>
 - EIP 0040A270 2_4.<ModuleEntryPoint>
- Stack (S):**
 - C 0 ES 002B 32bit 0(FFFFFFFF)
 - P 1 CS 0023 32bit 0(FFFFFFFF)
 - A 0 SS 002B 32bit 0(FFFFFFFF)
 - Z 1 DS 002B 32bit 0(FFFFFFFF)
 - S 0 FS 0053 32bit 3C5000(FFFF)
 - T 0 GS 002B 32bit 0(FFFFFFFF)
 - D 0
 - O 0 LastErr ERROR_ENVVAR_NOT_FOUND
 - EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LI)
 - ST0 empty 0.0
 - ST1 empty 0.0
 - ST2 empty 0.0
 - ST3 empty 0.0
 - ST4 empty 0.0
 - ST5 empty 0.0
 - ST6 empty 0.0
 - ST7 empty 0.0
- Memory Dump (M1-M5):**

Address	Hex dump	ASCII		0019FF74	769B0419	RETURN to kernel32.769B0419
0040B000	00 00 00 00 00 00 00 00		0019FF78	003C2000	
0040B008	02 8D 40 00 00 00 00 00	..@.....		0019FF7C	769B0400	kernel32.BaseThreadInitThunk
0040B010	00 00 00 00 00 00 00 00		0019FF80	0019FFDC	
0040B018	32 13 8B C0 00 8D 40 00	2.....@.		0019FF84	775E6620	RETURN to ntdll.775E6620
0040B020	00 8D 40 00 00 8D 40 00	..@....@.		0019FF88	003C2000	
0040B028	01 8D 40 00 00 00 00 00	..@.....		0019FF8C	ACA90050	
0040B030	00 00 00 00 84 1F 40 00@.		0019FF90	00000000	
				0019FF94	00000000	
- Registers (R):**
 - FST 0000 Cond 0 0 0 0 Err 0 0 0 0
 - FCW 027F Prec NEAR,53 Mask 1 1
- Bottom Status Bar:**
 - M1 M2 M3 M4 M5
 - Command: [dropdown]
 - ESP EBP NONE
 - Execute till return (Ctrl+F9) Top Paused

- ⊕ Tiếp tục chuyển chuỗi 8 bytes thành chuỗi 4 bytes và đếm so sánh với **0xb456b480**. Nếu sai thì báo lỗi.

The screenshot shows the assembly view of the debugger. The assembly window displays the following code snippet:

```

00400F18:    53          PUSH EBX
00400F19:    56          PUSH ESI
00400F1A:    8BF0        MOV ESI,ERAX
00400F1C:    83CB FF    OR EBX,FFFFFFFFFF
00400F1F:    8BC6        MOV ERAX,ESI
00400F21:    E8 DAAFFFFF CALL 2_4.00400F00
00400F26:    85C0        TEST ERAX,ERAX
00400F28:    7E 21       JLE SHORT 2_4.00400F4B
00400F29:    C1E0 01000000 MOV EDK,1
00400F2F:    > 33C9      XOR ECX,ECX
00400F31:    8A4C16 FF    MOU CL,BYTE PTR [ESI+EDX-1]
00400F35:    33CB        XOR ECX,EBX
00400F37:    81E1 FF000000 AND ECX,0FF
00400F3D:    C1E8 08      SHR EBX,8
00400F40:    331C8D ACC74 XOR EBX,DWORD PTR [ECX*4+40C7AC]
00400F47:    42          INC EDK
00400F48:    48          DEC ERAX
00400F49:    ^ 75 E4      JNZ SHORT 2_4.00400F2F
00400F4B:    > F7D3      NOT ERAX
00400F4D:    8BC3        MOV ERAX,EBX
00400F50:    . SE         POP ESI
00400F51:    . SB         POP EBX
00400F52:    . C3         RET
00400F54:    8BC0        MOU ERAX,ERAX
00400F54:    . 55         PUSH EBP

```

The registers window shows:

- ERX: 0019FFCC
- ECX: 0040A270 2_4.<ModuleEntryPoint>
- EDX: 0040A270 2_4.<ModuleEntryPoint>
- EBX: 003C2000
- ESP: 0019FF74
- EBP: 0019FF80
- ESI: 0040A270 2_4.<ModuleEntryPoint>
- EDI: 0040A270 2_4.<ModuleEntryPoint>
- EIP: 0040A270 2_4.<ModuleEntryPoint>

The memory dump window shows memory starting at address 00400000. The status bar indicates the program is Paused at 5:37 CH on 08/06/2019.

- ⊕ Sau đó, chương trình xử lý 8 kí tự còn lại. Nếu các kí tự không nằm trong phạm vi từ '**'0' → '9'**' hoặc từ '**'A' → 'F'**' hoặc từ '**'a' → 'f'**' thì báo lỗi.

The screenshot shows the assembly view of the debugger. The assembly window displays the following code snippet:

```

00409F49:    > B840 FC    MOU ECX,DWORD PTR [EBP-4]
00409F4C:    8A4C11 FF    MOU CL,BYTE PTR [ECX+EDX-1]
00409F50:    80C1 D0      ADD CL,0D0
00409F53:    80E9 0A      SUB CL,0A
00409F56:    72 08       JB SHORT 2_4.00409F60
00409F58:    > 80C1 F9    ADD CL,0F9
00409F5B:    80E9 06       SUB CL,6
00409F5E:    > 73 01       JNB SHORT 2_4.00409F61
00409F60:    43          INC EBX
00409F61:    > 42          INC EDK
00409F62:    48          DEC ERAX
00409F63:    ^ 75 E4      JNZ SHORT 2_4.00409F49
00409F65:    > 8845 FC    MOU ERAX,DWORD PTR [EBP-4]
00409F68:    E8 939FFFFF  CALL 2_4.00400F00
00409F6D:    3B0B        CMP EBX,ERAX
00409F6F:    0F94C0      SETE R
00409F72:    88D8        MOV ERAX,ERAX
00409F74:    33C0        XOR ERAX,ERAX
00409F76:    5A          POP EDX
00409F77:    . 59         POP ECX
00409F78:    . 59         POP ECX
00409F79:    64:8910      MOV DWORD PTR FS:[EAX],EDK
00409F7C:    60 919F4000  PUSH 2_4.00409F91
00409F81:    > 8045 FC    LEA EAX,DWORD PTR [EBP-4]

```

The registers window shows:

- ERX: 0019FFCC
- ECX: 0040A270 2_4.<ModuleEntryPoint>
- EDX: 0040A270 2_4.<ModuleEntryPoint>
- EBX: 003C2000
- ESP: 0019FF74
- EBP: 0019FF80
- ESI: 0040A270 2_4.<ModuleEntryPoint>
- EDI: 0040A270 2_4.<ModuleEntryPoint>
- EIP: 0040A270 2_4.<ModuleEntryPoint>

The memory dump window shows memory starting at address 00400000. The status bar indicates the program is Paused at 5:44 CH on 08/06/2019.

- ✚ Tiếp tục, chương trình kiểm tra 2 ký tự cuối cùng có phải là '*ID*' hoặc '*Id*' hay không.

```

00409F98:    .B8EA   CMP EBP,EDX
00409F9A:    .BC 9D01FC1E MOU ESP,IEFCD19A
00409F9C:    .8FC3   POP EBX
00409F9D:    .00C3   ADD BL,AL
00409F9E:    .98     NOP
00409F9F:    .0FC8   BSMPW EAX
00409F9F:    .C3     RET
00409FA0:    .99     NOP
00409FA1:    .55     PUSH EBP
00409FA2:    .8EC    MOU EBP,ESP
00409FA3:    .89C4 C8 ADD ESP,-38
00409FA4:    .53     PUSH EBX
00409FA5:    .56     PUSH ESI
00409FA6:    .39C9   XOR ECX,ECX
00409FA7:    .89AD C8 MOV DWORD PTR [EBP-38],ECX
00409FA8:    .894D CC MOV DWORD PTR [EBP-34],ECX
00409FA9:    .894D EC MOV DWORD PTR [EBP-14],ECX
00409FAA:    .8955 F8 MOV DWORD PTR [EBP-8],EDX
00409FAB:    .8945 FC MOV DWORD PTR [EBP-4],ERX
00409FAC:    .8845 FC MOV ERX,DWORD PTR [EBP-4]
00409FAD:    .E8 83A0FFFF CALL _2_4_0040404C
00409FAC:    .E8 84B5 F8 MOV ERX,DWORD PTR [EBP-8]
00409FAD:    .E8 7B80FFFF CALL _2_4_0040404C
00409FAD:    .33C8   XOR ERX,ERX

```

Local call from 0040A433

Address	Hex dump	ASCII
00408000	00 00 00 00 00 00 00 00
00408008	02 8D 40 00 00 00 00 00
00408010	00 00 00 00 00 00 00 00
00408018	32 13 88 C0 00 8D 40 00@.
00408020	00 8D 40 00 00 00 8D 40@.
00408028	01 8D 40 00 00 00 00 00@.
00408030	00 00 00 00 84 1F 40 00@.
00408038	14 21 40 00 94 24 40 00	.+@..\$@.

M1 M2 M3 M4 M5 Command: ESP EBP NONE

00409F98 > 00409F9A: [0x00000002 :- 00000002] bytes | OFFSET: 0x00009398 -> 0x0000939A Top Paused

8:49 CH ENG 08/06/2019

✚ Sau đó, chuyên 8 bytes ký tự này thành chuỗi 4 bytes (Cũng sử dụng hàm chuyên như bước trước đó). Sau đó so sánh với *0xE367DB06*.

```

0040A446:    .A1 4CCF4000 MOU EAX,DWORD PTR [40CF4C1]
0040A448:    .E8 649CFFFF CALL _2_4_004040B4
0040A450:    .8B45 D0 MOU EAX,DWORD PTR [EBP-30]
0040A453:    .E8 C0EAFFFF CALL _2_4_00405F18
0040A458:    .3D 0CDB67E3 CMP EAX,E367D00C
0040A45F:    .75 2C JNZ SHORT _2_4_0040A48B
0040A45F:    .804D CC LEA ECX,DWORD PTR [EBP-34]
0040A462:    .8B15 4CCF4000 MOU EDX,DWORD PTR [40CF4C1]
0040A468:    .B8 D8854000 MOU EAX,_2_4_0040A5D8
0040A46D:    .E8 A6F8FFFF CALL _2_4_00409D18
0040A472:    .8B55 CC MOU EDX,DWORD PTR [EBP-34]
0040A475:    .R1 0CBE4000 MOU EAX,DWORD PTR [40BE0C]
0040A477:    .E8 B59DFFFF CALL _2_4_00404184
0040A47F:    .E8 BC89FFFFFF CALL _2_4_004042E40
0040A484:    .E8 C381FFFF CALL _2_4_00404264C
0040A489:    .v EB 34 JMP SHORT _2_4_0040A4BF
0040A48B:    > R1 0CBE4000 MOU EAX,DWORD PTR [40BE0C]
0040A490:    .B8 34964000 MOU EDX,_2_4_0040A634
0040A495:    .E8 E99CFFFF CAL _2_4_00404184
0040A498:    .E8 A189FFFFFF CAL _2_4_004042E40
0040A49F:    .E8 A801FFFF CALL _2_4_00404264C
0040A4A4:    .v EB 19 JMP SHORT _2_4_0040A4BF
0040A4A6:    > R1 0CBE4000 MOU EAX,DWORD PTR [40BE0C]
0040A4B0:    .B8 34A64000 MOU EDX,_2_4_0040A634

```

ASCII "901DA97900EC?!"

ASCII "Invalid seria"

ASCII "Invalid seria"

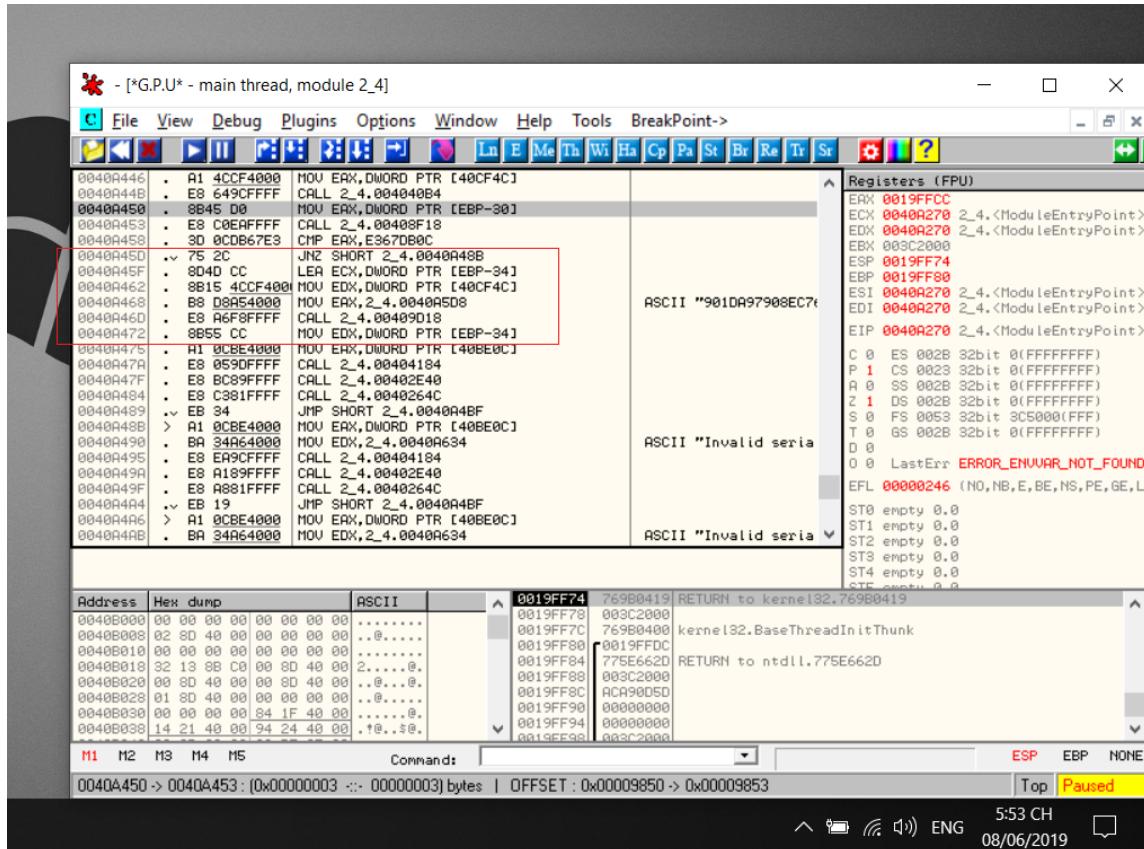
Address	Hex dump	ASCII
00408000	00 00 00 00 00 00 00 00
00408008	02 8D 40 00 00 00 00 00@.
00408010	00 00 00 00 00 00 00 00
00408018	32 13 88 C0 00 8D 40 00@.
00408020	00 8D 40 00 00 00 8D 40@.
00408028	01 8D 40 00 00 00 00 00@.
00408030	00 00 00 00 84 1F 40 00@.
00408038	14 21 40 00 94 24 40 00	.+@..\$@.

M1 M2 M3 M4 M5 Command: ESP EBP NONE

0040A450 > 0040A453: [0x00000003 :- 00000003] bytes | OFFSET: 0x00009850 -> 0x00009853 Top Paused

5:53 CH ENG 08/06/2019

♣ Cuối cùng, chương trình sẽ chạy đến *goodboy*.

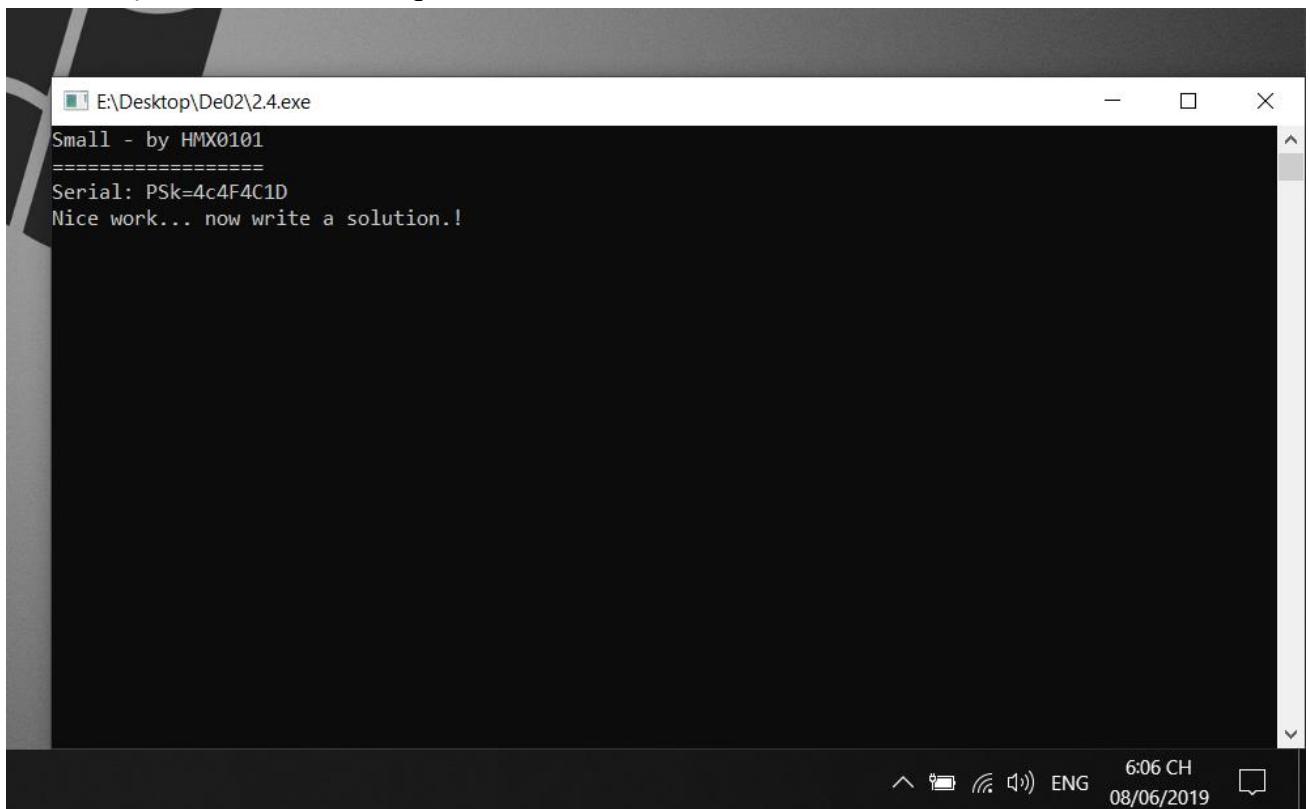


♣ Vì chương trình sử dụng thuật toán băm nên khó có thể truy ngược như bình thường được nên cách duy nhất là sử dụng thuật toán *Bruce Force*.

♣ Các key mà nhóm tìm được:

- Key = ‘PSk=4c4f4c1d’

⊕ Test lại lần nữa, ta có kết quả:

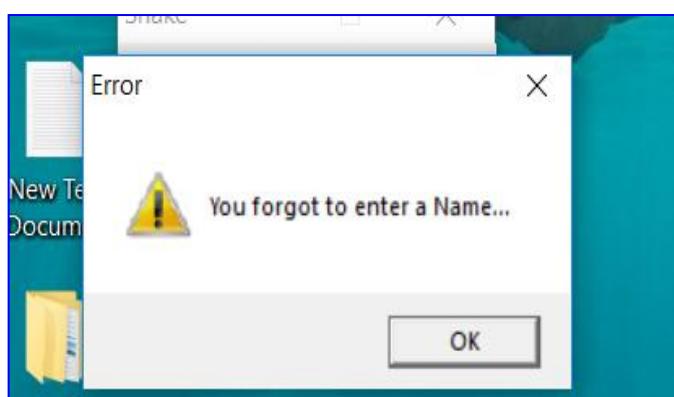


Bài 2.5

- Đối với bài này ta cần nhập cả NAME lẫn SERIAL . Trong đó NAME là ta tự nhập và từ NAME giải mã ra được SERIAL .
- Kiểm tra NAME :

```
0040109A . 68 E8030000 PUSH 3E8
0040109F . FF35 001F400 PUSH DWORD PTR [401F00]
004010A5 . FF15 5420400 CALL DWORD PTR [<&USER32.GetDlgItemTextA]
004010AB . 85C0 TEST EAX,EAX
004010AD . 75 1C JNZ SHORT 2_5.004010CB
004010B0 . 48 EC PUSH ECX
```

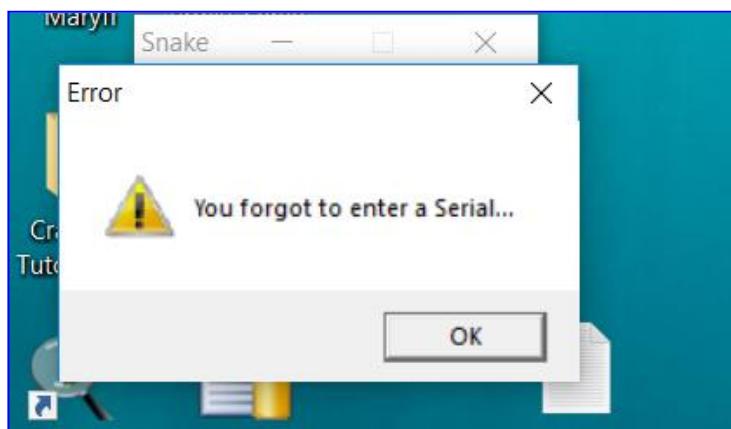
Dòng 004010A5 gọi hàm để kiểm tra NAME đã nhập chưa . Nếu tồn tại thì EAX trả về 1 còn rỗng thi trả về 0 và xuất ra thông báo :



- Kiểm tra SERIAL :

```
004010D2 . 68 E9030000 PUSH 3E9
004010D7 . FF35 001F400 PUSH DWORD PTR [401F00]
004010DD . FF15 5420400 CALL DWORD PTR [<&USER32.GetDlgItemTextA]
004010E3 . 85C0 TEST EAX,EAX
004010E5 . 75 1C JNZ SHORT 2_5.00401103
004010E7 . 6A 30 PUSH 30
004010E9 . 68 00144000 PUSH 2_5.00401400
004010EE . 68 44144000 PUSH 2_5.00401444
```

Dòng 004010DD gọi hàm để kiểm tra SERIAL đã nhập chưa . Nếu tồn tại thì EAX trả về 1 còn rỗng thi trả về 0 và xuất ra thông báo :



Nếu EAX trả về 1 là SERIAL tồn tại thì ở dòng 00401103 gọi đến hàm ở dòng 004012AA để kiểm tra SERIAL có bao gồm các kí tự sau không :

0 1 2 3 4 5 6 7 8 9 A B C D E F

CODE :

```

00401200: $ 56      PUSH EAX
00401200: . BE 00174000 PUSH ESI
00401200: > 8B06    MOV ESI,2_5.00401700
00401201: > 8B06    MOV EAX,DWORD PTR [ESI]
00401201: > 8B06    AND DWORD PTR [ESI],0
00401201: > 8B06    ADD ESI,4
00401201: > 8B06    TEST EAX,EAX
00401201: > 75 F4    JNE SHORT 2_5.004012B1
00401200: . BE 001D4000 MOU ESI,2_5.00401D00
00401200: > 8A06    MOU AL,BYTE PTR [ESI]
00401200: > 8A06    TEST AL,AL
00401200: > 74 18    JE SHORT 2_5.004012E0
00401200: > 3C 30    CMP AL,30
00401200: > 72 0C    JB SHORT 2_5.004012D0
00401200: > 3C 3A    CMP AL,3A
00401200: > 72 0D    JB SHORT 2_5.004012D0
00401200: > 3C 41    CMP AL,41
00401200: > 72 04    JB SHORT 2_5.004012D0
00401200: > 3C 47    CMP AL,47
00401200: > 72 05    JB SHORT 2_5.004012D0
00401200: > 5E      POP ESI
00401200: > B8 00    MOU AL,0
00401200: > C3      RET
00401200: > 46      INC ESI
00401200: ^ EB E2    JMP SHORT 2_5.004012C2
Local call from 00401103

```

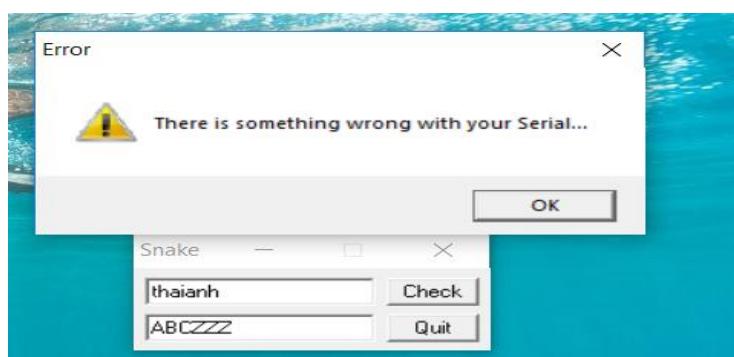
Address	Hex dump	UNICODE
0019FF84	76B98494	RETURN to kernel32.76B98494
0019FF88	0021D000	
0019FF8C	76B98470	kernel32.BaseThreadInitThunk
0019FF90	00C38ED7	
0019FF94	0019FFDC	
0019FF98	77EB41C8	RETURN to ntdll.77EB41C8
0019FF9C	0021D000	
0019FFAC	49314B93	
0019FFA4	00000000	
0019FFAB	00000000	
0019FFAC	0021D000	
0019FFB0	00000000	
0019FFB4	00000000	
0019FFB8	00000000	
0019FFBC	00000000	
0019FFC0	49314B93	
0019FFC4	0019FFA0	
0019FFC8	00000000	

Ở dòng này : 004012DD |> 46 |INC ESI

Tăng ESI lên 1 đơn vị và tương ứng với giá trị ESI bao gồm :

30 ('0'),31 ('1'),32 ('2'),33 ('3'),34 ('4'),35 ('5'),36 ('6'),37 ('7'),38 ('8'),39 ('9'),41 ('A'),42 ('B'),43 ('C'),44 ('D'),45 ('E'),46 ('F').

Nếu nhập ngoài các kí tự trên sẽ xuất thông báo :



Thêm theo ở dòng 00401173 tạo 1 bảng gồm 16x16.

00401185 |. B9 00010000 MOV ECX,100

Tức là 256 bytes - 16x16

00401172	C3	RET	ESI
00401173	\$ 50	PUSH EAX	EBP
00401174	. 51	PUSH ECX	ESI
00401175	. 57	PUSH EDI	EDI
00401176	. BF F01A4000	MOV EDI,2_5.00401AF0	EIP
0040117B	. FC	CLD	C
0040117C	. B9 10000000	MOV ECX,10	P
00401181	. B0 FF	MOU AL,0FF	A
00401183	. F3:AA	REP STOS BYTE PTR ES:[EDI]	Z
00401185	. B9 00010000	MOV ECX,100	S
0040118A	. B0 00	MOU AL,0	T
0040118C	. F3:AA	REP STOS BYTE PTR ES:[EDI]	D
0040118E	. B9 10000000	MOV ECX,10	O
00401193	. B0 FF	MOU AL,0FF	EFL
00401195	. F3:AA	REP STOS BYTE PTR ES:[EDI]	ST0
00401197	. 5F	POP EDI	ST1
00401198	. 59	POP ECX	ST2
00401199	. 58	POP EAX	
0040119A	. C3	RET	

Thêm theo, ở dòng 0040119B dùng để đưa NAME vào trong bảng 16x16

0040119B	\$ 50	PUSH EAX	F
0040119C	. 51	PUSH ECX	E
0040119D	. 52	PUSH EDX	E
0040119E	. 56	PUSH ESI	E
0040119F	. 57	PUSH EDI	E
004011A0	. 33C9	XOR ECX,ECX	E
004011A2	. BE 001E4000	MOV ESI,2_5.00401E00	E
004011A7	. BF 001B4000	MOV EDI,2_5.00401B00	E
004011AC	. 56	PUSH ESI	E
004011AD	. B2 00	MOV DL,0	E
004011AF	> 8A06	MOV AL,BYTE PTR [ESI]	E
004011B1	. 46	INC ESI	C
004011B2	. 02D0	ADD DL,AL	F
004011B4	. 84C0	TEST AL,AL	N
004011B6	.^ 75 F7	JNZ SHORT 2_5.004011AF	S
004011B8	. 5E	POP ESI	T
004011B9	> 0FB606	MOVZX EAX,BYTE PTR [ESI]	C
004011BC	. 32C2	XOR AL,DL	C
004011BE	> 2AD0	SUB DL,AL	C
004011C0	. 800C07 CC	OR BYTE PTR [EDI+EAX],0CC	C
004011C4	.~ 75 04	JNZ SHORT 2_5.004011CA	E
004011C6	. FEC0	DEC DL	E
004011C8	.^ EB F4	JMP SHORT 2_5.004011BE	E
004011CA	> 41	INC ECX	E
004011CB	. 46	INC ESI	E
004011CC	. 803E 00	CMP BYTE PTR [ESI],0	E

004011CC	. 803E 00	CMP BYTE PTR [ESI],0	F
004011CF	.^ 75 E8	JNZ SHORT 2_5.004011B9	E
004011D1	. 890D 041F4000	MOV DWORD PTR [401F04],ECX	E
004011D7	. 32D0	XOR DL,AL	E
004011D9	> 2AC2	SUB AL,DL	E
004011DB	. 803C07 CC	CMP BYTE PTR [EDI+EAX],0CC	E
004011DF	.~ 75 04	JNZ SHORT 2_5.004011E5	E
004011E1	. FEC0	DEC DL	E
004011E3	.^ EB F4	JMP SHORT 2_5.004011D9	E
004011E5	> C60407 DD	MOU BYTE PTR [EDI+EAX],0DD	E
004011E9	. 8AC2	MOV AL,DL	E
004011EB	> 803C07 CC	CMP BYTE PTR [EDI+EAX],0CC	E
004011EF	.~ 74 06	JE SHORT 2_5.004011F7	E
004011F1	. 803C07 DD	CMP BYTE PTR [EDI+EAX],0DD	E
004011F5	.~ 75 04	JNZ SHORT 2_5.004011FB	E
004011F7	> FEC8	DEC AL	E
004011F9	.^ EB F0	JMP SHORT 2_5.004011EB	E
004011FB	> 8D0407	LEA EAX,DWORD PTR [EDI+EAX]	E
004011FE	. C600 99	MOV BYTE PTR [EAX],99	E
00401201	. A3 00174000	MOU DWORD PTR [401700],EAX	E
00401206	. 5F	POP EDI	E
00401207	. 5E	POP ESI	E
00401208	. 5A	POP EDX	E
00401209	. 59	POP ECX	E
0040120A	. 58	POP EAX	E
0040120B	. C3	RET	E

NAME được đưa vô bảng theo thuật toán sau :

1. Tính tổng giá trị các kí tự NAME (hex) lưu vào DL.
2. Lấy từng kí tự lưu vào AL (8 bit cuối của EAX)
3. XOR AL,DL
4. Sau đó SUB DL,AL
5. Đưa CC vào bảng ở địa chỉ của EDI + EAX . EDI lúc này là 00401B00
6. Sau khi đưa hết CC vào bảng XOR DL,AL
7. SUB AL,DL
8. Đưa DD vào ở địa chỉ của EDI + EAX.
9. Tiếp tục MOV AL,DL . Lưu DL vào AL
10. Đưa 99 vào bảng ở địa chỉ EDI + EAX.

Với ví dụ NAME : anh ta có bảng sau :

Address	Hex dump	UNICODE
00401B00	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B20	00 00 99 00 00 00 00 00 00 00 00 00 00 00 00 00	' ..
00401B30	00 00 00 00 00 00 00 00 00 00 CC 00 00 00 00 00 00	..F..
00401B40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B50	00 00 00 00 00 00 CC 00 00 00 00 00 00 00 00 00 00	..F..
00401B60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B80	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 CC	..;
00401B90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BB0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BC0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Trong đó ta có thể hiểu CC ở đây là thức ăn Apple và DD là đích đến cuối cùng và 99 là vị trí Snake xuất phát .

➡ Tiếp theo, ở dòng 0040120C đến 004012A9 là giải mã SERIAL nhập vào để giải mã hướng đi cho Snake ăn hết Apple và đến đích.

00401228 AND AL,3

Nhận biết hướng di chuyển.

0040122A SHR CL,2

Số lần di chuyển cho hướng đi.

0040124B CMP AL,99

Hiểu đơn giản là không được đi lùi lại .

- Mỗi kí tự trong SERIAL biết được hướng di chuyển theo lệnh : AND AL,3

0 – Đi lên
1 – Đi xuống
2 – Sang trái
3 – Sang phải

- Số lần di chuyển : SHR CL,2

Kí tự	Số lần di chuyển
0->3	1
4->7	2
8->B	3
C->F	4

- Tổng hợp ta sẽ có :

Kí tự	Hướng	Số lần
0	0	1
1	1	1
2	2	1
3	3	1

Kí tự	Hướng	Số lần
4	0	2
5	1	2
6	2	2
7	3	2

Kí tự	Hướng	Số lần
8	0	3
9	1	3
A	2	3
B	3	3

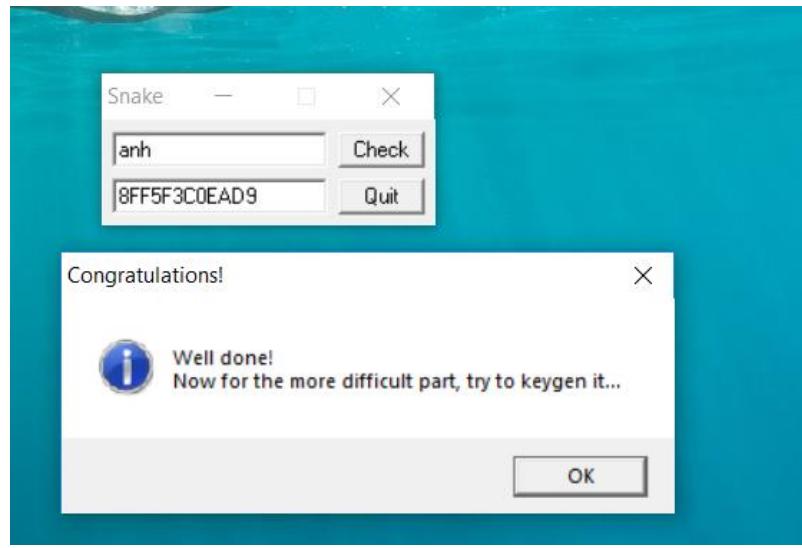
Kí tự	Hướng	Số lần
C	0	4
D	1	4
E	2	4
F	3	4

Ví dụ : NAME : anh

SERIAL : 8FF5F3C0EAD9

MOVE : 00033333331133333000002222222111111

Snake trong quá trình di chuyển ăn CC sẽ dài ra và SERIAL đúng nếu giúp Snake di chuyển ăn hết CC rồi sau đó mới ăn DD thì thành công và xuất ra thông báo :



Tuy nhiên với NAME nhập vào có 5 kí tự giống nhau liền kề thì trong quá trình đưa NAME vào bảng có thể dẫn đến sai vì có vị trí các Apple(CC) trùng nhau nên số lượng CC được đưa vào ít hơn số kí tự của NAME. Ví dụ với NAME : 11111

Address	Hex dump	UNICODE
00401B00	CC 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	Í.....
00401B10	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B20	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B30	00 99 00 00 00 00 00 00 00 00 00 00 00 00 00 00	'.....
00401B40	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B50	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B60	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B70	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B80	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401B90	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BA0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BB0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BC0	00 00 00 00 CC 00 00 00 00 00 00 00 00 00 00 00 00	...Í.....
00401BD0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BE0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401BF0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00401C00	FF	

Chỉ có 2 CC < 5 kí tự được đưa vào bảng .

Demo Keygen for Snake:

Di chuyển Snake ăn hết các Apple rồi sau đó đến đích cuối cùng.

Trong đó : 9 là Snake – C là Táo – D là đích đến

Nên : Di chuyển hợp lý để có được SERIAL đúng và tránh đi vòng tròn tại 1 điểm ăn.

Cách di chuyển : W – lên
S – Xuống
A – Trái
D – Phải

Sau khi ăn thành công :

MỨC ĐỘ HOÀN THIỆN

✓ *Hoàn thành 100% yêu cầu của đồ án*

TÀI LIỆU THAM KHẢO

❖ File hướng dẫn đồ án của giáo viên

<https://courses.fit.hcmus.edu.vn/mod/resource/view.php?id=63010>

❖ Hvaonline

<http://www.hvaonline.net/hvaonline/posts/list/456.hva>