



**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG HCM**  
**KHOA CÔNG NGHỆ THÔNG TIN**  
**MÔN: KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ**  
**LỚP: 17CTT1**

## **BÁO CÁO ĐỒ ÁN 3**

# **CRACKING**

**Sinh viên thực hiện:**

**Trương Nguyễn Anh Hoàng – 1712039**  
**Nguyễn Tân Gia Lợi – 1712086**  
**Trần Thị Tuyết Chung – 1712011**  
**Nguyễn Hoài Thi – 1612648**

**Giáo viên hướng dẫn: Lê Viết Long**

## Mục lục

(Note: Đề =  $(9+1+6+8) \bmod 3 + 1 = 1$ )

<b>1. Chương trình 1_1.exe</b>	<b>3</b>
1.1. Thông tin cơ bản về chương trình:	3
1.2. Tiến hành debug sơ bộ bằng Ollydbg – xác định goodboy/badboy:	4
1.3. Tiến hành debug vào vị trí của mà badboy được reference:	5
1.4. Tiến hành xác định password đúng của chương trình:	6
1.5. Tiến hành truy vết quá trình xử lý password:	8
1.6. Dịch ngược hashed password chuẩn của chương trình:	10
1.7. Thử nghiệm kết quả:	11
<b>2. Chương trình 1_2.exe</b>	<b>12</b>
2.1. Thông tin cơ bản về chương trình:	12
2.2. Tiến hành debug sơ bộ bằng Ollydbg – xác định goodboy/badboy – vị trí lưu trữ input serial:	12
2.3. Tiến hành truy vết từ badboy :	14
2.4. Tiếp tục xét chương trình xử lý part1:	16
2.5. Đề xuất thuật toán mã hoá của chương trình và hướng viết keygen:	18
2.6. Thử nghiệm kết quả:	19
<b>3. Chương trình 1_3.exe</b>	<b>20</b>
3.1. Thông tin cơ bản về chương trình:	20
3.2. Tiến hành debug sơ bộ bằng Ollydbg – xác định goodboy/badboy:	21
3.3. Tiến hành debug vào vị trí của mà badboy và goodboy được reference:	22
3.4. Tiến hành tìm kết phương thức xử lý và kiểm tra của chương trình:	22
3.5. Tiến hành truy vết quá trình xử lý Name:	24
3.6. Truy vết code xử lý serial nhập vào của chương trình:	25
3.7. Đề xuất thuật toán viết keygen:	25

## 1. Chương trình 1\_1.exe

### 1.1. Thông tin cơ bản về chương trình:

- Màn hình console hiển thị giao diện yêu cầu nhập password.
- Nếu nhập sai password sẽ thu được phản hồi (badboy) là **"Nope...try again"**

```

D:\Hoc tap\KTMT & HN\Project_03_crack\Project_03_crack\1\1_1.exe
      8      .oPYo. 8
      8      8 .o8 8
8oPYo. oPYo. 8 .P'8 8 .o .oPYo. odYo.
8      8 8 ` ' 8.d' 8 8oP' 8oooo8 8' `8
8      8 8      8o' 8 8 `b. 8.      8 8
`YooP' 8      `YooP' 8 `o. `Yooo' 8 8
:.....:.....:.....:.....:.....:.....:
:.....:.....:.....:.....:.....:.....:
:.....:.....:.....:.....:.....:.....:
CrackMe #4 (BruteforceMe)

Password : wrong password

Nope... try again.
    
```



### 1.3. Tiến hành debug vào vị trí của mà badboy được reference:

- Kiểm tra vị trí của badboy, ta thấy dòng code print badboy này được jump từ vị trí **004014CB**

```

00401409 | .  EB 0C      JMP SHORT 1_1.004014E7
0040140B | > C70424 87324 MOV DWORD PTR [ESP],1_1.00403287 | ASCII "No Nope... try again."
004014E2 | .  E8 99050000 CALL <JMP.&msvcrt.printf> | printf
004014E7 | > E8 D4040000 CALL <JMP.&msvcrt._getch> | _getch
004014EC | .  B8 00000000 MOV EAX,0
004014F1 | .  C9          LEAVE
004014F2 | .  C3          RET
004014F3 | .  90          NOP
004014F4 | .  90          NOP
004014F5 | .  90          NOP
004014F6 | .  90          NOP
004014F7 | .  90          NOP
004014F8 | .  90          NOP
004014F9 | .  90          NOP
004014FA | .  90          NOP
004014FB | .  90          NOP
004014FC | .  90          NOP
004014FD | .  90          NOP
004014FE | .  90          NOP
004014FF | .  90          NOP
00401500 | .  55          PUSH EBP
00401501 | .  B9 80334000 MOV ECX,1_1.00403380
00401506 | .  89E5        MOV EBP,ESP
00401508 | .  EB 14      JMP SHORT 1_1.0040151E
0040150A | .  8DB6 0000000 LEA ESI,DWORD PTR [ESI]
00401510 | > 8B51 04     MOV EDI,DWORD PTR [ECX+4]
00401513 | .  8B01      MOV EAX,DWORD PTR [ECX]
00403287=1_1.00403287 (ASCII "No Nope... try again.")
Jump from 004014CB

```

- Di chuyển tới dòng 004014CB ta thấy được đoạn code như sau:

```

004014C4 | .  E8 87050000 CALL <JMP.&msvcrt.strcmp> | strcmp
004014C9 | .  85C0      TEST EAX,EAX
004014CB | .  75 0E     JNZ SHORT 1_1.004014DB
004014CD | .  C70424 5C324 MOV DWORD PTR [ESP],1_1.0040325C | ASCII "That's right! Now write a small tut :)"
004014D4 | .  E8 A7050000 CALL <JMP.&msvcrt.printf> | printf
004014D9 | .  EB 0C      JMP SHORT 1_1.004014E7
004014DB | > C70424 87324 MOV DWORD PTR [ESP],1_1.00403287 | ASCII "No Nope... try again."
004014E2 | .  E8 99050000 CALL <JMP.&msvcrt.printf> | printf
004014E7 | > E8 D4040000 CALL <JMP.&msvcrt._getch> | _getch

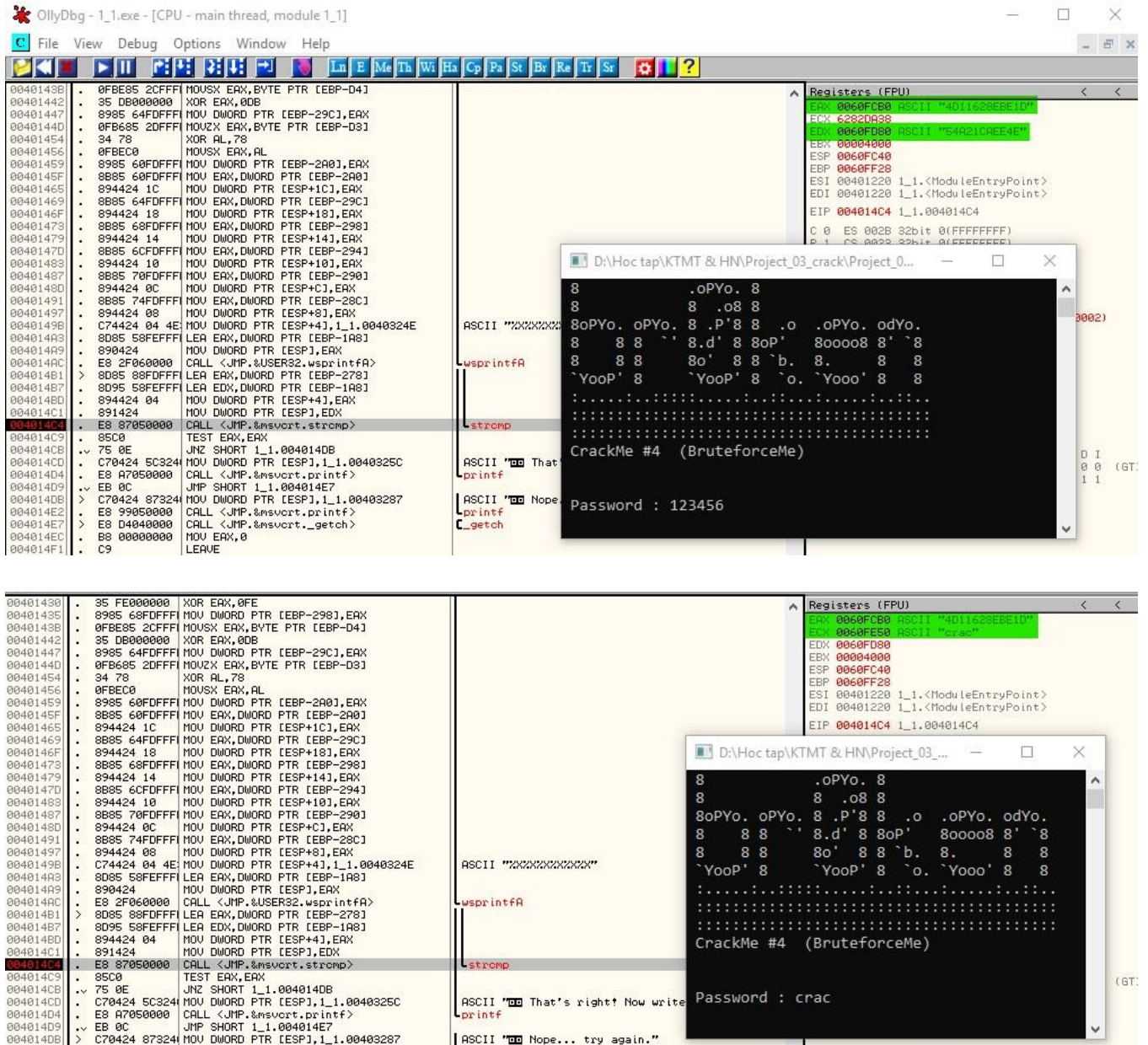
```

- Ta có thể nhận xét về cơ chế in ra kết quả của chương trình thông qua đoạn code:
  - Ở dòng **004014C4** (được tô xanh), tiến hành so sánh 2 chuỗi EAX và EDX (có thể là chuỗi nhập vào và password đúng của chương trình)
  - Ở dòng tiếp theo dựa vào kết quả mà set cờ cho dòng **004014CB**
  - Nếu nhập đúng pass, so sánh chuỗi giống nhau thì bỏ qua dòng **004014CB**
  - Nếu sai jump đến vị trí in ra "badboy" ở **004014DB**



#### 1.4. Tiến hành xác định password đúng của chương trình:

- Như đã biết ở phần trước, dòng **004014C4** (được tô xanh), tiến hành so sánh 2 chuỗi EAX và EDX, một trong 2 thanh ghi này có thể chứa password đúng của chương trình
- Ta tiến hành đặt breakpoint ở hàm so sánh và kiểm tra với các giá trị input khác nhau, là "123456", "crac", "bestcrack", "noobss". Thu được các kết quả như sau:



```

00401469 . 8B85 64DFFF MOV EAX,DMWORD PTR [EBP-29C]
0040146F . 894424 18 MOV DMWORD PTR [ESP+18],EAX
00401473 . 8B85 68DFFF MOV EAX,DMWORD PTR [EBP-298]
00401479 . 894424 14 MOV DMWORD PTR [ESP+14],EAX
0040147D . 8B85 6CFDFFF MOV EAX,DMWORD PTR [EBP-294]
00401483 . 894424 10 MOV DMWORD PTR [ESP+10],EAX
00401487 . 8B85 70DFFF MOV EAX,DMWORD PTR [EBP-290]
0040148D . 894424 0C MOV DMWORD PTR [ESP+C],EAX
00401491 . 8B85 74DFFF MOV EAX,DMWORD PTR [EBP-28C]
00401497 . 894424 08 MOV DMWORD PTR [ESP+8],EAX
0040149B . C74424 04 4E JMP DMWORD PTR [ESP+4],I_1.0040324E
004014A3 . 8085 58FEFF LEA EAX,DMWORD PTR [EBP-1A8]
004014A9 . 890424 MOV DMWORD PTR [ESP],EAX
004014AB . E8 2F060000 CALL <JMP.&USER32.wsprintfA>
004014B1 . 8085 58FEFF LEA EAX,DMWORD PTR [EBP-278]
004014B7 . 8085 58FEFF LEA EDI,DMWORD PTR [EBP-1A8]
004014BD . 894424 04 MOV DMWORD PTR [ESP+4],EAX
004014C1 . 891424 MOV DMWORD PTR [ESP],EDI
004014C4 . E8 87050000 CALL <JMP.&msvcrt.strcmp>
004014C9 . 85C0 TEST EAX,EAX
004014CB . 75 0E JNZ SHORT I_1.004014DB
004014CD . C70424 5C324 MOV DMWORD PTR [ESP],I_1.0040325C
004014D4 . E8 A7050000 CALL <JMP.&msvcrt.printf>
004014D9 . EB 0C JMP SHORT I_1.004014E7
004014DB . C70424 87324 MOV DMWORD PTR [ESP],I_1.00403287
004014E2 . E8 99050000 CALL <JMP.&msvcrt.printf>
004014E7 . E8 D4040000 CALL <JMP.&msvcrt._getch>
004014EC . B8 00000000 MOV EAX,0
004014F1 . C9 LEAVE
004014F2 . C3 RET
004014F3 . 90 NOP
004014F4 . 90 NOP
004014F5 . 90 NOP
004014F6 . 90 NOP
004014F7 . 90 NOP
004014F8 . 90 NOP
004014F9 . 90 NOP

```

Registers (FPU)

```

EAX 0060FCB0 ASCII "4011628EBE1D"
ECX 0060FE50 ASCII "bestcrack"
EDX 0060FD80
EDI 00004000
ESP 0060FC40
EBP 0060FF28
ESI 00401220 I_1.<ModuleEntryPoint>
EDI 00401220 I_1.<ModuleEntryPoint>

```

ASCII "That's right! Now we..."

YooP' 8 .oPYo. 8

CrackMe #4 (BruteforceMe)

Password : bestcrack

```

00401430 . 35 FE000000 XOR EAX,0FE
00401435 . 8985 68DFFF MOV DMWORD PTR [EBP-298],EAX
0040143B . 0FB85 2CFF MOVX EAX,BYTE PTR [EBP-D4]
00401442 . 35 DE000000 XOR EAX,0DE
00401447 . 8985 64DFFF MOV DMWORD PTR [EBP-29C],EAX
0040144D . 0FB85 2DFF MOVX EAX,BYTE PTR [EBP-D3]
00401454 . 34 78 XOR AL,78
00401456 . 0FBEC0 MOVX EAX,AL
00401459 . 8985 68DFFF MOV DMWORD PTR [EBP-2A0],EAX
0040145F . 8B85 68DFFF MOV EAX,DMWORD PTR [EBP-2A0]
00401465 . 894424 1C MOV DMWORD PTR [ESP+1C],EAX
00401469 . 8B85 64DFFF MOV EAX,DMWORD PTR [EBP-29C]
0040146F . 894424 18 MOV DMWORD PTR [ESP+18],EAX
00401473 . 8B85 68DFFF MOV EAX,DMWORD PTR [EBP-298]
00401479 . 894424 14 MOV DMWORD PTR [ESP+14],EAX
0040147D . 8B85 6CFDFFF MOV EAX,DMWORD PTR [EBP-294]
00401483 . 894424 10 MOV DMWORD PTR [ESP+10],EAX
00401487 . 8B85 70DFFF MOV EAX,DMWORD PTR [EBP-290]
0040148D . 894424 0C MOV DMWORD PTR [ESP+C],EAX
00401491 . 8B85 74DFFF MOV EAX,DMWORD PTR [EBP-28C]
00401497 . 894424 08 MOV DMWORD PTR [ESP+8],EAX
0040149B . C74424 04 4E JMP DMWORD PTR [ESP+4],I_1.0040324E
004014A3 . 8085 58FEFF LEA EAX,DMWORD PTR [EBP-1A8]
004014A9 . 890424 MOV DMWORD PTR [ESP],EAX
004014AB . E8 2F060000 CALL <JMP.&USER32.wsprintfA>
004014B1 . 8085 58FEFF LEA EAX,DMWORD PTR [EBP-278]
004014B7 . 8085 58FEFF LEA EDI,DMWORD PTR [EBP-1A8]
004014BD . 894424 04 MOV DMWORD PTR [ESP+4],EAX
004014C1 . 891424 MOV DMWORD PTR [ESP],EDI
004014C4 . E8 87050000 CALL <JMP.&msvcrt.strcmp>
004014C9 . 85C0 TEST EAX,EAX
004014CB . 75 0E JNZ SHORT I_1.004014DB
004014CD . C70424 5C324 MOV DMWORD PTR [ESP],I_1.0040325C
004014D4 . E8 A7050000 CALL <JMP.&msvcrt.printf>
004014D9 . EB 0C JMP SHORT I_1.004014E7
004014DB . C70424 87324 MOV DMWORD PTR [ESP],I_1.00403287
004014E2 . E8 99050000 CALL <JMP.&msvcrt.printf>
004014E7 . E8 D4040000 CALL <JMP.&msvcrt._getch>
004014EC . B8 00000000 MOV EAX,0
004014F1 . C9 LEAVE

```

Registers (FPU)

```

EAX 0060FCB0 ASCII "4011628EBE1D"
ECX 2C5E9A0C
EDX 0060FD80 ASCII "9417709C8B"
EDI 00004000
ESP 0060FC40
EBP 0060FF28
ESI 00401220 I_1.<ModuleEntryPoint>
EDI 00401220 I_1.<ModuleEntryPoint>
EIP 004014C4 I_1.004014C4
C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)

```

ASCII "That's right! Now we..."

YooP' 8 .oPYo. 8

CrackMe #4 (BruteforceMe)

Password : noobss

- Ta có thể kết luận như sau:

- Thanh ghi EAX chứa password đúng của chương trình (nhưng đã bị xử lý)
- Nếu chiều dài password != 6 thì sẽ không được xử lý (chắc chắn cho ra badboy) vậy password có chiều dài là 6
- Nếu chiều dài password bằng 6 thì xử lý theo cách của chương trình rồi so sánh với mật khẩu chuẩn đã được mã hoá

### 1.5. Tiến hành truy vết quá trình xử lý password:

- Ta tiến hành kiểm tra và thấy được, mật khẩu đã bị xử lý (hashed pass) của chương trình lưu ở ô nhớ **[EBP-278]** còn input sau khi được xử lý lưu ở **[EBP- 1A8]**.

00401481	>	8085 88FDFFF	LEA EAX,DWORD PTR [EBP-278]	CALL <JMP.&USER32.wsprintf>
00401487	.	8095 58FEFFF	LEA EDX,DWORD PTR [EBP-1A8]	
0040148D	.	894424 04	MOV DWORD PTR [ESP+4],EAX	
004014C1	.	891424	MOV DWORD PTR [ESP],EDX	
004014C4	.	E8 87050000	CALL <JMP.&msvcrt.strcmp>	strcmp
004014C9	.	85C0	TEST EAX,EAX	
004014CB	.	75 0E	JNZ SHORT 1.004014DB	

- Truy vết đối với hashed password (**EAX/[EBP-278]**) với ô nhớ **[EBP-278]** ta thấy hashed password này được khởi tạo trực tiếp, không có gì để khai thác.

004012C3	.	A1 00304000	MOV EAX,DWORD PTR [403000]	
004012C8	.	8985 88FDFFF	MOV DWORD PTR [EBP-278],EAX	
004012CE	.	A1 04304000	MOV EAX,DWORD PTR [403004]	
004012D3	.	8985 8CFDFFF	MOV DWORD PTR [EBP-274],EAX	
004012D9	.	A1 08304000	MOV EAX,DWORD PTR [403008]	
004012DE	.	8985 90FDFFF	MOV DWORD PTR [EBP-270],EAX	
004012E4	.	0FB605 0C304	MOVZX EAX,BYTE PTR [40300C]	
004012EB	.	8885 94FDFFF	MOV BYTE PTR [EBP-26C],AL	
004012F1	.	8D95 95FDFFF	LEA EDX,DWORD PTR [EBP-26B]	
004012F7	.	B8 BA000000	MOV EAX,0BA	
004012FC	.	894424 08	MOV DWORD PTR [ESP+8],EAX	
00401300	.	C74424 04 00	MOV DWORD PTR [ESP+4],0	
00401308	.	891424	MOV DWORD PTR [ESP],EDX	
0040130B	.	E8 30070000	CALL <JMP.&msvcrt.memset>	

- Ta tiến hành truy vết xem input của chúng ta bị xử lý như thế nào để dịch ngược lại hashed password. Ta tìm kiếm **[EBP-1A8]**:

004013EA	.	33F8 06	CMP EAX,6	
004013ED	.	0F85 BE000000	JNZ 1.004014B1	
004013F3	.	0FB685 28FFF	MOVZX EAX,BYTE PTR [EBP-D8]	
004013FA	.	34 34	XOR AL,34	
004013FC	.	0FBEC0	MOVZX EAX,AL	
004013FF	.	8985 74FDFFF	MOV DWORD PTR [EBP-28C],EAX	
00401405	.	0FB685 29FFF	MOVZX EAX,BYTE PTR [EBP-D7]	
0040140C	.	34 78	XOR AL,78	
0040140E	.	0FBEC0	MOVZX EAX,AL	
00401411	.	8985 70FDFFF	MOV DWORD PTR [EBP-290],EAX	
00401417	.	0FB685 2AFF	MOVZX EAX,BYTE PTR [EBP-D6]	
0040141E	.	34 12	XOR AL,12	
00401420	.	0FBEC0	MOVZX EAX,AL	
00401423	.	8985 6CFDFFF	MOV DWORD PTR [EBP-294],EAX	
00401429	.	0FB685 2BFFF	MOVZX EAX,BYTE PTR [EBP-D5]	
00401430	.	35 FE000000	XOR EAX,0FE	
00401435	.	8985 68FDFFF	MOV DWORD PTR [EBP-298],EAX	
0040143B	.	0FB685 2CFFF	MOVZX EAX,BYTE PTR [EBP-D4]	
00401442	.	35 D6000000	XOR EAX,0DB	
00401447	.	8985 64FDFFF	MOV DWORD PTR [EBP-29C],EAX	
0040144D	.	0FB685 2DFFF	MOVZX EAX,BYTE PTR [EBP-D3]	
00401454	.	34 78	XOR AL,78	
00401456	.	0FBEC0	MOVZX EAX,AL	
00401459	.	8985 60FDFFF	MOV DWORD PTR [EBP-2A0],EAX	
0040145F	.	8B85 60FDFFF	MOV EAX,DWORD PTR [EBP-2A0]	
00401465	.	894424 1C	MOV DWORD PTR [ESP+1C],EAX	
00401469	.	8B85 64FDFFF	MOV EAX,DWORD PTR [EBP-29C]	
0040146F	.	894424 18	MOV DWORD PTR [ESP+18],EAX	
00401473	.	8B85 68FDFFF	MOV EAX,DWORD PTR [EBP-298]	
00401479	.	894424 14	MOV DWORD PTR [ESP+14],EAX	
0040147D	.	8B85 6CFDFFF	MOV EAX,DWORD PTR [EBP-294]	
00401483	.	894424 10	MOV DWORD PTR [ESP+10],EAX	
00401487	.	8B85 70FDFFF	MOV EAX,DWORD PTR [EBP-290]	
0040148D	.	894424 0C	MOV DWORD PTR [ESP+C],EAX	
00401491	.	8B85 74FDFFF	MOV EAX,DWORD PTR [EBP-28C]	
00401497	.	894424 08	MOV DWORD PTR [ESP+8],EAX	
0040149B	.	C74424 04 4E	MOV DWORD PTR [ESP+4],1.0040324E	
004014A3	.	8085 58FEFFF	LEA EAX,DWORD PTR [EBP-1A8]	



- Đoạn code ở trong khung đỏ là đưa dữ liệu vào stack (có đúng 6 lần pull vào stack), không phải đoạn code xử lý, ta có thể thấy đoạn code xử lý ngay ở phía trên.
- Ta dễ thấy dòng **004013EA** có so sánh strlen với 6. Nếu khác thì jump đến dòng **004014B1** mà bỏ qua những dòng code xử lý. Đây chính là đoạn code kiểm tra xem input có đúng 6 ký tự hay không.
- Ta xem xét đoạn code từ **004013ED-00401456**. Đây là 6 lần liên tiếp thực hiện chu trình như sau:
  - Lấy từng kí tự của input ra
  - XORs kí tự đó với giá trị cho trước
  - Lưu kết quả vào biến tạm

```

004013F3 . 0FB685 28FFFF MOVZX EAX, BYTE PTR [EBP-D8]
004013FA . 34 34 XOR AL, 34
004013FC . 0FBEC0 MOVZX EAX, AL
004013FF . 8985 74FDFFF MOV DWORD PTR [EBP-28C], EAX
00401405 . 0FB685 29FFFF MOVZX EAX, BYTE PTR [EBP-D7]
0040140C . 34 78 XOR AL, 78
0040140E . 0FBEC0 MOVZX EAX, AL
00401411 . 8985 70FDFFF MOV DWORD PTR [EBP-290], EAX
00401417 . 0FB685 2AFFFF MOVZX EAX, BYTE PTR [EBP-D6]
0040141E . 34 12 XOR AL, 12
00401420 . 0FBEC0 MOVZX EAX, AL
00401423 . 8985 6CFDFFF MOV DWORD PTR [EBP-294], EAX
00401429 . 0FBE85 2BFFFF MOVZX EAX, BYTE PTR [EBP-D5]
00401430 . 35 FE000000 XOR EAX, 0FE
00401435 . 8985 68FDFFF MOV DWORD PTR [EBP-298], EAX
0040143B . 0FBE85 2CFFFF MOVZX EAX, BYTE PTR [EBP-D4]
00401442 . 35 DB000000 XOR EAX, 0DB
00401447 . 8985 64FDFFF MOV DWORD PTR [EBP-29C], EAX
0040144D . 0FB685 2DFFFF MOVZX EAX, BYTE PTR [EBP-D3]
00401454 . 34 78 XOR AL, 78
00401456 . 0FBEC0 MOVZX EAX, AL
    
```

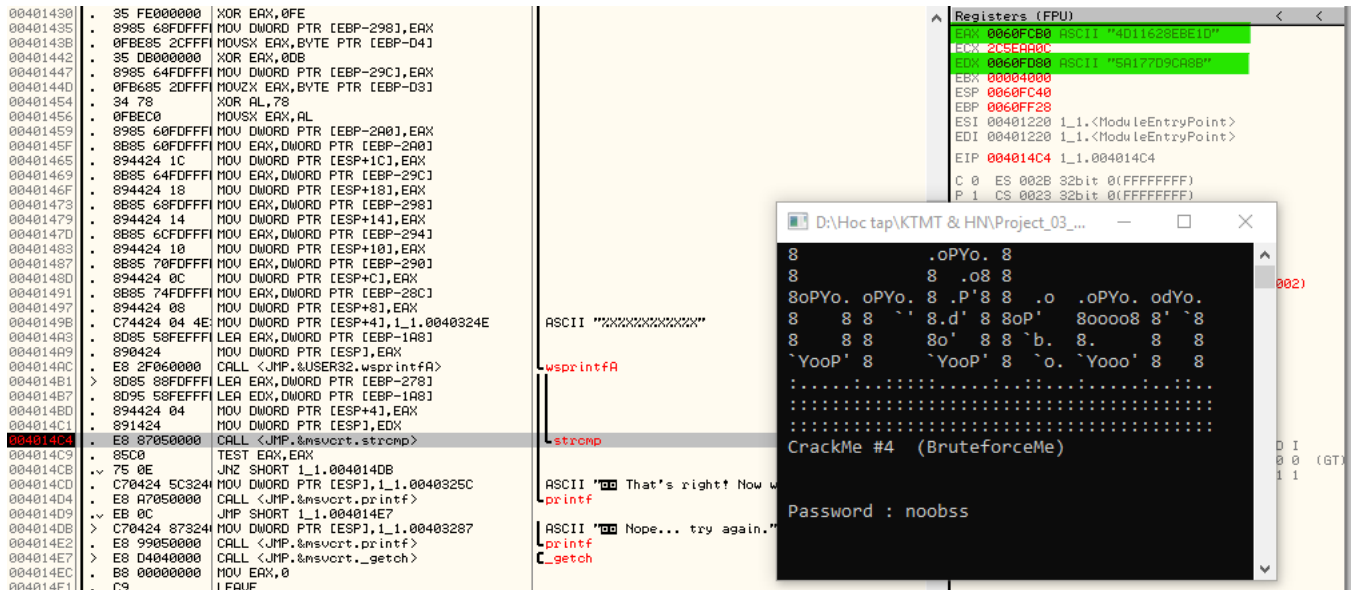
- Phân tích code ta được kết quả như sau:

	input[0]	input[1]	input[2]	input[3]	input[4]	input[5]
ASCII	...	...	...	...	...	...
XOR	34	78	12	0FE	0DB	78

- Xác nhận cách thức mã hoá của chương trình với input "noobss":

	input[0]	input[1]	input[2]	input[3]	input[4]	input[5]
ASCII	n	o	o	b	s	s
XOR	34	78	12	0FE	0DB	78
RES	5A	17	7D	9C	A8	B

- Ta thấy kết quả thu được giống với giá trị của EDX khi breakpoint:



- Vậy cách xử lý trên cũng là cách mã hoá của chương trình.

### 1.6. Dịch ngược hashed password chuẩn của chương trình:

- Sau khi xác định được cách xử lý của chương trình, ta nhận thấy mã hoá dựa trên phép toán XOR. Và phép toán XOR này có tính đối xứng. Tức là:

+ Nếu **"a XOR b = c"** thì **"c XOR b = a"** và **"a XOR c = b"**

- Dựa vào tính đối xứng này, ta tính toán password của chương trình. Vì độ dài hashed password là 12 và độ dài password là 6, nên ta thử lần lượt độ dài của các giá trị được hash và thu được password theo bảng sau

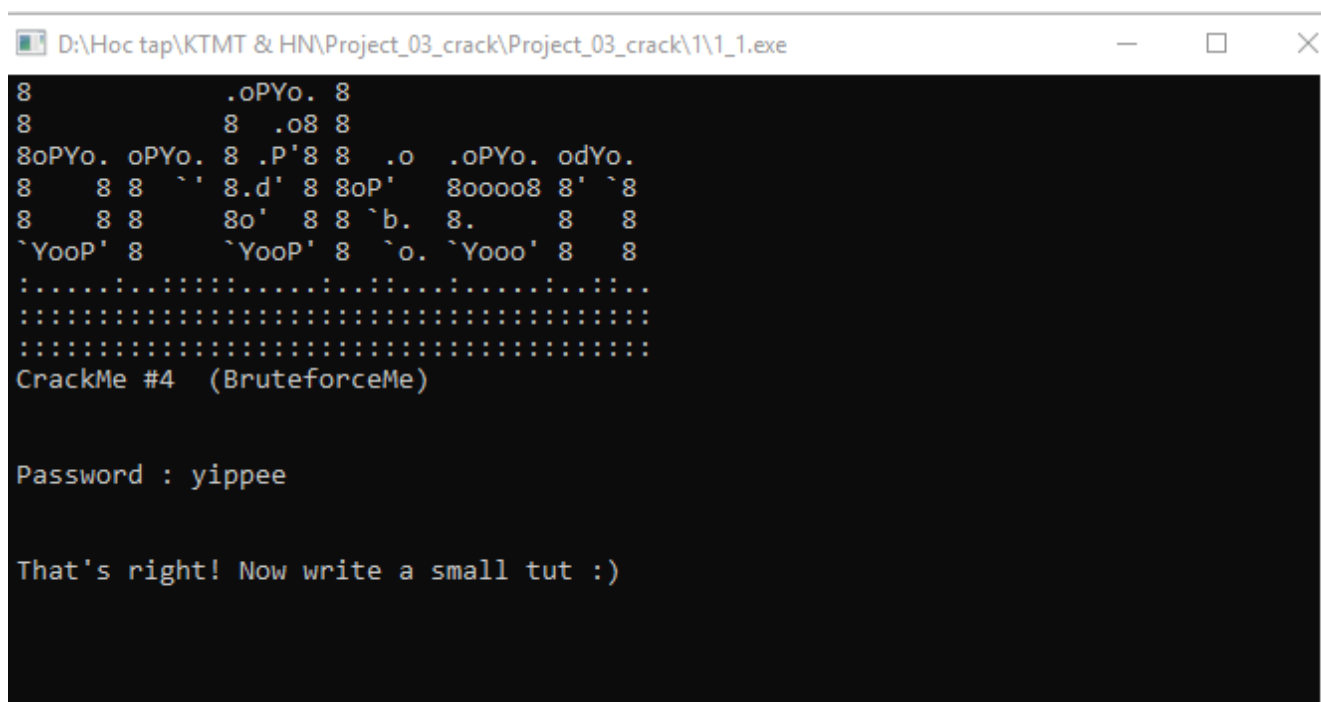
	hash[0]	hash[1]	hash[2]	hash[3]	hash[4]	hash[5]
RES	4D	11	62	8E	BE	1D
XOR	34	78	12	0FE	0DB	78
HEX	79	69	70	70	65	65
ASCII	y	i	p	p	e	e

### 1.7. Thử nghiệm kết quả:

- Ta xác định được password là "yippee" bằng cách dịch ngược hashed password "4D11628EBE1D". Theo bảng phía dưới, với độ dài mỗi ký tự sau khi hash là **2**.

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

- Ta thử nghiệm password thì được kết quả đúng:



```

8      .oPYo. 8
8      8 .o8 8
8oPYo. oPYo. 8 .P'8 8 .o .oPYo. odYo.
8      8 8 `'' 8.d' 8 8oP' 8oooo8 8' `8
8      8 8      8o' 8 8 `b. 8.      8 8
`YooP' 8      `YooP' 8 `o. `Yooo' 8 8
:.....:.....:.....:.....:.....:.....:
:.....:.....:.....:.....:.....:.....:
:.....:.....:.....:.....:.....:.....:
CrackMe #4 (BruteforceMe)

Password : yippee

That's right! Now write a small tut :)

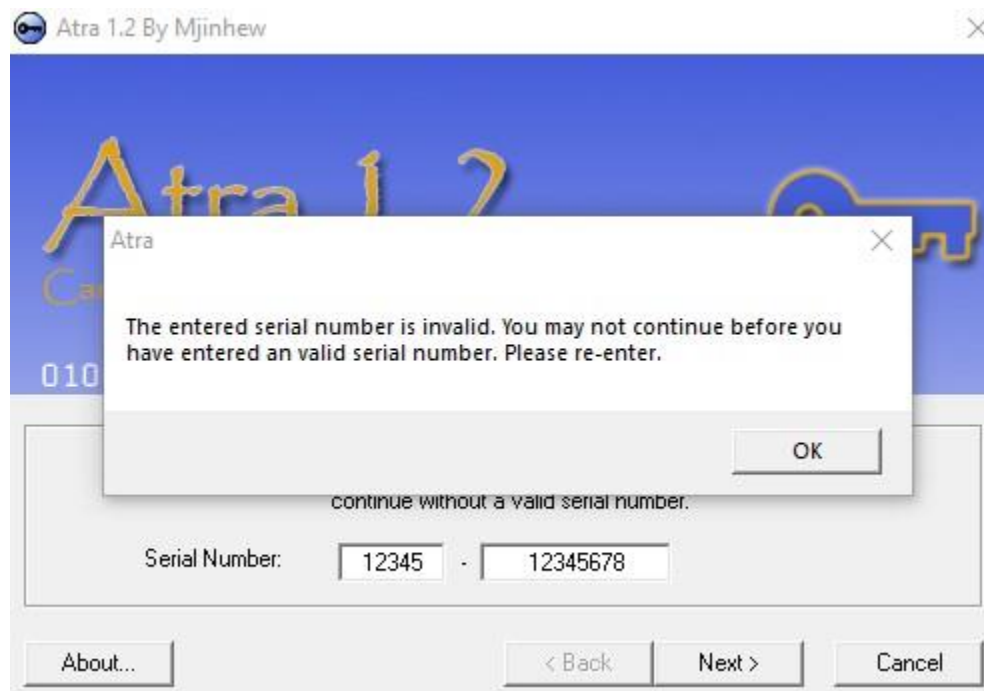
```

- **Vậy password của chương trình là "yippee"**. Vì chương trình chỉ có 1 password duy nhất nên ta sẽ không viết keygen

## 2. Chương trình 1\_2.exe

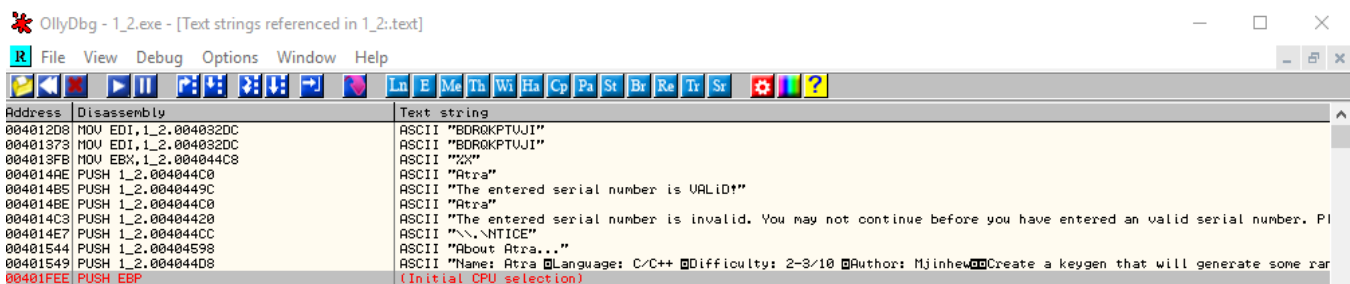
### 2.1. Thông tin cơ bản về chương trình:

- Màn hình hiển thị giao diện yêu cầu nhập serial gồm 2 part:
  - Part 1 có 5 ký tự
  - Part 2 có 8 ký tự
- Nếu nhập sai serial sẽ thu được phản hồi (badboy) là **"The entered ... re-enter"**



### 2.2. Tiến hành debug sơ bộ bằng Ollydbg – xác định goodboy/badboy – vị trí lưu trữ input serial:

- Ta sẽ tiến hành tìm tất cả text string được reference trong chương trình, kết quả thu được như sau



- Để thấy được message khi nhập pass đúng (goodboy) ở đây là **"The entered serial number is VALiD "**.
- Ta có thể nhận thấy truy vết từ đoạn code so sánh để xuất ra goodboy và badboy không thu được kết quả gì đáng kể. Ta tiến hành tìm kết chuỗi serial nhập vào
- Tiến hành đánh dấu và đặt breakpoint tại các dòng có lệnh liên quan tới xử lý chuỗi ta thu được địa chỉ lưu chuỗi serial input và vị trí lấy input. Cụ thể:
  - Part 1 được lấy vào tại dòng **00401292** với địa chỉ không cố định
  - Part 2 được lấy vào tại dòng **004012A5** với địa chỉ không cố định





### 2.3. Tiến hành truy vết từ badboy :

Ta tiến hành kiểm tra tại vị trí in ra badboy, ta thấy vị trí in ra badboy được jump từ rất nhiều dòng khác nhau. Cụ thể là **00401325**, **0040136A**, **004013A8**, **004013D1**, **004013DB**. Đây chính là những điều kiện cần của chương trình, nếu không thoả mãn, xuất ra badboy. Ta sẽ tiến hành kiểm tra các hàm này.

004014AC	. 6A 00	PUSH 0	
004014AE	. 68 C0444000	PUSH 1_2.004044C0	ASCII "Atra"
004014B3	✓ 75 0E	JNZ SHORT 1_2.004014C3	
004014B5	. 68 9C444000	PUSH 1_2.0040449C	ASCII "The entered serial number is UF
004014B7	. 68 00	JMP SHORT 1_2.004014C0	
004014BC	> 6A 00	PUSH 0	
004014BE	. 68 C0444000	PUSH 1_2.004044C0	ASCII "Atra"
004014C3	> 68 20444000	PUSH 1_2.00404420	ASCII "The entered serial number is n
004014C8	> 8B4D F8	MOV ECX,DWORD PTR [EBP-8]	
004014CB	. E8 9E0A0000	CALL <JMP.&MFC42.#4224>	
004014D0	> 5F	POP EDI	
004014D1	. 5E	POP ESI	
004014D2	. 5B	POP EBX	
004014D3	. C9	LEAVE	
004014D4	. C3	RET	
004014D5	§ 6A 00	PUSH 0	hTemplateFile = NULL
004014D7	. 68 80000000	PUSH 80	Attributes = NORMAL
004014DC	. 6A 03	PUSH 3	Mode = OPEN_EXISTING
004014DE	. 6A 00	PUSH 0	pSecurity = NULL
004014E0	. 6A 03	PUSH 3	ShareMode = FILE_SHARE_READ FILE_SHARE
004014E2	. 68 00000000	PUSH C0000000	Access = GENERIC_READ GENERIC_WRITE
004014E7	. 68 CC444000	PUSH 1_2.004044CC	FileName = "\\.\NTICE"
004014EC	. FF15 0C304000	CALL DWORD PTR [<&KERNEL32.CreateFileA>	CreateFileA
004014F2	. 83F8 FF	CMP EAX,-1	
004014F5	✓ 74 0B	JE SHORT 1_2.00401502	
Jumps from 00401325, 0040136A, 004013A8, 004013D1, 004013DB			

- Di chuyển tới dòng **0040136A** ta thấy được đoạn code như sau:

0040135E	. 8975 FC	MOV DWORD PTR [EBP-4],ESI	
00401361	. 50	PUSH EAX	
00401362	. E8 750C0000	CALL <JMP.&MSUCRT.strlen>	strlen
00401367	. 85C0	TEST EAX,EAX	
00401369	. 59	POP ECX	
0040136A	✓ 0F86 4C010000	JBE 1_2.004014BC	

+ Đây là đoạn code kiểm tra nếu len của part 1 = 0 thì báo lỗi

- Di chuyển tới dòng **004013D1** ta thấy được đoạn code như sau:

00401378	> 57	PUSH EDI	[S
00401379	. 33F6	XOR ESI,ESI	
0040137B	. E8 5C0C0000	CALL <JMP.&MSUCRT.strlen>	[strlen
00401380	. 85C0	TEST EAX,EAX	
00401382	. 59	POP ECX	
00401383	.~ 76 35	JBE SHORT 1_2.004013BA	
00401385	> 8A03	MOV AL,BYTE PTR [EBX]	
00401387	. 3A86 DC324000	CMP AL,BYTE PTR [ESI+4032DC]	
0040138D	.~ 75 05	JNZ SHORT 1_2.00401394	
0040138F	. FF45 F0	INC DWORD PTR [EBP-10]	
00401392	.~ EB 1A	JMP SHORT 1_2.004013AE	
00401394	> 3A86 D0324000	CMP AL,BYTE PTR [ESI+4032D0]	
0040139A	.~ 75 05	JNZ SHORT 1_2.004013A1	
0040139C	. FF45 EC	INC DWORD PTR [EBP-14]	
0040139F	.~ EB 0D	JMP SHORT 1_2.004013AE	
004013A1	> FF45 F4	INC DWORD PTR [EBP-C]	
004013A4	. 837D F4 2F	CMP DWORD PTR [EBP-C],2F	
004013A8	.~ 0F87 0E010000	JA 1_2.004014BC	
004013AE	> 57	PUSH EDI	[S
004013AF	. 46	INC ESI	
004013B0	. E8 270C0000	CALL <JMP.&MSUCRT.strlen>	[strlen
004013B5	. 3BF0	CMP ESI,EAX	
004013B7	. 59	POP ECX	
004013B8	.^ 72 CB	JB SHORT 1_2.004013B5	
004013BA	> FF45 FC	INC DWORD PTR [EBP-4]	
004013BD	. 8D45 E0	LEA EAX,DWORD PTR [EBP-20]	
004013C0	. 50	PUSH EAX	
004013C1	. 43	INC EBX	[S
004013C2	. E8 150C0000	CALL <JMP.&MSUCRT.strlen>	[strlen
004013C7	. 3945 FC	CMP DWORD PTR [EBP-4],EAX	
004013CA	. 59	POP ECX	
004013CB	.^ 72 AB	JB SHORT 1_2.00401378	
004013CD	. 837D F0 03	CMP DWORD PTR [EBP-10],3	
004013D1	.~ 0F85 E5000000	JNZ 1_2.004014BC	

+ Đoạn code này có tác dụng kiểm tra xem **[EBP-10]** có bằng 3 không, với **[EBP-10]** là số ký tự của part 1 mà xuất hiện trong chuỗi **"BDRQKPTVJI"**. Đoạn kiểm tra này được thực hiện từ dòng **00401385** đến dòng **004013B8**

+ Kiểm tra tương tự tại lệnh **004013DB** ta thấy đoạn code ở đây so sánh **[EBP-14]** với 2. với **[EBP-14]** là số ký tự của part 1 mà xuất hiện trong chuỗi **"0123456789"**. Đoạn kiểm tra này cũng được thực hiện từ dòng **00401385** đến dòng **004013B8**

- Ta có thể nhận xét về điều kiện để được chấp nhận của part1:
  - Part 1 có tổng cộng 5 ký tự
  - Trong đó 3 ký tự phải nằm trong chuỗi **"BDRQKPTVJI"**. 2 ký tự phải nằm trong chuỗi **"0123456789"**.
  - Việc này được kiểm tra trong đoạn code từ dòng **00401385** đến dòng **004013B8**. Tối đa lên đến 50 lần kiểm tra (5 ký tự, chuỗi 10 ký tự)

## 2.4. Tiếp tục xét chương trình xử lý part1:

- Ta nhận thấy sau khi kiểm tra điều kiện part 1, nếu đúng thì chương trình không jump về badboy mà di chuyển đến dòng **004013E1**, thực hiện đoạn code đến và jump đến dòng **00401D5E**. Ta tiến hành follow đến dòng này:
- Đoạn code từ dòng **00401D5E** thực chất là hash chuỗi part 1 với thuật toán phức tạp và kết thúc tại dòng **00401D15**. Kết quả thu được là **một số** sau khi xor output của thuật toán hash với nhau
- Tiếp đó, ta quan sát dòng **004013F4**. Đoạn code từ **004013F4** đến **00401405** là tiến hành chuyển kết quả thu được thành hệ hexa ở dạng string.

004013D7	. 837D EC 02	CMP DWORD PTR [EBP-14],2	
004013DB	. 0F85 DB000000	JNZ 1_2.004014BC	
004013E1	. 8D45 E0	LEA EAX,DWORD PTR [EBP-20]	
004013E4	. 50	PUSH EAX	
004013E5	. E8 F20B0000	CALL <JMP.&MSVCRT.strlen>	[s strlen
004013EA	. 50	PUSH EAX	
004013EB	. 8D45 E0	LEA EAX,DWORD PTR [EBP-20]	
004013EE	. 50	PUSH EAX	
004013EF	. E8 6A090000	CALL 1_2.00401D5E	
004013F4	. 8B3D B0314000	MOV EDI,DWORD PTR [<&MSVCRT.sprintf>]	MSVCRT.sprintf
004013FA	. 50	PUSH EAX	[<%X>
004013FB	. BB C8444000	MOV EBX,1_2.004044C8	ASCII "%X"
00401400	. 8D45 D4	LEA EAX,DWORD PTR [EBP-2C]	format => "%X"
00401403	. 53	PUSH EBX	[s
00401404	. 50	PUSH EAX	sprintf
00401405	. FFD7	CALL EDI	

- Ta tiếp tục trace code đến lệnh CALL 1\_2.0040147 ở dòng **00401429**

0040141B	> 8B45 FC	MOV EAX,DWORD PTR [EBP-4]	
0040141E	. 50	PUSH EAX	
0040141F	. 8D7405 D4	LEA ESI,DWORD PTR [EBP+EAX-2C]	
00401423	. 0FBF4405 D4	MOVSX EAX,BYTE PTR [EBP+EAX-2C]	
00401428	. 50	PUSH EAX	
00401429	. E8 19FEFFFF	CALL 1_2.00401247	
0040142E	. FF45 FC	INC DWORD PTR [EBP-4]	
00401431	. 8B06	MOV BYTE PTR [ESI],AL	
00401433	. 8D45 D4	LEA EAX,DWORD PTR [EBP-2C]	
00401436	. 50	PUSH EAX	
00401437	. E8 A00B0000	CALL <JMP.&MSVCRT.strlen>	[s strlen
0040143C	. 83C4 0C	ADD ESP,0C	
0040143F	. 3945 FC	CMP DWORD PTR [EBP-4],EAX	
00401442	. 72 D7	JB SHORT 1_2.0040141B	

- Dễ thấy đây là một hàm khởi tạo giá trực tiếp, với các giá trị lần lượt thu được qua việc đặt breakpoint là

```

00401247 8B4424 08 MOV EAX,DWORD PTR [ESP+8]
00401248 . 8BC8 MOV ECX,EAX
0040124D . C1E1 02 SHL ECX,2
00401250 . 8B81 E832400 MOV EAX,DWORD PTR [ECX+4032E8]
00401256 . 2B81 2040400 SUB EAX,DWORD PTR [ECX+404020]
0040125C . 334424 04 XOR EAX,DWORD PTR [ESP+4]
00401260 . C3 RET
    
```

{0xA6, 0x16, 0xAF, 0xFD, 0xD4, 0x07, 0x10, 0xF6} (8 lần chạy)

- Từ dòng **0040141B** đến dòng **00401442** tiến hành khởi tạo lần lượt các giá trị như trên và xor lần lượt với các ký tự thu được của chuỗi hexa từ dòng **004013F4**.
- Từ dòng **00401456** đến dòng **0040147D** tiến hành thực hiện thủ tục:
  - Chuỗi đầu vào là S
  - $S[i] = (S[i] \ll i) \mid S[i]$
  - $i < 8$

```

00401456 > 8B45 FC MOV EAX,DWORD PTR [EBP-4]
00401459 . 50 PUSH EAX
0040145A . 8D7405 D4 LEA ESI,DWORD PTR [EBP+EAX-2C]
0040145E . 0FBE4405 D4 MOVSX EAX,BYTE PTR [EBP+EAX-2C]
00401463 . 50 PUSH EAX
00401464 . E8 F8FDFFFF CALL 1_2.00401261
00401469 . FF45 FC INC DWORD PTR [EBP-4]
0040146C . 8B06 MOV BYTE PTR [ESI],AL
0040146E . 8D45 D4 LEA EAX,DWORD PTR [EBP-2C]
00401471 . 50 PUSH EAX
00401472 . E8 650B0000 CALL <JMP.&MSUCRT.strlen>
00401477 . 83C4 0C ADD ESP,0C
0040147A . 3945 FC CMP DWORD PTR [EBP-4],EAX
0040147D . ^ 72 D7 JB SHORT 1_2.00401456
    
```

- Cuối cùng ở dòng **0040148D** thực hiện lệnh **CALL Atra.004010C0**, đoạn code từ dòng **004010C0** thực chất là hash chuỗi S phía trên với thuật toán **CRC32**.
- Đến đây, chương trình mới thực hiện so sánh chuỗi S vừa generate từ part 1 sau hàng loạt biến đổi phức tạp tại với part 2 do người dùng nhập vào

0040148D	. E8 2EFCFFFF	CALL 1_2.004010C0	
00401492	. 50	PUSH EAX	
00401493	. 8D45 C8	LEA EAX,DWORD PTR [EBP-38]	
00401496	. 53	PUSH EBX	
00401497	. 50	PUSH EAX	
00401498	. FFD7	CALL EDI	
0040149A	. 8D45 C8	LEA EAX,DWORD PTR [EBP-38]	
0040149D	. 50	PUSH EAX	
0040149E	. 8D45 BC	LEA EAX,DWORD PTR [EBP-44]	
004014A1	. 50	PUSH EAX	
004014A2	. E8 2F0B0000	CALL <JMP.&MSUCRT.strcmp>	
004014A7	. 83C4 20	ADD ESP,20	
004014AA	. 85C0	TEST EAX,EAX	
004014AC	. 6A 00	PUSH 0	
004014AE	. 68 C0444000	PUSH 1_2.004044C0	ASCII "Atra"
004014B3	~ 75 0E	JNZ SHORT 1_2.004014C3	
004014B5	. 68 9C444000	PUSH 1_2.0040449C	ASCII "The entered serial number is UF"
004014BA	~ EB 0C	JMP SHORT 1_2.004014C8	
004014BC	> 6A 00	PUSH 0	
004014BE	. 68 C0444000	PUSH 1_2.004044C0	ASCII "Atra"
004014C3	> 68 20444000	PUSH 1_2.00404420	ASCII "The entered serial number is in"

## 2.5. Đề xuất thuật toán mã hoá của chương trình và hướng viết keygen

- Sau khi xác định được cách xử lý của chương trình, ta nhận thấy mã hoá dựa trên chu trình như sau:
  - Kiểm tra part 1 có đúng như điều kiện đề ra hay không ?
  - Hash part 1 bằng những biến đổi phức tạp
  - Dem kết quả thu được đưa về dạng hexa string
  - Xor lần lượt ký tự của string thu được với mảng cố định
  - Thực hiện thủ tục phối hợp dịch trái và or
  - Hash lần 2 với thuật toán CRC32
- Dựa trên thuật toán đề xuất nhóm tiến hành viết keygen như sau:
  - Yêu cầu nhập vào part 1 đúng như điều kiện của chương trình
  - Vì thuật toán hash lần 1 quá phức tạp, tiến hành convert từ vị từ asm (push, lea, mov, shl) thành toán tử có sẵn trong ngôn ngữ C. Chèn thẳng đoạn code hash part 1 đã được convert vào trong code C của chương trình
  - Các bước còn lại làm tương tự chương trình để sinh ra part 2 từ part 1



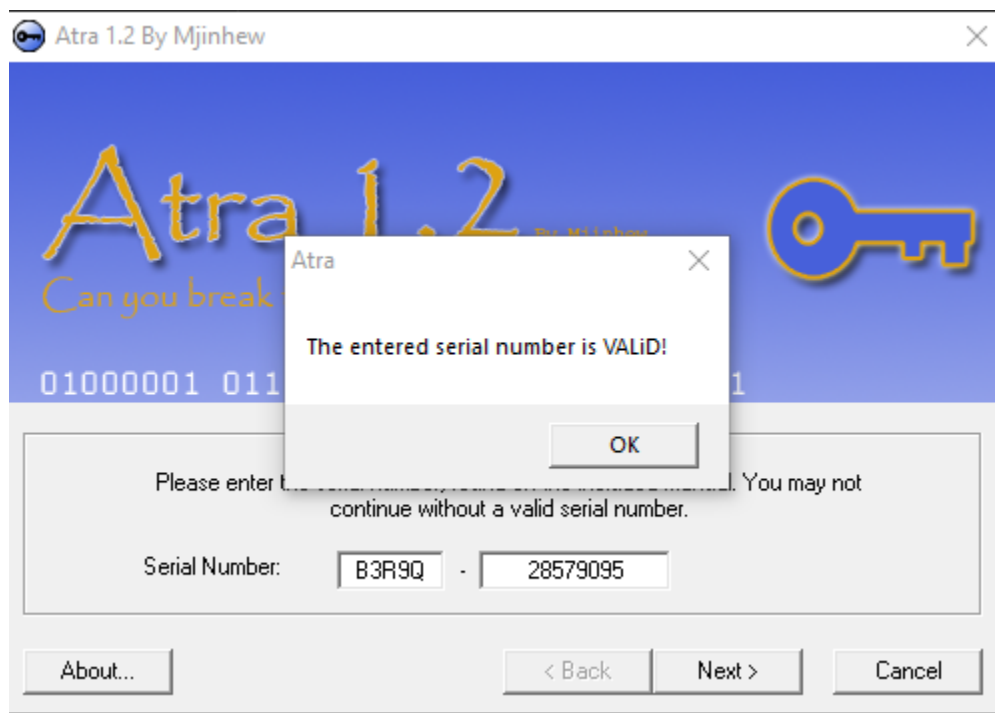
## 2.6. Thử nghiệm kết quả:

- Sau khi hoàn thiện keygen, tiến hành kiểm tra với part 1 = "B3R9Q"
- Kết quả thu được như sau:

```
D:\Hoc tap\KTMT & HN\Project_03_crack\keygen\keygen_1_2\x64\Release\keygen_1_2.exe

Part 1 include:
    3 characters in 'BDRQKPTVJI'
    2 characters in '0123456789'
Enter serial part 1: B3R9Q
    Serial part 2: 28579095
Press any key to continue . . .
```

- Ta thử nghiệm kết quả thu được:

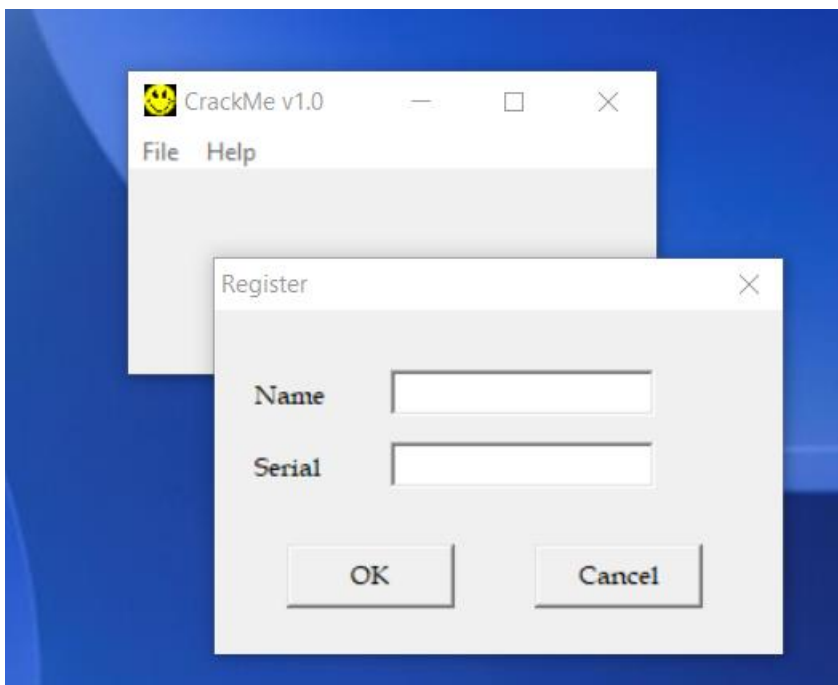


- Vậy keygen đề xuất là keygen chuẩn.

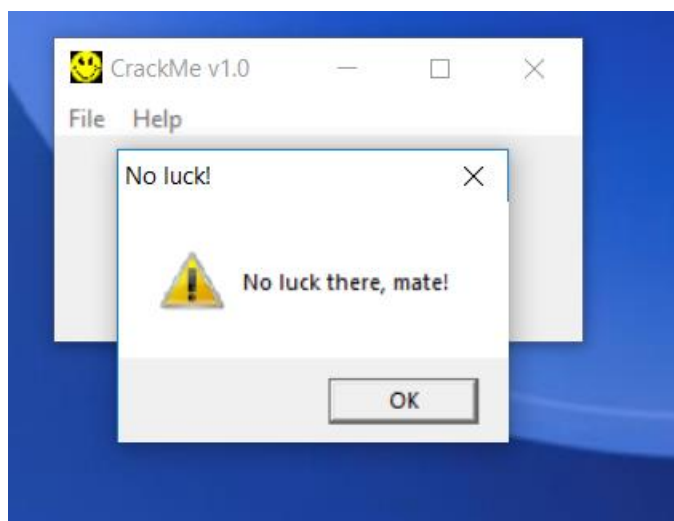
### 3. Chương trình 1\_3.exe

#### 3.1. Thông tin cơ bản về chương trình:

- Giao diện chương trình có nơi nhập serial và name đi kèm trong phần Register.

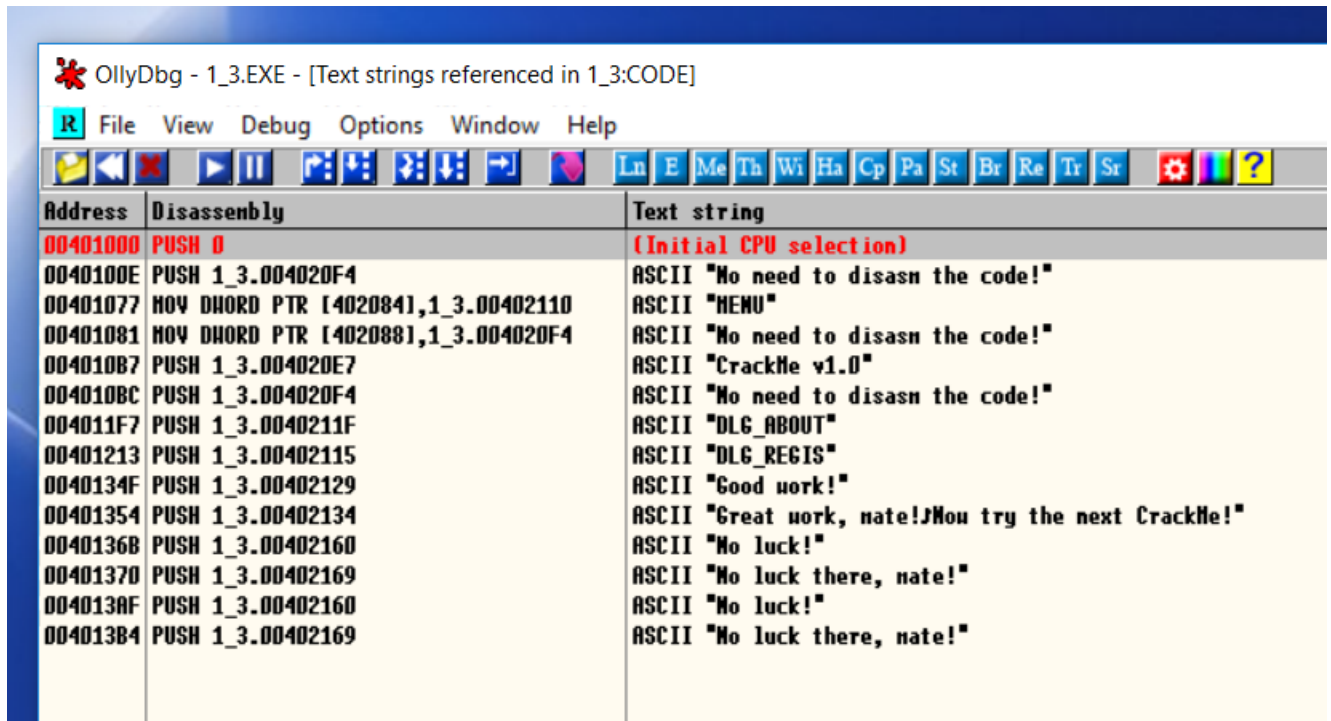


- Nếu nhập sai bộ serial/name sẽ thu được phản hồi (badboy) là **"No luck, mate"**



### 3.2. Tiến hành debug sơ bộ bằng Ollydbg – xác định goodboy/badboy:

- Vì đã có sẵn “badboy” nên ta truy vấn đến đoạn code asm cho ra kết quả sai
- Ta sẽ tiến hành tìm tất cả text string được reference trong chương trình, kết quả thu được như sau



- Dễ thấy được message khi nhập pass đúng (goodboy) ở đây là **“Great work, mate!Now try the next CrackMe”**.
- Và tổng cộng có 2 nơi in ra badboy

### 3.3. Tiến hành debug vào vị trí của mà badboy và goodboy được reference:

- Kiểm tra vị trí của badboy, ta thấy dòng code print badboy window ở vị trí **00401362** được gọi từ dòng **00401245**

00401362	\$ 6A 00	PUSH 0	BeepType = MB_OK
00401364	. E8 A0000000	CALL <JMP.&USER32.MessageBeep>	MessageBeep
00401369	. 6A 30	PUSH 30	Style = MB_OK;MB_ICONEXCLAMATION;MB_APPLMODAL
0040136B	. 68 60214000	PUSH 1_3.00402160	Title = "No luck!"
00401370	. 68 69214000	PUSH 1_3.00402169	Text = "No luck there, nate!"
00401375	. FF75 08	PUSH DWORD PTR [EBP+8]	hOwner
00401378	. E8 BD000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA

- Và ở print badboy window khác vị trí **004013AC** được jump từ dòng **0040138B**

004013AC	> 5E	POP ESI	
004013AD	. 6A 30	PUSH 30	Style = MB_OK;MB_ICONEXCLAMATION;MB_APPLMODAL
004013AF	. 68 60214000	PUSH 1_3.00402160	Title = "No luck!"
004013B4	. 68 69214000	PUSH 1_3.00402169	Text = "No luck there, nate!"
004013B9	. FF75 08	PUSH DWORD PTR [EBP+8]	hOwner
004013BC	. E8 79000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA

- Kiểm tra vị trí của badboy, ta thấy dòng code print goodboy window ở vị trí **0040134D** được gọi từ dòng **0040124C**

0040134D	\$ 6A 30	PUSH 30	Style = MB_OK;MB_ICONEXCLAMATION;MB_APPLMODAL
0040134F	. 68 29214000	PUSH 1_3.00402129	Title = "Good work!"
00401354	. 68 34214000	PUSH 1_3.00402134	Text = "Great work, nate! You try the next CrackMe!"
00401359	. FF75 08	PUSH DWORD PTR [EBP+8]	hOwner
0040135C	. E8 D9000000	CALL <JMP.&USER32.MessageBoxA>	MessageBoxA
00401361	. C3	RET	

- Ta có thể nhận xét về cơ chế in ra kết quả của chương trình thông qua đoạn code:
  - Đoạn code in ra badboy ở **00401362** và goodboy ở **0040134D** ở ngay cạnh nhau, còn đoạn in ra badboy ở **004013AC** nằm tách biệt
  - Vậy rất có thể đoạn code ở **00401362** và **0040134D** nằm ở đoạn code so sánh kết quả đúng để xuất ra phản hồi (điều kiện đủ). Còn đoạn code ở **004013AC** thuộc điều kiện cần của chương trình, nếu không thỏa điều kiện cần nào đó thì cho ra kết quả sai

### 3.4. Tiến hành tìm kết phương thức xử lý và kiểm tra của chương trình:

- Ta tiến hành đặt breakpoint ở trước các vị trí call goodboy và badboy ở điều kiện đủ là dòng **00401245**, **0040124C**. Đó là vị trí dòng **00401241 CMP EAX, EBX**

00401240	- 58	POP EAX
00401241	- 3BC3	CMP EAX,EBX
00401243	- 74 07	JE SHORT 1_3.0040124C
00401245	- E8 18010000	CALL 1_3.00401362
0040124A	- ^ EB 9A	JMP SHORT 1_3.004011E6
0040124C	> E8 FC000000	CALL 1_3.00401340

- Thử với name = "abcde" và serial = "12345" ta thu được kết quả như sau:

00401228	- 68 8E214000	PUSH 1_3.0040218E	ASCII "ABCDE"
0040122D	- E8 4C010000	CALL 1_3.0040137E	
00401232	- 5D	PUSH EAX	
00401233	- 68 7E214000	PUSH 1_3.0040217E	ASCII "12345"
00401238	- E8 9B010000	CALL 1_3.004013D8	
0040123D	- 83C4 04	ADD ESP,4	
00401240	- 58	POP EAX	
00401241	- 3BC3	CMP EAX,EBX	

- Ta có thể kết luận như sau:
  - Name input được lưu ở **0040218E**, Serial input được lưu ở **0040217E**
  - Dòng **0040137E** là vị trí xử lý Name, dòng **004013D8** là vị trí xử lý Serial
  - Sau khi xử lý Name và Serial nhập vào theo thuật toán của chương trình thì so sánh 2 chuỗi vừa biến đổi được, nếu giống nhau gọi in ra goodboy, nếu khác nhau in ra badboy



### 3.5. Tiến hành truy vết quá trình xử lý Name:

- Ta tiến hành kiểm tra tại vị trí **0040137E** và thấy được đoạn code xử lý name như sau:

0040137E	\$ 8B7424 04	MOV ESI,DWORD PTR [ESP+4]
00401382	. 56	PUSH ESI
00401383	> 8A06	MOV AL,BYTE PTR [ESI]
00401385	. 84C0	TEST AL,AL
00401387	~ 74 13	JE SHORT 1_3.0040139C
00401389	. 3C 41	CMP AL,41
0040138B	~ 72 1F	JB SHORT 1_3.004013AC
0040138D	. 3C 5A	CMP AL,5A
0040138F	~ 73 03	JNB SHORT 1_3.00401394
00401391	. 46	INC ESI
00401392	^ EB EF	JMP SHORT 1_3.00401383
00401394	> E8 39000000	CALL 1_3.004013D2
00401399	. 46	INC ESI
0040139A	^ EB E7	JMP SHORT 1_3.00401383
0040139C	> 5E	POP ESI
0040139D	. E8 20000000	CALL 1_3.004013C2
004013A2	. 81F7 78560000	XOR EDI,5678
004013A8	. 8BC7	MOV EAX,EDI
004013AA	~ EB 15	JMP SHORT 1_3.004013C1
004013AC	> 5E	POP ESI
004013AD	. 6A 30	PUSH 30
004013AF	. 68 60214000	PUSH 1_3.00402160
004013B4	. 68 69214000	PUSH 1_3.00402169
004013B9	. FF75 08	PUSH DWORD PTR [EBP+8]
004013BC	> E8 79000000	CALL <JMP.&USER32.MessageBoxA>

- Ta tiến hành truy vết đối với các lệnh trong đoạn code này và kết luận:
  - Từ dòng **00401383** đến **0040139A** tiến hành kiểm tra các ký tự nhập vào có phải là chữ cái (a-z, A-Z) không và upcase toàn bộ các ký tự trong name. Nếu có phần tử không phải là chữ thì nhảy đến badboy thứ 2 **004013AC**
  - Đoạn code ở dòng **004013C2** tiến hành cộng tất cả các ký tự đã được kiểm tra và upcase ở phía trước lại

004013C2	\$ 33FF	XOR EDI,EDI
004013C4	. 33DB	XOR EBX,EBX
004013C6	> 8A1E	MOV BL,BYTE PTR [ESI]
004013C8	. 84DB	TEST BL,BL
004013CA	~ 74 05	JE SHORT 1_3.004013D1
004013CC	. 03FB	ADD EDI,EBX
004013CE	. 46	INC ESI
004013CF	^ EB F5	JMP SHORT 1_3.004013C6
004013D1	> C3	RET

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

### 3.6. Truy vết code xử lý serial nhập vào của chương trình:

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

004013D8	\$	33C0	XOR EAX,EAX
004013DA	.	33FF	XOR EDI,EDI
004013DC	.	33DB	XOR EBX,EBX
004013DE	.	8B7424 04	MOV ESI,DWORD PTR [ESP+4]
004013E2	>	B0 0A	MOV AL,0A
004013E4	.	8A1E	MOV BL,BYTE PTR [ESI]
004013E6	.	84DB	TEST BL,BL
004013E8	~	74 0B	JE SHORT 1_3.004013F5
004013EA	.	80EB 30	SUB BL,30
004013ED	.	0FAFF8	INVL EDI,EAX
004013F0	.	03FB	ADD EDI,EBX
004013F2	.	46	INC ESI
004013F3	^	EB ED	JMP SHORT 1_3.004013E2
004013F5	>	81F7 34120000	XOR EDI,1234
004013FB	.	8BDF	MOV EBX,EDI
004013FD	.	C3	RET

- Trong đó đoạn code từ **004013E2** đến **004013F5** thực hiện convert chuỗi serial key thành số
- Sau đó tiến hành XOR kết quả thu được với 0x1234

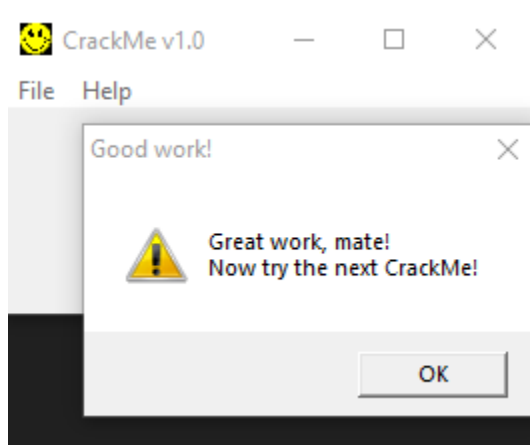
### 3.7. Đề xuất thuật toán viết keygen:

- Ta rõ ràng có thể thấy tương ứng 1 name chỉ có 1 serial và name phải là chữ cái. Và serial có thể được generate từ name bằng cách lợi dụng tính chất của phép XOR. Thay vì đem serial đem xor với 0x1234. Ta tiến hành như sau:
  - Kiểm tra name có toàn chữ cái không và upcase (name)
  - Đem cộng các ký tự của name lại với nhau
  - Đem xor kết quả với 0x5678
  - Đem xor kết quả tiếp với 0x1234
  - Kết quả thu được chính là serial ở dạng số

- Ta thử nghiệm sinh serial bằng keygen với name = "chung" thì được kết quả serial như sau:

```
D:\Hoc tap\KTMT & HN\Project_03_crack\keygen\Keygen_1_3\x64\Release\Keygen_1_3.exe
Name: chung
Serial: 17721
Press any key to continue . . .
```

- Thử với chương trình gốc thu được kết quả:



- Vậy keygen hoạt động chính xác.

### I. Đánh giá thành viên

MSSV	Họ và tên	Đánh giá hoàn thành	Đóng góp
1712039	Trương Nguyễn Anh Hoàng	Tốt	Keygen file 1_2, 1_1, viết báo cáo
1712011	Trần Thị Tuyết Chung	Tốt	File 1_2, 1_3, keygen file 1_2, 1_3
1712086	Nguyễn Tân Gia Lợi	Tốt	File 1_1, 1_3, hỗ trợ viết báo cáo
1612648	Nguyễn Hoài Thi	Tốt	File 1.3, keygen file 1.3

### II. Đánh giá đồ án

File	Độ phức tạp	Đánh giá hoàn thành	Note
1_1	Dễ	100%	
1_2	Rất phức tạp	90%	Hoàn thành keygen
1_3	Bình thường	100%	Hoàn thành keygen

### III. Tài liệu tham khảo

- [1]. Tài liệu giảng dạy môn KTMTvHN GV. Lê Viết Long, ĐH. KHTN
- [2]. [http://crackmes.dreamhosters.com/users/saitob/atra\\_1.2/](http://crackmes.dreamhosters.com/users/saitob/atra_1.2/)
- [3]. <https://rosettacode.org/wiki/CRC-32>
- [4]. <https://whitehat.vn/threads/re2-huong-dan-su-dung-ollydbg.883/>

