

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP HCM**  
**KHOA CÔNG NGHỆ THÔNG TIN**



**BÁO CÁO MÔN HỌC**  
**KIẾN TRÚC MÁY TÍNH VÀ HỢP NGỮ**

**ĐỒ ÁN 3**  
**CRACKING PHẦN MỀM ĐƠN GIẢN (ĐỀ 1)**

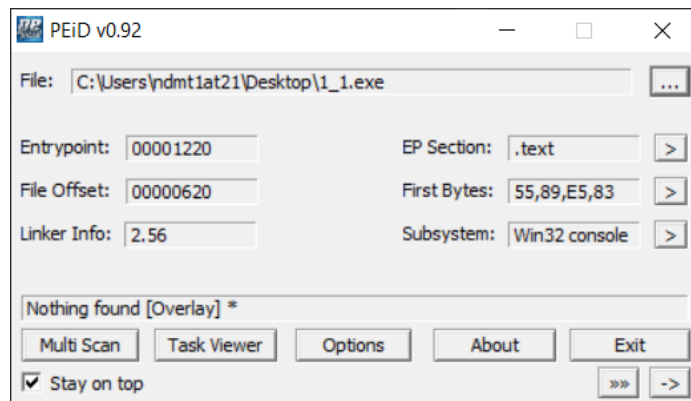
Nhóm sinh viên thực hiện:

Nguyễn Đức Minh Trí	18120612
Nguyễn Ngọc Năng Toàn	18120600
Triệu Trang Tòng	18120602
Trần Ngọc Tịnh	18120597
Nguyễn Tú Toàn	18120601

## 1. Chương trình 1\_1.exe

### 1.1. Kiểm tra chương trình với PEiD

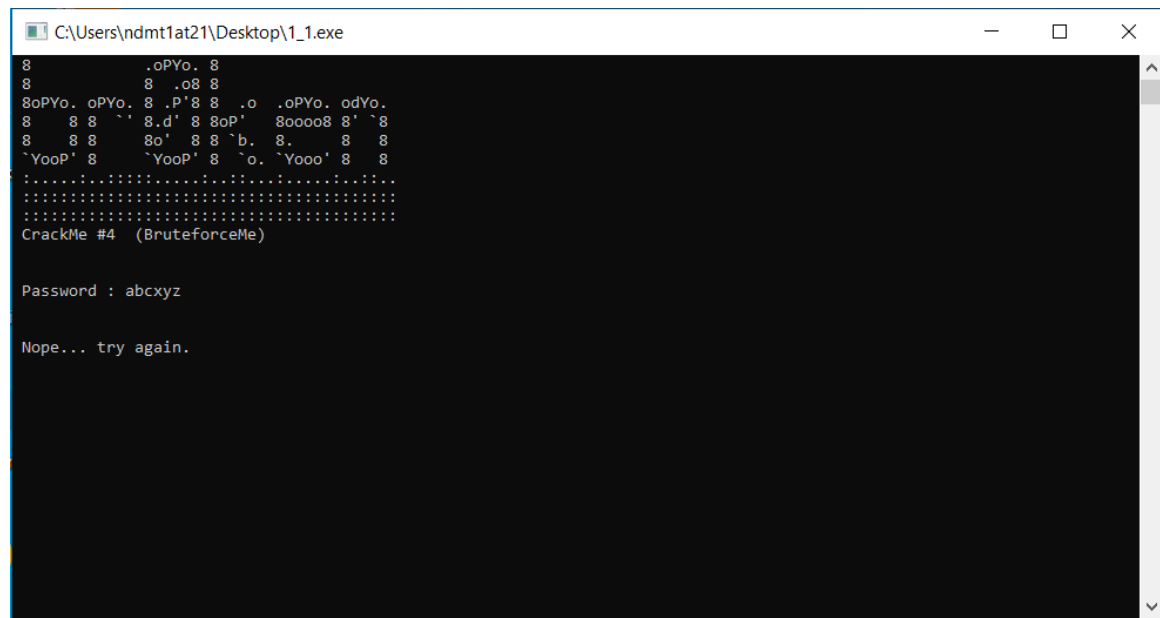
Mở PEiD lên kiểm tra chương trình có pack/protect không, kết quả thu được:



PEiD chưa thể xác định được -> ta tiếp tục thực hiện mở Olly lên.

### 1.2. Giao diện chương trình

Khi nhập một password ngẫu nhiên, chương trình hiển thị như sau:



Lưu ý đến chuỗi thông báo “Nope... try again”

### 1.3. Debug chương trình với OllyDBG

- Ta tìm tất các chuỗi trong chương trình

Address	Disassembly	Text string
00401220	PUSH EBP	(Initial CPU selection)
0040132E	MOV DWORD PTR [ESP],1_1.004030C8	ASCII " 8 .oPYo. 8 "
0040133A	MOV DWORD PTR [ESP],1_1.004030EC	ASCII " 8 8 .o8 8 "
00401346	MOV DWORD PTR [ESP],1_1.00403118	ASCII " 8oPYo. oPYo. 8 .P'8 8 .o .oPYo. odYo. "
00401352	MOV DWORD PTR [ESP],1_1.00403144	ASCII " 8 8 8 '\. 8.d' 8 8oP' 8oooo8 8' `8 "
0040135E	MOV DWORD PTR [ESP],1_1.00403170	ASCII " 8 8 8 8o' 8 8 `b. 8. 8 8 "
0040136A	MOV DWORD PTR [ESP],1_1.0040319C	ASCII " `YooP' 8 `YooP' 8 `o. `Yooo' 8 8 "
00401376	MOV DWORD PTR [ESP],1_1.004031C8	ASCII " : ..... :
00401382	MOV DWORD PTR [ESP],1_1.004031F4	ASCII " : ..... :
0040138E	MOV DWORD PTR [ESP],1_1.004031F4	ASCII " : ..... :
0040139A	MOV DWORD PTR [ESP],1_1.0040321F	ASCII " CrackMe #4 (BruteforceMe)"
004013A6	MOV DWORD PTR [ESP],1_1.0040323C	ASCII " Password : "
004013BC	MOV DWORD PTR [ESP],1_1.0040324B	ASCII "%s"
00401498	MOV DWORD PTR [ESP+4],1_1.0040324E	ASCII "%X%X%X%X%X%X"
004014CD	MOV DWORD PTR [ESP],1_1.0040325C	ASCII " That's right! Now write a small tut :))"
004014DB	MOV DWORD PTR [ESP],1_1.00403287	ASCII " Nope... try again."
004016B7	MOV ECX,1_1.004032C4	ASCII "w32_sharedptr-size == sizeof(W32_EH_SHARED)"
004016C9	MOV DWORD PTR [ESP],1_1.004032F1	ASCII "%s:%u: failed assertion '%s'"
004016D0	MOV EAX,1_1.00403310	ASCII ".../gcc/gcc/config/i386/w32-shared-ptr.c"
004016DE	MOV EAX,1_1.0040333C	ASCII "GetAtomNameA (atom, s, sizeof(s)) != 0"

- Để thấy chuỗi “Nope... try again”, ta đi đến chuỗi. Quan sát thấy có dòng lệnh *TEST EAX, EAX* và *JNZ 004014DB* (Badboy). Đến đây ta đã hình dung rằng: khi không thỏa điều kiện so sánh nhảy đến Badboy, ngược lại thực hiện Goodboy.

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.&userc.printf>	printf
004014D9	.. EB 0C	JMP SHORT 1.1.004014E7	
004014DB	> C70424 87324	MOV DWORD PTR [ESP],1.1.00403287	ASCII " Nope... try again."

- Lướt lên phía trên, ta đặt breakpoint (BP) tại dòng nhập password (abcxyz) và quan sát, debug dần

004013A6	C70424 3C3241	MOV DWORD PTR [ESP],1_1.0040323C	ASCII " Password : "
004013AD	E8 CE060000	CALL <JMP.&svcrnt.printf>	printf
004013B2	8D85 28FFFFF	LEA EAX,DWORD PTR [EBP-D8]	
004013B8	894424 04	MOV DWORD PTR [ESP+4],EAX	
004013BC	C70424 4B3241	MOV DWORD PTR [ESP],1_1.0040324B	ASCII "%s"
004013C3	E8 A8060000	CALL <JMP.&svcrnt scanf>	scanf

- Quan sát phía dưới có điều kiện so sánh. Lúc này EAX chứa strlen của chuỗi vừa nhập, nếu khác 6 chương trình sẽ bỏ qua một loạt các bước xử lý. Ở đây ta nhập abcxyz nên len = 6 nên tiếp tục debug.

004013E9	. 83F8 06	CMP EAX,6
004013ED	.. 0F85 BE000000	JNZ 1.004014B1

- Xuống phía dưới ta thấy đoạn code như sau:

```

004013EA . 83F8 06      CMP EAX,6
004013ED . 0F85 BE000001 JNZ 1_1.004014B1
004013F3 . 0FB685 28FFF1 MOVZX EAX,BYTE PTR [EBP-D8]
004013FA . 34 34        XOR AL,34
004013FC . 0FBEC0       MOVSX EAX,AL
004013FF . 8985 74FDFFF1 MOV DWORD PTR [EBP-28C],EAX
00401405 . 0FB685 29FFF1 MOVZX EAX,BYTE PTR [EBP-D7]
0040140C . 34 78        XOR AL,78
0040140E . 0FBEC0       MOVSX EAX,AL
00401411 . 8985 70FDFFF1 MOV DWORD PTR [EBP-290],EAX
00401417 . 0FB685 2AFFF1 MOVZX EAX,BYTE PTR [EBP-D6]
0040141E . 34 12        XOR AL,12
00401420 . 0FBEC0       MOVSX EAX,AL
00401423 . 8985 6CFDFFF1 MOV DWORD PTR [EBP-294],EAX
00401429 . 0FB685 2BFFF1 MOVZX EAX,BYTE PTR [EBP-D5]
00401430 . 35 FE000000   XOR EAX,0FE
00401435 . 8985 68FDFFF1 MOV DWORD PTR [EBP-298],EAX
0040143B . 0FB685 2CFFF1 MOVZX EAX,BYTE PTR [EBP-D4]
00401442 . 35 DB000000   XOR EAX,0DB
00401447 . 8985 64FDFFF1 MOV DWORD PTR [EBP-29C],EAX
0040144D . 0FB685 2DFFF1 MOVZX EAX,BYTE PTR [EBP-D3]

```

Khi debug, đoạn code này lấy từng ký tự của chuỗi vừa nhập đi XOR với một số. Cụ thể, XOR theo thứ tự sau:

0	1	2	3	4	5
0x34	0x78	0x12	0xFE	0xDB	0x78

- Debug tiếp tục chương trình, ta thấy đoạn mã

```

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_1_0040324E ASCII "%X%X%X%X%X%X%"
004014A3 . 8D85 58FEFFF LEA EAX,DWORD PTR [EBP-1A8]
004014A9 . 890424 MOV DWORD PTR [ESP],EAX

```

Khi debug ta thấy đoạn mã này chuyển password nhập (dạng string) thành chuỗi hex.

- Tiếp tục, chương trình thực hiện so sánh chuỗi hex nhập với một chuỗi hex nào đó. Cụ thể đối số hiển thị khi strcmp:

```

Registers (FPU)
EAX 0060FCB0 ASCII "4D11628EBE1D"
ECX 3B6EBEC4
EDX 0060FD80 ASCII "551A7186A22"

```

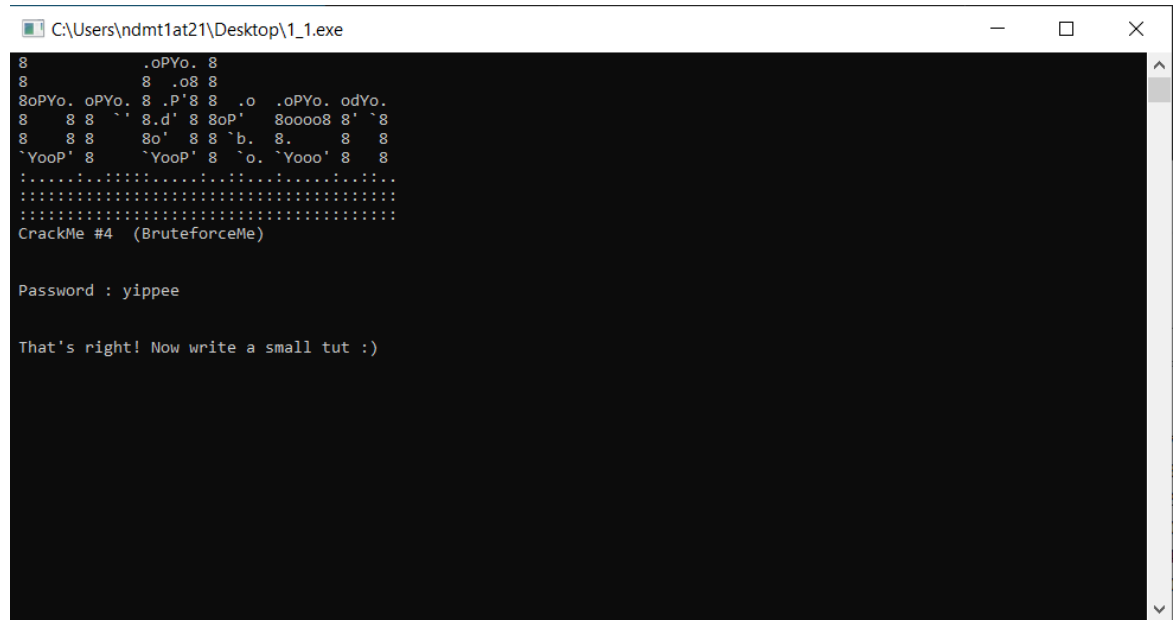
Chuỗi "551A7186A22" là chuỗi hex của chuỗi "abcxyz". Vậy dễ dàng suy ra chuỗi hex "4D11628EBE1D" là password dạng hex của chương trình. (vì lệnh so sánh quyết định nhảy đến badboy hay không)

- Chuyển đổi ngược lại (a XOR b = c -> c XOR b = a):

idx	0	1	2	3	4	5
SRC	0x4D	0x11	0x62	0x8E	0xBE	0x1D
XOR	0x34	0x78	0x12	0xFE	0xDB	0x78
RESULT	0x79	0x69	0x70	0x70	0x65	0x65

Tra bảng mã ASCII ta được chuỗi password cần tìm: "yippee"

- Kiểm tra kết quả:



#### 1.4. Thuật toán chương trình phát sinh key

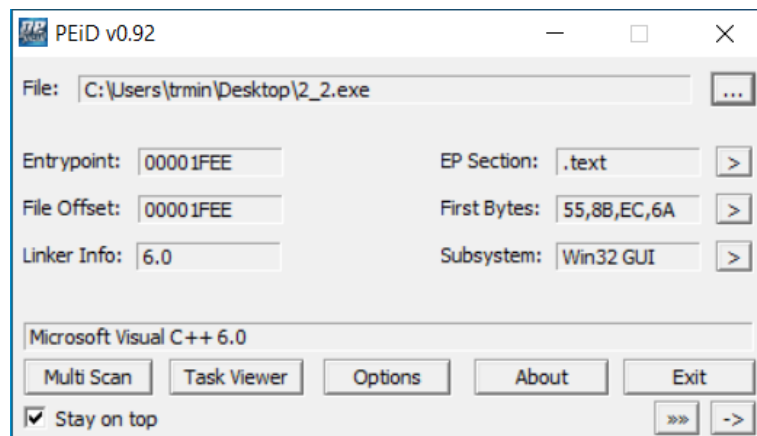
**B1.** Lấy mỗi 2 ký tự (mã hex) trong chuỗi “4D11628EBE1D” xor lần lượt các giá trị trong bảng hash.

**B2.** In ra màn hình kết quả.

## 2. Chương trình 1\_2.exe

### 2.1. Kiểm tra chương trình với PEiD

Tương tự, ta mở PEiD lên kiểm tra chương trình có pack/protect không, kết quả thu được:

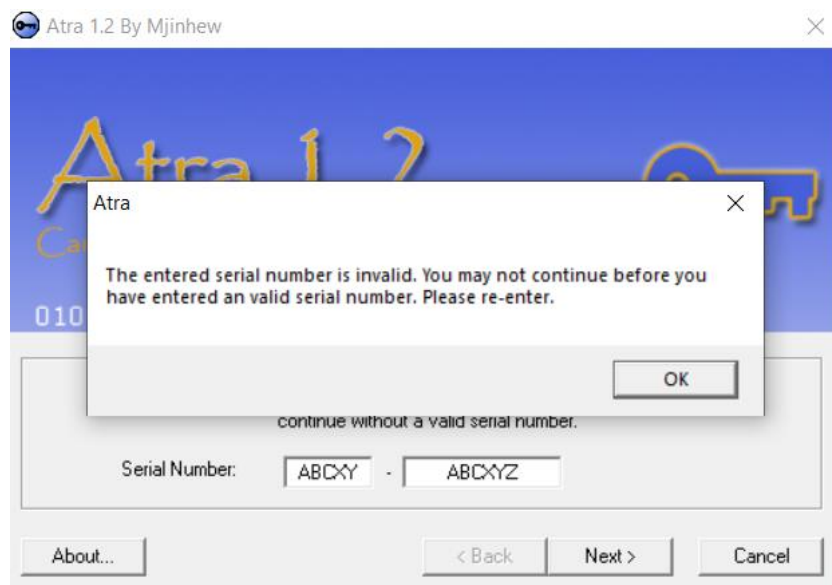


Ta xác định được chương trình được viết bằng C++. Mở OllyDbg phân tích.

## 2.2. Giao diện chương trình



Khi nhập một password bất kỳ, ta thấy ô đầu tiên cho phép nhập 5 ký tự, ô thứ 2 nhập được 6 ký tự. Nhập ABCXY-ABCXYZ, chương trình hiển thị như sau



Lưu ý đến chuỗi hiển thị “The entered....”

## 2.3. Debug chương trình với OllyDbg

- Tìm tất chuỗi có trong chương trình (thao tác như phần 1\_1.exe), đi đến phần badboy tức chuỗi “The entered serial number is invalid...”. Quan sát lên trên một chút, ta thấy so sánh s1 và s2 để quyết định badboy hay goodboy nhưng chưa rõ 2 chuỗi là gì.

0040149D	50	PUSH EAX	
0040149E	8D45 BC	LEA EAX,DWORD PTR [EBP-44]	s2
004014A1	50	PUSH EAX	s1
004014A2	E8 2F0B0000	CALL <JMP.&MSUCRT.strcmp>	strcmp
004014A7	83C4 20	ADD ESP,20	
004014AA	85C0	TEST EAX,EAX	
004014AC	6A 00	PUSH 0	
004014AE	68 C0444000	PUSH 2_2.004044C0	ASCII "Atra"
004014B3	75 0E	JNZ SHORT 2_2.004014C3	
004014B5	68 9C444000	PUSH 2_2.0040449C	ASCII "The entered serial number is vALiD!"
004014BA	EB 0C	JMP SHORT 2_2.004014C8	
004014BC	6A 00	PUSH 0	
004014BE	68 C0444000	PUSH 2_2.004044C0	ASCII "Atra"
004014C3	68 20444000	PUSH 2_2.00404420	ASCII "The entered serial number is invalid. You may

- Ta chưa xác định phải đặt BP ở đâu, vì còn rất nhiều dòng lệnh ở trên. Thử kéo lên một chút, ta thấy được một loạt các xử lý liên quan đến chuỗi. Thử đặt BP ở đầu hàm này (0x00401270).

00401270	55	PUSH EBP	
00401271	8BEC	MOV EBP,ESP	
00401273	83EC 44	SUB ESP,44	
00401276	53	PUSH EBX	
00401277	56	PUSH ESI	
00401278	57	PUSH EDI	
00401279	8D45 E0	LEA EAX,DWORD PTR [EBP-20]	
0040127C	6A 06	PUSH 6	
0040127E	33F6	XOR ESI,ESI	
00401280	50	PUSH EAX	
00401281	894D F8	MOV DWORD PTR [EBP-8],ECX	
00401284	68 E0300000	PUSH 3E8	
00401289	8975 F0	MOV DWORD PTR [EBP-10],ESI	
0040128C	8975 EC	MOV DWORD PTR [EBP-14],ESI	
0040128F	8975 F4	MOV DWORD PTR [EBP-C],ESI	
00401292	E8 DD0C0000	CALL <JMP.&MFC42.#3098>	
00401297	8B4D F8	MOV ECX,DWORD PTR [EBP-8]	
0040129A	8D45 BC	LEA EAX,DWORD PTR [EBP-44]	
0040129D	6A 0C	PUSH 0C	
0040129F	50	PUSH EAX	
004012A0	68 E9030000	PUSH 3E9	
004012A5	E8 CA0C0000	CALL <JMP.&MFC42.#3098>	
004012AA	E8 26020000	CALL 1_2.004014D5	
004012AF	85C0	TEST EAX,EAX	
004012B1	0F84 A4000000	JE 1_2.0040135B	
004012B7	8D45 E0	LEA EAX,DWORD PTR [EBP-20]	
004012BA	50	PUSH EAX	s
004012BB	E8 1C0D0000	CALL <JMP.&MSUCRT.strlen>	strlen
004012C0	85C0	TEST EAX,EAX	
004012C2	59	POP ECX	
004012C3	0F86 92000000	JBE 1_2.0040135B	
004012C9	8A45 E0	MOV AL,BYTE PTR [EBP-20]	
004012CC	B1 03	MOV CL,3	
004012CE	04 08	ADD AL,8	
004012D0	8975 E8	MOV DWORD PTR [EBP-18],ESI	
004012D3	F6E9	IMUL CL	
004012D5	8A45 E0	MOV BYTE PTR [EBP-20],AL	
004012D8	BF DC324000	MOV EDI,1_2.004032DC	ASCII "BDR0KPTUJI"
004012DD	EB 02	JMP SHORT 1_2.004012E1	
004012DF	33F6	XOR ESI,ESI	
004012E1	8D45 E0	LEA EAX,DWORD PTR [EBP-20]	
004012E4	8975 FC	MOV DWORD PTR [EBP-4],ESI	
004012E7	50	PUSH EAX	s
004012E8	E8 EF0C0000	CALL <JMP.&MSUCRT.strlen>	strlen
004012ED	85C0	TEST EAX,EAX	
004012EF	59	POP ECX	
004012F0	76 58	JBE SHORT 1_2.0040134A	
004012F2	8D5D E0	LEA EBX,DWORD PTR [EBP-20]	

Nhấn F9 chạy đến BP, nhập chuỗi key “12345 – 12345678”, chương trình dừng lại.

- Khi debug tiếp tục, ta thấy một đoạn code, tương tự như 2 dòng for lồng nhau:

00401378	> 57	PUSH EDI	s = "BDRQKPTVJI"
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	strlen
004013B0	. E8 270C0000	CALL <JMP.&MSUCRT.strlen>	strlen
004013B5	. 3BF0	CMP ESI,EAX	
004013B7	. 59	POP ECX	
004013B8	^ 72 CB	JB SHORT 1_2.00401385	
004013BA	> FF45 FC	INC DWORD PTR [EBP-4]	
004013BD	. 8D45 E0	LEA EAX,DWORD PTR [EBP-20]	
004013C0	. 50	PUSH EAX	s
004013C1	. 43	INC EBX	strlen
004013C2	. E8 150C0000	CALL <JMP.&MSUCRT.strlen>	strlen
004013C7	. 3945 FC	CMP DWORD PTR [EBP-4],EAX	

Khi chạy từng dòng một, để rút ra kết luận: dòng for trong cùng biến chạy j (0 ≤ j < 9). Dòng lặp này làm hai nhiệm vụ: so sánh ký tự thứ i từ dòng for đầu (0 ≤ i < chiều dài key ô thứ nhất) với ký tự "BDRQKPTVJI"[j] và so sánh với số từ '0' đến '9'.

Nếu điều kiện so sánh đầu đúng tăng count1, điều kiện thứ 2 đúng tăng count2.

- Sau khi kết thúc 2 dòng lặp, ta thấy đoạn so sánh:

004013C0	. 837D F0 03	CMP DWORD PTR [EBP-10],3	
004013D1	~ 0F85 E5000000	JNZ 1_2.004014BC	
004013D7	. 837D EC 02	CMP DWORD PTR [EBP-14],2	
004013DB	~ 0F85 D6000000	JNZ 1_2.004014BC	

Ý nghĩa: so sánh count1 với 3 và count2 với 2, nếu một trong hai điều kiện không thỏa nhảy đến badboy

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 invalid. You may r
004014C5	< 0040 50	CALL EBX NUMBER OF RECORDS	

Vậy để chương trình có thể kiểm tra key2, thì key1 phải đảm bảo: **có 3 ký tự trong chuỗi "BDRQKPTVJI" và 2 ký tự từ '0' - '9'.**

- Nhập lại key1 = "BDR12", key2 = "12345678", sau đó chạy đến đoạn so sánh, chạy tiếp tục:

004013EF	. E8 6A090000	CALL 1_2.00401D5E	
004013F4	. 8B3D B0314000	MOV EDI,DWORD PTR [;&MSUCRT.sprintf>]	MSUCRT.sprintf

Sau khi CALL, thì thanh ghi EAX có một giá trị lạ, không còn thấy chuỗi "BDR12" trước khi gọi hàm.



- Nhập lại key1, key2, vào hàm 1\_2.00401D5E:

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

Để thấy trong hàm này có 3 hàm con được gọi, lần lượt đi vào các hàm.

- Ý nghĩa hàm 1.2\_004015A7:

004015A7	8B4424 04	MOV EAX, DWORD PTR [ESP+4]
004015AB	8360 14 00	AND DWORD PTR [EAX+14],0
004015AF	8360 10 00	AND DWORD PTR [EAX+10],0
004015B3	C700 01234567	MOV DWORD PTR [EAX],67452301
004015B9	C740 04 89ABCD	MOV DWORD PTR [EAX+4],EFCDA8B9
004015C0	C740 08 FEDCBA	MOV DWORD PTR [EAX+8],98BADCFE
004015C7	C740 0C 76543210	MOV DWORD PTR [EAX+C],10325476
004015CC	C3	RET

Hàm này có tác dụng gán 6 giá trị vào dãy địa chỉ liên tục, kết quả sau gán:

0019F7BC	67452301
0019F7C0	EFCDA8B9
0019F7C4	98BADCFE
0019F7C8	10325476
0019F7CC	00000000
0019F7D0	00000000

Như những dãy số vô nghĩa. Tìm thử thì **giá trị 0x67452301, 0xEFCDA8B9 có liên quan đến mã MD5**. Đây là một mảng, ta đặt tên lần lượt là: state[0] state[1], state[2], state[3], count[0], count[1].

- Ý nghĩa hàm 1.2\_004015CF:

004015CF	55	PUSH EBP
004015D0	8BEC	MOV EBP,ESP
004015D2	53	PUSH EBX
004015D3	56	PUSH ESI
004015D4	8B75 08	MOV ESI, DWORD PTR [EBP+8]
004015D7	57	PUSH EDI
004015D8	8B7D 10	MOV EDI, DWORD PTR [EBP+10]
004015DB	8B4E 10	MOV ECX, DWORD PTR [ESI+10]
004015DE	8BD7	MOV EDX, EDI
004015E0	8BC1	MOV EAX, ECX
004015E2	C1E8 03	SHR EAX,3
004015E5	8D0CF9	LEA ECX, DWORD PTR [ECX+EDI*8]
004015E8	83E0 3F	AND EAX,3F
004015EB	C1E2 03	SHL EDX,3
004015EE	3BCA	CMPL ECX,EDX
004015F0	894E 10	MOV DWORD PTR [ESI+10],ECX
004015F3	73 03	JNB SHORT 1_2.004015F8
004015F5	FF46 14	INC DWORD PTR [ESI+14]
004015F8	6A 40	PUSH 40
004015FA	8BCF	MOV ECX, EDI
004015FC	5B	POP EBX
004015FD	C1E9 10	SHR ECX,10
00401600	014E 14	ADD DWORD PTR [ESI+14],ECX
00401603	2BD8	SUB EBX,EAX
00401605	3BFB	CMPL EDI,EBX

00401607	72 42	JB SHORT 1_2.0040164B	[n src dest memcpy
00401609	53	PUSH EBX	
0040160A	8D4430 18	LEA EAX,DWORD PTR [EAX+ESI+18]	
0040160E	FF75 0C	PUSH DWORD PTR [EBP+C]	
00401611	50	PUSH EAX	
00401612	E8 CB090000	CALL <JMP.&MSUCRT.memcpy>	
00401617	8D46 18	LEA EAX,DWORD PTR [ESI+18]	
0040161A	50	PUSH EAX	
0040161B	56	PUSH ESI	
0040161C	E8 4C000000	CALL 1_2.0040166D	
00401621	83C4 14	ADD ESP,14	[n src dest memcpy
00401624	895D 08	MOV DWORD PTR [EBP+8],EBX	
00401627	83C3 3F	ADD EBX,3F	
0040162A	3BD F	CMP EBX,EDI	
0040162C	73 19	JNB SHORT 1_2.00401647	
0040162E	8B45 0C	MOV EAX,DWORD PTR [EBP+C]	
00401631	8D4418 C1	LEA EAX,DWORD PTR [EAX+EBX-3F]	
00401635	50	PUSH EAX	
00401636	56	PUSH ESI	
00401637	E8 31000000	CALL 1_2.0040166D	
0040163C	8345 08 40	ADD DWORD PTR [EBP+8],40	[n src dest memcpy
00401640	59	POP ECX	
00401641	59	POP ECX	
00401642	83C3 40	ADD EBX,40	
00401645	EB E3	JMP SHORT 1_2.0040162A	
00401647	33C0	XOR EAX,EAX	
00401649	EB 04	JMP SHORT 1_2.0040164F	
0040164B	8365 08 00	AND DWORD PTR [EBP+8],0	
0040164F	8B4D 08	MOV ECX,DWORD PTR [EBP+8]	
00401652	8B55 0C	MOV EDX,DWORD PTR [EBP+C]	
00401655	2BF9	SUB EDI,ECX	[n src dest memcpy
00401657	03CA	ADD ECX,EDX	
00401659	57	PUSH EDI	
0040165A	8D4430 18	LEA EAX,DWORD PTR [EAX+ESI+18]	
0040165E	51	PUSH ECX	
0040165F	50	PUSH EAX	
00401660	E8 7D090000	CALL <JMP.&MSUCRT.memcpy>	
00401665	83C4 0C	ADD ESP,0C	
00401668	5F	POP EDI	
00401669	5E	POP ESI	
0040166A	5B	POP EBX	[n src dest memcpy
0040166B	5D	POP EBP	
0040166C	C3	RET	

Thứ tự thực hiện hàm:

+ Thanh ghi ECX load giá trị ở địa chỉ **0x0019F7CC** (được gán ở hàm thứ nhất): đặt giá trị thanh ghi là a.

+ Thanh ghi EDI chứa độ dài của chuỗi key1: gọi giá trị trong thanh ghi là lenKey1.

+ Thanh ghi EDX = EDI (chứa tạm thời): tmpLen

+ Gán EAX = ECX, gọi giá trị của thanh ghi EAX là b.

Ý nghĩa các bước xử lý từ **0x004015D4** đến **0x004015EB**:

b = (a >> 3)

a = a + lenKey1 \* 8

b = b & 0x3f

tmpLen = lenKey << 3

+ Gọi giá trị chứa ở địa chỉ **0x0019F7D0** (được gán ở hàm thứ nhất): c

Ý nghĩa các bước xử lý từ **0x004015EE** đến **0x0040166C**:

if (a < tmpLen)

c++;

else

EBX = 0x40

c += lenKey1 >> 29

d = EBX - a = 64 - a

```

if (lenKey1 >= d)
{
    memcpy(&buffer[b], input, d)
    Gọi 1.2_0040166D, tham số buffer, &state)
    I = d
    for (I = d; I + 63 < inputLen; I += 64)
        Gọi 1.2_0040166D, tham số input[I], &state)
}
else I = 0
memcpy(&buff[b], &input, z - I)

```

- Ý nghĩa hàm **1\_2.0040166D**:  
3 tham số x: buffer, y: địa chỉ state  
+ Gọi hàm **1\_2.00401C6F**: 3 tham số x: buffer, y: mảng 16 byte, z = 64  
+ Hàng loạt xử lý phức tạp (copy code asm phần này)

0040166D	55	PUSH EBP
0040166E	8BEC	MOV EBP,ESP
00401670	83EC 48	SUB ESP,48
00401673	53	PUSH EBX
00401674	56	PUSH ESI
00401675	8B75 08	MOV ESI,DWORD PTR [EBP+8]
00401678	57	PUSH EDI
00401679	6A 40	PUSH 40
0040167B	8B06	MOV EAX,DWORD PTR [ESI]
0040167D	FF75 0C	PUSH DWORD PTR [EBP+C]
00401680	8B7E 04	MOV EDI,DWORD PTR [ESI+4]
00401683	8B5E 08	MOV EBX,DWORD PTR [ESI+8]
00401686	8945 08	MOV DWORD PTR [EBP+8],EAX
00401689	8B46 0C	MOV EAX,DWORD PTR [ESI+C]
0040168C	8945 F8	MOV DWORD PTR [EBP-8],EAX
0040168F	8D45 B8	LEA EAX,DWORD PTR [EBP-48]
00401692	50	PUSH EAX
00401693	E8 D7050000	CALL 1_2.00401C6F
00401698	8BC7	MOV EAX,EDI
0040169A	8BCB	MOV ECX,EBX
0040169C	F7D0	NOT EAX
0040169E	2345 F8	AND EAX,DWORD PTR [EBP-8]
004016A1	23CF	AND ECX,EDI
004016A3	0BC1	OR EAX,ECX
004016A5	8B4D 08	MOV ECX,DWORD PTR [EBP+8]
004016A8	0345 B8	ADD EAX,DWORD PTR [EBP-48]
004016AB	03C9	ADD ECX,EAX
004016AD	8BC1	MOV EAX,ECX
004016AF	C1E8 1D	SHR EAX,1D
004016B2	C1E1 03	SHL ECX,3
004016B5	0BC1	OR EAX,ECX
004016B7	8BCF	MOV ECX,EDI
004016B9	8BD0	MOV EDX,EAX
004016BB	23C8	AND ECX,EAX
004016BD	F7D2	NOT EDX

- Ý nghĩa hàm **1\_2.004016CF**: 3 tham số (DWORD\* x, BYTE\* y, DWORD z)

00401C6F	837C24 0C 00	CMP DWORD PTR [ESP+C],0
00401C74	76 3E	JBE SHORT 1_2.00401CB4
00401C76	8B5424 08	MOV EDX,DWORD PTR [ESP+8]
00401C7A	8B4C24 04	MOV ECX,DWORD PTR [ESP+4]
00401C7E	56	PUSH ESI
00401C7F	57	PUSH EDI
00401C80	6A FE	PUSH -2
00401C82	8D42 02	LEA EAX,DWORD PTR [EDX+2]
00401C85	5E	POP ESI
00401C86	2BF2	SUB ESI,EDX
00401C88	0FB678 FF	MOVZX EDI,BYTE PTR [EAX-1]
00401C8C	33D2	XOR EDX,EDX
00401C8E	8A70 01	MOV DH,BYTE PTR [EAX+1]
00401C91	8A10	MOV DL,BYTE PTR [EAX]
00401C93	83C0 04	ADD EAX,4
00401C96	C1E2 08	SHL EDX,8
00401C99	0BD7	OR EDX,EDI
00401C9B	0FB678 FA	MOVZX EDI,BYTE PTR [EAX-6]
00401C9F	C1E2 08	SHL EDX,8
00401CA2	0BD7	OR EDX,EDI
00401CA4	8911	MOV DWORD PTR [ECX],EDX
00401CA6	8D1406	LEA EDX,DWORD PTR [ESI+EAX]
00401CA9	83C1 04	ADD ECX,4
00401CAC	3B5424 14	CMP EDX,DWORD PTR [ESP+14]
00401CB0	72 D6	JB SHORT 1_2.00401C88
00401CB2	5F	POP EDI
00401CB3	5E	POP ESI
00401CB4	C3	RET

```

for (i = 0, j = 0; j < len; i++, j += 4)
{
    output[i] = ((DWORD)input[j]) | (((DWORD)input[j+1]) << 8) |
    (((DWORD)input[j+2]) << 16) | (((DWORD)input[j+3]) << 24);
}

```

- Ý nghĩa hàm **1\_2.00401CB5**:

+ Gọi hàm: **1\_2.00401D19**, 3 tham số x, y, z. Trong đó x: địa chỉ count, y: địa chỉ phần tử đầu của mảng 2 phần tử (8 byte), z = 8.

+ EAX = giá trị count[0] = e; e = (e >> 3) & 0x3f

+ f = (e < 56) ? (56 - e) : (120 - e)

+ Gọi hàm **1\_2.004015CF** với 3 tham số truyền: địa chỉ state, 1 mảng x và f. Với x là 1 mảng 64 phần tử 1 byte, địa chỉ phần tử đầu **0x004045A8**.

004045A8	80 00 00 00 00 00 00 00	.....
004045B0	00 00 00 00 00 00 00 00	.....
004045B8	00 00 00 00 00 00 00 00	.....
004045C0	00 00 00 00 00 00 00 00	.....
004045C8	00 00 00 00 00 00 00 00	.....
004045D0	00 00 00 00 00 00 00 00	.....
004045D8	00 00 00 00 00 00 00 00	.....
004045E0	00 00 00 00 00 00 00 00	.....

+ Gọi hàm **1\_2.004015CF** với 3 tham số x, y, z. Trong đó x: địa chỉ state, 1 mảng y 8 byte và z = 8

+ Gọi hàm **1\_2.00401D19**, 3 tham số x, y, z. Trong đó x: địa chỉ state, y là mảng kết quả (16 byte), z = 4

- Ý nghĩa chính của hàm **1.2.00401D19**

00401D19	837C24 0C 00	CMP DWORD PTR [ESP+C],0
00401D1E	76 3D	JBE SHORT 1_2.00401D5D
00401D20	8B5424 04	MOV EDX,DWORD PTR [ESP+4]
00401D24	8B4C24 08	MOV ECX,DWORD PTR [ESP+8]
00401D28	56	PUSH ESI
00401D29	83CE FF	OR ESI,FFFFFFFF
00401D2C	8D42 01	LEA EAX,DWORD PTR [EDX+1]
00401D2F	2BF2	SUB ESI,EDX
00401D31	8A11	MOV DL,BYTE PTR [ECX]
00401D33	8B50 FF	MOV BYTE PTR [EAX-1],DL
00401D36	8B11	MOV EDX,DWORD PTR [ECX]
00401D38	C1EA 08	SHR EDX,8
00401D3B	8B10	MOV BYTE PTR [EAX],DL
00401D3D	8B11	MOV EDX,DWORD PTR [ECX]
00401D3F	C1EA 10	SHR EDX,10
00401D42	8B50 01	MOV BYTE PTR [EAX+1],DL
00401D45	8B11	MOV EDX,DWORD PTR [ECX]
00401D47	C1EA 18	SHR EDX,18
00401D4A	8B50 02	MOV BYTE PTR [EAX+2],DL
00401D4D	83C0 04	ADD EAX,4
00401D50	83C1 04	ADD ECX,4
00401D53	8D1406	LEA EDX,DWORD PTR [ESI+EAX]
00401D56	3B5424 10	CMP EDX,DWORD PTR [ESP+10]
00401D5A	72 D5	JB SHORT 1_2.00401D31
00401D5C	5E	POP ESI
00401D5D	C3	RET

Truyền vào 3 tham số(unsigned char\* x, DWORD\* y, z).

```

for (i = 0, j = 0; j < len; i++, j += 4)
{
    y[j] = x[i] & 0xff;
}

```

```

y[j+1] = (x[i] >> 8) & 0xff;
y[j+2] = (x[i] >> 16) & 0xff;
y[j+3] = (x[i] >> 24) & 0xff;
}
(Vì dùng thanh ghi DL (1byte) để chứa nên ta sẽ thêm & 0xff)

```

**Kết luận:** Từ ý nghĩa của các hàm, ta thấy được hàm **1\_2.00401D5E** sử dụng thuật toán gần giống thuật toán hash md5.

- Kết thúc hàm hash md5, sau đó là chuyển kết quả trong thanh ghi EAX về dạng chuỗi hex. Gọi là S.

004013F4	. 8B3D B0314000	MOV EDI,DWORD PTR [<&MSUCRT.sprintf>]	MSUCRT.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	printf
00401405	. FFD7	CALL EDI	

- Tiếp theo là một dòng for:

0040141B	> 8B45 FC	MOV EAX,DWORD PTR [EBP-4]	
0040141E	. 50	PUSH EAX	
0040141F	. 8D7405 D4	LEA ESI,DWORD PTR [EBP+EAX-2C]	
00401423	. 0FBE4405 D4	MOVSX EAX,BYTE PTR [EBP+EAX-2C]	
00401428	. 50	PUSH EAX	
00401429	. E8 19FEFFFF	CALL 1_2.00401247	
0040142E	. FF45 FC	INC DWORD PTR [EBP-4]	
00401431	. 8B06	MOV BYTE PTR [ESI],AL	
00401433	. 8D45 D4	LEA EAX,DWORD PTR [EBP-2C]	
00401436	. 50	PUSH EAX	[s
00401437	. E8 A00B0000	CALL <JMP.&MSUCRT.strlen>	strlen
0040143C	. 83C4 0C	ADD ESP,0C	
0040143F	. 3945 FC	CMP DWORD PTR [EBP-4],EAX	
00401442	. ^ 72 D7	JB SHORT 1_2.0040141B	

Có dòng gọi hàm **1\_2.00401247**

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

Từ hàm con và vòng lặp ngoài, khi debug ta rút ra:

- + Biến đếm chạy từ 0 đến length(S)
- + Trong hàm 1\_2\_00401247, thực hiện load từng ký tự và xor với một số x được load lên từ ram. Kết hợp với việc kết quả từ hàm này chỉ lấy 1 byte nên ta có mảng các số x (gọi là table):

Table[8] = {0xA6, 0x16, 0xAF, 0xFD, 0xD4, 0x07, 0x10, 0xF6}

- + Nói gọn lại, ý nghĩa hàm 1\_2.00401247:

```

for (int i = 0; i < S.length(); i++)
    S[i] = S[i] ^ table[i];

```

- Tương tự, tiếp tục đi xuống cũng có một dòng for, khá tương tự ở trên:

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	. 8806	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>	[s strlen
00401477	. 83C4 0C	ADD ESP,0C	
0040147A	. 3945 FC	CMP DWORD PTR [EBP-4],EAX	
0040147D	. ^ 72 D7	JB SHORT 1_2.00401456	

Gọi hàm 1\_2.00401261

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

Ý nghĩa của dòng lặp khá đơn giản:

```
for (int i = 0; i < S.length(); i++)
    S[i] = (S[i] << 1) | S[i];
```

- Tiếp theo, chương trình gọi hàm 1\_2.004010C0

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

Khi debug, dễ thấy câu lệnh MOV EAX, DWORD PTR [EAX\*4 + 404020] load lên giá trị như: 0x4DB26158, 0x65B0D9C6, D1BB67F1...(đương nhiên là khi nhập key1 là BDR12). Khi tìm kiếm ta thấy các giá trị này nằm trong lookupTable của hash CRC32. Với các câu lệnh như trên ta có thể viết lại như sau:

```
uint32_t result = 0xFFFFFFFF;
int index = 0;
for (int i = 0; i < S.length(); i++)
{
    index = (S[i] ^ result) & 0x000000FF (lấy byte cuối);
    result = (result >> 8) ^ crc32Table[index];
}
result = ~result;
```



Trong đó crc32Table là mảng, được thể hiện ở hex dump tại địa chỉ **0x00404020**

00404020	00 00 00 00 96 30 07 77	.....0.w
00404028	2C 61 0E EE BA 51 09 99	,a...Q..
00404030	19 C4 6D 07 8F F4 6A 70	..m...jp
00404038	35 A5 63 E9 A3 95 64 9E	5.c...d.
00404040	32 88 DB 0E A4 B8 DC 79	2.....y
00404048	1E E9 D5 E0 88 D9 D2 97	.....
00404050	2B 4C B6 09 BD 7C B1 7E	+L...!.~
00404058	07 2D B8 E7 91 1D BF 90	.-.....
00404060	64 10 B7 1D F2 20 B0 6A	d.... .j
00404068	48 71 B9 F3 DE 41 BE 84	Hq...A..
00404070	7D D4 DA 1A EB E4 DD 6D	}......M
00404078	51 B5 D4 F4 C7 85 D3 83	Q.....

(mảng này sẽ có 256 phần tử, hình trên thể hiện một vài giá trị)

- Sau khi thực hiện công việc hash CRC32 chuỗi S, ta được kết quả là một giá trị 4 byte. Thực hiện chuyển giá trị này về hex string, và cuối cùng là so sánh chuỗi S này với key2 mà người dung nhập

0040149D	. 50	PUSH EAX	s2 s1 = "" strcmp
0040149E	. 8D45 BC	LEA EAX,DWORD PTR [EBP-44]	
004014A1	. 50	PUSH EAX	
004014A2	. E8 2F0B0000	CALL <JMP.&MSVCRT.strcmp>	

Nếu đúng đi đến goodboy, ngược lại là badboy

## 2.4. Thuật toán chương trình phát sinh key

**B1.** Phát sinh key1 có đủ 3 chữ cái trong chuỗi "BDRQKPTVJI" và 2 chữ số từ 0 – 9.

**B2.** Hash chuỗi key1 với thuật toán gần giống md5.

MD5\_Init (0x004015A7)

MD5\_Update (0x004015CF)

MD5\_Final (0x00401CB5)

MD5\_Final trả về chuỗi char MD5\_Digest[16].

**B3.** Thực hiện copy vùng nhớ

UINT32 A = 0;

for (int i = 0; i < 4; i++)

{

UINT32 tmp = memcpy((void\*)&tmp, MD5\_Digest[i \* 4];

A = A ^ tmp;

}

**B4.**

S = intToHexStr(A);

for (int i = 0; i < S.length(); i++)

S[i] = (S[i] << 1) | S[i];

```
for (int i = 0; i < S.length(); i++)  
    S[i] = (S[i] << 1) | S[i];
```

**B5.** `UINT32 crc = GetCrcStr(S);`

**B6.** `string key2 = intToHexStr(crc)`

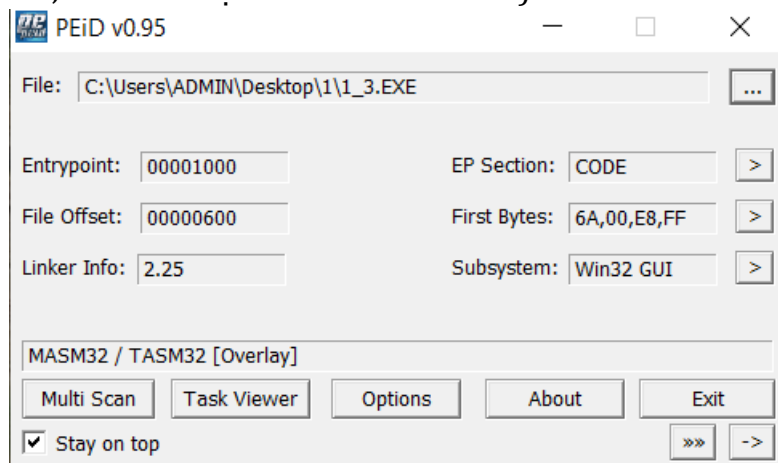
In ra màn hình key2. Hoàn tất.

**Key minh họa:** 22IVD - 9255E152

### 3. Chương trình 1\_3.exe

#### 3.1. Kiểm tra chương trình với PEiD

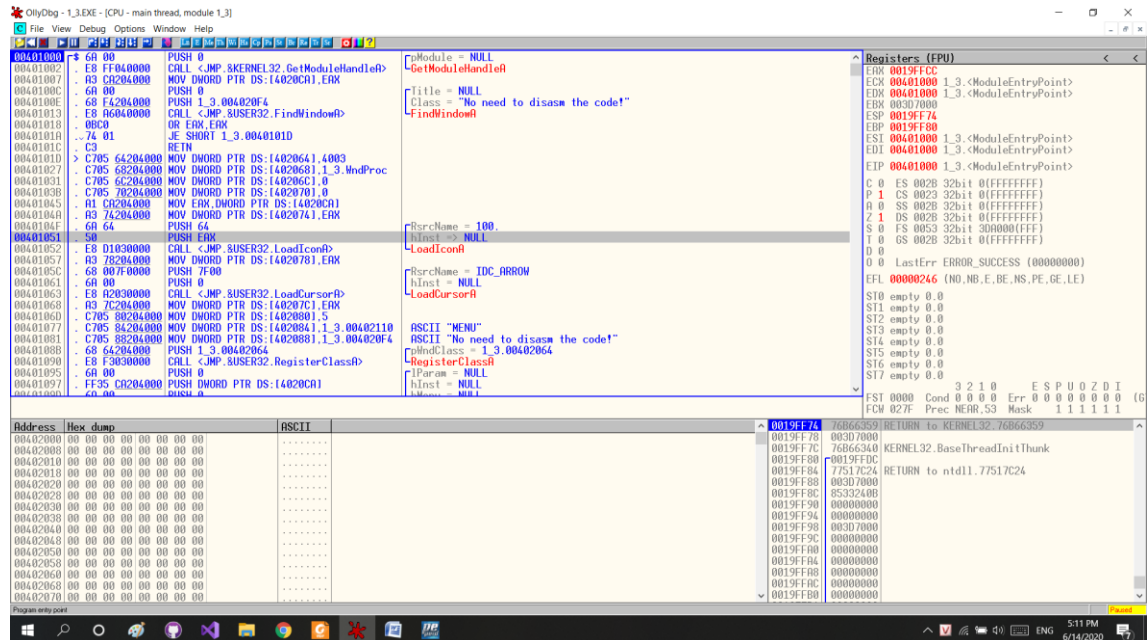
(Packer là một kiểu chương trình nén hoặc che dấu file thực thi (executable file). Các chương trình này ra đời bắt nguồn từ mục đích giảm kích thước của file, làm cho việc tải file nhanh hơn)



Sau khi Scan xong ra kết quả thì ta có thể thấy kết quả là chương trình không bị pack và có thể được code bằng MASM32 hoặc TASM32.



## 3.2. Debug chương trình với OllyDbg



Có nhiều cách để tiếp cận như

- Thông qua việc nhập dữ liệu(Hàm GetDlgItemTextA)
- Hàm MessageBoxA
- Các từ khóa GoodBoy,BadBoy

Trong bài này mình chọn theo cách 1 là GetDlgItemTextA

Sơ lược về hàm GetDlgItemTextA

(<https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-getdlgitemtext>)

### Syntax

C++

Copy

```
UINT GetDlgItemTextA(  
    HWND hDlg,  
    int nIDDlgItem,  
    LPSTR lpString,  
    int cchMax  
);
```

## Parameters

hDlg

Type: **HWND**

A handle to the dialog box that contains the control.

nIDDlgItem

Type: **int**

The identifier of the control whose title or text is to be retrieved.

lpString

Type: **LPTSTR**

The buffer to receive the title or text.

cchMax

Type: **int**

The maximum length, in characters, of the string to be copied to the buffer pointed to by *lpString*. If the length of the string, including the null character, exceeds the limit, the string is truncated.

## Return value

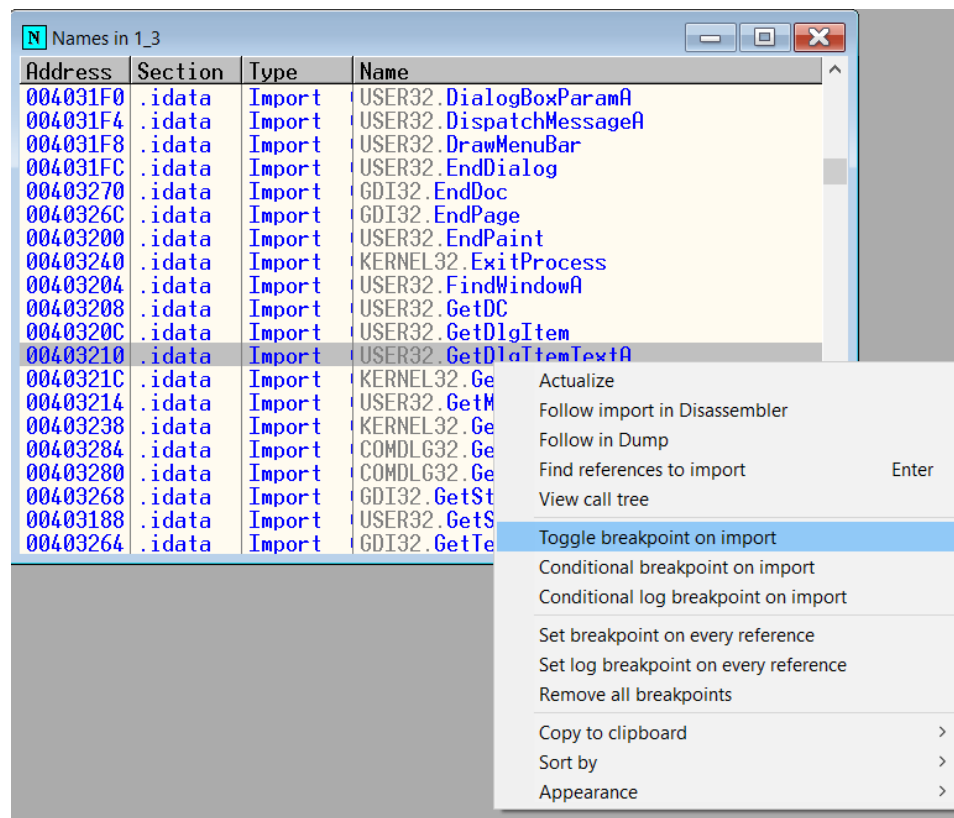
Type: **UINT**

If the function succeeds, the return value specifies the number of characters copied to the buffer, not including the terminating null character.

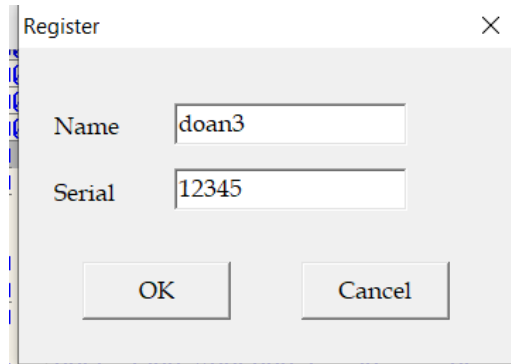
If the function fails, the return value is zero. To get extended error information, call [GetLastError](#).

Để hiểu rõ hơn các dữ liệu khi nhập vào sẽ được lưu trữ ở đâu chúng ta đặt 1 BreakPoint vào Hàm

## *GetDlgItemTextA*

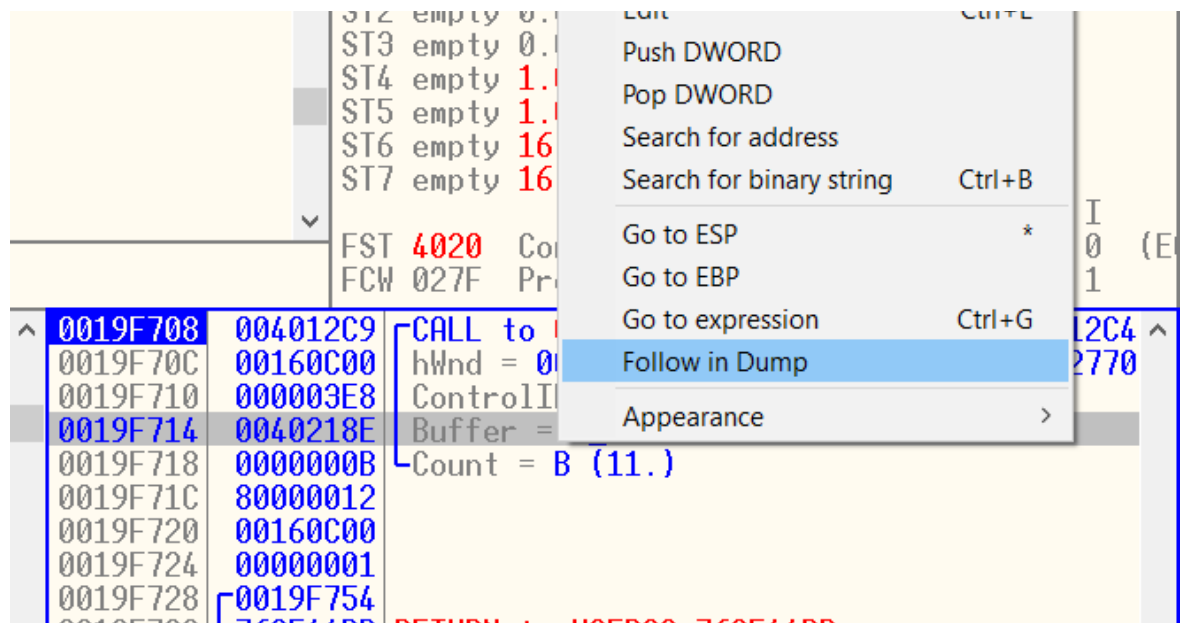


Nhấn F9 để Debug chương trình



Nhập 1 bộ ngẫu nhiên để test

Sau khi nhấn OK chương trình sẽ dừng lại ngay hàm mình đã đặt BP. Ở dòng Count = 11 cho chúng ta biết dữ liệu tối đa là 11. Suy ra số ký tự Name < 11



Nhấn vào Follow in Dump để xem dữ liệu

Address	Hex dump	ASCII
0040218E	00 00 00 00 00 00 00 00	.....
00402196	00 00 00 00 00 00 00 00	.....
0040219E	00 00 00 00 00 00 00 00	.....
004021A6	00 00 00 00 00 00 00 00	.....
004021AE	00 00 00 00 00 00 00 00	.....
004021B6	00 00 00 00 00 00 00 00	.....
004021BE	00 00 00 00 00 00 00 00	.....
004021C6	00 00 00 00 00 00 00 00	.....
004021CE	00 00 00 00 00 00 00 00	.....
004021D6	00 00 00 00 00 00 00 00	.....
004021DE	00 00 00 00 00 00 00 00	.....
004021E6	00 00 00 00 00 00 00 00	.....
004021EE	00 00 00 00 00 00 00 00	.....
004021F6	00 00 00 00 00 00 00 00	.....
004021FE	00 00 00 00 00 00 00 00	.....

Dữ liệu chưa có gì tức là Hàm chưa được chạy. Nhấn F8 để trace qua các câu lệnh để thực thi hàm

76A444CB	EB 707EFAFF	CALL USER32.GetWindowTextA	
76A444D0	EB 0E	JMP SHORT USER32.76A444E0	
76A444D2	837D 14 00	CMP DWORD PTR SS:[EBP+14],0	
76A444D6	74 06	JE SHORT USER32.76A444DE	
76A444D8	8B45 10	MOV EAX,DWORD PTR SS:[EBP+10]	
76A444DB	C600 00	MOV BYTE PTR DS:[EAX],0	
76A444DE	33C0	XOR EAX,EAX	
76A444E0	5D	POP EBP	
76A444E1	C2 1000	RETN 10	
76A444E4	CC	INT3	
76A444E5	CC	INT3	
76A444E6	CC	INT3	
76A444E7	CC	INT3	
76A444E8	CC	INT3	
76A444E9	CC	INT3	
76A444EA	CC	INT3	
76A444E0=USER32.76A444E0			
Address	Hex dump	ASCII	
0040218E	64 6F 61 6E 33 00 00 00	doan3...	
00402196	00 00 00 00 00 00 00 00	.....	

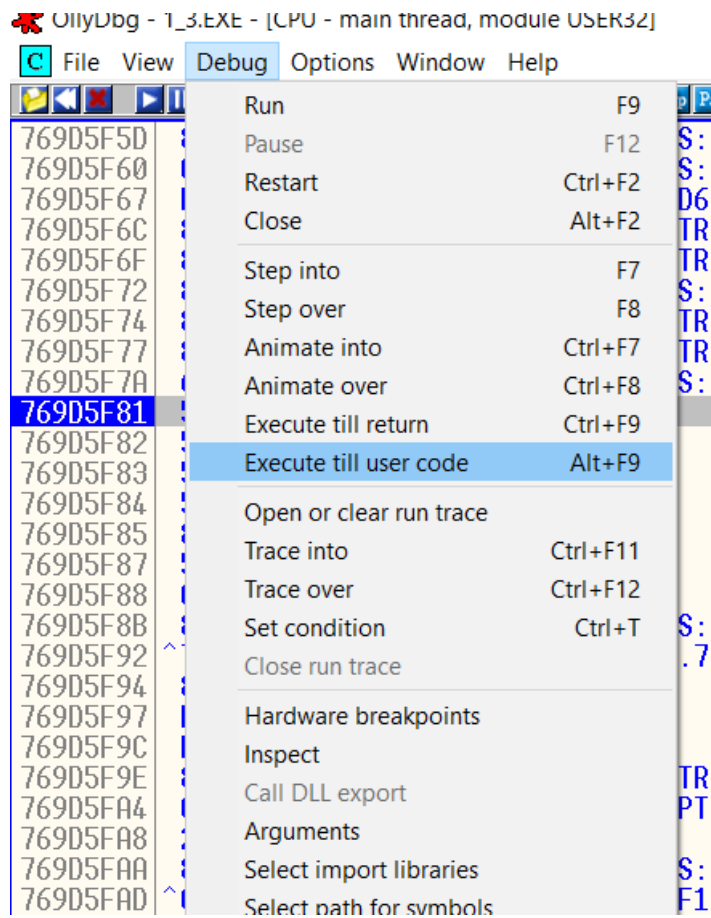
F8 đến câu lệnh này thì ta thấy Name đã được thêm vào .Chú ý đến thanh ghi EAX bằng 5 bằng với số kí tự của Name là Doan3.

Registers (FPU)
EAX 00000005
ECX 00000000

Tiếp tục F8 thu được giá trị của serial

0040217B	2C 20 0D 01 74 05 21 00	, mate!.
0040217E	31 32 33 34 35 00 00 00	12345...
00402186	00 00 00 00 00 00 00 00	.....
0040218E	64 6F 61 6E 33 00 00 00	doan3...

Sau khi tìm hiểu xong hàm GetDlgItemtextA ta sẽ quay trở lại code chính bằng cách



00401223	. 83F8 00	CMP EAX,0	
00401226	. 74 BE	JE SHORT 1_3.004011E6	
00401228	. 68 8E214000	PUSH 1_3.0040218E	ASCII "doan3"
0040122D	. E8 4C010000	CALL 1_3.0040137E	
00401232	. 50	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	
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_3.0040134D	
00401251	. EB 93	JMP SHORT 1_3.004011E6	
00401253	. C8 000000	ENTER 0,0	
00401257	. 53	PUSH EBX	

Ta nhận thấy sau khi PUSH name vào STACK thì sẽ thực hiện 1 lệnh Call,và sau khi PUSH Serial cũng thực hiện 1 lệnh Call.Ta tiến hành phân tích lệnh CALL 1\_3.0040137E trước

Khi F8 tới đoạn CALL ta nhấn F7 để vào trong hàm CALL

0040137E	8B 74 24 04	MOV ESI,DWORD PTR SS:[ESP+4]	1_3.0040218E
00401382	56	PUSH ESI	
00401383	> 8A 06	MOV AL,BYTE PTR DS:[ESI]	
00401385	84 C0	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 39 00 00 00	CALL 1_3.004013D2	
00401399	46	INC ESI	
0040139A	EB E7	JMP SHORT 1_3.00401383	
0040139B	5F	POP ESI	

Phân tích đoạn code trên như sau.

MOV dữ liệu từ STACK[ESP+4] mà cụ thể ở đây dữ liệu là Name vào ESI

PUSH ESI vào STACK

MOV 1 BYTE từ ESI vào AL(Thanh ghi Low của AX)

So sánh các Byte vừa lấy với 41.( 41 trong mã Hex = 65 trong mã Dec và là mã của Ký tự A trong ASCII)

So sánh các Byte vừa lấy với 5A.(5A trong mã Hex = 90 trong mã Dec và là mã ký tự Z trong ASCII).

Tóm tắt 1 cách dễ hiểu thì đoạn code trên có nhiệm vụ đọc từng ký tự trong chuỗi Name có nằm trong khoảng từ A->Z hay không.Nếu không thì sẽ CALL đến 1\_3.004013D2 để CONVERT

00402170	2C 20 00 01	74 03 21 00	, name: .	
0040217E	31 32 33 34	35 00 00 00	12345...	
00402186	00 00 00 00	00 00 00 00	.....	
0040218E	44 4F 41 4E	33 00 00 00	DOAN3...	

Sau khi CONVERT xong thì sẽ đến lệnh CALL

00401399	46	INC ESI	
0040139A	EB E7	JMP SHORT 1_3.00401383	
0040139C	> 5E	POP ESI	
0040139D	E8 20 00 00 00	CALL 1_3.004013C2	
004013A2	81 F7 78 56 00 00	XOR EDI,5678	

Nhấn F7 để xem lệnh CALL này làm gì

004013C2	\$ 33 FF	XOR EDI,EDI	EDI=0
004013C4	33 DB	XOR EBX,EBX	EBX=0
004013C6	> 8A 1E	MOV BL,BYTE PTR DS:[ESI]	MOV 1 BYTE ESI -> BL(LOW BX)
004013C8	84 DB	TEST BL,BL	BL AND BL. IF BL=0 -> ZF =1
004013CA	74 05	JE SHORT 1_3.004013D1	JE IF ZF =1
004013CC	03 FB	ADD EDI,EBX	EDI=EDI+EBX
004013CE	46	INC ESI	ESI+1
004013CF	EB F5	JMP SHORT 1_3.004013C6	JMP.JUMP K DIEU KIEN.VONG LAP
004013D1	C3	RETN	
004013D2	2C 20 00 01	SUB DI,20	

Đoạn code này làm nhiệm vụ cộng dồn toàn bộ các ký tự trong chuỗi Name lại vào lưu vào thanh ghi edi.

```

EBP 0019FDDC
ESI 0040218E ASCII "DOAN3"
EDI 00000111

```

111(Hex)=273(Dec)

Sau khi kết thúc hàm CALL

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 SS:[EBP+8]	
004013BC	. E8 79000000	CALL <JMP.&USER32.MessageBoxA>	Style = MB_OK MB_ICONEXCLAMATION MB_APPLMODAL Title = "No luck!" Text = "No luck there, mate!" hOwner = 00130BB6 ('CrackMe v1.0',class='No need to disasm the code!')
004013C1	. C3	RETN	MessageBoxA
004013C2	. 33FF	XOR EDI,EDI	EDX=0

Giá trị EDI sẽ được XOR với 5678.Kết quả thu được sẽ MOV vào EAX

0040122D	. E8 4C010000	CALL 1_3.0040137E	
00401232	. 50	PUSH EAX	
00401233	. 68 7E214000	PUSH 1_3.0040217E	
00401238	. E8 9B010000	CALL 1_3.004013D8	ASCII "12345"
0040123D	. 83C4 04	ADD ESP,4	
00401240	. 58	POP EAX	

Kết thúc hàm Call xử lí Name.Kết quả sau khi được lưu vào EAX sẽ được push lên STACK

Tiếp tục PUSH Serial lên STACK và gọi 1 hàm Call để xử lí Serial

Nhấn F7 để vào hàm CALL này

004013D8	. 33C0	XOR EAX,EAX	EAX = 0
004013DA	. 33FF	XOR EDI,EDI	EDI =0
004013DC	. 33DB	XOR EBX,EBX	EBX =0
004013DE	. 8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]	ESI = SERIAL
004013E2	> B0 0A	MOV AL,0A	AL = 0A
004013E4	. 8A1E	MOV BL,BYTE PTR DS:[ESI]	MOV 1 BYTE FROM ESI -> BL
004013E6	. 84DB	TEST BL,BL	AND BL,BL. IF BL = 0 -> FLAG ZF =1
004013E8	. 74 0B	JE SHORT 1_3.004013F5	EXIT LOOP
004013EA	. 80EB 30	SUB BL,30	BL = BL -30
004013ED	. 0FAFF8	IMUL EDI,EAX	EDI = EDI *EAX
004013F0	. 03FB	ADD EDI,EBX	EDI = EDI + EBX
004013F2	. 46	INC ESI	ESI++
004013F3	. EB ED	JMP SHORT 1_3.004013E2	
004013F5	> 81F7 34120000	XOR EDI,1234	
004013FB	. 8BDF	MOV EBX,EDI	
004013FD	. C3	RETN	

Đoạn code trên có nhiệm vụ Chuyển chuỗi Serial về dạng Hex và lưu vào thanh ghi EDI .

Sau đó sẽ đem EDI XOR với 1234 rồi lưu vào EBX

```

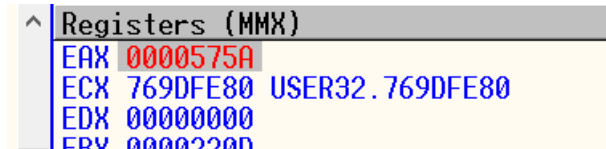
EDI 0000220D

```

Sau đó trở về lại màn hình chính và tới phần so sánh giữa EAX và EBX

00401238	. E8 9B010000	CALL 1_3.004013D8
0040123D	. 83C4 04	ADD ESP,4
00401240	. 58	POP EAX
00401241	. 3BC3	CMP EAX,EBX
00401243	. 74 07	JE SHORT 1_3.0040124C
00401245	. E8 18010000	CALL 1_3.00401362
00401248	. 5B	MOV EBX,EBX

Bây giờ ta sẽ tìm Serial chính xác với chuỗi Name vừa nhập. Vì lúc ta nhập test là doan3 có kí tự 3 không thuộc A-Z nên sẽ bị thông báo lỗi. Nên ta sẽ kiểm tra lại sau với Name là doan. Và sẽ chạy lại chương trình để tìm lại giá trị EAX. Giá trị của EAX khi có Name là doan là

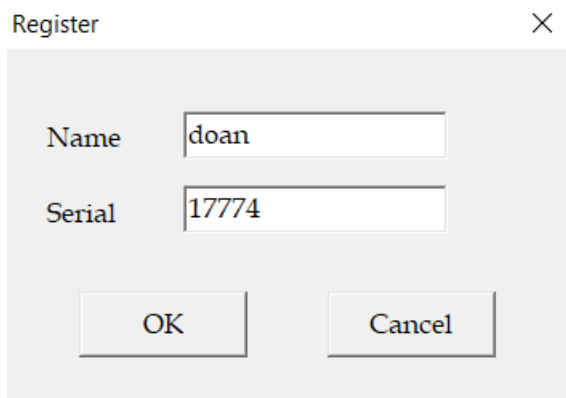


575A(HEX)

Serial của chúng ta sẽ là giá trị tính toán được của chuỗi Name và đem xor với 0x1234

Lấy 575A XOR 1234 = 456e( Chuyển sang Dec = 17774)

Thử chạy lại chương trình với Name doan và Serial 17774



Ta nhận được thông báo thành công

CHÚ Ý: Dữ liệu Name phải là kí tự chữ và có độ dài < 11

