



# KIẾN TRÚC MÁY TÍNH & HỢP NGỮ



## MÔ TẢ ĐỒ ÁN

### CRACKING



#### Mục lục

1. Thành viên .....	2
2. Nội dung .....	3



## 1. Thành viên

STT	MSSV	Họ và tên	Email
1	1712930	Trần Văn Vỹ	1712930@student.hcmus.edu.vn
2	1712905	Nguyễn Hoàng Việt	1712905@student.hcmus.edu.vn
3	1712891	Trần Thúy Tuyền	1712891@student.hcmus.edu.vn

## 2. Nội dung

**Đề = [số cuối (1712930 + 1712905 + 1712891) mod 3] + 1 = [6 mod 3] + 1 = 1**

### Đề 1

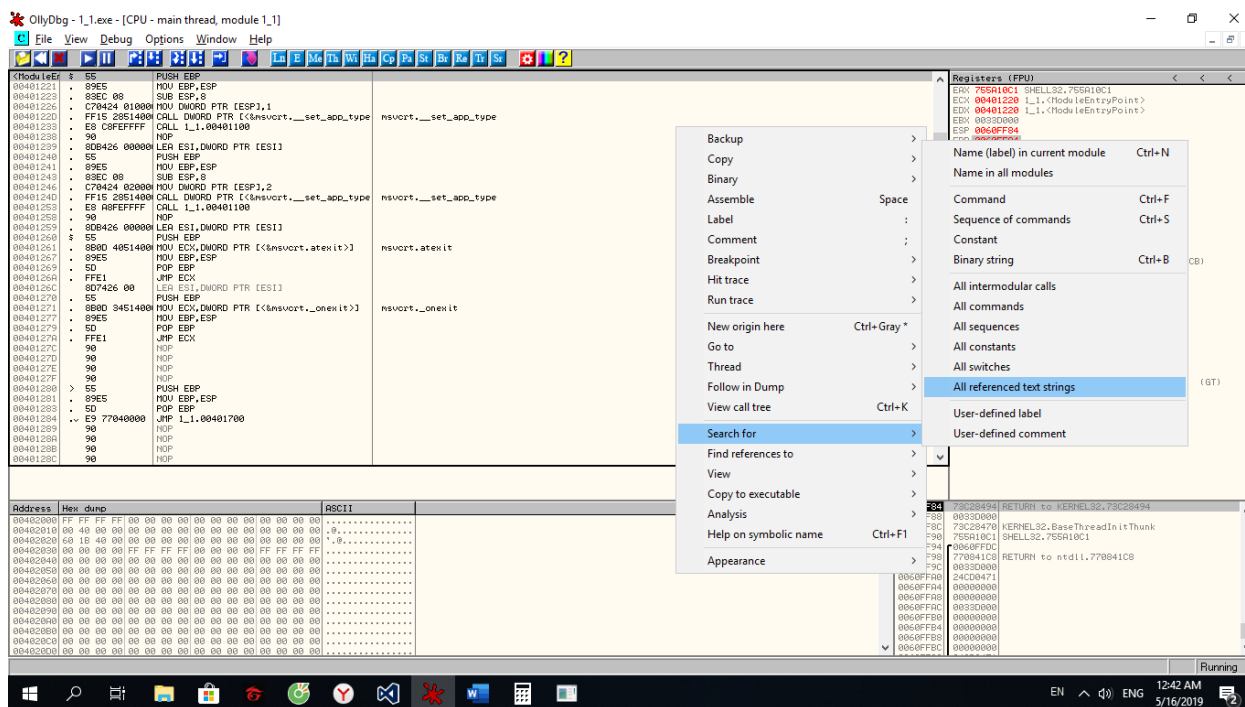
#### 2.1. Bài 1: tutotial cracking 1\_1

Chạy OllyDbg.exe

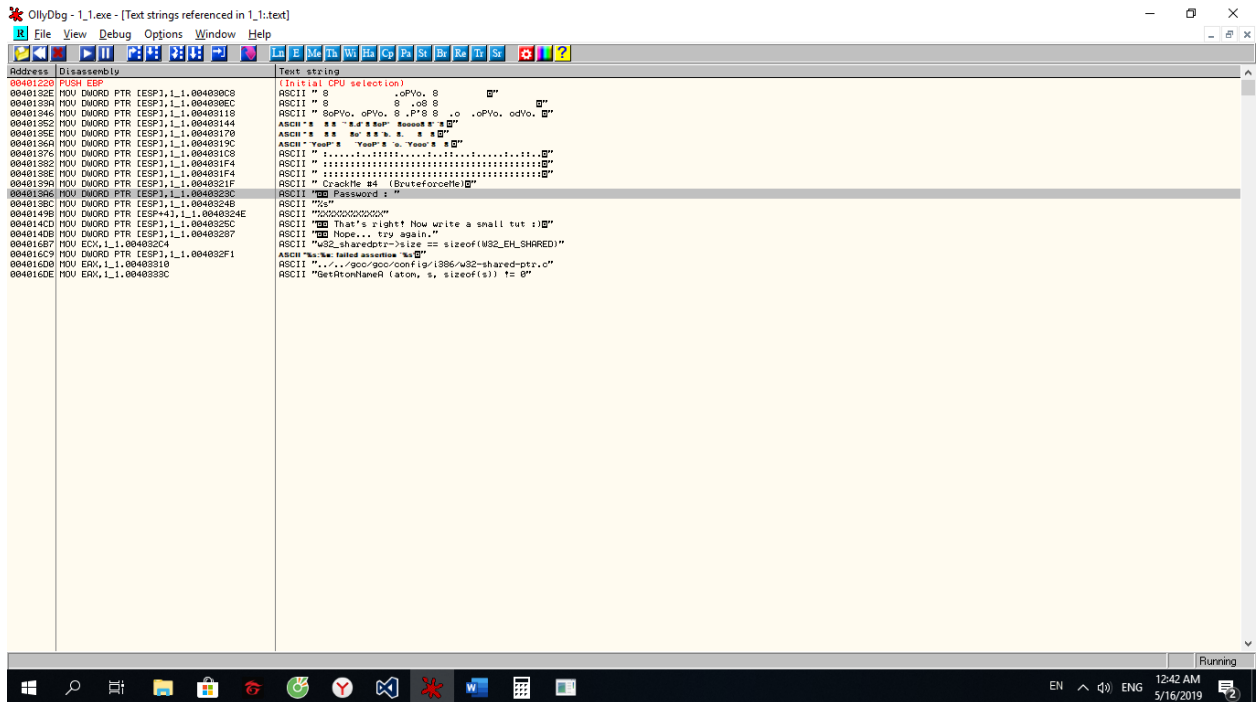
Từ ứng dụng mở file 1\_1.exe

Ấn F9 để chạy chương trình lần đầu

Click chuột phải trong cửa sổ CPU trong Olly chọn Search for tiếp tục chọn All referenced text string

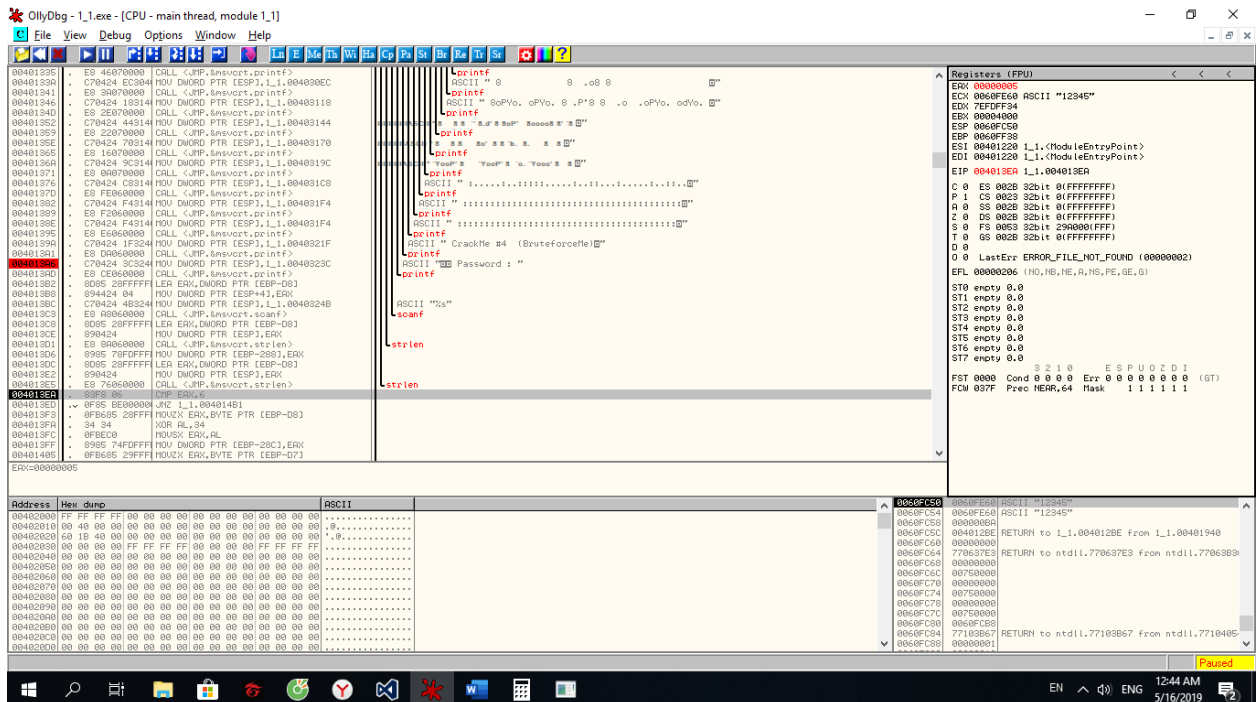


Ấn đúp vào dòng chữ “password:” để trở lại màn hình CPU



Sau đó ta đặt Break Point(BP) tại vị trí đó và ấn F9 một lần nữa. Chương trình sẽ chạy đến vị trí ta đã đặt BP, sau đó ta tiếp tục bấm F8 khi chương trình yêu cầu nhập mật khẩu, ta chọn một mật khẩu tùy ý, chọn “12345” và bấm F8 chạy tiếp các bước tiếp theo.

Tới chỗ cột 3 của CPU xuất hiện câu lệnh: `CMP EAX,6` ta nhìn bảng Register bên phải hiện tại giá trị của EAX là 5 tương ứng với số kí tự trong password => ta biết được mật khẩu có 6 kí tự.



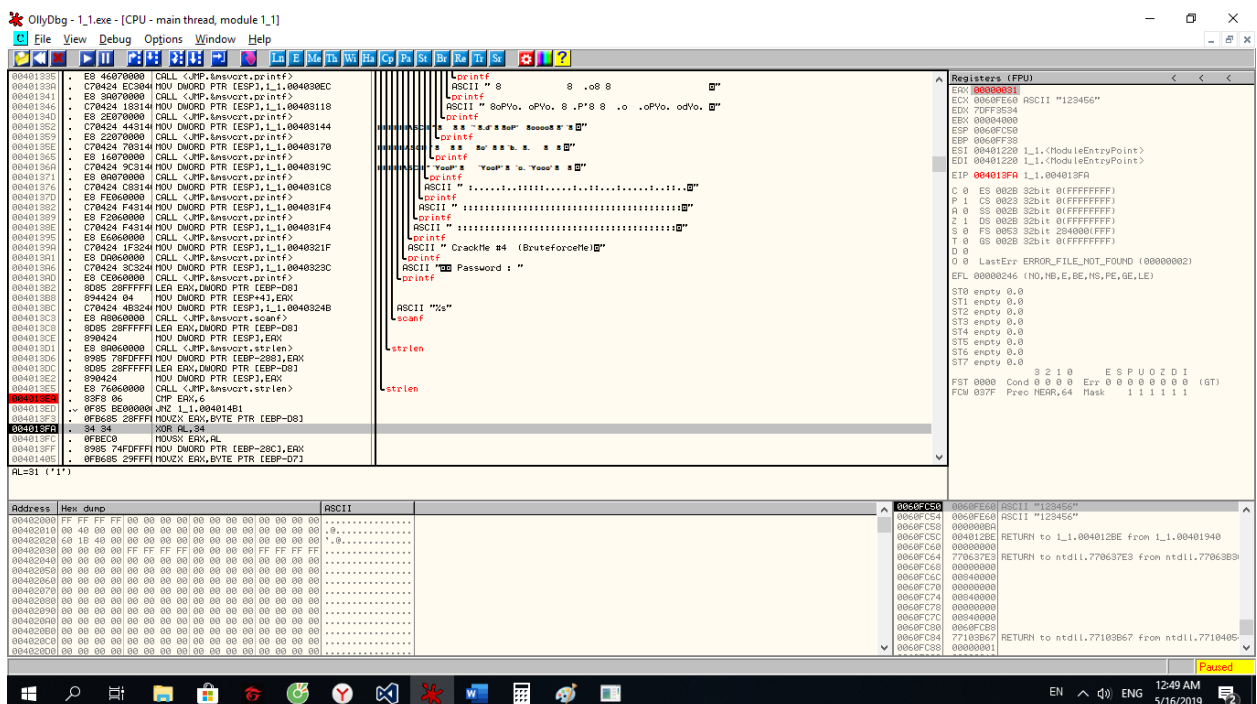


```
Registers (FPU)
EAX 00000005
ECX 0060FE60 ASCII "12345"
EDX 7EFDFF34
EBX 00004000
ESP 0060FC50
EBP 0060FF38
ESI 00401220 1_1.<ModuleEntryPoint>
EDI 00401220 1_1.<ModuleEntryPoint>
EIP 004013EA 1_1.004013EA

C 0 ES 002B 32bit 0(FFFFFFFF)
P 1 CS 0023 32bit 0(FFFFFFFF)
A 0 SS 002B 32bit 0(FFFFFFFF)
Z 0 DS 002B 32bit 0(FFFFFFFF)
S 0 FS 0053 32bit 29A000(FFF)
T 0 GS 002B 32bit 0(FFFFFFFF)
D 0
O 0 LastErr ERROR_FILE_NOT_FOUND (00000002)
EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
```

Ta nhấn Ctrl+F2 để chạy lại chương trình lần này ta đặt BP tại vị trí có câu lệnh CMP EAX,6 . Lần này ta nhập mật khẩu là "123456", bấm F9 cho chương trình chạy tới BP.

Sau đó nhấn F8 đến lệnh XOR AL,34 ta thấy giá trị của của EAX là 31(Hex) tương ứng với kí tự "1" (ASCII) đầu tiên của chuỗi mật khẩu. Lệnh XOR AL,34 thực hiện lấy 31 XOR 34 = 5(Hex) sau đó lưu giá trị có được vào địa chỉ [EBP-28C] (hay 60FF38-28C=60FCAC).



Nhấn F8 đến lệnh XOR AL,78 giá trị của EAX là 32 tương ứng với "2"(ASCII) kí tự thứ 2 của chuỗi. Lệnh XOR AL,78 = 4A(Hex) sau đó lưu giá trị có được vào địa chỉ [EBP-290]



OllyDbg - 1\_1.exe - [CPU - main thread, module 1\_1]

Registers (FPU)

EAX 00000033  
ECX 000FE608 ASCII "123456"  
EDX 7DFF3E34  
EBX 00004000  
ESP 000FCF50  
EBP 0000FF38  
ESI 00401220 1\_1.<ModuleEntryPoint>  
EDI 00401220 1\_1.<ModuleEntryPoint>  
EIP 0040140C 1\_1.0040140C  
C 0 ES 002B 32bit 0xFFFFFFFF  
P 1 CS 002B 32bit 0xFFFFFFFF  
A 0 SS 002B 32bit 0xFFFFFFFF  
Z 0 DS 002B 32bit 0xFFFFFFFF  
O 0 FS 0053 32bit 20400001FFF  
T 0 GS 002B 32bit 0xFFFFFFFF  
0 0 LastErr ERROR\_FILE\_NOT\_FOUND (00000002)  
EFL 00000206 (NO,NB,NE,A,HS,PE,SE,G)  
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  
FST 0000 Cnd 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)  
FDM B37F Prec NEAR,64 Flask 1 1 1 1 1 1

Address Hex dump ASCII

00402000 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402010 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402020 60 18 40 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402030 00 00 00 FF FF FF FF 00 00 00 00 FF FF FF FF  
00402040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Nhấn F8 đến lệnh XOR AL,12 giá trị của EAX là 33 tương ứng “3”(ASCII) kí tự thứ 3 của chuỗi.

Lệnh XOR AL,12=21(Hex) sau đó lưu giá trị có được vào địa chỉ [EBP-294]

OllyDbg - 1\_1.exe - [CPU - main thread, module 1\_1]

Registers (FPU)

EAX 00000033  
ECX 000FE608 ASCII "123456"  
EDX 7DFF3E34  
EBX 00004000  
ESP 000FCF50  
EBP 0000FF38  
ESI 00401220 1\_1.<ModuleEntryPoint>  
EDI 00401220 1\_1.<ModuleEntryPoint>  
EIP 0040141E 1\_1.0040141E  
C 0 ES 002B 32bit 0xFFFFFFFF  
P 0 CS 002B 32bit 0xFFFFFFFF  
A 0 SS 002B 32bit 0xFFFFFFFF  
Z 0 DS 002B 32bit 0xFFFFFFFF  
O 0 FS 0053 32bit 20400001FFF  
T 0 GS 002B 32bit 0xFFFFFFFF  
0 0 LastErr ERROR\_FILE\_NOT\_FOUND (00000002)  
EFL 00000206 (NO,NB,NE,A,HS,PE,SE,G)  
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  
FST 0000 Cnd 0 0 0 0 Err 0 0 0 0 0 0 0 0 (GT)  
FDM B37F Prec NEAR,64 Flask 1 1 1 1 1 1

Address Hex dump ASCII

00402000 FF FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402010 00 40 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402020 60 18 40 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402030 00 00 00 FF FF FF FF 00 00 00 00 FF FF FF FF  
00402040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402080 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
00402090 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  
004020D0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Tương tự với kí tự thứ tư “4”(ASCII) thực hiện lệnh XOR AL,0FE=CA(Hex) lưu vào địa chỉ [EBP-298]

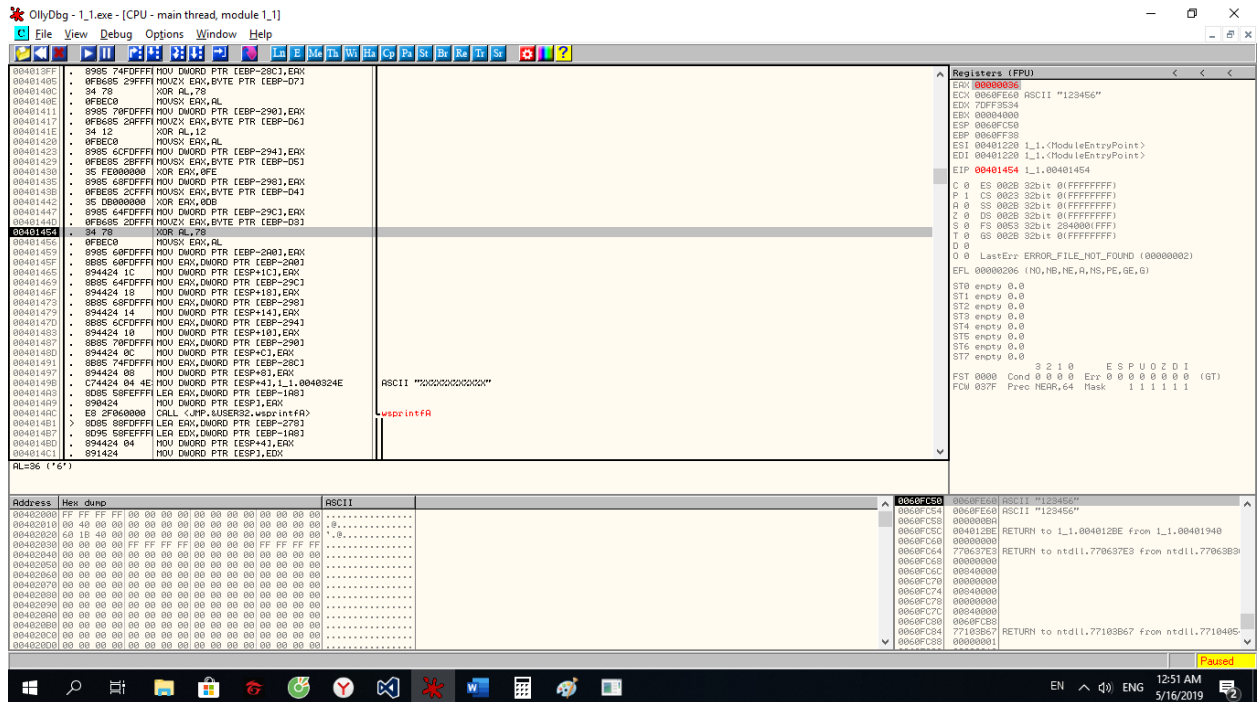


[illegible]

Tương tự với kí tự thứ năm “5”(ASCII) thực hiện lệnh XOR AL,DB=**EE**(Hex) lưu vào địa chỉ[EBP -29C]

The screenshot shows OllyDbg running on a Windows 7 desktop. The main window displays assembly code for a program named '01.exe'. The assembly window shows instructions such as `C78424 9C3241 MOV DWORD PTR [ESP], L_1.0040323C` and `8005 C06000 CALL <MP.8movect,printf>`. The registers window shows `EAX=00000035`, `ECX=006FCE50`, and `ESI=00401220`. The stack window shows a hex dump of memory starting at address 00402000.

Tương tự với kí tự thứ sáu “6” (ASCII) thực hiện lệnh XOR AL,78=4E(Hex) lưu vào địa chỉ [EBP-2A0]



5 4A 21 CA EE 4E

Từ các bảng mã Hex ta chuyển về dạng ASCII “54A21CAEE4E”

Nhấn F8 tới câu lệnh TEST EAX,EAX tại đây so sánh 2 chuỗi là

0060FC48	0060FF38	
0060FC4C	004014C9	1_1.004014C9
0060FC50	0060FD90	ASCII "54A21CAEE4E"
0060FC54	0060FCC0	ASCII "4D11628EBE10"
0060FC58	00000005	
0060FC5C	0000004A	
0060FC60	00000021	
0060FC64	000000CA	
0060FC68	000000EE	
0060FC6C	0000004E	
0060FC70	00000000	
0060FC74	00840000	
0060FC78	00000000	
0060FC7C	00840000	
0060FC80	0060FCB8	

“54A21CAEE4E” (chuỗi chuyển đổi từ mật khẩu)

“4D11628EBE10” (chuỗi cho sẵn)

⇒ Để có mật khẩu đúng ta cần biến đổi sao cho chuỗi đó trùng với chuỗi cho sẵn.

Ta thấy trong chuỗi cho sẵn “4D11628EBE10” có 12 ký tự nên ta chia đều chúng cho 6 ta được 2. Vậy mỗi ký tự trong mật khẩu tương ứng với 2 ký tự trong mã cho sẵn.

Từ đây ta truy ngược lại từ lệnh XOR.

1. 4D XOR 34=79(hex) => ký tự “y”(ASCII)

2. 11 XOR 78= 69(Hex) => kí tự “i” (ASCII)
3. 62 XOR 12= 70(hex) => kí tự “p” (ASCII)
4. 8E XOR 0FE=70(Hex) => kí tự “p” (ASCII)
5. BE XOR DB=65(Hex)=> kí tự “e” (ASCII)
6. 10 XOR 78=65(hex) => kí tự “e”(ASCII)

Vậy mật khẩu cần tìm là “yippee”

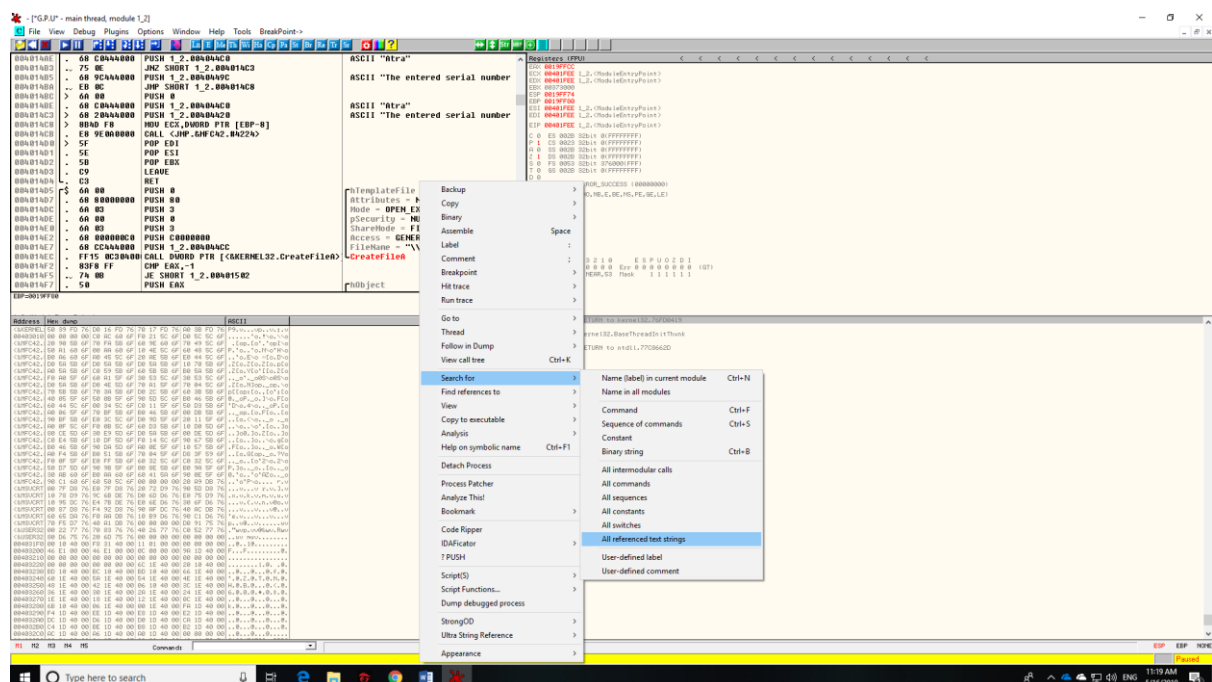
## 2.2. Bài 2: tutorial cracking 1\_2

Chạy OllyDbg.exe

Từ ứng dụng mở file 1\_2.exe

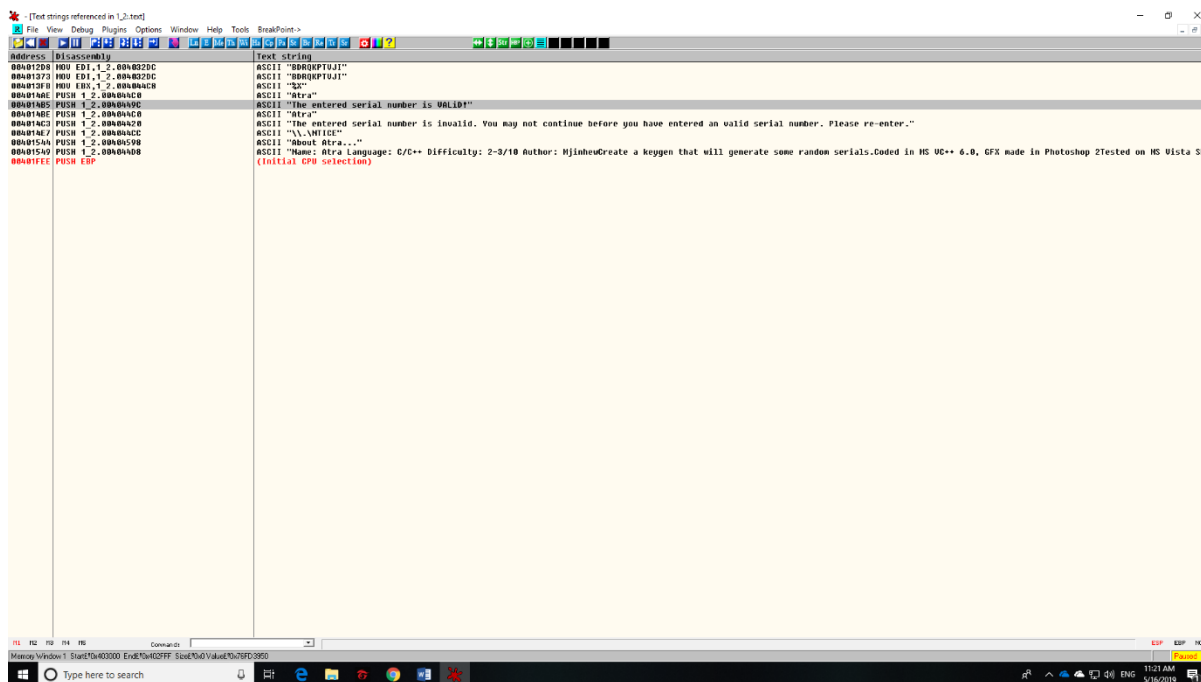
Ấn F9 để chạy chương trình lần đầu

Click chuột phải trong cửa sổ CPU trong Olly chọn Search for tiếp tục chọn All referenced text string



Ta để ý đến dòng chữ “The entered serial number is VALID!” Ấn đúp vào dòng chữ để trở lại màn hình CPU





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"
004014BA	- EB 0C	JMP SHORT 1_2.004014C8	

Dò lên phía trên ta thấy dòng JNZ Short 1\_2.004014C3 nhảy đến dòng chữ báo sai Serial Number!

Vậy điều kiện để nhảy đến là gì? JNZ : Jump if not zero (điều kiện nhảy là cờ Z = 0).

Dịch lên 3 dòng, ta thấy lệnh TEST EAX,EAX (đây là lệnh kiểm tra EAX = 0, nếu EAX = 0, ZF = 1 => điều ta MUỐN)

Vậy khi nào thì EAX = 0, để ý thấy dòng trên nữa ta có hàm Strcmp, hàm này trả về EAX = 0 khi hai chuỗi so sánh bằng nhau!

Từ đó dự đoán, ta phải so sánh chuỗi của ta, với một chuỗi có sẵn, xem chúng có bằng nhau hay không!

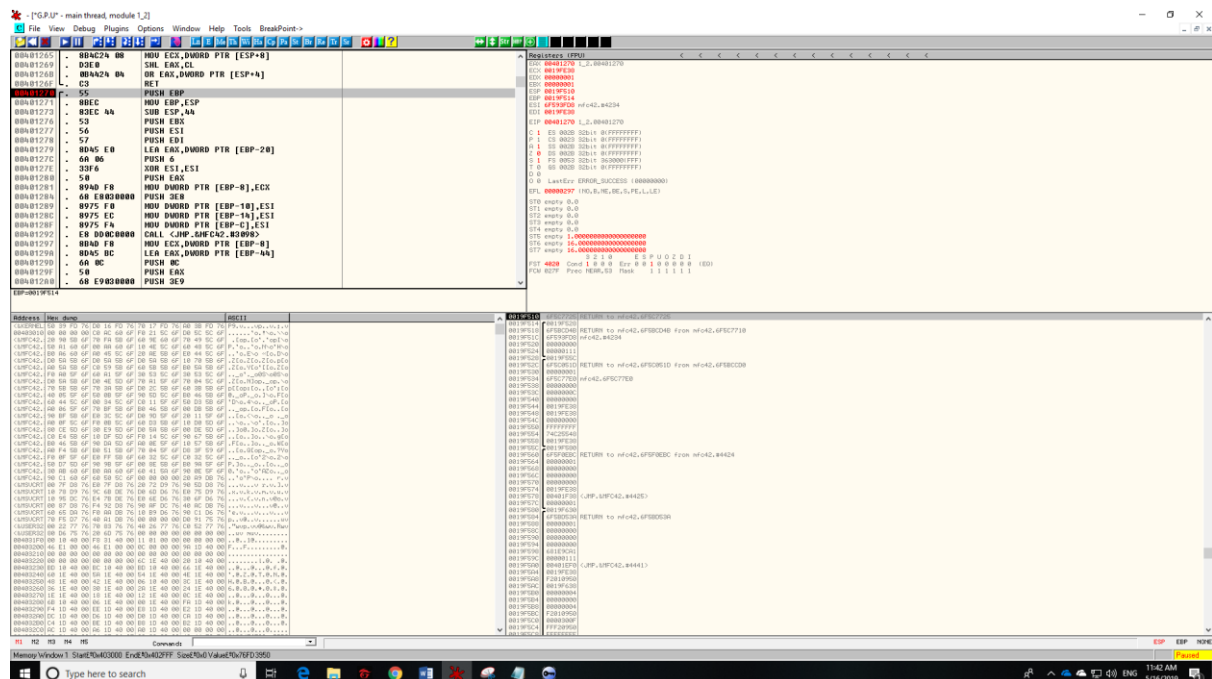
Tuy nhiên, sau nhiều lần mò chạy thử (Đã đặt Breakpoint tại đó), nhưng chương trình vẫn không xét được tới đó, vậy... có thể là cái chuỗi CÓ SẴN đó, thật chất không phải có sẵn, mà là được tạo ra từ một cái gì đó, có thể? Là serial number ta nhập vào!

Chú ý thấy khi mở chương trình 1\_2.exe lên, Serial Number ta nhập vào có hai phần: tạm gọi là P1 và P2. P1 có 5 chữ số và P2 có 8 chữ số.

À, có thể đây là mấu chốt để ta có thể tìm thấy đoạn kiểm tra từng phần Serial Number ta nhập vào.



Quay lên phía trên trên xa xa tận cùng kia, ta thử đặt Breakpoint tại một vị trí.



Và nhấn chạy chương trình. Oh, được này.

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 E8030000	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]

Nhấn F8 để theo dõi quá trình thực thi của chương trình.

Ta để ý sau khi qua khỏi dòng Call <JMP.&MFC42.#3098> EAX trả về 5.

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
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 E8030000	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

**Registers (FPU)**  
EAX 00000005  
ECX 76759646 user32.76759646  
EDX 00000000  
EBX 00000001  
ESP 0019F4BC  
EBP 0019F50C  
ESI 00000000  
EDI 0019FE38  
EIP 00401297 1.2.00401297  
C 0 ES 002B 32bit 0(FFFFFFFF)  
P 0 CS 002B 32bit 0(FFFFFFFF)  
A 1 SS 002B 32bit 0(FFFFFFFF)  
Z 0 DS 002B 32bit 0(FFFFFFFF)  
S 0 FS 005B 32bit 0(FFFFFFFF)  
T 0 GS 002B 32bit 0(FFFFFFFF)  
D 0  
O 0  
LastErr ERROR\_SUCCESS (00000000)  
EFL 00000212 (NO, NB, NE, L, PE, LE)  
ST0 empty 0.0  
ST1 empty 0.0  
ST2 empty 0.0  
ST3 empty 0.0  
ST4 empty 0.0  
ST5 empty 1.000000000000000000000000  
ST6 empty 16.000000000000000000000000  
ST7 empty 16.000000000000000000000000  
FPU 4820 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EO)  
FCW 027F Prec NEAR, 53 Mask 1 1 1 1 1 1

Điều này khiến ta liên tưởng ngay đến, số lượng ký tự của P1!

Vậy có thể đây là dòng lấy P1 của Serial Number ta nhập vào.

Tiếp tục F8 nào.

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 E8030000	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 D00C0000	CALL <JMP.&HFC42.03098>	
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.&HFC42.03098>	
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	
004012BB	E8 1C000000	CALL <JMP.&MSUCRT.strlen>	[5 strlen
004012C0	85C0	TEST EAX,EAX	
004012C2	59	POP ECX	
004012C3	0F86 92000000	JBE 1_2.0040135B	

Registers (FPU)	
EAX	00000000
ECX	76763646 user32.76759646
EDX	00000000
EBX	00000001
ESP	0019F40C
EBP	0019F50C
ESI	00000000
EDI	0019FE38
EIP	0040129A 1_2.0040129A
C 0	ES 0028 32bit 0(FFFFFFFF)
P 0	CS 0023 32bit 0(FFFFFFFF)
A 1	SS 002B 32bit 0(FFFFFFFF)
Z 0	DS 0028 32bit 0(FFFFFFFF)
S 0	FS 0053 32bit 363000(FFF)
T 0	GS 002B 32bit 0(FFFFFFFF)
O 0	
0 0	LastErr ERROR_SUCCESS (00000000)
EFL	00000212 (NO,NB,NE,A,NS,PO,GE,G)
ST0	empty 0.0
ST1	empty 0.0
ST2	empty 0.0
ST3	empty 0.0
ST4	empty 0.0
ST5	empty 1.000000000000000000
ST6	empty 16.000000000000000000
ST7	empty 16.000000000000000000
3 2 1 0 E S P U O Z D I	
FST 4020	Cond 1 0 0 0 Err 0 0 1 0 0 0 0 (EQ)
FCW 027F	Prec NEAR,S3 Mask 1 1 1 1 1 1

Dự đoán của ta đã đúng, khi phía dưới cũng có dòng Call tương tự, nhưng EAX trả về lần này là 8!

Vậy đây chắc chắn là dòng lấy P2 của Serial Number ta nhập vào.

Suy nghĩ một chút, vậy dự đoán ban đầu, P1 và P2 được chia ra để xử lý là ĐÚNG.

Vậy vấn đề bây giờ là tìm ra đoạn xử lý của từng Part!

Tiếp tục F8,

Ta thấy xuất hiện lệnh gọi hàm strlen (kiểm tra độ dài chuỗi) và dễ nhận thấy là nó đang kiểm tra độ dài P1 ta nhập vào và trả về độ dài lưu ở EAX.

00401361	50	PUSH EAX	
00401362	E8 750C0000	CALL <JMP.&MSUCRT.strlen>	[5 strlen
00401367	85C0	TEST EAX,EAX	
00401369	59	POP ECX	
0040136A	0F86 4C010000	JBE 1_2.004014BC	

JBE: Jump if Below or Equal (nhảy nếu cờ C = 1 hay Z = 1), trước đó ta có lệnh TEST EAX,EAX (bật cờ Z = 1 nếu EAX = 0)

Lệnh nào nhảy đến thông báo sai Serial Number, vậy tới đây nếu ta không nhập ký tự nào ở P1 thì sẽ bị báo lỗi ngay lập tức!

Có thể dự đoán việc kiểm tra này nhằm nhận biết chuỗi nhập vào rỗng -> ta sẽ không cần kiểm tra độ hợp lệ của P1.

00401370	8D5D E0	LEA EBX,DWORD PTR [EBP-20]	
00401373	BF DC324000	MOV EDI,1_2.004032DC	ASCII "BDRQKPTVJI"
00401378	57	PUSH EDI	[5
00401379	33F6	XOR ESI,ESI	
0040137B	E8 5C0C0000	CALL <JMP.&MSUCRT.strlen>	strlen
00401380	85C0	TEST EAX,EAX	

Đoạn trên ta xét độ dài chuỗi có sẵn trong memory 004032DC "BDRQKPTVJI"

EAX trả về = 10.

Và tất nhiên vì là chuỗi có sẵn thì EAX làm gì có thể = 0, trừ khi có người can thiệp vào.

Tiếp theo ta đi vào đoạn loop.

00401383	< 76 35	JBE SHORT 1_2.0040138A	
00401385	> 8A 03	MOV AL, BYTE PTR [EBX]	
00401387	< 3A 86 DC 32 40 00	CMP AL, BYTE PTR [ESI+4032DC]	
0040138D	< 75 05	JNZ SHORT 1_2.00401394	
0040138F	< FF 45 F0	INC DWORD PTR [EBP-10]	
00401392	< EB 1A	JMP SHORT 1_2.004013AE	
00401394	> 3A 86 DC 32 40 00	CMP AL, BYTE PTR [ESI+4032D0]	
0040139A	< 75 05	JNZ SHORT 1_2.004013A1	
0040139C	< FF 45 EC	INC DWORD PTR [EBP-14]	
0040139F	< EB 0D	JMP SHORT 1_2.004013AE	
004013A1	> FF 45 F4	INC DWORD PTR [EBP-C]	
004013A4	< 83 7D F4 2F	CMP DWORD PTR [EBP-C], 2F	
004013A8	< 0F 87 0E 01 00 00	JA 1_2.004014BC	
004013AE	> 57	PUSH EDI	[ <sup>5</sup> strlen
004013AF	< 46	INC ESI	
004013B0	< E8 27 0C 00 00	CALL <JMP.&MSUCRT.strlen>	
004013B5	< 3B F0	CMP ESI, EAX	
004013B7	< 59	POP ECX	
004013B8	< 72 CB	JB SHORT 1_2.00401385	

EBX lúc này đang giữ giá trị ASCII của P1 ta nhập vào.

MOV AL, BYTE PTR [EBX] lệnh này có nghĩa là, lấy 1 byte của EBX lưu vào trong byte thấp của EAX.

Registers (FPU)	
EAX	0000000A
ECX	004032DC ASCII "BDRQKPTVJI"
EDX	7EFF4849
EBX	0019F4EC ASCII "12"
ESP	0019F4BC
EBP	0019F50C
ESI	00000000
EDI	004032DC ASCII "BDRQKPTVJI"
EIP	00401385 1_2.00401385

EAX đang giữ giá trị A. (như đã tính ở trên).

Sau khi chạy lệnh MOV.

EAX	00000031
ECX	004032DC ASCII "BDRQKPTVJI"
EDX	7EFF4849
EBX	0019F4EC ASCII "12"
ESP	0019F4BC
EBP	0019F50C
ESI	00000000
EDI	004032DC ASCII "BDRQKPTVJI"

Byte thấp của EAX giữ giá trị ASCII Code (ví dụ ở đây: số 1 có ASCII Hex Code = 31).

Lúc này ta thấy phía dưới là lệnh CMP AL, BYTE PTR [ESI+4032DC], bởi vì ESI đã XOR với chính nó ở lệnh phía trên, nên lúc này ESI = 0. Mà 4032DC chính là vị trí thanh ghi đang giữ giá trị chuỗi "BDRQKPTVJI"

Nên có thể hiểu lệnh CMP ở đây là so sánh xem if ( P1[counter1] == [counter2 + 4032DC] ).

Hai dòng tiếp theo: JNZ SHORT 1\_2.00401394



## INC DWORD PTR [EBP - 10]

JNZ: Jump if not Zero, mà lệnh CMP chỉ trả về Zero khi 2 chuỗi so sánh bằng nhau.

Có thể hiểu, ở đây là nếu so sánh, trong P1 ta nhập vào có kí tự giống với kí tự trong chuỗi cho sẵn, thì ta cộng cái gì đó thêm 1 đơn vị. Vậy đó có thể là gì?

```
004013CD | . 837D F0 03 | CMP DWORD PTR [EBP-10],3
004013D1 | ~ 0F85 E5 00 00 | JNZ 1.2.004014BC
004013D7 | . 837D EC 02 | CMP DWORD PTR [EBP-14],2
004013DB | ~ 0F85 D8 00 00 | JNZ 1.2.004014BC
```

Đề ý sau đoạn loop, ta có so sánh [EBP - 10] với 3. Và nhảy đến đoạn báo lỗi key nếu không bằng nhau.

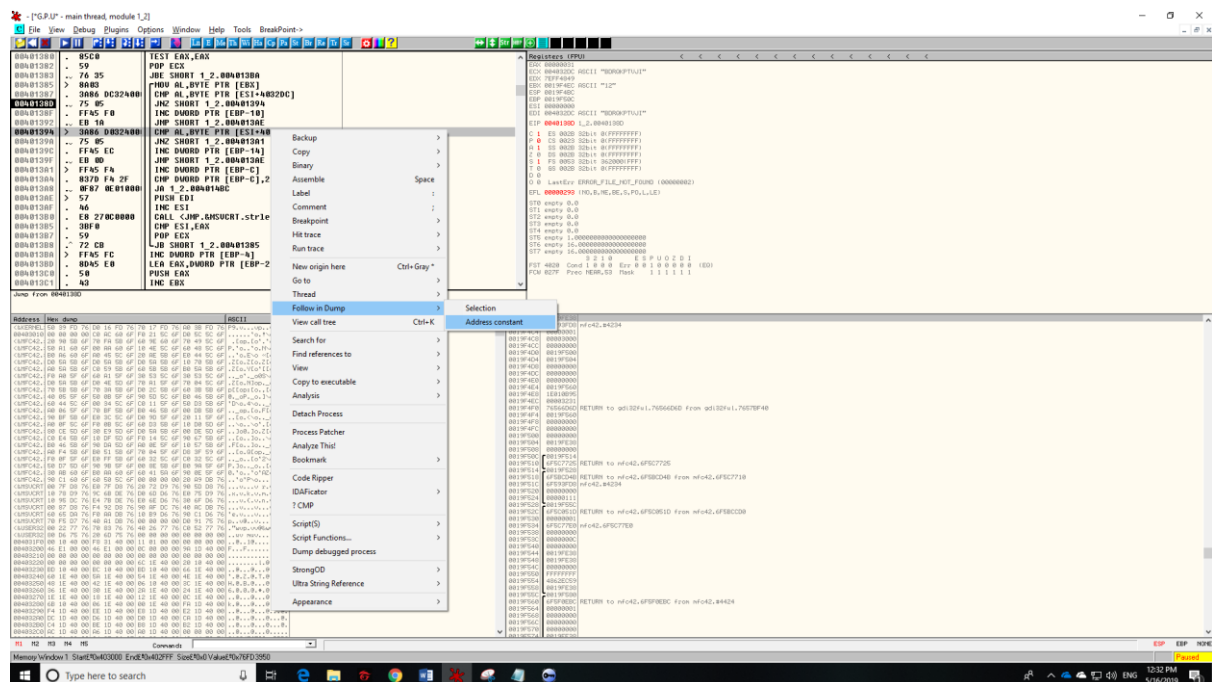
Vậy có thể đây là một biến đếm. số lượng ký tự bằng nhau với chuỗi trong 4032DC chẳng hạn?

Mà  $3 + 2 = 5$ , vậy có thể đoạn loop trên chính là kiểm tra xem, trong chuỗi P1, ta có đúng 3 ký tự bất kì trong chuỗi “BDRQKPTVJI” hay không.

Và ta phải đúng 2 ký tự nữa trong 1 chuỗi có sẵn khác? Vậy chuỗi đó là gì?

Nhìn lại sang hình đoạn Loop, ta thấy có xuất hiện memory 4032D0.

Kiểm tra trong 4032D0 đang chứa chuỗi gì bằng cách dưới đây.



Và ta biết được đó là chuỗi “012345679” !

Kết luận, P1 ta phải có 3 ký tự trong chuỗi “BDRQKPTVJI” và 2 ký tự trong chuỗi “0123456789”.

Rồi, xử lý xong P1. Ta đã biết được tính hợp lệ của P1.





Giờ chuyển sang tìm hiểu xem, P2 được sinh ra như thế nào?

Ta chạy lại chương trình, nhập vào 5 ký tự P1 với tiêu chuẩn đã tìm ra ở trên.

004013E4	8045 E0	LEA EAX,DWORD PTR [EBP-20]		Registers (FPU)
004013E5	50	PUSH EAX	[5	EAX 00000000
004013E6	E8 F20B0000	CALL <JMP.&MSUCRT.strlen>	strlen	EAX 0019F4EC ASCII "12B0T"
004013E7	50	PUSH EAX		EDX F554FF53
004013E8	8045 E0	LEA EAX,DWORD PTR [EBP-20]		EBX 0019F4F1
004013E9	50	PUSH EAX		ESP 0019F4B0
004013EA	E8 6A090000	CALL 1_2.00401D5E		EBP 0019F50C
004013EB	8B3D 80314000	MOV EDI,DWORD PTR [<&MSUCRT.sprintf>]	msvcrt.sprintf	ESI 00000000
004013EC	50	PUSH EAX		EDI 004032DC ASCII "BDR0KPTUJI"
004013ED	BB C8444000	MOV EBX,1_2.004044C8	<%X>	EIP 004013E1 1_2.004013E1
004013EE	8045 D4	LEA EAX,DWORD PTR [EBP-2C]	ASCII "%X"	C 0 ES 002B 32bit 0(FFFFFFFF)
004013EF	53	PUSH EBX	Format => "%X"	P 1 CS 0023 32bit 0(FFFFFFFF)
004013F0	50	PUSH EAX	S	A 0 SS 002B 32bit 0(FFFFFFFF)
004013F1	FFD7	CALL EDI	sprintf	Z 1 DS 002B 32bit 0(FFFFFFFF)
004013F2	8365 FC 00	AND DWORD PTR [EBP-4],0		S 0 FS 0053 32bit 362000(FFF)
004013F3	8045 D4	LEA EAX,DWORD PTR [EBP-2C]		T 0 GS 002B 32bit 0(FFFFFFFF)
004013F4	50	PUSH EAX		D 0
004013F5	E8 C80B0000	CALL <JMP.&MSUCRT.strlen>	[5	D 0 LastErr ERROR_FILE_NOT_FOUND (00000002)
004013F6	83C4 1C	ADD ESP,1C	strlen	EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)
004013F7	85C0	TEST EAX,EAX		ST0 empty 0.0
004013F8	76 29	JBE SHORT 1_2.00401444		ST1 empty 0.0
004013F9	8045 FC	MOV EAX,DWORD PTR [EBP-4]		ST2 empty 0.0
004013FA	50	PUSH EAX		ST3 empty 0.0
004013FB	807405 D4	LEA ESI,DWORD PTR [EBP+EAX-2C]		ST4 empty 0.0
004013FC	0FBE4405 D4	MOVSX EAX,BYTE PTR [EBP+EAX-2C]		ST5 empty 1.00000000000000000000
004013FD				ST6 empty 16.00000000000000000000
004013FE				ST7 empty 16.00000000000000000000
004013FF				FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EO)
00401400				FCW 027F Prec NEAR,S3 Mask 1 1 1 1 1 1

Đặt Breakpoint sau đoạn kiểm tra P1 xong.

Ta thấy lại có 1 đoạn kiểm tra độ dài (strlen).

F8 thấy là kiểm tra độ dài P1.

Dưới đó có 1 lệnh gọi hàm CALL 1\_2.00401D5E.

004013D7	837D EC 02	CMPL DWORD PTR [EBP-14],2		Registers (FPU)
004013D8	0F85 D8000000	JNZ 1_2.004014BC		EAX 238E0084
004013D9	8045 E0	LEA EAX,DWORD PTR [EBP-20]		ECX 00000000
004013DA	50	PUSH EAX	[5	EDX 00000000
004013DB	E8 F20B0000	CALL <JMP.&MSUCRT.strlen>	strlen	EBX 0019F4F1
004013DC	50	PUSH EAX		ESP 0019F4B0
004013DD	8045 E0	LEA EAX,DWORD PTR [EBP-20]		EBP 0019F50C
004013DE	50	PUSH EAX		ESI 00000000
004013DF	E8 6A090000	CALL 1_2.00401D5E		EDI 004032DC ASCII "BDR0KPTUJI"
004013E0	8B3D 80314000	MOV EDI,DWORD PTR [<&MSUCRT.sprintf>]	msvcrt.sprintf	EIP 004013F4 1_2.004013F4
004013E1	50	PUSH EAX	<%X>	C 0 ES 002B 32bit 0(FFFFFFFF)
004013E2	BB C8444000	MOV EBX,1_2.004044C8	ASCII "%X"	P 1 CS 0023 32bit 0(FFFFFFFF)
004013E3	8045 D4	LEA EAX,DWORD PTR [EBP-2C]	Format => "%X"	A 0 SS 002B 32bit 0(FFFFFFFF)
004013E4	53	PUSH EBX	S	Z 0 DS 002B 32bit 0(FFFFFFFF)
004013E5	50	PUSH EAX	sprintf	S 0 FS 0053 32bit 362000(FFF)
004013E6	FFD7	CALL EDI		T 0 GS 002B 32bit 0(FFFFFFFF)
004013E7	8365 FC 00	AND DWORD PTR [EBP-4],0		D 0
004013E8	8045 D4	LEA EAX,DWORD PTR [EBP-2C]		D 0 LastErr ERROR_FILE_NOT_FOUND (00000002)
004013E9	50	PUSH EAX	[5	EFL 00000206 (NO,NB,NE,A,NS,PE,GE,G)
004013EA	E8 C80B0000	CALL <JMP.&MSUCRT.strlen>	strlen	ST0 empty 0.0
004013EB	83C4 1C	ADD ESP,1C		ST1 empty 0.0
004013EC	85C0	TEST EAX,EAX		ST2 empty 0.0
004013ED	76 29	JBE SHORT 1_2.00401444		ST3 empty 0.0
004013EE	8045 FC	MOV EAX,DWORD PTR [EBP-4]		ST4 empty 0.0
004013EF	50	PUSH EAX		ST5 empty 1.00000000000000000000
004013F0				ST6 empty 16.00000000000000000000
004013F1				ST7 empty 16.00000000000000000000
004013F2				FST 4020 Cond 1 0 0 0 Err 0 0 1 0 0 0 0 0 (EO)
004013F3				FCW 027F Prec NEAR,S3 Mask 1 1 1 1 1 1

Và sau khi gọi hàm xong, kết quả trả về trong EAX là một chuỗi.

Đây có thể là gì đây?

Theo suy đoán, để biến đổi từ một chuỗi ban đầu thành một chuỗi khác. Người ta đã sử dụng hàm băm.

Nhưng chính xác thì nó sử dụng hàm băm gì, bởi vì hàm băm cũng có nhiều loại.

Sau tìm hiểu, thì ta nhận thấy hàm băm này có sự tương đồng với MD5.

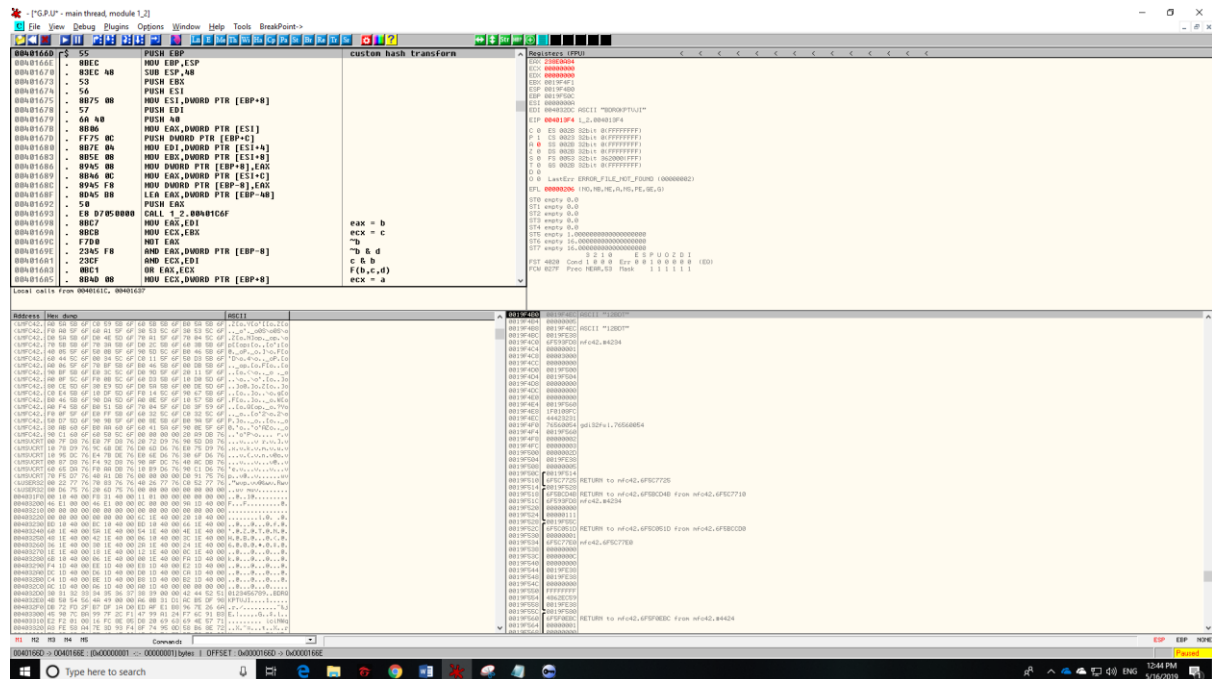
Bởi vì ta tìm thấy đoạn Hash Init của nó.



004015A7	8B4424 04	MOV EAX,DWORD PTR [ESP+4]	custom hash init
004015AB	8360 14 00	AND DWORD PTR [EAX+14],0	
004015AF	8360 10 00	AND DWORD PTR [EAX+10],0	
004015B3	C700 0123456	MOV DWORD PTR [EAX],67452301	
004015B9	C740 04 89AB	MOV DWORD PTR [EAX+4],EFCDB89	
004015C0	C740 08 FEDC	MOV DWORD PTR [EAX+8],98BADCFE	
004015C7	C740 0C 7654	MOV DWORD PTR [EAX+C],10325476	

Nhưng nó được thay đổi ở đoạn Transform.

Cùng xem nó đã tùy biến như thế nào nhé.



Vị trí 1\_2.0040166D là nơi bắt đầu của đoạn Transform của hàm băm.

Ta lướt đến vị trí bắt đầu băm, 1\_2.00401698

Theo lý thuyết, hàm băm trong MD5 sử dụng 4 dãy 8bit: a, b, c, d.

Và bắt đầu băm bằng cách thay đổi lần lượt a, b, c, d trong mỗi lần gọi hàm.

Bắt đầu là thay đổi a.

Và mỗi một lượt băm ta sẽ giữ nguyên hàm băm, sang lượt tiếp theo hàm băm có thể đã thay đổi.

Ban đầu a = 0x67452301, b = 0xEFCDAB89, c = 0x10325476, d = 0x98BADCFE

00401698	8BC7	MOV EAX,EDI	eax = b
0040169A	8BCB	MOV ECX,EBX	ecx = c
0040169C	F7D0	NOT EAX	~b
0040169E	2345 F8	AND EAX,DWORD PTR [EBP-8]	~b & d
004016A1	23CF	AND ECX,EDI	c & b
004016A3	0BC1	OR EAX,ECX	F(b,c,d)
004016A5	8B4D 08	MOV ECX,DWORD PTR [EBP+8]	ecx = a
004016A8	0345 B8	ADD EAX,DWORD PTR [EBP-48]	F(b,c,d) + x
004016AB	03C8	ADD ECX,EAX	a + F(b,c,d) + x
004016AD	8BC1	MOV EAX,ECX	eax = &a
004016AF	C1E8 1D	SHR EAX,1D	
004016B2	C1E1 03	SHL ECX,3	
004016B5	0BC1	OR EAX,ECX	eax = a



Lúc này F8 ta thấy EDI đang giữ giá trị b, EBX đang giữ giá trị c.

Và cách hoạt động của hàm băm được chú thích trong hình,

Ta thấy hàm  $F(x,y,z) = (x \& y) | (\sim x \& z)$ .

Từ đó ta viết được hàm băm cho lượt I.

Tương tự ta có hàm băm ở lượt II, lượt III.

Bắt đầu tại các vị trí trong hình.

Lượt II.

00401874	- 8BD0	MOV EDX,EAX	edx = c
00401876	- 894D FC	MOV DWORD PTR [EBP-4],ECX	cap nhât lai b
00401879	- 0BD1	OR EDX,ECX	b   c
0040187B	- 8BC8	MOV ECX,EAX	
0040187D	- 23D3	AND EDX,EBX	
0040187F	- 234D FC	AND ECX,DWORD PTR [EBP-4]	
00401882	- 0BD1	OR EDX,ECX	
00401884	- 0355 B8	ADD EDX,DWORD PTR [EBP-48]	
00401887	- 8DBC17 99698	LEA EDI,DWORD PTR [EDI+EDX+5A826999]	
0040188E	- 8BCF	MOV ECX,EDI	
00401890	- 8BD7	MOV EDX,EDI	
00401892	- 8B7D FC	MOV EDI,DWORD PTR [EBP-4]	
00401895	- C1E9 1D	SHR ECX,1D	
00401898	- C1E2 03	SHL EDX,3	
0040189B	- 0BCA	OR ECX,EDX	ecx = a

Lượt III.

00401A03	- 8BD3	MOV EDX,EBX	
00401A05	- 33D7	XOR EDX,EDI	
00401A07	- 33D0	XOR EDX,EAX	
00401A09	- 0355 B8	ADD EDX,DWORD PTR [EBP-48]	
00401A0C	- 8D8C11 A1FBD	LEA ECX,DWORD PTR [ECX+EDX+6ED9FBA1]	
00401A0B	- 8BD1	MOV EDX,ECX	
00401A05	- C1EA 1D	SHR EDX,1D	
00401A08	- C1E1 03	SHL ECX,3	
00401A0B	- 0BD1	OR EDX,ECX	
00401ABD	- 8BCF	MOV ECX,EDI	
00401ABF	- 33C8	XOR ECX,EAX	
00401AC1	- 33CA	XOR ECX,EDX	
00401AC3	- 034D D8	ADD ECX,DWORD PTR [EBP-28]	
00401AC6	- 8D8C0B A1FBD	LEA ECX,DWORD PTR [EBX+ECX+6ED9FBA1]	
00401ACD	- 8BD9	MOV EBX,ECX	
00401ACF	- C1EB 17	SHR EBX,17	
00401AD2	- C1E1 09	SHL ECX,9	
00401AD5	- 0BD9	OR EBX,ECX	
00401AD7	- 8BCB	MOV ECX,EBX	
00401AD9	- 895D 08	MOV DWORD PTR [EBP+8],EBX	
00401ADC	- 33C8	XOR ECX,EAX	
00401ADE	- 33CA	XOR ECX,EDX	
00401AE0	- 034D C8	ADD ECX,DWORD PTR [EBP-38]	
00401AE3	- 8D8C0F A1FBD	LEA ECX,DWORD PTR [EDI+ECX+6ED9FBA1]	
00401AEA	- 8BF9	MOV EDI,ECX	

Kết thúc hàm băm, ta có được kết quả khi:

Kết quả ban đầu a += a;

Kết quả ban đầu b += b;

Kết quả ban đầu c += c;

Kết quả ban đầu d += d;

Sau đó  $a \wedge b \wedge c \wedge c$ , theo đoạn code dưới đây.

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]	

Sau khi thoát khỏi hàm băm,

004013EF	- E8 6A090000	CALL 1_2.00401D5E	
004013F4	- 8B3D B031400	MOV EDI,DWORD PTR [&MSUCRT.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	
00401407	- 8365 FC 00	AND DWORD PTR [EBP-4],0	
0040140B	- 8D45 D4	LEA EAX,DWORD PTR [EBP-2C]	
0040140E	- 50	PUSH EAX	s
0040140F	- E8 C80B0000	CALL <JMP.&MSUCRT.strlen>	strlen
00401414	- 83C4 1C	ADD ESP,1C	
00401417	- 85C0	TEST EAX,EAX	
00401419	- 76 29	JBE SHORT 1_2.00401444	

Kết quả trả về được lưu lại dưới dạng ASCII Code.

Ví dụ: CC32DDA3 lưu lại trong thanh ghi là 0x44444133 0x43433332.

Hàm Strlen gọi ở dưới để lấy độ dài chuỗi vừa lưu lại dưới dạng ASCII Code ở trên.

Tiếp theo, F8 đến đoạn này.

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	

Lại là một vòng lặp, con đường mã hóa từ P1 sang P2 còn dài trước mắt.

Nhớ lại sau lần Hash trước ta có được một chuỗi 8 byte, nhưng được lưu trong thanh ghi dưới dạng ASCII Code. (Tạm gọi đây là S1)

Thuật toán sử dụng trong vòng lặp này:

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

While ( i < S1.length() )

{

S1[i] = S1[i] xor Table[i];

i++;



}

(Kết quả trong S[i] chỉ lấy tối đa 2 byte).

Tiếp theo, F8 đến đoạn này.

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

Này tương tự cũng là một vòng lặp, xử lý biến đổi S1 sau lần biến đổi ở trên.

Thuật toán sử dụng ở đây là:

While ( i < S1.length() )

{

S1[i] = (S1[i] shl i) or S1[i];

i++;

}

(Kết quả mỗi lần lưu trong S[i] lấy tối đa 2 byte).

Sau khi thực hiện 2 vòng lặp, ta thu được S1.

F8 tiếp đến 1 câu lệnh gọi hàm.

Hàm lần này được gọi, lại là một hàm băm, nhưng lần này là CRC32.

Lí do ta nhận biết được.

Là do Hash Lookup Table được lưu trong memory từ 404020 – 40441C

0040148C	. 50	PUSH EAX	
0040148D	. E8 2EFCFFFF	CALL 1_2.004010C0	

Đây là Hash Lookup Table được lưu sẵn trong thanh ghi:





```
00404020 00 00 00 00 96 30 07 77 2C 61 0E EE BA 51 09 99 .....0.w,a...Q..
00404030 19 C4 6D 07 8F F4 6A 70 35 A5 63 E9 A3 95 64 9E ...m...jp5.c...d.
00404040 32 88 DB 0E A4 B8 DC 79 1E E9 D5 E0 88 D9 D2 97 2.....y.....
00404050 2B 4C B6 09 BD 7C B1 7E 07 2D B8 E7 91 1D BF 90 +L...!..-.....
00404060 64 10 B7 1D F2 20 B0 6A 48 71 B9 F3 DE 41 BE 84 d.... .jHq...A..
00404070 7D 04 DA 1A EB E4 DD 6D 51 B5 D4 F4 C7 85 D3 83 }.....mQ.....
00404080 56 98 6C 13 C0 A8 68 64 7A F9 62 FD EC C9 65 8A U,l...kdz.b...e.
00404090 4F 5C 01 14 D9 6C 06 63 63 3D 0F FA F5 0D 08 8D 0\...l.oc=.....
004040A0 C8 20 6E 3B 5E 10 69 4C E4 41 60 D5 72 71 67 A2 . n;^..iL.A'.rqg.
004040B0 D1 E4 03 3C 47 04 04 4B FD 85 0D D2 6B B5 0A A5 ...<G..K....k...
004040C0 FA A8 B5 35 6C 98 B2 42 D6 C9 BB DB 40 F9 BC AC ...5l..B....@...
004040D0 E3 6C D8 32 75 5C DF 45 CF 0D D6 DC 59 3D D1 AB .l.2u\..E....Y=..
004040E0 AC 30 D9 26 3A 00 DE 51 80 51 D7 C8 16 61 D0 BF .0.&:...Q.Q...a..
004040F0 B5 F4 B4 21 23 C4 B3 56 99 95 BA CF 0F A5 BD B8 ...!#...U.....
00404100 9E B8 02 28 08 88 05 5F B2 D9 0C 24 E9 0B B1 ...(. ...$. ....
00404110 87 7C 6F 2F 11 4C 68 58 AB 1D 61 C1 3D 2D 66 B6 .lo/.LhX..a.=f..
00404120 90 41 DC 76 06 71 D8 01 BC 20 D2 98 2A 10 D5 EF .A.v.q... ..*...
00404130 89 85 B1 71 1F B5 B6 06 A5 E4 BF 9F 33 D4 B8 E8 ...q4.....3...
00404140 A2 C9 07 78 34 F9 00 0F 8E A8 09 96 18 98 0E E1 ...x4.....
00404150 BB 0D 6A 7F 2D 3D 6D 08 97 6C 64 91 01 5C 63 E6 .j..=m...ld...\c.
00404160 F4 51 6B 68 62 61 6C 1C D8 30 65 85 4E 00 62 F2 .Qkkbal..0e.N.b.
00404170 ED 95 06 6C 7B A5 01 1B C1 F4 08 82 57 C4 0F F5 ...l(. ....w...
00404180 C6 D9 B0 65 50 E9 B7 12 EA B8 BE 8B 7C 88 B9 FC ...eP.....!...
00404190 DF 1D D0 62 49 2D DA 15 F3 7C D3 8C 65 4C D4 FB ...bI-...!...eL..
004041A0 58 61 B2 4D CE 51 B5 3A 74 00 BC A3 E2 30 B8 D4 Xa.M.Q.:t....0..
004041B0 41 A5 DF 4A D7 95 D8 3D 6D C4 D1 A4 FB F4 D6 D3 A..J...=m.....
004041C0 6A E9 69 43 FC D9 6E 34 46 88 67 AD D0 B8 60 DA j..iC...n4F.g...'.
004041D0 73 2D 04 44 E5 1D 03 33 5F 4C 0A AA C9 7C 0D 0D s-.D...3_L...i..
004041E0 3C 71 05 50 AA 41 02 27 10 10 0B BE 86 20 0C C9 <q.P.A.'..... ..
004041F0 25 B5 68 57 B3 85 6F 20 09 D4 66 B9 9F E4 61 CE %..hW...o ..f...a.
00404200 0E F9 DE 5E 98 C9 D9 29 22 98 D0 B0 B4 A8 D7 C7 ...^...)".....
00404210 17 3D B3 59 81 0D B4 2E 3B 5C BD B7 AD 6C BA C0 .=.V....;\...l..
00404220 20 83 B8 ED B6 B3 BF 9A 0C E2 B6 03 9A D2 B1 74 .....t.....
00404230 39 47 D5 EA AF 77 D2 9D 15 26 D8 04 83 16 DC 73 9G...w...&.....s
00404240 12 0B 63 E3 84 3B 64 94 3E 6A 6D 0D A8 5A 6A 7A ..c...;d.>jm..Zjz
00404250 0B CF 0E E4 9D FF 09 93 27 AE 00 0A B1 9E 07 7D .....<.....}
00404260 44 93 0F F0 D2 A3 08 87 68 F2 01 1E FE C2 06 69 D.....h.....i
00404270 5D 57 62 F7 CB 67 65 80 71 36 6C 19 E7 06 6B 6E J\Wb..ge.q6l...kn
00404280 76 1B D4 FE 0E 2B D3 89 5A 7A DA 10 CC 4A D0 67 v...+...Zz...J.g
00404290 6F DF B9 F9 F9 EF BE 8E 43 BE B7 17 D5 8E B0 60 o.....C.....'
004042A0 E8 A3 D6 D6 7E 93 D1 A1 C4 C2 D8 38 52 F2 DF 4F ....~.....8R...0
004042B0 F1 67 B8 D1 67 57 BC A6 D0 06 B5 3F 4B 36 B2 48 .g...gW.....?K6.H
004042C0 DA 2B 0D D8 4C 1B 0A AF F6 4A 03 36 60 7A 04 41 .+..L....J.6^z.A
```

Và đây là đoạn code của hàm băm CRC32:

004010C0	\$ 56	PUSH ESI	crc
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	

Lưu ý đoạn MOV EAX,DWORD PTR [EAX\*4+404020]

Như đã đề cập, ta phát hiện Hash Lookup Table được lưu trong thanh ghi từ 404020 đến 40441C



Tại sao vậy? để ý thấy AND EAX, ECX mà ECX = 0xFF, nên ta biết EAX chỉ nằm trong giá trị khoảng [0x0, 0xFF] tức chỉ có 2 byte. Giá trị thanh ghi ta load ở dòng (1) nằm trong [404020, 40441C].

Và giá trị của Hash Lookup Table này trùng khớp với trong hàm băm CRC32.

⇒ Ở đây phải sử dụng hàm băm CRC32 để xử lý S1.

Kết quả thu được, ta gọi là C1.

Đoạn code tiếp theo để định dạng C1 thành hệ 16.

Và lưu dưới dạng chuỗi.

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	

F8 tiếp thì thấy ta đã đến đoạn so sánh mà ta đã đề cập từ đầu.

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	

s2

s1

strcmp

if EAX = 0 -> VALID KEY

Vậy từ đây, ta đã biết, hai chuỗi so sánh với nhau là chuỗi nhận được sau khi biến đổi P1 (cũng chính là P2 hợp lệ) và chuỗi P2 ta nhập vào. Nếu P2 = S2 => **Thành Công**, else => **Thất bại**.

Các nguồn tham khảo chương trình keygen 1\_2:

<https://barrgroup.com/Embedded-Systems/How-To/CRC-Calculation-C-Code>

<http://www.zedwood.com/article/cpp-md5-function>

<https://social.msdn.microsoft.com/Forums/en-US/d50184d2-313b-4944-8307-e0343e865879/cc-putting-the-window-in-center-of-screen?forum=vcgeneral&fbclid=IwAR3763EPFMcFzEYrayBfOPPPZXSTPyqLzSnmvE2DgRI2OBV-xyEYA4ShtA>

[xyEYA4ShtA](https://social.msdn.microsoft.com/Forums/en-US/d50184d2-313b-4944-8307-e0343e865879/cc-putting-the-window-in-center-of-screen?forum=vcgeneral&fbclid=IwAR3763EPFMcFzEYrayBfOPPPZXSTPyqLzSnmvE2DgRI2OBV-xyEYA4ShtA)

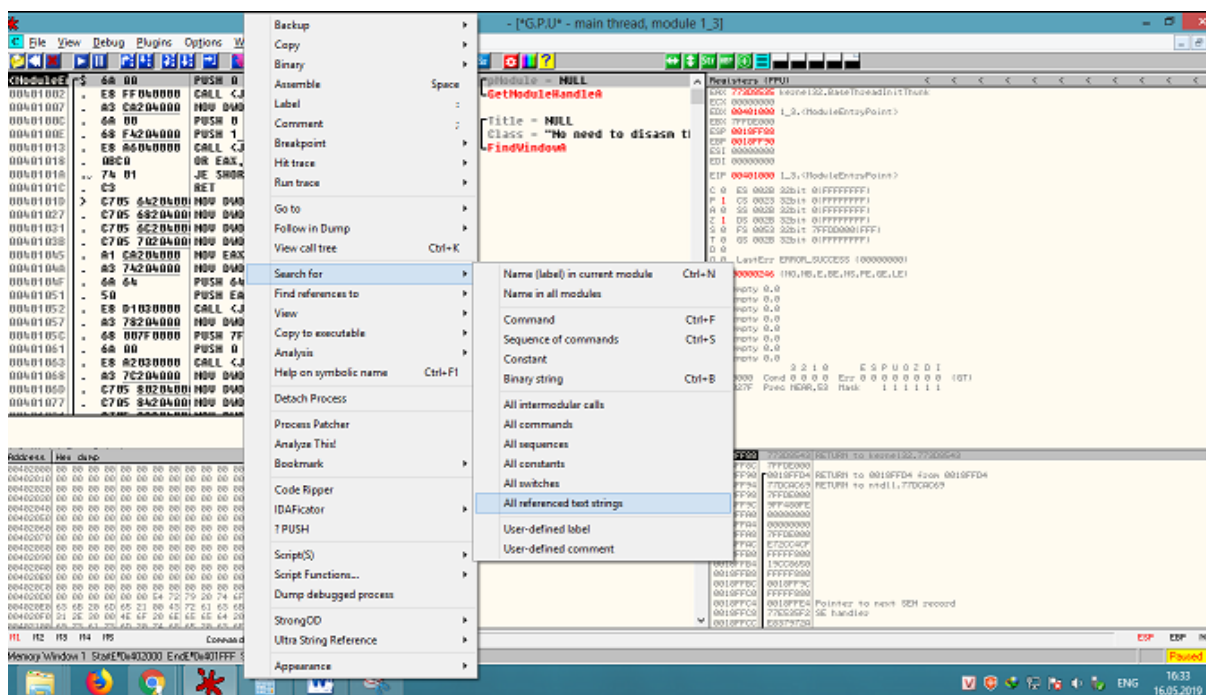
### 2.3. Bài 3: tutorial cracking 1\_3

Chạy OllyDbg.exe

Từ ứng dụng mở file 1\_3.exe

Ấn F9 để chạy chương trình lần đầu

Click chuột phải trong cửa sổ CPU trong Olly chọn Search for tiếp tục chọn All referenced text string



Ta để ý đến dòng chữ “Great work, mate! Now try the next CrackMe!” (đây chính là dấu hiệu crack thành công). Ấn đúp vào dòng chữ để trở lại màn hình CPU.

Address	Disassembly	Text string
00401000	PUSH 0	(Initial CPU selection)
0040100E	PUSH 1_3.004020F4	ASCII "No need to disasm the code!"
00401077	MOV DWORD PTR [402084], 1_3.00402110	ASCII "MENU"
00401081	MOV DWORD PTR [402088], 1_3.004020F4	ASCII "No need to disasm the code!"
004010B7	PUSH 1_3.004020E7	ASCII "CrackMe v1.0"
004010BC	PUSH 1_3.004020F4	ASCII "No need to disasm the code!"
004011F7	PUSH 1_3.0040211F	ASCII "DLG_ABOUT"
00401213	PUSH 1_3.00402115	ASCII "DLG_REGIS"
0040134F	PUSH 1_3.00402129	ASCII "Good work!"
00401354	PUSH 1_3.00402134	ASCII "Great work, mate! Now try the next CrackMe!"
0040136B	PUSH 1_3.00402160	ASCII "No luck!"
00401370	PUSH 1_3.00402169	ASCII "No luck there, mate!"
004013AF	PUSH 1_3.00402160	ASCII "No luck!"
004013B4	PUSH 1_3.00402169	ASCII "No luck there, mate!"



Lúc này, ta quan tâm đến dòng lệnh CALL <JMP.&USER32.MessageBoxA>.

Phải chăng đây là dòng lệnh gọi thông báo gì đó.

Ta thấy ở phía dưới, nếu ta nhập sai key, thì cũng gọi một lệnh tương tự.

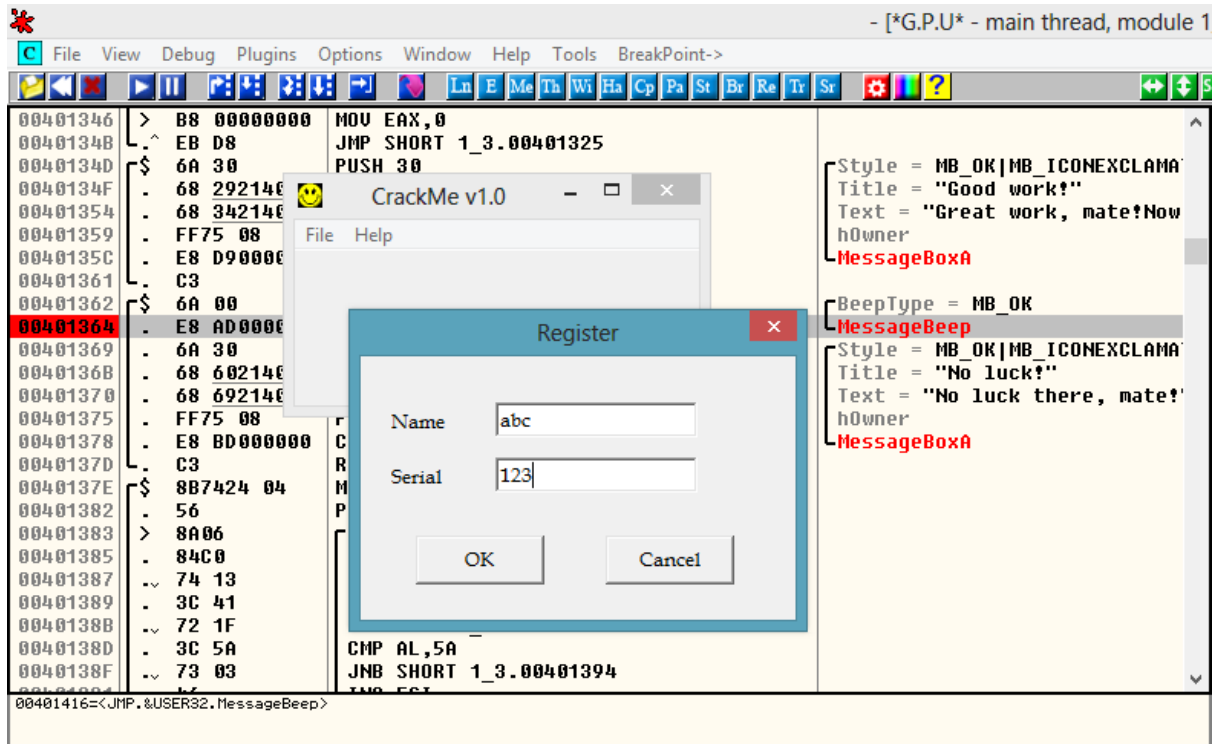
Nhưng trước thông báo nhập sai, ta để ý thấy có dòng lệnh CALL <JMP.&USER32.MessageBeep>

Cảm giác nên đặt Breakpoint tại đó, như hình:

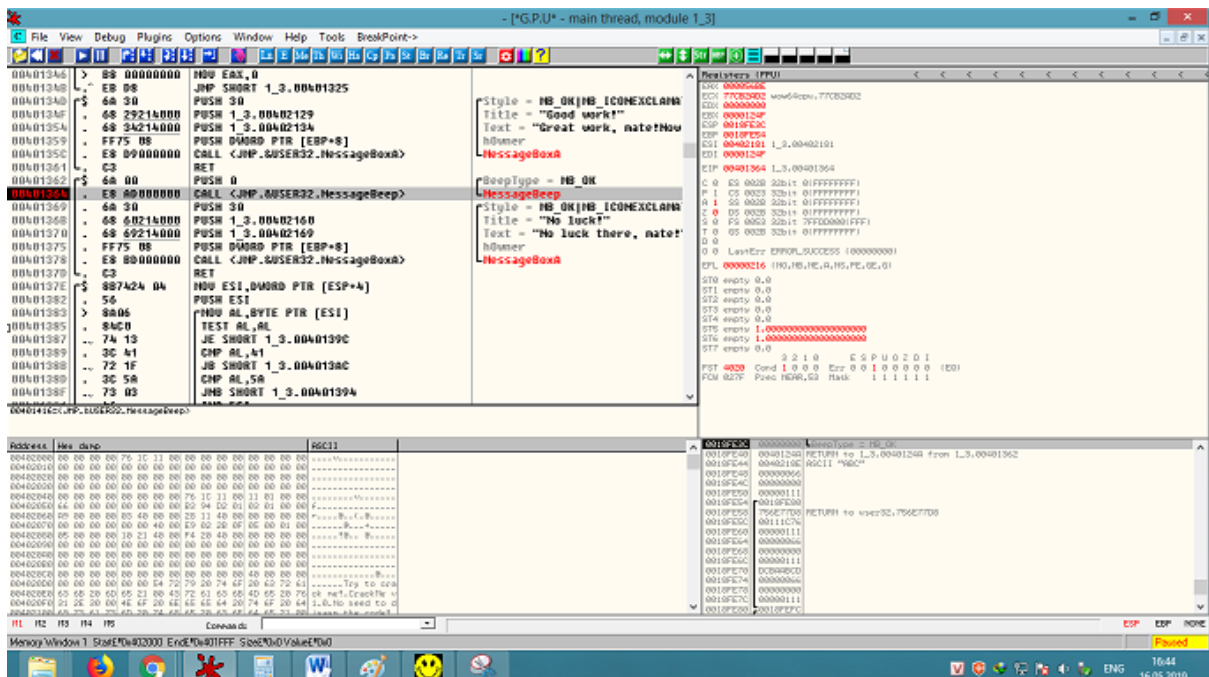
Và chạy thử chương trình.

Nhập username và password.





Nhấn OK.



Như vậy ta đã có thể Trace chương trình.

Kéo lên phía trên một chút, ta để ý thấy.

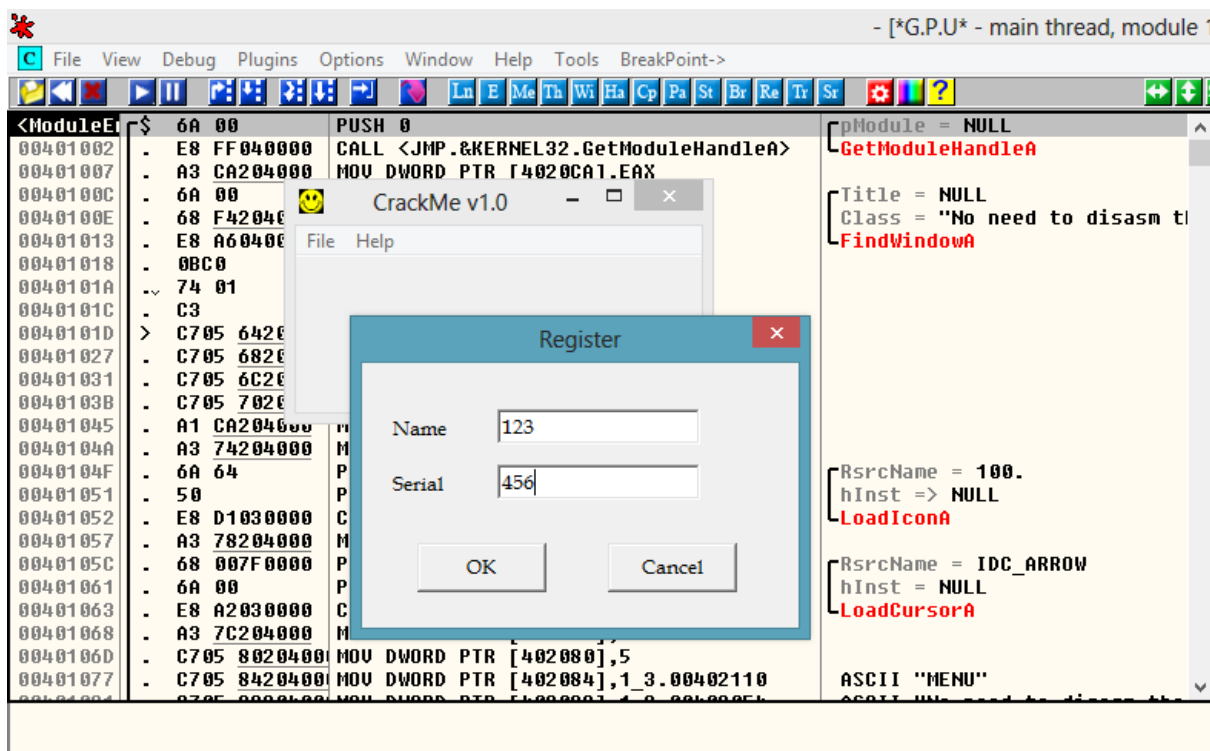


00401228	68 8E214000	PUSH 1_3.0040218E	ASCII "ABC"
0040122D	E8 4C010000	CALL 1_3.0040137E	
00401232	50	PUSH EAX	
00401233	68 7E214000	PUSH 1_3.0040217E	ASCII "123"
00401238	E8 9B010000	CALL 1_3.004013D8	
0040123D	83C4 04	ADD ESP,4	
00401240	58	POP EAX	
00401241	3BC3	CMP EAX,EBX	
00401243	74 07	JE SHORT 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	
00401258	56	PUSH ESI	
00401259	57	PUSH EDI	
0040125A	817D 0C 1001	CMP DWORD PTR [EBP+C],110	
00401261	74 34	JE SHORT 1_3.00401297	
00401263	817D 0C 1101	CMP DWORD PTR [EBP+C],111	
0040126A	74 35	JE SHORT 1_3.004012A1	
0040126C	837D 0C 10	CMP DWORD PTR [EBP+C],10	
00401270	0F84 81000000	JE 1_3.004012F7	
00401276	817D 0C 0102	CMP DWORD PTR [EBP+C],201	
0040127D	74 0C	JE SHORT 1_3.0040128B	

Hình như có gì đó quen quen, vâng, đó chính là username và password ta vừa nhập vào.

Nhưng điểm đáng lưu ý là, ta nhập vào “abc”, nhưng chương trình đã biến đổi toàn bộ chuỗi username là viết hoa.

Vậy ta thử nhập username với toàn số, và username gồm số và chữ xem kết quả như thế nào.



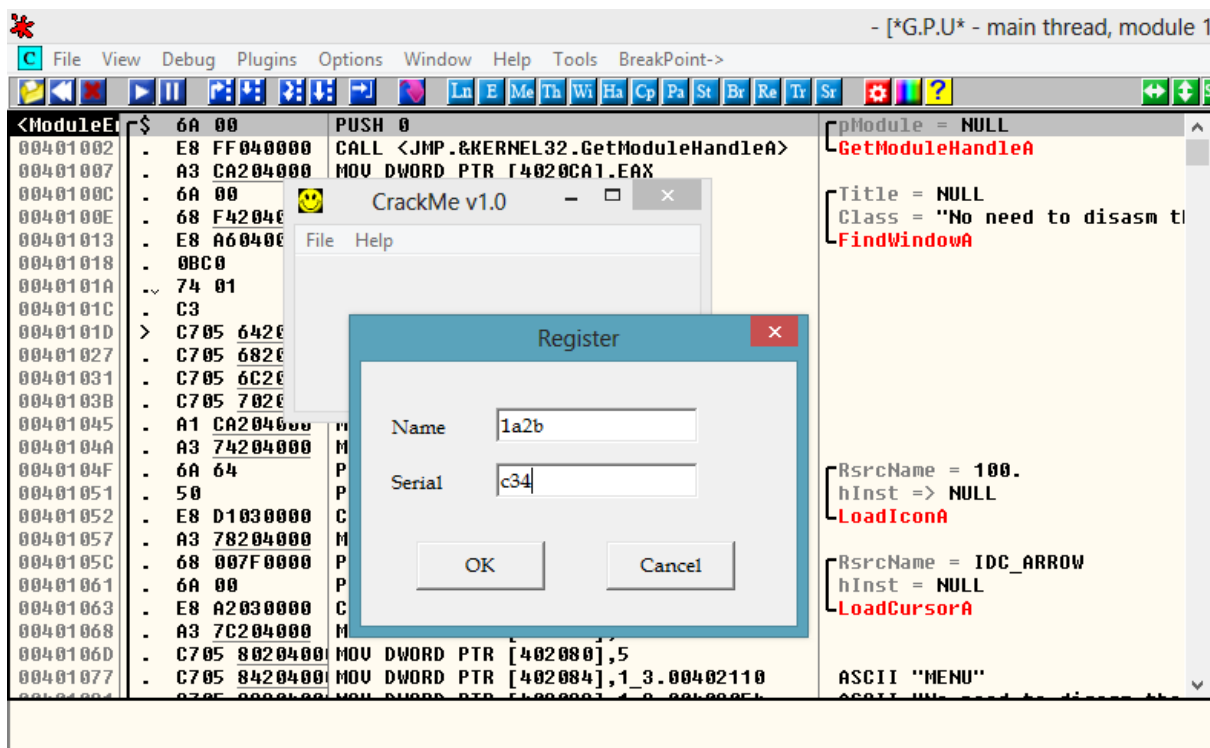
Trường hợp Username toàn số.

Kết quả:

00401228	- 68 8E214000	PUSH 1_3.0040218E	ASCII "123"
0040122D	- E8 4C010000	CALL 1_3.0040137E	
00401232	- 50	PUSH EAX	
00401233	- 68 7E214000	PUSH 1_3.0040217E	ASCII "456"
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	
00401258	- 56	PUSH ESI	
00401259	- 57	PUSH EDI	
0040125A	- 817D 0C 1001	CMP DWORD PTR [EBP+C],110	
00401261	- 74 34	JE SHORT 1_3.00401297	
00401263	- 817D 0C 1101	CMP DWORD PTR [EBP+C],111	
0040126A	- 74 35	JE SHORT 1_3.004012A1	
0040126C	- 837D 0C 10	CMP DWORD PTR [EBP+C],10	
00401270	- 0F84 81000000	JE 1_3.004012F7	
00401276	- 817D 0C 0102	CMP DWORD PTR [EBP+C],201	
0040127D	- 74 0C	JE SHORT 1_3.0040128B	

Oh, không có gì thay đổi.

Sang trường hợp gồm số và chữ, ở đây ta nhân tiện thử luôn với password.

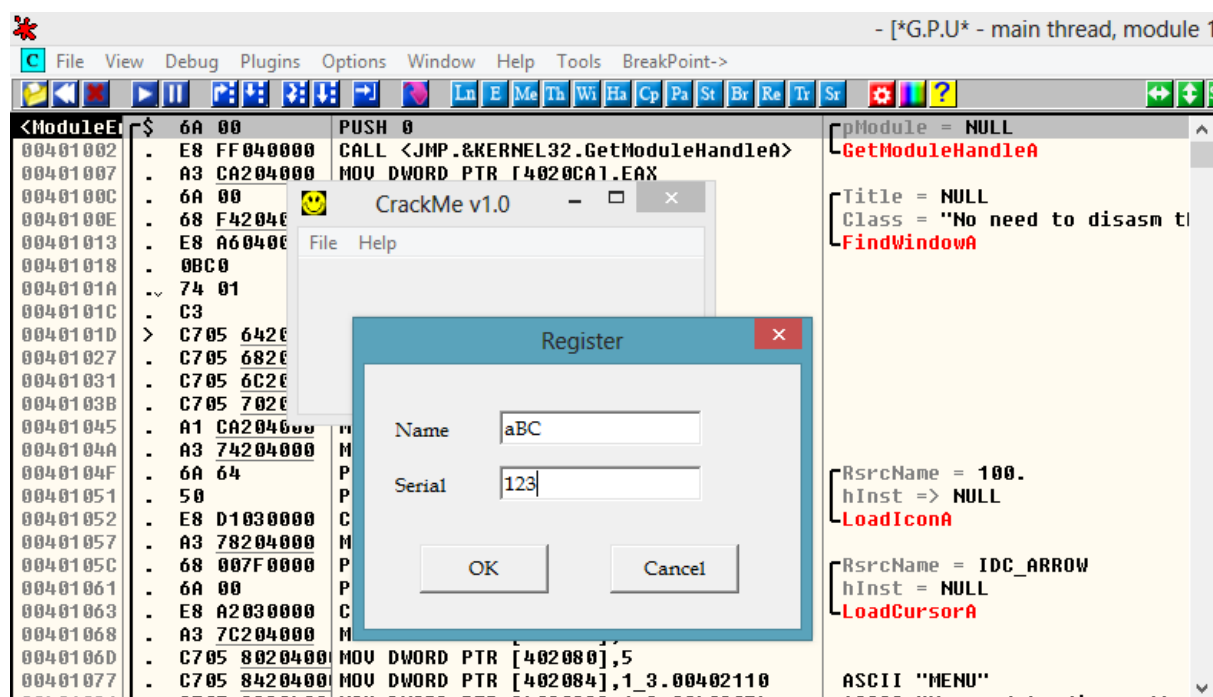


Kết quả:

00401228	. 68 8E214000	PUSH 1_3.0040218E	ASCII "1a2b"
0040122D	. E8 4C010000	CALL 1_3.0040137E	
00401232	. 50	PUSH EAX	
00401233	. 68 7E214000	PUSH 1_3.0040217E	ASCII "c34"
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	
00401258	. 56	PUSH ESI	
00401259	. 57	PUSH EDI	
0040125A	. 817D 0C 1001	CMP DWORD PTR [EBP+C],110	
00401261	~ 74 34	JE SHORT 1_3.00401297	
00401263	. 817D 0C 1101	CMP DWORD PTR [EBP+C],111	
0040126A	~ 74 35	JE SHORT 1_3.004012A1	
0040126C	. 837D 0C 10	CMP DWORD PTR [EBP+C],10	
00401270	~ 0F84 81000000	JE 1_3.004012F7	
00401276	. 817D 0C 0102	CMP DWORD PTR [EBP+C],201	
0040127D	~ 74 0C	JE SHORT 1_3.0040128B	

Hmm, cũng không có gì thay đổi, cả username lẫn password.

Thử thêm trường hợp toàn chữ, nhưng có chữ hoa với chữ thường.



The screenshot shows a debugger window with the following components:

- Assembly View (Left):** Displays assembly instructions with addresses and hex values. For example, at address 00401002, there is a `PUSH 0` instruction.
- Register Dialog Box (Center):** A modal dialog titled "Register" with two input fields: "Name" containing "aBC" and "Serial" containing "123". It has "OK" and "Cancel" buttons.
- Disassembly View (Right):** Shows the disassembled code, including instructions like `pModule = NULL`, `GetModuleHandleA`, `Title = NULL`, `Class = "No need to disasm t`, `FindWindowA`, `RsrcName = 100.`, `hInst => NULL`, `LoadIconA`, `RsrcName = IDC_ARROW`, `hInst = NULL`, `LoadCursorA`, and `ASCII "MENU"`.

00401228	. 68 8E214000	PUSH 1_3.0040218E	ASCII "ABC"
0040122D	. E8 4C010000	CALL 1_3.0040137E	
00401232	. 50	PUSH EAX	
00401233	. 68 7E214000	PUSH 1_3.0040217E	ASCII "123"
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	
00401258	. 56	PUSH ESI	
00401259	. 57	PUSH EDI	
0040125A	. 817D 0C 1001	CMP DWORD PTR [EBP+C],110	
00401261	~ 74 34	JE SHORT 1_3.00401297	
00401263	. 817D 0C 1101	CMP DWORD PTR [EBP+C],111	
0040126A	~ 74 35	JE SHORT 1_3.004012A1	
0040126C	. 837D 0C 10	CMP DWORD PTR [EBP+C],10	
00401270	~ 0F84 81000000	JE 1_3.004012F7	
00401276	. 817D 0C 0102	CMP DWORD PTR [EBP+C],201	
0040127D	~ 74 0C	JE SHORT 1_3.0040128B	

Oh, ta thấy chữ a cũng được chuyển sang chữ A, trong khi các kí tự viết hoa sẵn thì không thay đổi.

Tới đây, ta dự đoán. Trong chương trình tồn tại một hàm biến đổi username chữ thường thành chữ hoa.

Và nó chỉ hoạt động khi ta nhập Username toàn chữ. Đồng thời điều này có ý nghĩa, chương trình không phân biệt Username chữ hoa hay chữ thường.

Nhân tiện nói luôn, chỉ Username bị biến đổi. Password ko ảnh hưởng.

00401228	. 68 8E214000	PUSH 1_3.0040218E	ASCII "123"
0040122D	. E8 4C010000	CALL 1_3.0040137E	
00401232	. 50	PUSH EAX	
00401233	. 68 7E214000	PUSH 1_3.0040217E	ASCII "abc"
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	
00401258	. 56	PUSH ESI	
00401259	. 57	PUSH EDI	
0040125A	. 817D 0C 1001	CMP DWORD PTR [EBP+C],110	
00401261	~ 74 34	JE SHORT 1_3.00401297	
00401263	. 817D 0C 1101	CMP DWORD PTR [EBP+C],111	
0040126A	~ 74 35	JE SHORT 1_3.004012A1	
0040126C	. 837D 0C 10	CMP DWORD PTR [EBP+C],10	
00401270	~ 0F84 81000000	JE 1_3.004012F7	
00401276	. 817D 0C 0102	CMP DWORD PTR [EBP+C],201	
0040127D	~ 74 0C	JE SHORT 1_3.0040128B	

Ok. Vậy ta sẽ bắt đầu tìm hiểu xem hàm UpCase trong chương trình có tác dụng gì.

Liệu rằng phải ám chỉ ta phải nhập username toàn chữ mới là hợp lệ.



00401228	. 68 8E214000	PUSH 1_3.0040218E	ASCII "abc"
0040122D	. E8 4C010000	CALL 1_3.0040137E	
00401232	. 50	PUSH EAX	
00401233	. 68 7E214000	PUSH 1_3.0040217E	ASCII "123"
00401238	. E8 9B010000	CALL 1_3.004013D8	

Sau khi push Username vào stack, lệnh CALL nhảy đến đoạn này.

0040137E	8B7424 04	MOV ESI,DWORD PTR [ESP+4]	Xu ly Username
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>	

Đặt Breakpoint tại đây để theo dõi chương trình.

0040137E	8B7424 04	MOV ESI,DWORD PTR [ESP+4]	Xu ly Username
00401382	. 56	PUSH ESI	
00401383	> 8A06	MOV AL,BYTE PTR [ESI]	ESI lúc này = Username
00401385	. 84C0	TEST AL,AL	

Ở đây ta lấy ASCII Code của ký tự đầu trong chuỗi Username.

AL = ASCII (ký tự đầu trong Username).

Kiểm tra AL = 0 hay không bằng lệnh TEST AL,AL. Ở đây mục đích là để xem Username ta nhập vào có phải chuỗi rỗng hay không.

0040137E	8B7424 04	MOV ESI,DWORD PTR [ESP+4]	Xu ly Username
00401382	. 56	PUSH ESI	
00401383	> 8A06	MOV AL,BYTE PTR [ESI]	ESI lúc này = Username
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	

JE: Jump if Equal, trong TH TEST AL,AL mà AL = 0, thì cờ Z sẽ bật lên 1. Và JE sẽ thực hiện nếu ZF = 1. Trong TH ta không nhập gì ở Username. Thì chương trình sẽ không thực thi xử lý Username gì cả.





Tiếp đến, CMP AL,41. Nếu  $AL < 0x41$  thì nhảy. Ở đây là nhảy ra khỏi đoạn xử lý Username.

0x41 trong ASCII là ký tự A (Ở đây AL cũng đang là ASCII Code). Như vậy, nghi vấn hỏi này của ta sắp được giải đáp. Nhìn vào bảng ASCII:

46	056	2E	00101110	.
47	057	2F	00101111	/
48	060	30	00110000	0
49	061	31	00110001	1
50	062	32	00110010	2
51	063	33	00110011	3
52	064	34	00110100	4
53	065	35	00110101	5
54	066	36	00110110	6
55	067	37	00110111	7
56	070	38	00111000	8
57	071	39	00111001	9
58	072	3A	00111010	:
59	073	3B	00111011	;
60	074	3C	00111100	<
61	075	3D	00111101	=
62	076	3E	00111110	>
63	077	3F	00111111	?
64	100	40	01000000	@
65	101	41	01000001	A

Và nhìn lệnh CMP phía dưới: CMP AL,5A.

0x5A trong ASCII là chữ Z. Sau đó là lệnh JNB, có nghĩa là Nhảy nếu không bé hơn (Nhảy nếu lớn hơn).

Như vậy, thuật toán so sánh ở trên là  $A < x < Z$ , ta xét xem ký tự x này có phải ký tự CHỮ và là CHỮ HOA hay không.

Nếu  $x < A$  thì x chắc chắn không phải ký tự chữ.

Nếu  $x > A$  và  $x > Z$ , thì x có cơ hội là ký tự chữ thường (Chữ a có ASCII là 61). (2)

Nếu  $A < x < Z$ , thì x là ký tự chữ hoa. (3)



Lệnh JNB thực thi khi x rơi vào trường hợp (2) -> bỏ qua 2 câu lệnh: INC ESI và JMP SHORT 1\_3.401383.

ESI lúc này đang chỉ vào dãy Username, vậy INC ESI có nghĩa là ta sẽ xét ký tự tiếp theo, bằng chứng là lệnh JMP nhảy lên lại đoạn xét ký tự phía trên.

Vậy xét TH (2), x có cơ hội là chữ thường.

Ta thấy hàm được gọi:

004013D2	\$ 2C 20	SUB AL,20	
004013D4	. 88 06	MOV BYTE PTR [ESI],AL	
004013D6	. C3	RET	

Một đoạn code vô cùng ngắn, ta sẽ thực hiện  $AL = AL - 20$ .

Vậy 0x20 ở đây là gì? Vừa nãy có đề cập, x (AL) có cơ hội là chữ thường. Vậy 0x20 ở đây có lẽ là khoảng cách giữa chữ A và chữ a (0x41 và 0x61 trong ASCII).

Nếu x đúng là chữ thường, thì sao khi trừ 0x20, ta sẽ x chữ hoa!

Vậy đây chính xác là hàm UpCase mà ta đã đề cập ở trước đó.

0040137E	\$ 8B7424 04	MOV ESI,DWORD PTR [ESP+4]	Xu ly Username
00401382	. 56	PUSH ESI	
00401383	> 8A 06	MOV AL,BYTE PTR [ESI]	ESI lúc này = Username
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	1_3.0040218E
0040139A	^ EB E7	JMP SHORT 1_3.00401383	

Ta vẫn có hai dòng lệnh ta đã bỏ qua trước đó: INC ESI và JMP SHORT 1\_3.00401383.

Như đã giải thích. Ta sẽ xét tiếp ký tự tiếp theo trong Username.

Vậy khi nào thì dòng lặp này sẽ ngừng?

Đáp án đã được đề cập trước đó: Nếu ký tự đó là ký tự rỗng, hoặc ký tự đó không phải ký tự chữ.

Và trong TH phát hiện 1 ký tự trong Username không phải ký tự chữ, chương trình sẽ nhảy đến đoạn thông báo nhập sai! Như hình phía dưới:



0040137E	8B7424 04	MOV ESI,DWORD PTR [ESP+4]	Xu ly Username
00401382	56	PUSH ESI	
00401383	8A06	MOV AL,BYTE PTR [ESI]	ESI luc nay = Username
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>	
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	

Ok. Ta bước tiếp.

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	

Lại một lệnh gọi hàm.

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	

Ta có 2 lệnh XOR, kết quả 2 lệnh đều = 0, như ta đã biết, nếu XOR 2 giá trị giống nhau thì kết quả trả về là 0 (XOR là thuật toán so sánh khác).

Ta có 1 vòng lặp nhỏ.

ESI lúc này vẫn đang giữ địa chỉ lưu Username (sau khi xử lý viết hoa ở trên).

Ta cũng xét từng ký tự trong Username.

TEST BL,BL kiểm tra ký tự đó có phải ký tự rỗng hay không.

Nếu rỗng thì ta thoát hàm.

Nếu không thì EDI = EDI + EBX. (2 byte thấp lúc này của EBX đã có giá trị là ASCII Code của ký tự đang xét, trong khi EDI vẫn như ban đầu = 0).

INC ESI -> xét ký tự kế tiếp.

Như vậy toàn bộ dòng code này ta sẽ tính tổng ASCII Code của các ký tự.

Ví dụ:  $ABC = 41+42+43$ .

F8 tiếp,

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	

Sau khi có tổng ASCII của các ký tự (lưu trong EDI), ta lấy kết quả XOR với 0x5678.

Lưu lại kết quả tại EAX.

F8 tiếp,

00401228	- 68 8E214000	PUSH 1_3.0040218E	ASCII "ABC"
0040122D	- E8 4C010000	CALL 1_3.0040137E	
00401232	- 50	PUSH EAX	
00401233	- 68 7E214000	PUSH 1_3.0040217E	ASCII "123"
00401238	- E8 9B010000	CALL 1_3.004013D8	

Ta đã xử lý xong phần Username? Câu trả lời ta sẽ tìm thấy ở trong lệnh CALL thứ 2: CALL 1\_3.004013D8

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	> 00 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	IMUL 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	

EAX đang giữ giá trị kết quả ở phần trước. (như đã tính ở trên). Nhưng ở đây XOR EAX,EAX tức trả EAX = 0.

Ta lại có 2 dòng lệnh XOR quen thuộc, không cần giải thích lại vì đã giải thích ở trên.

Load địa chỉ lưu Password vào ESI (MOV ESI,DWORD PTR [ESP +4]).

Ta có 1 vòng lặp ngắn.

Điều kiện dừng là khi xét hết chuỗi Password:

TEST BL,BL và JE SHORT 1\_3.004013F5

Xét lần lượt từng ký tự của chuỗi Password,

AL = 0A, mà EAX đang giữ giá trị tổng, ví dụ EAX = 547A thì EAX lúc sau = 540A.

BL đang giữ giá trị ASCII của ký tự -> SUB BL 30 -> BL = BL - 0x30.



Vậy 0x30 ở đây là gì? Nếu ta đã từng tiếp cận lập trình Chuyển chữ sang số, ta sẽ để ý, ASCII của ký tự số trừ đi một khoảng 0x30 (tức ký tự '0') thì ta sẽ được giá trị của con số đó.

Ví dụ: '1' có ASCII Code là 0x31 mà trừ đi 0x30 ta sẽ có giá trị 1 (số 1).

Cho nên ta có thể hiểu bước làm này, là chuyển từ ký tự số, sang giá trị số!

IMUL EDI,EAX (EDI ban đầu khởi tạo = 0 và EAX với 2 byte thấp giữ giá trị 0A) -> EDI = EDI \* 10

ADD EDI,EBX -> EDI = EDI + EBX.

Như suy đoán, vòng lặp này để chuyển ký tự số sang số.

F8 tiếp, sau khi chuyển Password sang số. Ta có những suy nghĩ, liệu rằng Password có phải được định dạng dưới dạng số?

Chúng ta sắp gần đến đáp án rồi.

004013F3	. ^ EB ED	L JMP SHORT 1_3.004013E2	
004013F5	> 81F7 34120000	XOR EDI,1234	
004013F8	. 8BDF	MOV EBX,EDI	

Trước khi thoát khỏi hàm, ta thấy lệnh XOR EDI,1234

Trong khi EDI đang lưu giữ giá trị Password. (Ví dụ Password = '123' thì EDI = 123)

MOV EBX,EDI -> Lưu lại kết quả trong EBX.

F8 tiếp để thoát khỏi hàm.

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	

Điểm đáng chú ý ở đoạn code này, CMP EAX,EBX.

Như ta đã biết, EBX đang lưu giữ giá trị Password XOR 1234.

Vậy EAX sẽ lưu trữ cái gì? Nhớ lại thanh ghi Stack.

00401232	. 50	PUSH EAX	
00401233	. 68 7E214000	PUSH 1_3.0040217E	ASCII "123"
00401238	. E8 9B010000	CALL 1_3.004013D8	
0040123D	. 83C4 04	ADD ESP,4	

Trước khi gọi hàm 1\_3.004013D8, ta có push vào stack giá trị mà EAX đang lưu giữ (ESP + 4) và push vào stack chuỗi Password (ESP)

Vậy ADD ESP,4 và POP EAX, có nghĩa là ta sẽ lấy giá trị ESP+4 trong stack lưu vào EAX.

➔ EAX giữ giá trị tổng ta đã xử lý ở phần Username.





So sánh EAX và EBX:

Lệnh JE SHORT 1\_3.004124C: Nhảy khi EAX = EBX.

⇒ CALL 1\_3.004134D cũng chính là xuất thông báo Key Hợp Lệ -> **Thành Công**.

Nếu không, CALL 1\_3.00401362 là xuất thông báo Key Không Hợp Lệ -> *Thất bại*.

Như vậy, ta rút ra kết luận:

Tính hợp lệ của Key:

Username phải là ký tự chữ (Không phân biệt hoa thường)

Password = SumASCII(UpCase(Username)) XOR 0x5678 XOR 0x1234 (Dạng thập phân)