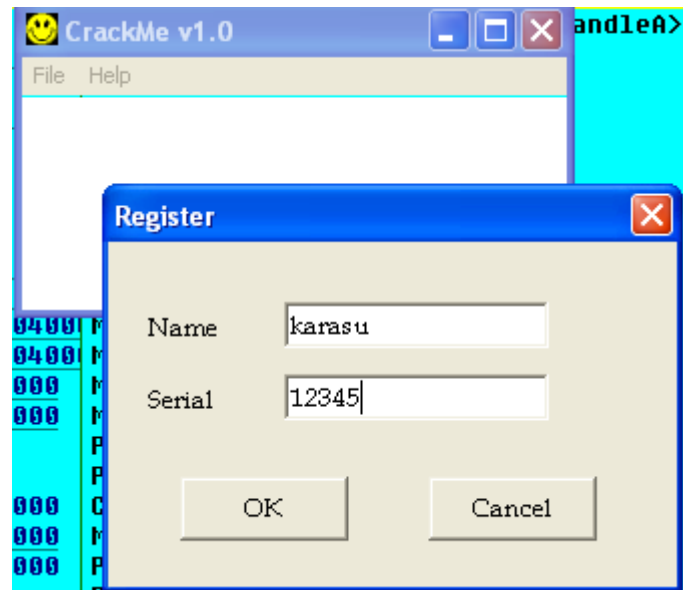


Particularmente Ricardo Narvaja mostraba en sus viejos tute cuando se usaba Olly como resolverlo – en INTRODUCCIÓN AL CRACKING CON OLLYDBG PARTE 16 – Si bien muestra cómo llegar a una solución, mi intención es abordarlo nuevamente después de tanto y realizar un keygen para que lo tome con cualquier nombre.

Bueno primero voy a poner un Bp en GetDlgItemTextA – esto se debe a que “Recupera el título o el texto asociado a un control en un cuadro de diálogo.” Según MSDN

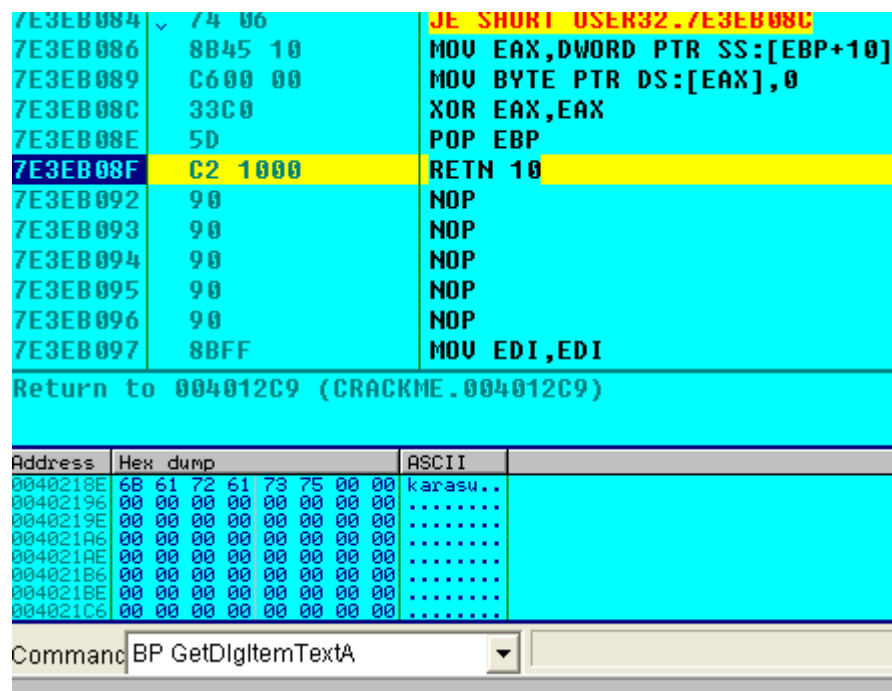
[illegible]

Le damos RUN y llegamos a la ventana correspondiente -> si damos en la opción "Help"-
 > Register aparece el famoso registro donde nos pide un nombre y un serial



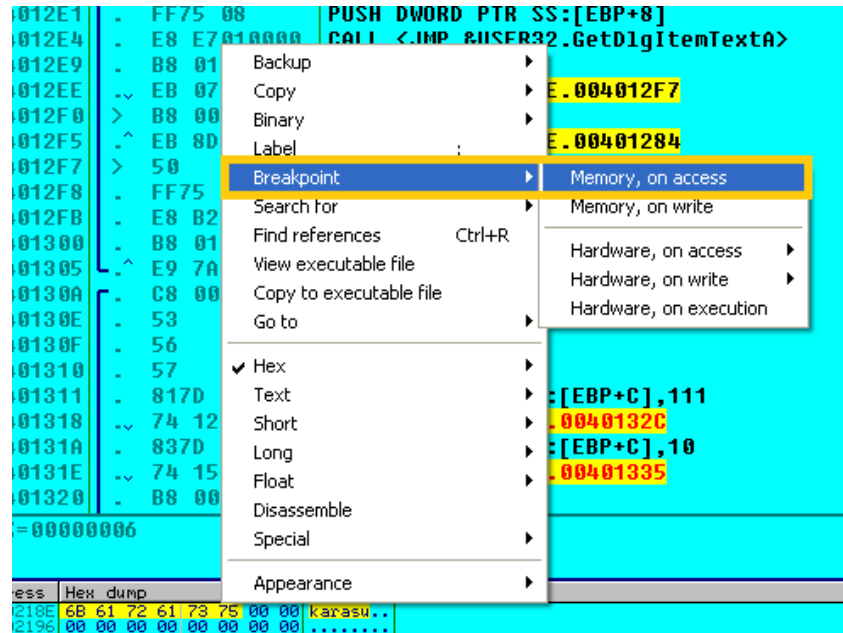
Así que comenzare a probar – el nombre usare “karasu” – el serial 12345 y le doy OK

Se detiene en la instrucción MOV EDI,EDI -> pero lo importante aquí es lo que aparece en el STACK -> allí se aloja lo que se ingresó en el Nombre – es decir “karasu” – bueno sigamos -> Click derecho -> Follow in Dump y si veo la ventana del dump está en 0 – esto es, como lo explica Ricardo – porque todavía no escribió, aun no se ejecutó la api todavía – le damos Ctrl + F9 o bien -> en la opción superior -> Debug -> Execute till return

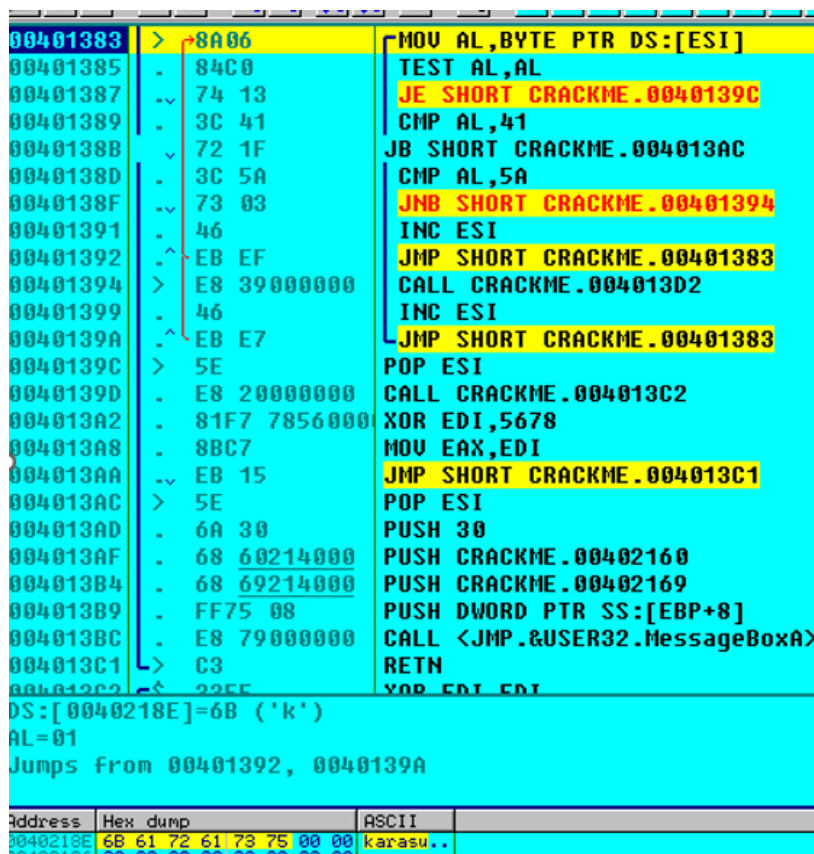


A partir de aquí voy a apartarme un poco de lo que se venía viendo en el tute original de Ricardo

Ahora si vemos que escribió el nombre – OK -> F7 – ahora lo que voy a hacer es marcar el valor hexa decimal del dump – marco 6B 61 72 61 73 75 – Luego voy a poner un Breakpoint On ACCES -> para ello -> click derecho ->



Y le damos F9 o RUN (x2 es decir dos veces para llegar a la instrucción) ->



Recordemos que para una instrucción antes –es decir que aún no se ejecuta “MOV AL, BYTE PTR DS:[EDI]” -> si vemos en la aclaración del Olly indica que [004018E] vale 6B (que en hexa es el valor de ‘k’ en la tabla ASCII) y se guarda en AL – sigo traceando y llego a este punto, creo yo que es importante:

The screenshot shows the OllyDbg interface with the assembly window displaying the following code:

```

004013C1 > C3                RETN
004013C2 $ 33FF             XOR EDI,EDI
004013C4 . 33DB             XOR EBX,EBX
004013C6 > 8A1E             MOV BL,BYTE PTR DS:[ESI]
004013C8 . 84DB             TEST BL,BL
004013CA ~ 74 05             JE SHORT CRACKME.004013D1
004013CC . 03FB             ADD EDI,EBX
004013CE . 46                INC ESI
004013CF ^ EB F5             JMP SHORT CRACKME.004013C6
004013D1 > C3                RETN
004013D2 $ 2C 20             SUB AL,20
004013D4 . 8806             MOV BYTE PTR DS:[ESI],AL
004013D6 . C3                RETN
004013D7 . C3                RETN
004013D8 $ 33C0             XOR EAX,EAX
004013DA . 33FF             XOR EDI,EDI
004013DC . 33DB             XOR EBX,EBX
004013DE . 8B7424 04        MOV ESI,DWORD PTR SS:[ESP+4]
004013E2 > B0 0A             MOV AL,0A
004013E4 . 8A1E             MOV BL,BYTE PTR DS:[ESI]
004013E6 . 84DB             TEST BL,BL
004013E8 ~ 74 0B             JE SHORT CRACKME.004013F5
004013EA . 80EB 30           SUB BL,30
004013ED . 0FAFF8           IMUL EDI,EAX
004013EF . 0000             ADD EDI,EBX

```

Below the assembly window, the register window shows:

```

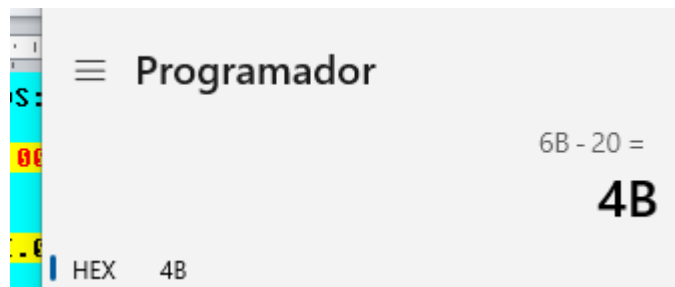
AL=4B ('K')
DS:[0040218E]=6B ('k')

```

At the bottom, the memory dump shows:

Address	Hex dump	ASCII
0040218E	6B 61 72 61 73 75 00 00	karasu..

Recordemos que AL = 6B -> luego le resta 20 en SUB AL, 20 = entonces esto quiere decir que ahora AL vale 4B



Y eso quiere decir que entonces que k – ahora es 4B (En ASCII “K”) y luego hay un RETUN (RET) -> entonces vuelve → Return to 00401399 (CRACKME.00401399)

Al seguir traceando -> llego otra vez al bucle -> observar que ahora toma la 'a' y AL conserva el valor 4B('K')

Address	Disassembly	Comment
00401383	> 8A 06	MOV AL, BYTE PTR DS:[ESI]
00401385	. 84 C0	TEST AL, AL
00401387	~ 74 13	JE SHORT CRACKME.0040139C
00401389	. 3C 41	CMP AL, 41
0040138B	~ 72 1F	JB SHORT CRACKME.004013AC
0040138D	. 3C 5A	CMP AL, 5A
0040138F	~ 73 03	JNB SHORT CRACKME.00401394
00401391	. 46	INC ESI
00401392	^ EB EF	JMP SHORT CRACKME.00401383
00401394	> E8 39 00 00 00	CALL CRACKME.004013D2
00401399	. 46	INC ESI
0040139A	^ EB E7	JMP SHORT CRACKME.00401383
0040139C	> 5E	POP ESI
0040139D	. E8 20 00 00 00	CALL CRACKME.004013C2
004013A2	. 81 F7 78 56 00 00	XOR EDI, 5678
004013A8	. 8B C7	MOV EAX, EDI
004013AA	~ EB 15	JMP SHORT CRACKME.004013C1
004013AC	> 5E	POP ESI
004013AD	. 6A 30	PUSH 30
004013AF	. 68 60 21 40 00	PUSH CRACKME.00402160
004013B4	. 68 69 21 40 00	PUSH CRACKME.00402169
004013B9	. FF 75 08	PUSH DWORD PTR SS:[EBP+8]
004013BC	. E8 79 00 00 00	CALL <JMP.&USER32.MessageBoxA>
004013C1	> C3	RETN
004013C2	^ 33 FF	XOR EDI, EDI

DS:[0040218F]=61 ('a')

AL=4B ('K')

Jumps from 00401392, 0040139A

En la siguiente instrucción AL = 61 sigo traceando y otra vez:

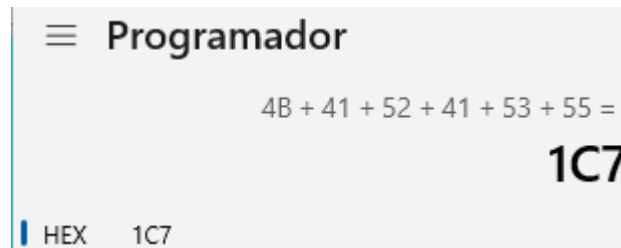
Address	Disassembly	Comment
004013D2	\$ 2C 20	SUB AL, 20
004013D4	. 88 06	MOV BYTE PTR DS:[ESI], AL
004013D6	. C3	RETN
004013D7	. C3	RETN
004013D8	\$ 33 C0	XOR EAX, EAX
004013DA	. 33 FF	XOR EDI, EDI
004013DC	. 33 DB	XOR EBX, EBX
004013DE	. 8B 74 24 04	MOV ESI, DWORD PTR SS:[ESP+4]
004013E2	> B0 0A	MOV AL, 0A
004013E4	. 8A 1E	MOV BL, BYTE PTR DS:[ESI]
004013E6	. 84 DB	TEST BL, BL
004013E8	~ 74 0B	JE SHORT CRACKME.004013F5
004013EA	. 80 EB 30	SUB BL, 30
004013ED	. 0F AF F8	IMUL EDI, EAX
004013F0	. 03 FB	ADD EDI, EBX
004013F2	. 46	INC ESI
004013F3	^ EB ED	JMP SHORT CRACKME.004013E2
004013F5	> 81 F7 34 12 00 00	XOR EDI, 1234
004013FB	. 8B DF	MOV EBX, EDI
004013FD	. C3	RETN
004013FE	~ FF 25 84 31 40 00	JMP DWORD PTR DS:[<&USER32.KillTimer>]
00401404	~ FF 25 88 31 40 00	JMP DWORD PTR DS:[<&USER32.GetSystemMetrics>]
0040140A	\$~ FF 25 8C 31 40 00	JMP DWORD PTR DS:[<&USER32.LoadCursorA>]
00401410	~ FF 25 90 31 40 00	JMP DWORD PTR DS:[<&USER32.LoadAcceleratorA>]
00401416	\$~ FF 25 94 31 40 00	JMP DWORD PTR DS:[<&USER32.MessageBoxA>]

AL=41 ('A')

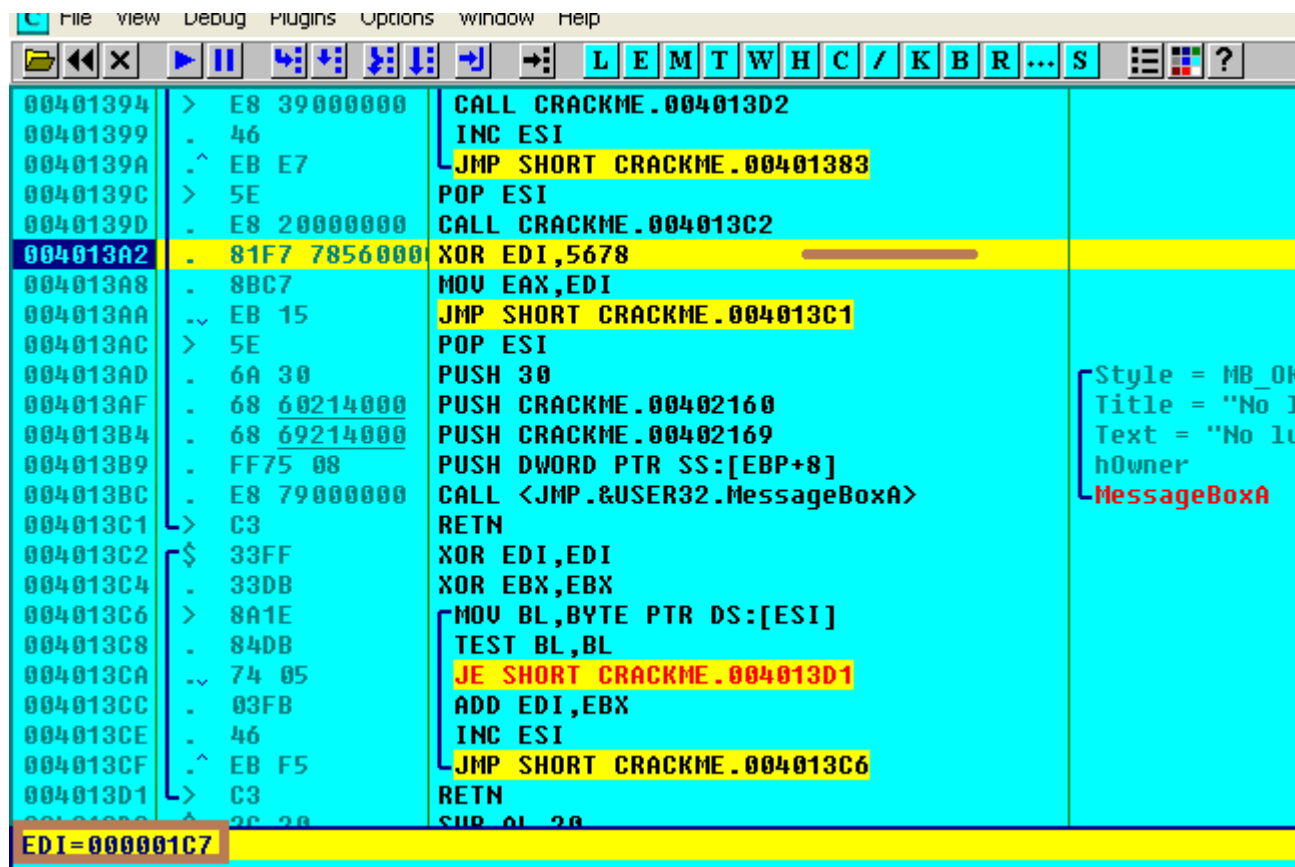
DS:[0040218F]=61 ('a')

Es decir, otra vez toma la siguiente letra y la convierte en mayúscula -> y luego va juntando el resultado en EDI – para no hacerlo tedioso, lo que hace es que recorre la string que se haya puesto en el nombre -> lo pone todo en mayúsculas -> Luego suma cada carácter – entonces si yo ingrese “karasu” -> lo va a convertir en “KARASU” básicamente y luego suma cada carácter → KARASU = 4B 41 52 41 53 55 (en Hexa)

Luego suma carácter por carácter



Una vez que recorre toda la string y sale del loop llegamos aquí:



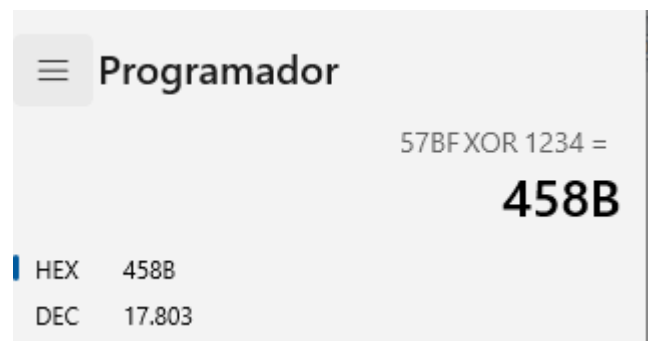
Notar que efectivamente EDI = 1C7 → Luego hace un XOR EDI, 5678 = 57BF posteriormente mueve ese valor a EAX ->

004013A8	. 8BC7	MOV EAX,EDI
004013AA	> EB 15	JMP SHORT CRACKME.004013B4
004013AC	> 5E	POP ESI
004013AD	. 6A 30	PUSH 30
004013AF	. 68 60214000	PUSH CRACKME.00402160
004013B4	. 68 69214000	PUSH CRACKME.00402169
004013B9	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
004013BC	. E8 79000000	CALL <JMP.&USER32.MessageBox>
004013C1	> C3	RETN
004013C2	\$ 33FF	XOR EDI,EDI
004013C4	. 33DB	XOR EBX,EBX
004013C6	> 8A1E	MOV BL,BYTE PTR DS:[ESI]
004013C8	. 84DB	TEST BL,BL
004013CA	> 74 05	JE SHORT CRACKME.004013D1
004013CC	. 03FB	ADD EDI,EBX
004013CE	. 46	INC ESI
004013CF	^ EB F5	JMP SHORT CRACKME.004013D1
004013D1	> C3	RETN
004013D2	\$ 3C 30	SUB AL,30
EDI=000057BF		
EAX=00000000		

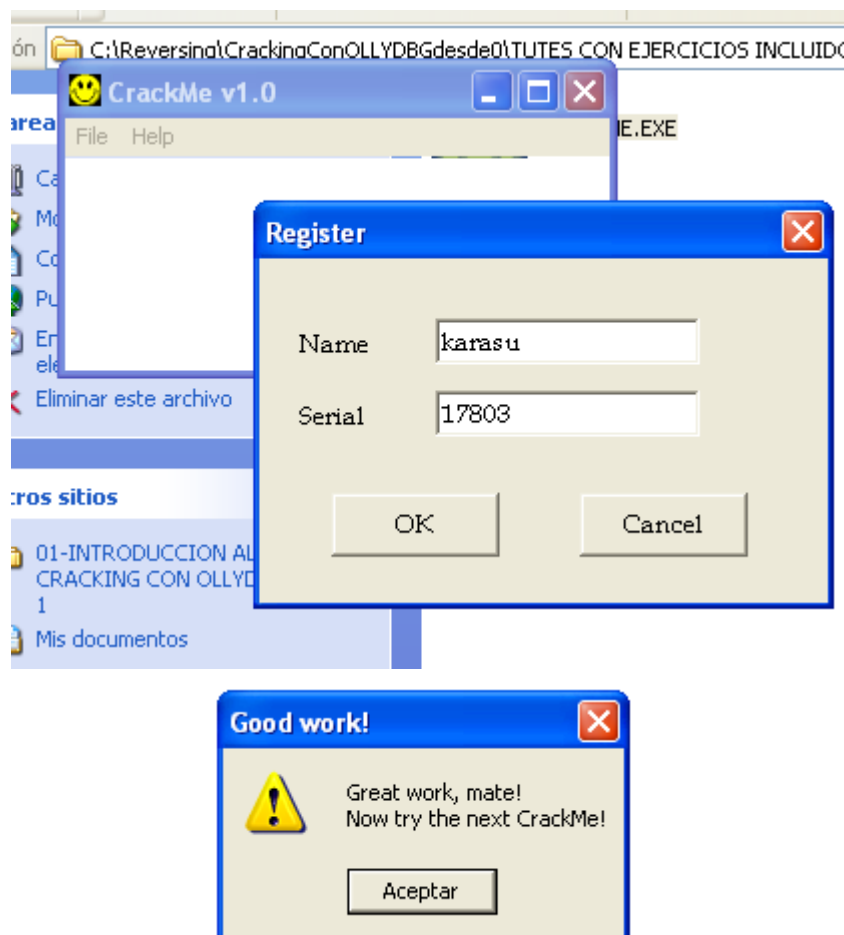
Luego seguimos traceando – es la parte donde evalúa nuestro serial – no lo mostrare ya que mi serial ya sé que es falso y llegamos a esta instancia:

004013D8	\$ 33C0	XOR EAX,EAX	
004013DA	. 33FF	XOR EDI,EDI	
004013DC	. 33DB	XOR EBX,EBX	
004013DE	. 8B7424 04	MOV ESI,DWORD PTR SS:[ESP+4]	
004013E2	> B0 0A	MOV AL,0A	
004013E4	. 8A1E	MOV BL,BYTE PTR DS:[ESI]	
004013E6	. 84DB	TEST BL,BL	
004013E8	> 74 0B	JE SHORT CRACKME.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 CRACKME.004013E2	
004013F5	> 81F7 34120000	XOR EDI,1234	
004013FB	. 8BDF	MOV EBX,EDI	
004013FD	. C3	RETN	
004013FE	. FF25 84314000	JMP DWORD PTR DS:[<&USER32.KillTimer>]	USER32.KillTimer
00401404	. FF25 88314000	JMP DWORD PTR DS:[<&USER32.GetSystemMetrics>]	USER32.GetSystemMetrics
0040140A	\$- FF25 8C314000	JMP DWORD PTR DS:[<&USER32.LoadCursorA>]	USER32.LoadCursorA
00401410	. FF25 90314000	JMP DWORD PTR DS:[<&USER32.LoadAccelerator>]	USER32.LoadAccelerator
00401416	\$- FF25 94314000	JMP DWORD PTR DS:[<&USER32.MessageBeep>]	USER32.MessageBeep
0040141C	. FF25 98314000	JMP DWORD PTR DS:[<&USER32.GetWindowRect>]	USER32.GetWindowRect
00401422	. FF25 9C314000	JMP DWORD PTR DS:[<&USER32.LoadStringA>]	USER32.LoadStringA
00401428	\$- FF25 A0314000	JMP DWORD PTR DS:[<&USER32.LoadIconA>]	USER32.LoadIconA
0040142E	. FF25 A4314000	JMP DWORD PTR DS:[<&USER32.LoadBitmapA>]	USER32.LoadBitmapA
EDI=00003039			
Jump from 004013E8			
Address	Hex dump	ASCII	
0040218E	4B 41 52 41 53 55 00 00	KARASU..	
00402196	00 00 00 00 00 00 00 00	

Ahí básicamente lo que hace es XOR EDI, 1234 – entonces eso quiere decir, que SI EDI = 57BF => 57BF XOR 1234 =



Es decir que ahora valdrá 458B = lo que quiere decir que el serial seria 17803 – el programa sigue y es cuando devuelve el cartel de chico malo – pero podemos probarlo –



Al dar OK -> devuelve el mensaje “Great work, mate! Now try the next Crackme!” → entonces le ganamos :D – algo que quizás no está demás, es que la clave 17803 sirve tanto como para la string “KARASU” como para “karasu”

Bueno llego el momento de hacer un script

Repasando entonces básicamente lo que hay que hacer es que el script me pida un nombre(string) -> Luego convertirlo a mayúsculas -> luego sumar carácter por carácter -> luego que haga XOR 5678 y a ese resultado que aplique XOR 1234 y luego que devuelva el valor numérico

```
MisProgramasEnPython > KeyGenCrackmeCrushead1999.py > ...
1  usu = input('Indique nombre de usuario: ')
2  usu = usu.upper()
3  key = 0
4  if usu.isalpha() == True:
5      for c in usu:
6          key = key + ord(c)
7          key = key^int(0x444C)
8      print('Su serial es: ' + str(key))
9  else:
10     print ("El usuario debe contener solo letras")
11
12
13
14
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS COMENTARIOS

```
PS D:\Eugenio\Propio\Desarrollo\MisProgramas> & D:/Eugenio/Propio/AreaTrabajo/Desarrollo/MisProgramas/MisProgramasEnPython/key.py
Indique nombre de usuario: karasu
Su serial es: 17803
PS D:\Eugenio\Propio\Desarrollo\MisProgramas> 
```

Corrección (15/08/2024)

Alguien me hizo notar que si se usa por ejemplo Zapallo (hacer una mención sencilla – con zapallo el anterior keygen funciona sin problemas) como usuario, al generar el serial en ese keygen inicial – no funcionaría y daría cartel de chico malo.

Por ello me puse a investigar, por que algo se me había pasado por alto y efectivamente así fue:

Al realizar el traceo nuevamente me doy cuenta que el problema es la Z

	Hex dump													ASCII	
E	5A	61	70	61	6C	6C	6F	00	00	00	00	00	00	00	Zapallo..
E	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Observar que Z es 5A en Hexa → bueno por que existe una línea que complica las cosas:

00401385	. 84C0	TEST AL,AL
00401387	. 74 13	JE SHORT CRACKME.0040139C
00401389	. 3C 41	CMP AL,41
0040138B	. 72 1F	JB SHORT CRACKME.004013AC
0040138D	. 3C 5A	CMP AL,5A
0040138F	. 73 03	JNB SHORT CRACKME.00401394
00401391	. 46	INC ESI
00401392	. ^EB EF	JMP SHORT CRACKME.00401383
00401394	> E8 39000000	CALL CRACKME.004013D2
00401399	. 46	INC ESI
0040139A	. ^EB E7	JMP SHORT CRACKME.00401383
0040139C	> 5E	POP ESI
0040139D	. E8 20000000	CALL CRACKME.004013C2
004013A2	. 81F7 78560000	XOR EDI,5678
004013A8	. 8BC7	MOV EAX,EDI
004013AA	. ^EB 15	JMP SHORT CRACKME.004013C1
004013AC	> 5E	POP ESI
004013AD	. 6A 30	PUSH 30
004013AF	. 68 60214000	PUSH CRACKME.00402160
004013B4	. 68 69214000	PUSH CRACKME.00402169
004013B9	. FF75 08	PUSH DWORD PTR SS:[EBP+8]
004013BC	. E8 79000000	CALL <JMP.&USER32.MessageBoxA>
004013C1	> C3	RETN
004013C2	. \$ 33FF	XOR EDI,EDI
004013C4	. 33DB	XOR EBX,EBX
004013C6	> 8A1E	MOV BL,BYTE PTR DS:[ESI]
AL=5A ('Z')		

Allí cuando compara y comienza a hacer el bucle de convertir los caracteres a mayúsculas y etc.- Z(Siendo 5A) estando en AL – es comparado contra 5A esto causa que cuando llegue a la instrucción

004013C4	. 33DB	XOR EBX,EBX	Registers (FP EAX 0000003A ECX 0012FDE4 EDX 7C91E4F4 EBX 00000000 ESP 0012FE98 EBP 0012FEB4 ESI 0040218E EDI 0012FF1C EIP 004013D4 C 0 ES 0023 P 1 CS 001B A 0 SS 0023 Z 0 DS 0023 S 0 FS 003B T 0 GS 0000 D 0 O 0 LastErr EFL 00010206 ST0 empty -?? ST1 empty -?? ST2 empty -?? ST3 empty -?? ST4 empty -NA ST5 empty 1.0 ST6 empty 1.0 ST7 empty 1.0
004013C6	> 8A1E	MOV BL, BYTE PTR DS: [ESI]	
004013C8	. 84DB	TEST BL, BL	
004013CA	. 74 05	JE SHORT CRACKME.004013D1	
004013CC	. 03FB	ADD EDI, EBX	
004013CE	. 46	INC ESI	
004013CF	. ^EB F5	JMP SHORT CRACKME.004013C6	
004013D1	> C3	RETN	
004013D2	\$ 2C 20	SUB AL, 20	
004013D4	. 8806	MOV BYTE PTR DS: [ESI], AL	
004013D6	. C3	RETN	
004013D7	. C3	RETN	
004013D8	\$ 33C0	XOR EAX, EAX	
004013DA	. 33FF	XOR EDI, EDI	
004013DC	. 33DB	XOR EBX, EBX	
004013DE	. 8B7424 04	MOV ESI, DWORD PTR SS: [ESP+4]	
004013E2	> B0 0A	MOV AL, 0A	
004013E4	. 8A1E	MOV BL, BYTE PTR DS: [ESI]	
004013E6	. 84DB	TEST BL, BL	
004013E8	. 74 0B	JE SHORT CRACKME.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 CRACKME.004013E2	
004013F5	> 81F7 34120000	XOR EDI, 1234	
004013FB	. 8BDF	MOV EBX, EDI	
AL=3A (':')			
DS: [0040218E]=5A ('Z')			

Al llegar al SUB AL,20 -> causa que 5A -20 = 3ª y eso quiere decir que eso es representado

la salida seria así:

Address	Hex dump	ASCII
0218E	3A 41 50 41 4C 4C 4F 00 00 00 00 00 00 00 00 00	: APALLO.....
0219E	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Entonces hay que modificar el script por que eso no lo tiene contemplado, por ello se modifica para que cuando se usa un nombre que comience con "Z" -> lo reemplace con ":" y realice las operaciones desde allí.

Usando "Zorro" como nombre: realicé los cambios en el script y tuve estos resultados:

```
MisProgramasEnPython > KeygenCrackmeCRUSHEAD1999.py > ...
1 | import re
2 | usu = input('Indique nombre de usuario: ')
3 | permitido = r"[a-zA-Z0-9:]"
4 | usu = usu.upper()
5 | if re.match(f"^{permitido}+$", usu):
6 |     if usu[0] == 'Z':
7 |         usu = ':' + usu[1:]
8 |     key = 0
9 |     if usu:
10 |         for c in usu:
11 |             key = key + ord(c)
12 |             key = key^int(0x444C)
13 |             print('Su serial es: ' + str(key))
14 | else:
15 |     print ("El usuario debe contener solo letras")
16 |
17 |
18 |
19 |
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL PUERTOS COMENTARIOS

```
PS D:\Programas\Programacion\MisProgramas> & D:/Programas/Python/Python3
USHEAD1999.py
Indique nombre de usuario: Zorro
Su serial es: 17712
```

