

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]



Software	Keygenme v1.0 por ZLT
Protección	Serial.
Herramientas	Windows 7 Home Premium SP1 x32 Bits (S.O donde trabajamos.) X64DBG (Feb 14 2018) RDG Packer Detector v0.7.6.2017 Microsoft Visual Studio 2017 DESCARGAR HERRAMIENTAS DESCARGAR TUTO+ARCHIVOS
SOLUCIÓN	KEYGEN
AUTOR	LUISFECAB
RELEASE	Septiembre 8 2018 [TUTORIAL 008]

INTRODUCCIÓN

En el tutorial anterior, [1662](#); quedé comprometido en completar el Crack y Patch para la versión de x64 Bits del <Nitro Pro v12.1.0.195> pero como sigo sin tener acceso a un Windows x64 Bits, pues lógicamente no he podido cumplir ese compromiso, y mientras eso pase no me pienso quedarme quieto sin poder escribir otro tuto, así que voy a aprovechar y hacer el reto de Zelt@ con su <Keygenme v1.0 por ZLT> que unos meses largos atrás lo puso en la lista **CracksLatinoS**. Aprovecharé para hacerlo con ayuda del **x64DBG** y con eso empezar a usarlo para ir cogiéndole "el chuate" para mis futuras aventuras en Reversing, sobre todo para aplicaciones de x64 Bits.

Como siempre saludos para toda la lista, en especial para todos aquellos que pasan, comentan y ayudan a resolver dudas; o dejando sus tutoriales para beneficio de todos.

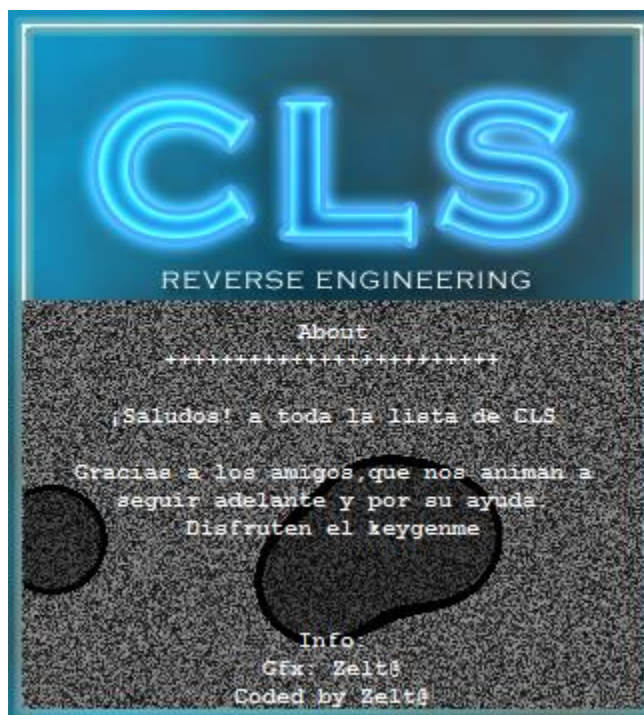
Espero que este pequeño aporte sea otro granito de arena que enriquezca, aún más, la gran biblioteca que es **CracksLatinoS**.

ANALISIS INICAL

Ejecutemos el <Keygenme v1.0 por ZLT> para verlo en acción.

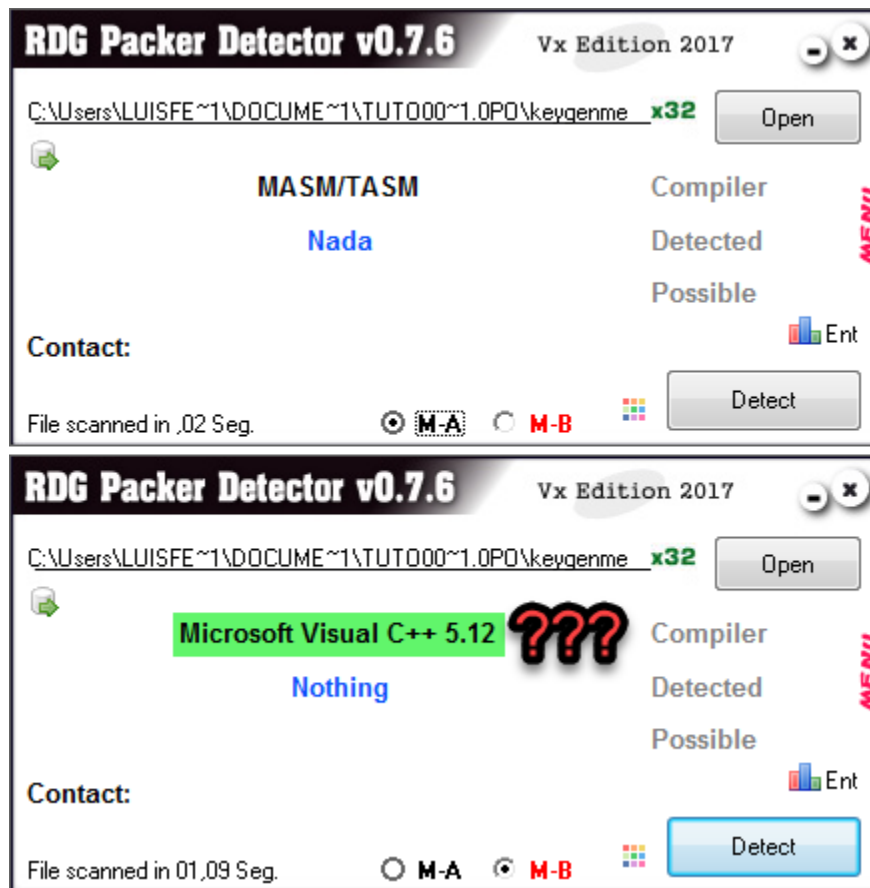


Es de lo más atractivo visualmente, me gusta su presentación, creo que tiene que ver que el azul es mi color favorito y este tiene tremendos contrastes de azul. Sería muy vacano poder hacer cosas como esta, bueno espero lograrlo algún día. Revisemos su "About".



[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

Muy chulo el "About" también, buena música y el efecto me recuerda al T1000, en Terminator #2. Desde este tuto aprovecho para responder tus palabras del "About", saludos y agradecerte por compartir tus conocimientos con todos nosotros. Resulta que Zelt@ lo programó en ASM, más sin embargo lo chequearemos con el <RDG Packer Detector v0.7.6.2017> para sí trae algo más.

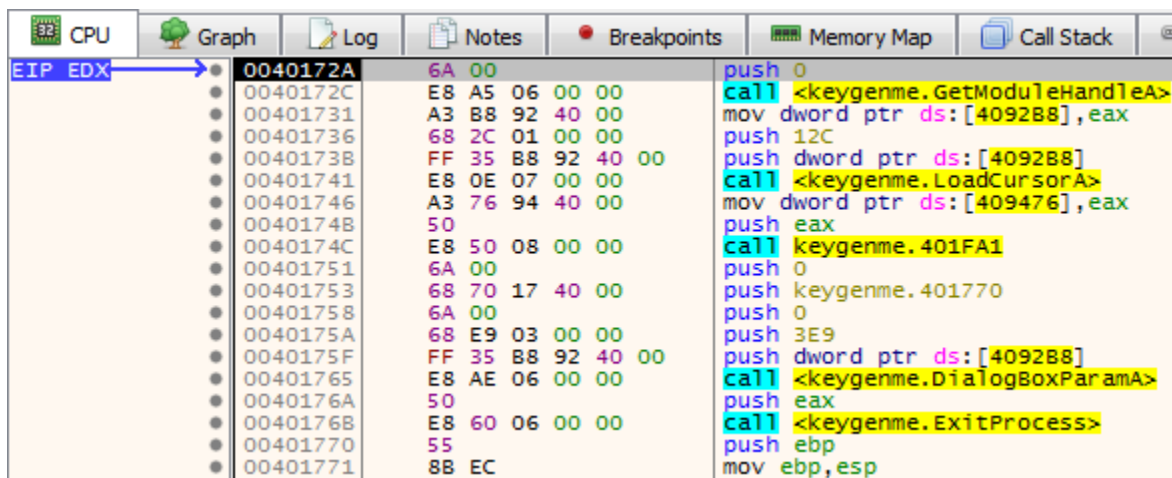


El **M-A** confirma lo que dice Zelt@ en el "About", pero al revisar el scan con **M-B** dice que está programado un <Microsoft Visual C++ 5.12> pues creo que ahí se confunde el <RDG Packer Detector v0.7.6.2017>.


Ya lo examinamos y es un KeyGenMe sin empacar y al parecer sin ningún truco antidebugging. Ya con eso lo podemos cerrar y cargarlo en nuestro **x64DBG**.

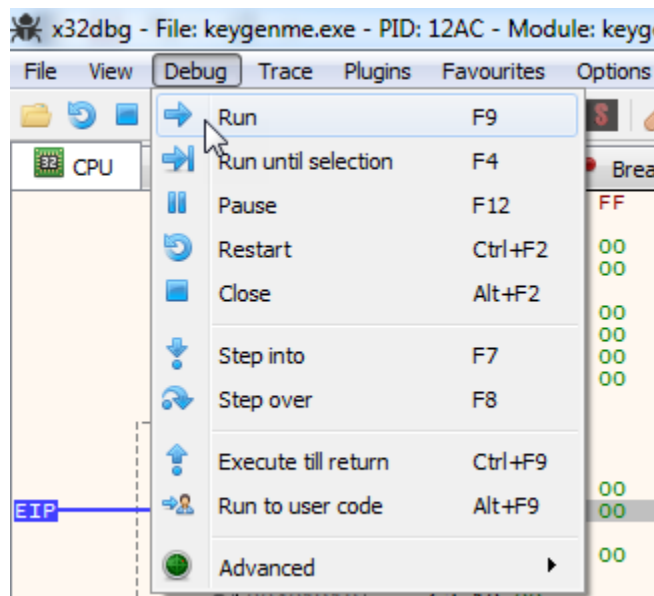
AL ATAQUE

Lo abrimos con el **x64DBG** y si miramos en ensamblado tiene toda la pinta de un ASM, solo pienso cómo se puede equivocar el **<RDG Packer Detector v0.7.6.2017>** si este KeyGenMe está limpio sin nada raro, o es que yo soy el perdido.



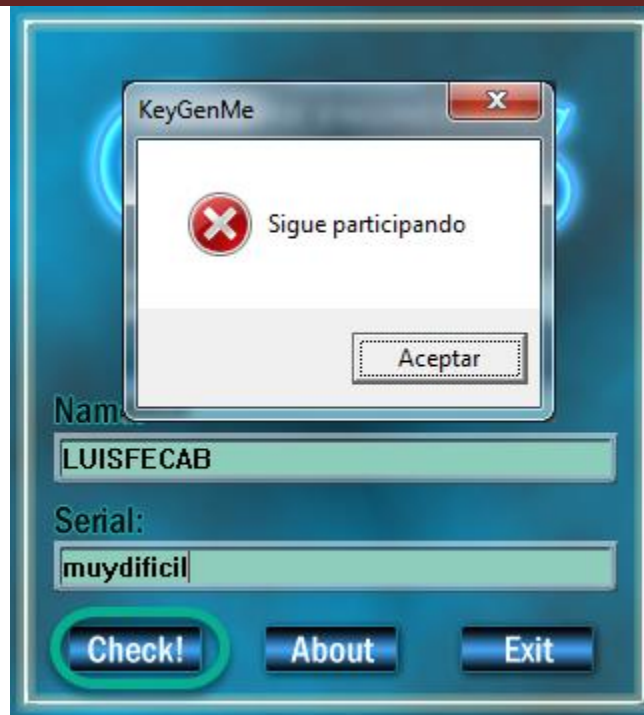
Address	Disassembly	Comment
0040172A	6A 00	push 0
0040172C	E8 A5 06 00 00	call <keygenme.GetModuleHandleA>
00401731	A3 B8 92 40 00	mov dword ptr ds:[4092B8],eax
00401736	68 2C 01 00 00	push 12C
0040173B	FF 35 B8 92 40 00	push dword ptr ds:[4092B8]
00401741	E8 0E 07 00 00	call <keygenme.LoadCursorA>
00401746	A3 76 94 40 00	mov dword ptr ds:[409476],eax
0040174B	50	push eax
0040174C	E8 50 08 00 00	call keygenme.401FA1
00401751	6A 00	push 0
00401753	68 70 17 40 00	push keygenme.401770
00401758	6A 00	push 0
0040175A	68 E9 03 00 00	push 3E9
0040175F	FF 35 B8 92 40 00	push dword ptr ds:[4092B8]
00401765	E8 AE 06 00 00	call <keygenme.DialogBoxParamA>
0040176A	50	push eax
0040176B	E8 60 06 00 00	call <keygenme.ExitProcess>
00401770	55	push ebp
00401771	8B EC	mov ebp,esp

Me quedó faltando en el **ANÁLISIS INICIAL** meterles unos datos y ver qué me salía, así que eso haremos ahora, corramos el programa con **<F9>** o , o también desde la barra de herramientas **<Debug>**.

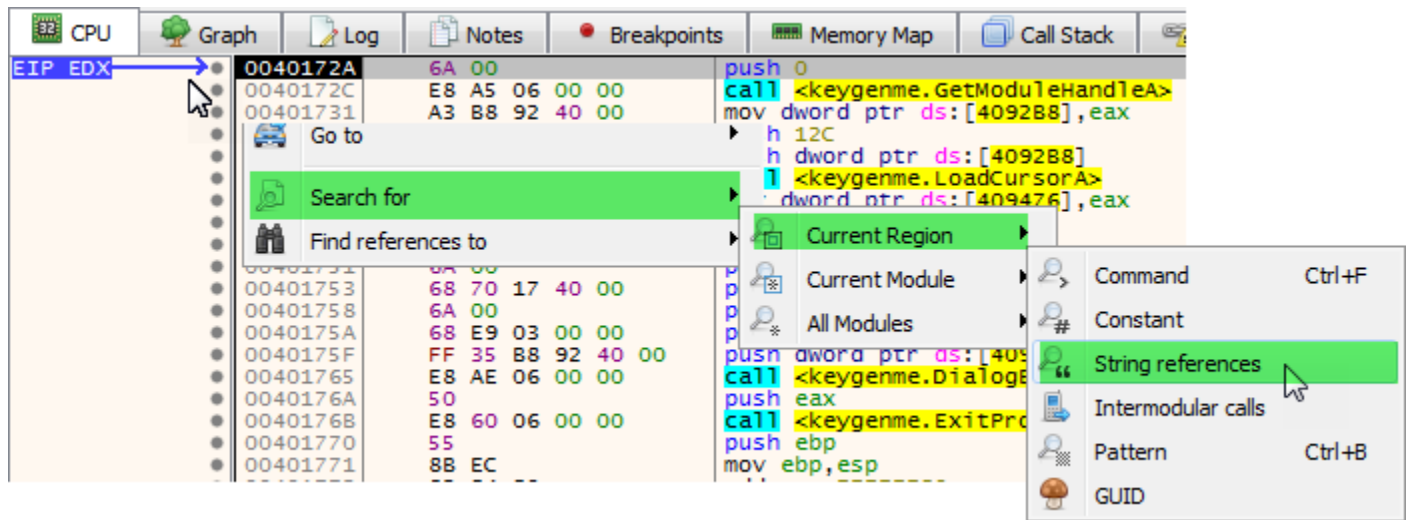


Podemos ver que ahí tenemos todas las opciones de trazo básico y sus accesos rápidos. Mis datos son los de siempre, mi nick **LUISFECAB** y mi serial de batalla **"muydifícil"**.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]



Ingresé mis datos y clic al botón "Check!" para ver al "CHICO MALO". "Sigue participando", pensé que me diría "Sigue practicando", pero ahora que lo pienso tiene mucha razón participa y ganarás satisfacción y experiencia. Listo, iniciaremos buscando por las Strings, <Click derecho->Search for->Current Region->String references>.



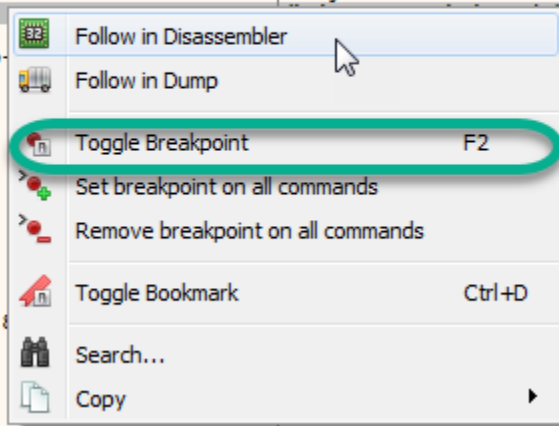
El **x64DBG** ofrece más opciones para buscar, yo voy a escoger que me busque en "Current Region" y luego les hacharé un vistazo a las otras para ver que muestra y así de a poquitos vamos conociendo el **x64DBG**.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

Address	Disassembly	String
00401391	mov eax,dword ptr ds:[40911B]	".310"
004013B9	mov dword ptr ds:[40911B],ecx	".310"
004017B7	push keygenme.409000	"KeyGenMe V 1.0"
00401D06	push keygenme.409158	"KeyGenMe"
00401D0B	push keygenme.409123	"Sigue participando"
00401D1E	push keygenme.409158	"KeyGenMe"
00401F81	mov dword ptr ss:[ebp-8],keygenme.40919C	"AniGIF"
00402271	push keygenme.40919C	"AniGIF"
00402276	push keygenme.4091D9	"This file has not a valid Gif signature."
00402516	push keygenme.40919C	"AniGIF"
0040251B	push keygenme.409202	"Fatal error."
00402616	push keygenme.4091A5	"21F9"
0040262A	push keygenme.4091AA	"3B"
004027F7	push keygenme.40919C	"AniGIF"
004027FC	push keygenme.4091AD	"Error loading file."
004028AF	push keygenme.40919C	"AniGIF"
004028B4	push keygenme.4091C1	"Error loading resource."
00402E9B	push keygenme.40921F	"open"
004064A5	mov esi,keygenme.406280	"mc@"

Ahí tenemos las **Strings** que nos arroja el **x64DBG** y resaltado en **VERDE** está "**Sigue participando**" que es el mensaje del **<CHICO MALO>**. Vallamos a esa dirección **00401D0B**.

Address	Disassembly	String
00401391	mov eax,dword ptr ds:[40911B]	".310"
004013B9	mov dword ptr ds:[40911B],ecx	".310"
004017B7	push keygenme.409000	"KeyGenMe V 1.0"
00401D06	push keygenme.409158	"KeyGenMe"
00401D0B	push keygenme.409123	"Sigue participando"
00401D1E	push keygenme.409158	"KeyGenMe"
00401F81	mov dword ptr ss:[ebp-8],keygenme.40919C	"AniGIF"
00402271	push keygenme.40919C	"AniGIF"
00402276	push keygenme.4091D9	"This file has not a valid Gif signature."
00402516	push keygenme.40919C	"AniGIF"
0040251B	push keygenme.409202	"Fatal error."
00402616	push keygenme.4091A5	"21F9"
0040262A	push keygenme.4091AA	"3B"
004027F7	push keygenme.40919C	"AniGIF"
004027FC	push keygenme.4091AD	"Error loading file."
004028AF	push keygenme.40919C	"AniGIF"
004028B4	push keygenme.4091C1	"Error loading resource."
00402E9B	push keygenme.40921F	"open"
004064A5	mov esi,keygenme.406280	"mc@"



Podemos seguirla si la seleccionamos y presionamos **<ENTER>**, también con **<Click Derecho->Follow in Disassembler>**. Tiene otras opciones y que por fortuna es muy parecido a nuestro querido y un poco viejito **<OllYDBG v1.10>**. Resalte **<Toggle Breakpoint>** porque al final es lo que quería hacer para que pare cuando pase por ahí y es cuando está preparando el mensaje del **<CHICO MALO>**.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

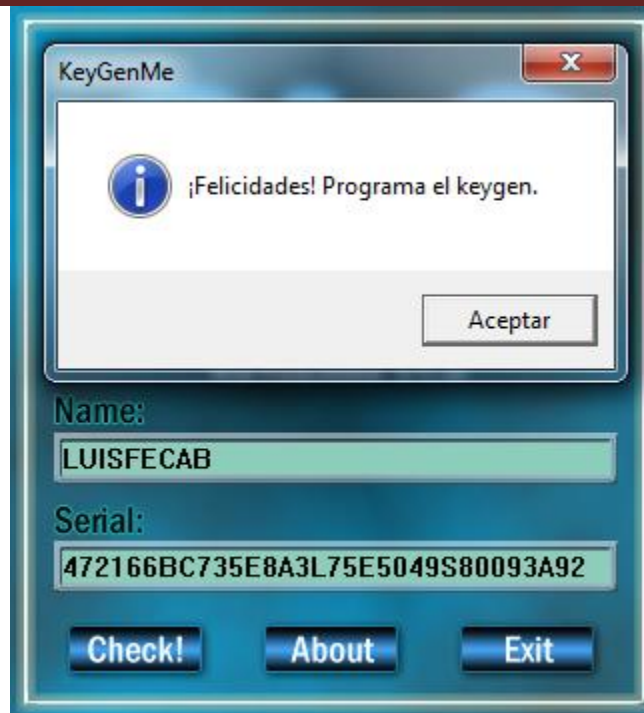
00401CB4	FF 35 E4 92 40 00	push dword ptr ds:[4092E4]	
00401CBA	E8 DD 01 00 00	call <keygenme.ShowWindow>	
00401CBF	6A 32	push 32	
00401CC1	68 18 93 40 00	push keygenme.409318	
00401CC6	68 F5 03 00 00	push 3F5	
00401CC8	FF 75 08	push dword ptr ss:[ebp+8]	
00401CCE	E8 69 01 00 00	call <keygenme.GetDlgItemTextA>	OBTIENE EL USER INGRESADO
00401CD3	FF 75 08	push dword ptr ss:[ebp+8]	
00401CD6	E8 CA F7 FF FF	call keygenme.4014A5	ORIGINA SERIAL CON NUESTRO USER INGRESADO
00401CDB	6A 32	push 32	
00401CDD	68 07 B5 40 00	push keygenme.40B507	
00401CE2	68 F6 03 00 00	push 3F6	
00401CE7	FF 75 08	push dword ptr ss:[ebp+8]	
00401CEA	E8 4D 01 00 00	call <keygenme.GetDlgItemTextA>	OBTIENE EL SERIAL INGRESADO
00401CEF	68 07 A5 40 00	push keygenme.40A507	
00401CF4	68 07 B5 40 00	push keygenme.40B507	
00401CF9	E8 02 01 00 00	call <keygenme.lstrcmp>	COMPARA EL SERIAL
00401CFE	85 C0	test eax, eax	
00401D00	74 1A	je keygenme.401D1C	SALTO DESCIVO
00401D02	EB 00	jmp keygenme.401D04	
00401D04	6A 10	push 10	
00401D06	68 58 91 40 00	push keygenme.409158	409158: "KeyGenMe"
00401D08	68 23 91 40 00	push keygenme.409123	409123: "Sigue participando"
00401D10	FF 75 08	push dword ptr ss:[ebp+8]	
00401D13	E8 48 01 00 00	call <keygenme.MessageBoxA>	
00401D18	C9	leave	
00401D19	C2 10 00	ret 10	
00401D1C	6A 40	push 40	
00401D1E	68 58 91 40 00	push keygenme.409158	409158: "KeyGenMe"
00401D23	68 36 91 40 00	push keygenme.409136	
00401D28	FF 75 08	push dword ptr ss:[ebp+8]	

Con la imagen de arriba tenemos todo explicado, creo que podemos decir que hemos hecho todo el análisis estático. Me enfocaré en lo resaltado en VERDE en la dirección 00401CFE test eax, eax. Para evitar al <CHICO MALO> la API lstrcmp debe retornar 0x0 y para que eso suceda, que en este caso son los dos seriales deben ser iguales y de esa forma se activará el FLAG-Z a 1 y el salto será tomado, y bye bye <CHICO MALO>. Podríamos cambiar la dirección 00401D00 je keygenme.401D1C por un salto incondicional (jmp), así siempre saltaremos al <CHICO BUENO>, 00401D00 jmp keygenme.401D1C. Podemos colocar un <BREAKPOINT> en la API lstrcmp y colocamos nuestros datos y los probamos.

Graph	Log	Notes	Breakpoints	Memory Map	Call Stack	SEH	Script	Symbols	Source	Referer
00401CBF	6A 32	push 32								
00401CC1	68 18 93 40 00	push keygenme.409318								409318: "LUISFECAB"
00401CC6	68 F5 03 00 00	push 3F5								
00401CC8	FF 75 08	push dword ptr ss:[ebp+8]								
00401CCE	E8 69 01 00 00	call <keygenme.GetDlgItemTextA>								
00401CD3	FF 75 08	push dword ptr ss:[ebp+8]								
00401CD6	E8 CA F7 FF FF	call keygenme.4014A5								
00401CDB	6A 32	push 32								
00401CDD	68 07 B5 40 00	push keygenme.40B507								40B507: "muydifícil"
00401CE2	68 F6 03 00 00	push 3F6								
00401CE7	FF 75 08	push dword ptr ss:[ebp+8]								
00401CEA	E8 4D 01 00 00	call <keygenme.GetDlgItemTextA>								
00401CEF	68 07 A5 40 00	push keygenme.40A507								40A507: "472166BC735E8A3L75E5049S80093A92"
00401CF4	68 07 B5 40 00	push keygenme.40B507								40B507: "muydifícil"
00401CF9	E8 02 01 00 00	call <keygenme.lstrcmp>								
00401CFE	85 C0	test eax, eax								
00401D00	74 1A	je keygenme.401D1C								
00401D02	EB 00	jmp keygenme.401D04								
00401D04	6A 10	push 10								
00401D06	68 58 91 40 00	push keygenme.409158								409158: "KeyGenMe"
00401D08	68 23 91 40 00	push keygenme.409123								409123: "Sigue participando"
00401D10	FF 75 08	push dword ptr ss:[ebp+8]								
00401D13	E8 48 01 00 00	call <keygenme.MessageBoxA>								
00401D18	C9	leave								
00401D19	C2 10 00	ret 10								

Ahí tenemos las dos variables y resaltado en VERDE el serial verdadero para nuestro user, "LUISFECAB", que sería "472166BC735E8A3L75E5049S80093A92". Probémoslo a ver cómo nos va.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]



Ese es nuestro serial, pero no es suficiente. Hemos analizado un poco lo que podríamos hacer y vencer el KeyGenMe, pero el reto es hacer un KeyGen como lo dice el "CHICO BUENO" y las instrucciones .txt.

Reglas:

- 1.- No parchar
- 2.- Encontrar rutina para generar serial correcto
- 3.- Programar keygen en lenguaje preferido
- 4.- Escribir tutorial

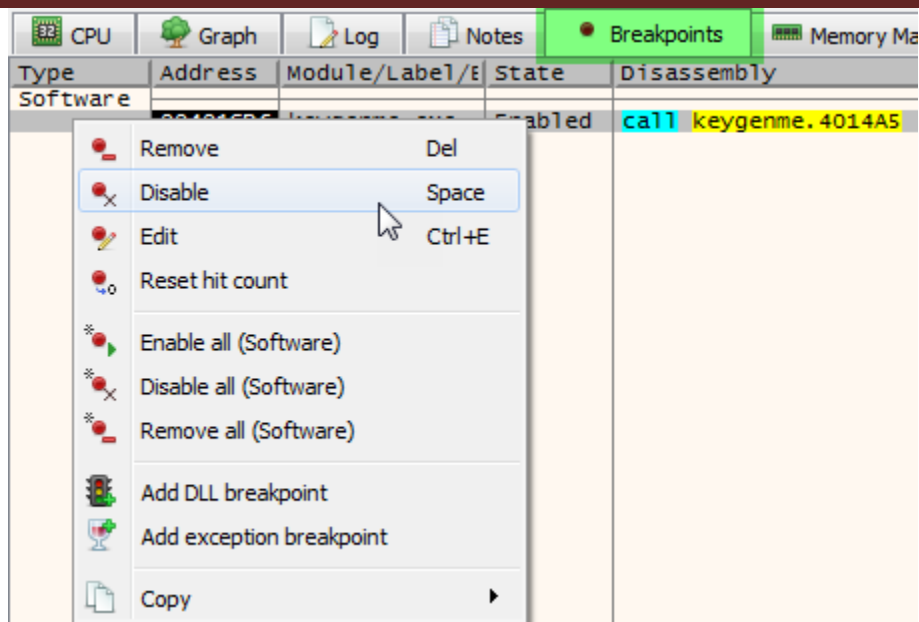
Feliz crackeo!!!!

Listo, como ya sabemos dónde está la rutina donde se origina el serial, entonces pongámosle un <BREAKPOINT> ahí y quitemos los que habíamos puesto.

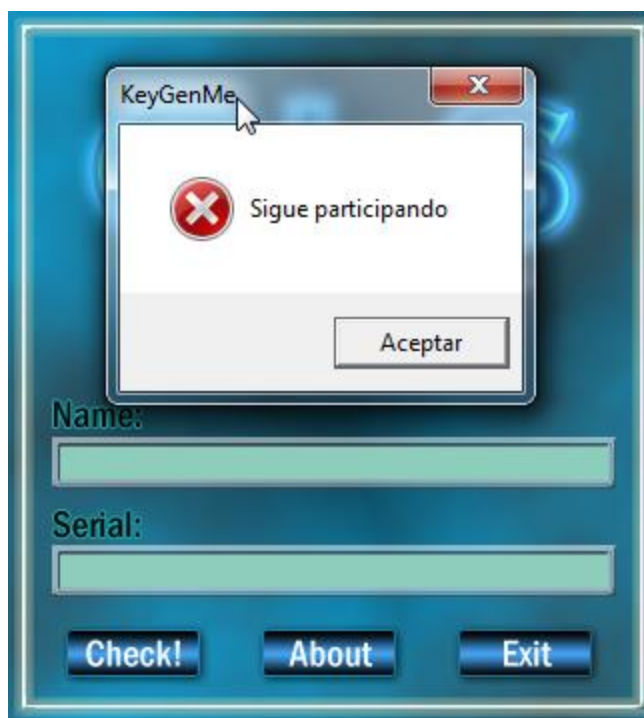
●	00401CCB	FF 75 08	push dword ptr ss:[ebp+8]
●	00401CCE	E8 69 01 00 00	call <keygenme.GetDlgItemTextA>
●	00401CD3	FF 75 08	push dword ptr ss:[ebp+8]
●	00401CD6	E8 CA F7 FF FF	call keygenme.4014A5
●	00401CDB	6A 32	push 32
●	00401CDD	68 07 B5 40 00	push keygenme.40B507
●	00401CE2	68 F6 03 00 00	push 3F6
●	00401CE7	FF 75 08	push dword ptr ss:[ebp+8]
●	00401CEA	E8 4D 01 00 00	call <keygenme.GetDlgItemTextA>
●	00401CEF	68 07 A5 40 00	push keygenme.40A507
●	00401CF4	68 07 B5 40 00	push keygenme.40B507
●	00401CF9	E8 02 01 00 00	call <keygenme.lstrcmp>
●	00401CFE	85 C0	test eax, eax
--	00401D00	74 1A	je keygenme.401D1C

Aquí pensando, vamos a deshabilitar nuestro <BREAKPOINT> utilizando para eso la pestaña "Breakpoints".

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]




Ahí podemos ver el resto de opciones que tenemos. Lo deshabilitamos y después con el KeyGeyMe sin meter ningún dato, todo en blanco y vamos a "Check!".



Muestra el <CHICO MALO>. No hace ninguna comprobación de nombre o serial vacíos, pensaba que diría falta ingresar user o algo parecido. Bueno ahora sí, habilitemos nuestro <BREAKPOINT>, metamos nuestros datos y los revisamos con "Check!".

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

Graph	Log	Notes	Breakpoints	Memory Map	Call Stack	SEH	Script	Symbol
00401CBF	6A 32				push 32			
00401CC1	68 18 93 40 00				push keygenme.409318			409318: "LUISFECAB"
00401CC6	68 F5 03 00 00				push 3F5			
00401CCB	FF 75 08				push dword ptr ss:[ebp+8]			
00401CCE	E8 69 01 00 00				call <keygenme.GetDlgItemTextA>			
00401CD3	FF 75 08				push dword ptr ss:[ebp+8]			
00401CD6	E8 CA F7 FF FF				call keygenme.4014A5			
00401CDB	6A 32				push 32			
00401CDD	68 07 B5 40 00				push keygenme.408507			
00401CE2	68 F6 03 00 00				push 3F6			
00401CE7	FF 75 08				push dword ptr ss:[ebp+8]			
00401CEA	E8 4D 01 00 00				call <keygenme.GetDlgItemTextA>			

Estamos parados en la entrada de la rutina que genera el serial. Entremos traciando con <F7> o .

004014A5	55				push ebp			
004014A6	8B EC				mov ebp,esp			
004014A8	68 1F 91 40 00				push keygenme.40911F			
004014AD	68 07 B1 40 00				push keygenme.408107			
004014B2	E8 27 0A 00 00				call <keygenme.GetUserNameA>			
004014B7	68 18 93 40 00				push keygenme.409318			
004014BC	68 07 95 40 00				push keygenme.409507			
004014C1	E8 40 09 00 00				call <keygenme.lstrncpy>			
004014C6	68 61 91 40 00				push keygenme.409161			
004014CB	68 07 95 40 00				push keygenme.409507			
004014D0	E8 25 09 00 00				call <keygenme.lstrcatA>			
004014D5	68 64 91 40 00				push keygenme.409164			
004014DA	68 07 95 40 00				push keygenme.409507			
004014DF	E8 16 09 00 00				call <keygenme.lstrcatA>			
004014E4	68 68 91 40 00				push keygenme.409168			
004014E9	68 07 95 40 00				push keygenme.409507			
004014EE	E8 07 09 00 00				call <keygenme.lstrcatA>			
004014F3	68 6C 91 40 00				push keygenme.40916C			
004014F8	68 07 95 40 00				push keygenme.409507			
004014FD	E8 F8 08 00 00				call <keygenme.lstrcatA>			
00401500	68 07 91 40 00				push keygenme.409167			
00401505	68 07 95 40 00				push keygenme.409507			
00401508	E8 4C 08 00 00				call <keygenme.lstrlenA>			
0040150C	A3 07 99 40 00				mov dword ptr ds:[409907],eax			
0040150E	33 C0				xor eax,eax			
00401510	33 C9				xor ecx,ecx			
00401512	0F BE 88 07 9D 4				movsx ecx,byte ptr ds:[eax+409D07]			
00401514	80 F1 1E				xor cl,1E			
00401516	88 88 07 A1 40 0				mov byte ptr ds:[eax+40A107],cl			
00401518	40				inc eax			
0040151A	3B 05 07 99 40 0				cmp eax,dword ptr ds:[409907]			
0040151C	72 E7				jb keygenme.4015C9			

TODO ESTO PARA COCATENAR
(UNIR)
"LUISFECAB" + "CrackS-Latinos"

CALCULA LONGITUD
"LUISFECAB CrackS-Latinos"
EAX=0x18 (24)

La parte resaltada en VERDE concatena (une) nuestro serial, "LUISFECAB" con la constante "CrackS-Latinos" quedando, "LUISFECAB CrackS-Latinos". Luego halla la longitud de lo anterior, que es 0x18 (24 caracteres, contando el espacio) y con eso entra a un LOOP resaltado en la parte ROJA donde crea una constante igual a nuestra longitud.

Algo que se me había pasado mencionar es que lo primero que hace es averiguar el administrador de la cuenta con la API_GetUserNameA en la dirección 004014B2 y que la usará más adelante para calcular el serial. En mi caso lo que retorna la API_GetUserNameA es igual a mi nombre ingresado porque tienen mi nombre de usuario, "LUISFECAB".

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

00401663	mov dword ptr ds:[409907],eax	Muevo 0x18 (longitud cadena) a memoria.
00401668	xor eax,eax	Prepara EAX=0 para ser el contador.
0040166A	xor ecx,ecx	Prepara ECX=0 para recibir nuestro serial de a un carácter.
0040166C	movsx ecx,byte ptr ds:[eax+409D07]	ECX recibe nuestro valor HEXA del serial, Uno en uno.
00401673	xor cl,1E	Xorea BYTE serial con 0x1E.
00401676	mov byte ptr ds:[eax+40A107],cl	Guarda el valor xoreado, va creando una constante.
0040167C	inc eax	Incrementa EAX. Nuestro contador.
0040167D	cmp eax,dword ptr ds:[409907]	Compara contador con 0x18 (longitud cadena)
00401683	jb keygenme.40166C	Mientras EAX < 0x18 salta para repetir LOOP

Nuestra constante queda así:

BYTES: 524B574D585B5D5F5C3E5D6C7F7D754D33527F6A7770716D

ASCII: RKWMX[]_>]l.}uM3R.jwpqm

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x=] Lo
Address	Hex				ASCII	
0040A107	52 4B 57 4D	58 5B 5D 5F	5C 3E 5D 6C	7F 7D 75 4D	RKWMX[]_>]l.}uM	
0040A117	33 52 7F 6A	77 70 71 6D	00 00 00 00	00 00 00 00	3R.jwpqm.....	
0040A127	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
0040A137	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	

La rutina del ese **LOOP** fue lo sencillo de hacer. En el **SRC-KeyGen** he dejado explicado casi todo, prácticamente me evitaría casi de hacer todo este tutorial pero dejaré la "flojera" a un lado y también trataré de explicar lo mejor que pueda aquí en el tuto.

```

If Len(UsEr) > 49 Then
    UsEr = Mid(UsEr, 1, 49) + " CrackS-Latinos"
Else
    UsEr = UsEr + " CrackS-Latinos"
End If

'Crea nuestra constante consUsErHEXA en función de la longitud de UsEr cocatenado con "CrackS-Latinos"
For i = 1 To Len(UsEr)
    consUsErHEXA = consUsErHEXA + Hex(CLng("&H" + Hex(Asc(Mid(UsEr, i, 1)))) Xor CLng("&H1E"))
Next

```

Yo voy a hacer mi KeyGen en VB.NET y pensaba que la cosa me iba a ser sencilla pero no sabía la pesadilla oculta tras un **CALL**.

Lo que has leído hasta aquí lo hacía a medida que lo estaba resolviendo pero me encontré con una instrucción nueva que no me dejó avanzar y luego ahí mismo una rutina muy larga que me ponía al límite de mi paciencia, así que opté por programar las rutinas que generaban el serial y de esa forma poder entender las vueltas que daba para luego hacer un tuto más claro y ordenado.

Bueno, después de generar una constante a la que llamé "**consUsErHEXA**" pasa a **004015E7 CALL Keygenme.406118** para cargar cuatro constantes.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

```

004015F2  A2 07 A5 40 00  mov dword ptr ds:[40A507],eax
004015E7  E8 2C 4B 00 00  call keygenme.406118
004015EC  FF 33 07 A3 00 00 push dword ptr ds:[40A307]
004015F3  C8 07 A4 00 00  push keygenme.40A407
00406118  57              push edi
00406119  33 C0           xor eax,eax
00406118  A3 D0 B9 40 00 mov dword ptr ds:[40B9D0],eax
00406120  33 C0           xor eax,eax
00406122  A3 D4 B9 40 00 mov dword ptr ds:[40B9D4],eax
00406127  BF 80 B9 40 00 mov edi,keygenme.40B980
0040612C  B9 10 00 00 00 mov ecx,10
00406131  F3 AB           repe stosd
00406133  B8 C0 B9 40 00 mov eax,keygenme.40B9C0
00406138  C7 00 01 23 45 6 mov dword ptr ds:[eax],67452301
0040613E  C7 40 04 89 AB C mov dword ptr ds:[eax+4],EFCDA889
00406145  C7 40 08 FE DC B mov dword ptr ds:[eax+8],98BADCFE
0040614C  C7 40 0C 76 54 3 mov dword ptr ds:[eax+C],10325476
00406153  5F             pop edi
00406154  C3             ret

```

Ahí vemos en lo resaltado con **AMARILLO** que en **EAX** colocará la dirección **keygenme.40B9C0** y a partir de ahí colocará esas cuatro constantes.

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	Watch 1	[x]=L
Address	Hex				ASCII	
0040B980	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	
0040B9C0	01 23 45 67	89 AB CD EF	FE DC BA 98	76 54 32 10	..#Eg.«ïpÛ°.vT2.	
0040B9D0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	

Saliendo de ese **CALL**, iremos a otro donde hará comparaciones con la longitud de nuestro nombre concatenado y dependiendo del resultado seguirá el camino.

```

004015F2  68 07 A1 40 00  push keygenme.40A107
004015F7  E8 5C 4B 00 00  call keygenme.406158
00406158  55              push ebp
00406159  8B EC           mov ebp,esp
00406158  56              push esi
0040615C  57              push edi
0040615D  53              push ebx
0040615E  8B 5D 0C         mov ebx,dword ptr ss:[ebp+C]
00406161  8B 75 08         mov esi,dword ptr ss:[ebp+8]
00406164  01 1D D0 B9 40 0 add dword ptr ds:[40B9D0],ebx
0040616A  EB 40            jmp keygenme.4061AC
0040616C  A1 D4 B9 40 00  mov eax,dword ptr ds:[40B9D4]
00406171  B9 40 00 00 00  mov ecx,40
00406176  2B C8           sub ecx,eax
00406178  8D 88 80 B9 40 0 lea edi,dword ptr ds:[eax+40B980]
0040617E  3B C8           cmp ecx,ebx
00406180  77 1E           ja keygenme.4061A0
00406182  2B D9           sub ebx,ecx
00406184  F3 A4           repe movsb
00406186  E8 B5 F9 FF FF  call keygenme.405840
00406188  33 C0           xor eax,eax
0040618D  A3 D4 B9 40 00  mov dword ptr ds:[40B9D4],eax
00406192  BF 80 B9 40 00  mov edi,keygenme.40B980
00406197  B9 10 00 00 00  mov ecx,10
0040619C  F3 AB           repe stosd
0040619E  EB 0C           jmp keygenme.4061AC
004061A0  8B C8           mov ecx,ebx
004061A2  F3 A4           repe movsb
004061A4  01 1D D4 B9 40 0 add dword ptr ds:[40B9D4],ebx
004061AA  EB 04           jmp keygenme.406180
004061AC  0B DB           or ebx,ebx
004061AE  75 BC           jne keygenme.40616C
004061B0  5B             pop ebx
004061B1  5F             pop edi
004061B2  5E             pop esi
004061B3  C9             leave
004061B4  C2 08 00       ret 8

```

LONGITUD NAME

TOMA LA LONGITUD DE NUESTRO USER Y LO COMPARA CON 0x40. SI NUESTRO USER ES MENOR SALTARÁ.

40: '@'
Le restaEAX

COMPARA CON NUESTRA LONGITUD. SALTA SI ECX>EBX

Mueve nuestra constante
Pesadilla

MUEVE NUESTRA CONSTANTE

El siguiente **CALL keygenme.406158** lo encontramos en la dirección **004015F7**, entremos en el con **<F7>**. En la imagen de arriba podemos ver en **0040615E** que pasa nuestra longitud a **EBX** para más adelante compararlo con **0x40**, pero primero hace un salto incondicional a **004061AC** OR **EBX,EBX** y dependiendo del resultado saltará hacia atrás **0040616E** **JNE keygenme.4061AC** y esto solamente es para hacer un **LOOP**, que en este caso saltará una sola vez; eso se los digo porque después de repetir muchos traceos, sé que al comparar la longitud de nuestro nombre con **0x40**, toma el salto que nos lleva por otro camino y entonces no pasaremos por **004061AE** **JNE keygenme.4061AC**,

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

pero que más adelante volverá a pasar por aquí y ahí sí tomará el salto, pero no usará nuestra longitud, si no un valor constante de **0x400** al cual le irá restando el **0x40**, repitiendo el **LOOP**. Eso lo podemos ver en **00416182 SUB EBX,ECX**. Si observamos la instrucción anterior **00406180 JA keygenme.4061A0**, vemos que en este caso saltará porque en **0040617E CMP ECX(0x40),EBX(0x18)** observamos que **EBX es menor a ECX**, o podemos decir también que **ECX es mayor que EBX**, y por ese motivo nunca entramos a **00416182 SUB EBX,ECX**. Como dije hace ratito, cuando entremos de nuevo a esta rutina la cosa cambia porque lo que era nuestra longitud pasa a ser **0x400**, eso quiere decir que **EBX es mayor que ECX**, o **ECX menor que EBX**.

Vamos a hacerlo paso a paso, y no me importa que se me alargue el tuto porque de eso se trata este escrito, de explicar las rutas que emulará nuestro KeyGen. Lleguemos al salto incondicional.

00406161	8B 75 08	mov esi,dword ptr ss:[ebp+8]
00406164	01 1D D0 B9 40 0	add dword ptr ds:[408900],ebx
0040616A	EB 40	jmp keygenme.4061AC
0040616C	A1 D4 B9 40 00	mov eax,dword ptr ds:[408904]
00406171	B9 40 00 00 00	mov ecx,40
00406176	2B C8	sub ecx,eax
00406178	8D B8 80 B9 40 0	lea edi,dword ptr ds:[eax+408980]
0040617E	3B C8	cmp ecx,ebx
00406180	77 1E	ja keygenme.4061A0
00406182	2B D9	sub ebx,ecx
00406184	F3 A4	repe movsb
00406186	E8 B5 F9 FF FF	call keygenme.405840
00406188	33 C0	xor eax,eax
0040618D	A3 D4 B9 40 00	mov dword ptr ds:[408904],eax
00406192	BF 80 B9 40 00	mov edi,keygenme.408980
00406197	B9 10 00 00 00	mov ecx,10
0040619C	F3 AB	repe stosd
0040619E	EB 0C	jmp keygenme.4061AC
004061A0	8B C8	mov ecx,ebx
004061A2	F3 A4	repe movsb
004061A4	01 1D D4 B9 40 0	add dword ptr ds:[408904],ebx
004061AA	EB 04	jmp keygenme.406180
004061AC	0B DB	or ebx,ebx
004061AE	75 BC	jne keygenme.40616C

Estamos parados **0040616A JMP keygenme.4061AC** y saltará a **004061AC OR EBX,EBX**. Pues con <F7> saltemos.

0040616A	EB 40	jmp keygenme.4061AC	EAX	004089C0	keygenme.004089C0
0040616C	A1 D4 B9 40 00	mov eax,dword ptr ds:[408904]	EBX	00000018	
00406171	B9 40 00 00 00	mov ecx,40	ECX	00000000	
00406176	2B C8	sub ecx,eax	EDX	00409508	"UISFECAB Crack5-Latinos"
00406178	8D B8 80 B9 40 0	lea edi,dword ptr ds:[eax+408980]	EBP	0012FC24	
0040617E	3B C8	cmp ecx,ebx	ESP	0012FC18	
00406180	77 1E	ja keygenme.4061A0	ESI	0040A107	keygenme.0040A107
00406182	2B D9	sub ebx,ecx	EDI	0012FCFC	
00406184	F3 A4	repe movsb			
00406186	E8 B5 F9 FF FF	call keygenme.405840			
00406188	33 C0	xor eax,eax			
0040618D	A3 D4 B9 40 00	mov dword ptr ds:[408904],eax			
00406192	BF 80 B9 40 00	mov edi,keygenme.408980			
00406197	B9 10 00 00 00	mov ecx,10			
0040619C	F3 AB	repe stosd			
0040619E	EB 0C	jmp keygenme.4061AC			
004061A0	8B C8	mov ecx,ebx			
004061A2	F3 A4	repe movsb			
004061A4	01 1D D4 B9 40 0	add dword ptr ds:[408904],ebx			
004061AA	EB 04	jmp keygenme.406180			
004061AC	0B DB	or ebx,ebx			
004061AE	75 BC	jne keygenme.40616C			

Al llegar hacemos el **004061AC OR EBX,EBX** y como **EBX** diferente de 0, entonces saltará. Tomemos el salto nuevamente.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

0040616A	EB 40	jmp keygenme.4061AC
0040616C	A1 D4 B9 40 00	mov eax,dword ptr ds:[40B9D4]
00406171	B9 40 00 00 00	mov ecx,40
00406176	2B C8	sub ecx,eax
00406178	8D B8 80 B9 40 0	lea edi,dword ptr ds:[eax+40B980]
0040617E	3B C8	cmp ecx,ebx
00406180	77 1E	ja keygenme.4061A0
00406182	2B D9	sub ebx,ecx
00406184	F3 A4	repe movsb
00406186	E8 B5 F9 FF FF	call keygenme.405840
00406188	33 C0	xor eax,eax
0040618D	A3 D4 B9 40 00	mov dword ptr ds:[40B9D4],eax
00406192	BF 80 B9 40 00	mov edi,keygenme.40B980
00406197	B9 10 00 00 00	mov ecx,10
0040619C	F3 AB	repe stosd
0040619E	EB 0C	jmp keygenme.4061AC
004061A0	8B CB	mov ecx,ebx
004061A2	F3 A4	repe movsb
004061A4	01 1D D4 B9 40 0	add dword ptr ds:[40B9D4],ebx
004061AA	EB 04	jmp keygenme.406180
004061AC	0B DB	or ebx,ebx
004061AE	75 BC	jne keygenme.40616C

Resumiendo lo resaltado en **AZUL**, Mueve a **EAX** lo que hay en la dirección **[40B9D4]**, que en este caso es **0x0** y servirá como inicio para una dirección de memoria donde moveremos nuestra constante. Luego moverá **0x40** a **ECX** y si recuerdan es para compararlo con nuestra longitud. Hace una resta a **ECX** que no afecta en nada ya que **EAX=0**. Sigue con un **LEA** para **EDI** con **[EAX+40B980]** y con eso mueve a **EDI** la dirección de memoria **40B980** para que a partir de ahí mover nuestra constante. Ya terminando compara nuestra longitud, y como ya sabemos que es menor, el salto será tomado. Y finalmente el salto, así que lleguemos a este.

0040616A	EB 40	jmp keygenme.4061AC
0040616C	A1 D4 B9 40 00	mov eax,dword ptr ds:[40B9D4]
00406171	B9 40 00 00 00	mov ecx,40
00406176	2B C8	sub ecx,eax
00406178	8D B8 80 B9 40 0	lea edi,dword ptr ds:[eax+40B980]
0040617E	3B C8	cmp ecx,ebx
00406180	77 1E	ja keygenme.4061A0
00406182	2B D9	sub ebx,ecx
00406184	F3 A4	repe movsb
00406186	E8 B5 F9 FF FF	call keygenme.405840
00406188	33 C0	xor eax,eax
0040618D	A3 D4 B9 40 00	mov dword ptr ds:[40B9D4],eax
00406192	BF 80 B9 40 00	mov edi,keygenme.40B980
00406197	B9 10 00 00 00	mov ecx,10
0040619C	F3 AB	repe stosd
0040619E	EB 0C	jmp keygenme.4061AC
004061A0	8B CB	mov ecx,ebx
004061A2	F3 A4	repe movsb
004061A4	01 1D D4 B9 40 0	add dword ptr ds:[40B9D4],ebx
004061AA	EB 04	jmp keygenme.406180
004061AC	0B DB	or ebx,ebx
004061AE	75 BC	jne keygenme.40616C

Ahí vemos que saltaremos a la dirección **004061A0 MOV ECX,EBX**. Como estamos paso a paso, seguimos con **<F7>** para saltar.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

0040616A	EB 40	jmp keygenme.4061AC	EAX 00000000
0040616C	A1 D4 B9 40 00	mov eax,dword ptr ds:[40B9D4]	EBX 00000018
00406171	B9 40 00 00 00	mov ecx,40	ECX 00000040
00406176	2B C8	sub ecx,eax	EDX 00409508
00406178	8D B8 80 B9 40 0	lea edi,dword ptr ds:[eax+40B980]	EBP 0012FC24
0040617E	3B C8	cmp ecx,ebx	ESP 0012FC18
00406180	77 1E	ja keygenme.4061A0	ESI 0040A107 keygenme.0040A107
00406182	2B D9	sub ebx,ecx	EDI 0040B980 keygenme.0040B980
00406184	F3 A4	repe movsb	
00406186	E8 B5 F9 FF FF	call keygenme.405B40	
00406188	33 C0	xor eax,eax	
0040618D	A3 D4 B9 40 00	mov dword ptr ds:[40B9D4],eax	
00406192	BF 80 B9 40 00	mov edi,keygenme.40B980	
00406197	B9 10 00 00 00	mov ecx,10	
0040619C	F3 AB	repe stosd	
0040619E	EB 0C	jmp keygenme.4061AC	
004061A0	8B C8	mov ecx,ebx	
004061A2	F3 A4	repe movsb	
004061A4	01 1D D4 B9 40 0	add dword ptr ds:[40B9D4],ebx	
004061AA	EB 04	jmp keygenme.406180	
004061AC	0B DB	or ebx,ebx	
004061AE	75 BC	jne keygenme.40616C	

Address	Hex
0040B980	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B990	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9C0	01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
0040B9D0	18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Lo resaltado en **AZUL**, cortico pero es sustancioso. Primero moveremos nuestra longitud a **ECX**, la cual será un contador para mover los **0x18 BYTES** que componen nuestra longitud y los moverá con el **repe movsb**, y los pasará de la dirección de memoria que tiene **ESI=0040A107** que fue el primer lugar donde la guardó, y que ahora los guardará en **EDI=0040B980**. Pasemos todo con **<F7>** y observemos paso a paso cómo mueve nuestra constante que yo llame "**consUsErHEXA**" es movida. Lleguemos hasta el salto incondicional.

0040616A	EB 40	jmp keygenme.4061AC
0040616C	A1 D4 B9 40 00	mov eax,dword ptr ds:[40B9D4]
00406171	B9 40 00 00 00	mov ecx,40
00406176	2B C8	sub ecx,eax
00406178	8D B8 80 B9 40 0	lea edi,dword ptr ds:[eax+40B980]
0040617E	3B C8	cmp ecx,ebx
00406180	77 1E	ja keygenme.4061A0
00406182	2B D9	sub ebx,ecx
00406184	F3 A4	repe movsb
00406186	E8 B5 F9 FF FF	call keygenme.405B40
00406188	33 C0	xor eax,eax
0040618D	A3 D4 B9 40 00	mov dword ptr ds:[40B9D4],eax
00406192	BF 80 B9 40 00	mov edi,keygenme.40B980
00406197	B9 10 00 00 00	mov ecx,10
0040619C	F3 AB	repe stosd
0040619E	EB 0C	jmp keygenme.4061AC
004061A0	8B C8	mov ecx,ebx
004061A2	F3 A4	repe movsb
004061A4	01 1D D4 B9 40 0	add dword ptr ds:[40B9D4],ebx
004061AA	EB 04	jmp keygenme.406180
004061AC	0B DB	or ebx,ebx
004061AE	75 BC	jne keygenme.40616C
004061B0	5F	pop ebx
004061B1	5E	pop edi
004061B2	5E	pop esi
004061B3	C9	leave
004061B4	C2 08 00	ret 8

Address	Hex
0040B980	52 48 57 4D 58 5B 5D 5F 5C 3E 5D 6C 7F 7D 75 4D
0040B990	33 52 7F 6A 77 70 71 6D 00 00 00 00 00 00 00 00
0040B9A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9C0	01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
0040B9D0	18 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Lo primero es notar que no tomamos el **LOOP** y salimos directo rumbo al **RET 8**. También nos evitamos entrar a ese **CALL keygenme.405B40** que fue mi tortura en este reto. Les adelanto un poco más de cosillas que descubriremos un poquillo más adelante. Resulta

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

que si vemos en nuestro DUMP, he resaltado en **ROJO** 16 DOWRD que equivalen a **0x40=64** caracteres de longitud de nuestro **usuario**; podemos ver que tenemos **0x18=24** caracteres de longitud. Ya hasta aquí podemos concluir algo muy importante, y es que si longitud menor a **0x40=64** seguimos este camino y nos evitamos la **Pesadilla**, de lo contrario a **Pesadilla** nos vamos. Pasemos el **RET 8** para ver a dónde salimos.

004061B4	C2 08 00	ret 8
004015F2	68 07 A1 40 00	push keygenme.40A107
004015F7	E8 5C 4B 00 00	call keygenme.406158
004015FC	E8 B7 4B 00 00	call keygenme.4061B8
00401601	68 07 AD 40 00	push keygenme.40AD07
00401606	6A 10	push 10
00401608	50	push eax
00401609	E8 02 4C 00 00	call keygenme.406210
0040160E	68 07 B1 40 00	push keygenme.40B107
00401613	68 07 AD 40 00	push keygenme.40AD07

Salimos a **004015FC CALL keygenme.4061B8**. Es otro **CALL**, pues no hay de otra, entremos con **<F7>** para ver qué más sigue.

004061B8	push esi	
004061B9	push edi	
004061BA	mov ecx,dword ptr ds:[40B9D4]	Mueve nuestra longitud a ECX.
004061C0	mov byte ptr ds:[ecx+40B980],80	Mueve 0x80 al final de nuestra constante "consUsErHEXA"
004061C7	cmp ecx,38	Compara nuestra longitud con 0x38
004061CA	jnb keygenme.4061E4	Si es menor a 38 salta
004061CC	call keygenme.405B40	Pesadilla. Como 0x18<38 nos libramos.

Arriba vemos las primeras instrucciones, que carga nuestra longitud y que hace "consUsErHEXA" + **0x80**. Así que lleguemos al salto el cual tomaremos porque **0x18<0x38**.

004061BA	8B 0D D4 B9 40 0	mov ecx,dword ptr ds:[40B9D4]
004061C0	C6 81 80 B9 40 0	mov byte ptr ds:[ecx+40B980],80
004061C7	83 F9 38	cmp ecx,38
004061CA	72 18	jnb keygenme.4061E4
004061CC	E8 6F F9 FF FF	call keygenme.405B40
004061D1	33 C0	xor eax,eax
004061D3	A3 D4 B9 40 00	mov dword ptr ds:[40B9D4],eax
004061D8	BF 80 B9 40 00	mov edi,keygenme.40B980
004061DD	B9 10 00 00 00	mov ecx,10
004061E2	F3 AB	repe stosd
004061E4	A1 D0 B9 40 00	mov eax,dword ptr ds:[40B9D0]
004061E9	33 D2	xor edx,edx
004061EB	0F A4 C2 03	shld edx,eax,3
004061EF	C1 E0 03	shl eax,3
004061F2	A3 B8 B9 40 00	mov dword ptr ds:[40B9B8],eax
004061F7	89 15 BC B9 40 0	mov dword ptr ds:[40B9BC],edx
004061FD	E8 3E F9 FF FF	call keygenme.405B40

A PESADILLA NOS
FUIMOS. NO NOS
LIBRAMOS

Arriba vemos que tomaremos el salto y pasará a **EXA** nuestra longitud con **[40B9D0]** y con eso calculará un valor que lo agregará en la posición **0x39** en el **DUMP**. Lo relevante resaltado en **ROJO** es la dirección **004061EF SHL EAX,3** porque las anteriores no aportan en nada.

Antes de llegar a **Pesadilla**, voy a plantear las otras rutas a seguir con respecto a nuestra longitud de usuario, pero a groso modo. Si ustedes colocan una longitud mayor a **0x38** y menor a **0x40** pasarán por los dos **CALL Keygenme.405B40** de arriba, pasaremos por esas dos **Pesadillas**, pero al salir de **004061CC CALL Keygenme.405B40** crea una nueva constante "consUsErHEXA" para trabajar con en el otro **004061FD CALL Keygenme.405B40**

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

Y nos queda faltando cuando longitud igual a **0x40**, que hace rato vimos una parte del camino y que también toma dos **Pesadillas**, la primera **Pesadilla** **CALL Keygenme.405B40** trabaja con nuestra constante "**consUsErHEXA**" original y después de salir crea otra "**consUsErHEXA**" para hacer la segunda **CALL Keygenme.405B40**. Ya les di unas buenas pistas para que ustedes hallen esas nuevas constantes que siempre serán iguales. En el **SRC-KeyGen** está todo explicado. Bueno, entremos a la **Pesadilla** **CALL Keygenme.405B40**.

Address	Hex	Assembly	Comments
00405B40	60	pushad	
00405B41	BE C0 B9 40 00	mov esi, keygenme.40B9C0	Carga inicio de las constante
00405B46	BF 80 B9 40 00	mov edi, keygenme.40B980	Carga nuestra constante
00405B48	88 06	mov eax, dword ptr ds:[esi]	!Pasa primer constante EAX
00405B4D	88 5E 04	mov ebx, dword ptr ds:[esi+4]	Pasa segunda constante a EBX
00405B50	88 4E 08	mov ecx, dword ptr ds:[esi+8]	Pasa tercera constante a ECX
00405B53	88 EF	mov ebp, edi	Pasa nuestra constante a EBP
00405B55	88 56 0C	mov edx, dword ptr ds:[esi+C]	Pasa cuarta constante a EDX
00405B58	88 F9	mov edi, ecx	Mueve tercera constante a EDI
00405B5A	33 FA	xor edi, edx	XOREA 3cons/4cons
00405B5C	23 FB	and edi, ebx	
00405B5E	33 FA	xor edi, edx	
00405B60	03 45 00	add eax, dword ptr ss:[ebp]	Suma 1cons+nuestrascons
00405B63	8D 84 38 78 A4 5	lea eax, dword ptr ds:[eax+edi*289]	
00406108	01 06	add dword ptr ds:[esi], eax	EAX 4A445309
0040610A	01 5E 04	add dword ptr ds:[esi+4], ebx	EBX 3F3C6DA5
0040610D	01 4E 08	add dword ptr ds:[esi+8], ecx	ECX 3040F827
00406110	01 56 0C	add dword ptr ds:[esi+C], edx	EDX A2D44F66
00406113	61	popad	EBP 0040B980 keygenme.0040B980
00406114	C3	ret	ESI 0040B9C0 keygenme.0040B9C0

Address	Hex
0040B980	52 48 57 4D 58 5B 5D 5F 5C 3E 5D 6C 7F 7D 75 4D
0040B990	33 52 7F 6A 77 70 71 6D 80 00 00 00 00 00 00 00
0040B9A0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9B0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040B9C0	01 23 45 67 89 AB CD EF FE DC BA 98 76 54 32 10
0040B9D0	18 00 00 00 18 00 00 00 00 00 00 00 00 00 00 00
0040B9E0	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Ya el **<KeyGenMe v1.0>** teniendo nuestra constante "**consUsErHEXA**", que es lo resaltado en **ROJO** en el **DUMP** y las cuatro más que obtuvimos en el **CALL Keygenme.406118** y que las tenemos resaltado en **AMARILLO** en el **DUMP**.

Pues esta rutina que la llame **Pesadilla** por lo larga y que me tocó programar una por una todas esas instrucciones que solo se dedicaban a coger nuestra "**consUsErHEXA**" por partes de **DWORD** en **DWORD**, en orden del **1** al **16**, y junto a las cuatro constantes trabaja para calcular unos nuevos valores, y que esos nuevos valores son utilizados para calcular otros nuevos junto con nuestra "**consUsErHEXA**", pero ya los **DWORD** no son en orden, y de paso cada calculo tiene una suma o resta de un valor constante. Ya después de todas esas trapisondas llegamos casi al final **00406108** y ahí suma los registros **EAX, EBX, ECX, EDX** a las cuatro constantes en ese mismo orden y esas pasan a ser las nuevas cuatro constantes con las cuales la rutina **Pesadilla** realiza nuevos cálculos si es llamada. Entonces resumiendo, cada vez que entramos a esta **Pesadilla** tendremos cuatro nuevas constantes. Leguemos al **00406114** **RET**.

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5
Address	Hex			
0040B980	52 48 57 4D	58 5B 5D 5F	5C 3E 5D 6C	7F 7D 75 4D
0040B990	33 52 7F 6A	77 70 71 6D	80 00 00 00	00 00 00 00
0040B9A0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0040B9B0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00
0040B9C0	0A 76 89 B1	2E 19 0A 2F	25 D5 FB C8	DC A3 06 B3
0040B9D0	18 00 00 00	18 00 00 00	00 00 00 00	00 00 00 00
0040B9E0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

Ahí tenemos las nuevas cuatro constantes y que serán usadas si **Pesadilla** entra en acción. Bueno, pasemos el **00406114 RET**.

```

00406189 57          push edi
0040618A 8B 0D D4 B9 40 0 mov ecx,dword ptr ds:[40B9D4]
004061C0 8B 0D D4 B9 40 0 mov byte ptr ds:[ecx+40B980],80
004061C7 83 F9 38      cmp ecx,38
004061CA 72 18         jb keygenme.4061E4
004061CC E8 6F F9 FF FF call keygenme.405840
004061D1 33 C0         xor eax,eax
004061D3 A3 D4 B9 40 00 mov dword ptr ds:[40B9D4],eax
004061D8 BF 80 B9 40 00 mov edi,keygenme.40B980
004061DD B9 10 00 00 00 mov ecx,10
004061E2 F3 AB        repe stosd
004061E4 A1 D0 B9 40 00 mov eax,dword ptr ds:[40B9D0]
004061E9 33 D2         xor edx,edx
004061EB 0F A4 C2 03   shld edx,eax,3
004061EF C1 E0 03     shl eax,3
004061F2 A3 B8 B9 40 00 mov dword ptr ds:[40B988],eax
004061F7 89 15 BC B9 40 0 mov dword ptr ds:[40B98C],edx
004061FD E8 3E F9 FF FF call keygenme.405840
00406202 B8 C0 B9 40 00 mov eax,keygenme.40B9C0
00406207 5F          pop edi
00406208 5E          pop esi
00406209 C3          ret

```

Salimos a la rutina donde comparaba la longitud a **0x38**. Si recordamos, dependiendo del resultado cogemos las dos **Pesadillas** y si fuera así, la segunda **Pesadilla** trabajaría con esas nuevas cuatro constantes. Sigamos traceando y pasemos ese **00406209 RET**, a ver a dónde salimos.

004015F7	E8 5C 4B 00 00	call keygenme.406158
004015FC	E8 B7 4B 00 00	call keygenme.4061B8
00401601	68 07 AD 40 00	push keygenme.40AD07
00401606	6A 10	push 10
00401608	50	push eax
00401609	E8 02 4C 00 00	call keygenme.406210
0040160E	68 07 B1 40 00	push keygenme.40B107
00401613	68 07 AD 40 00	push keygenme.40AD07
00401618	E8 DD 07 00 00	call <keygenme.1strcatA>
0040161D	68 00 04 00 00	push 400
00401622	68 07 99 40 00	push keygenme.409907
00401627	E8 B6 07 00 00	call <keygenme.RtlZeroMemory>
0040162C	68 00 04 00 00	push 400
00401631	68 07 9D 40 00	push keygenme.409D07
00401636	E8 A7 07 00 00	call <keygenme.RtlZeroMemory>
0040163B	68 00 04 00 00	push 400
00401640	68 07 A1 40 00	push keygenme.40A107
00401645	E8 98 07 00 00	call <keygenme.RtlZeroMemory>
0040164A	68 07 AD 40 00	push keygenme.40AD07
0040164F	68 07 9D 40 00	push keygenme.409D07
00401654	E8 AD 07 00 00	call <keygenme.1strcpy>
00401659	68 07 AD 40 00	push keygenme.40AD07
0040165E	E8 A9 07 00 00	call <keygenme.1strlenA>
00401663	A3 07 99 40 00	mov dword ptr ds:[409907],eax
00401668	33 C0	xor eax,eax
0040166A	33 C9	xor ecx,ecx
0040166C	0F BE 88 07 9D 40	movsx ecx,byte ptr ds:[eax+409D07]
00401673	80 F1 3C	xor cl,3C
00401676	8B 88 07 A1 40 00	mov byte ptr ds:[eax+40A107],cl
0040167C	40	inc eax
0040167D	3B 05 07 99 40 00	cmp eax,dword ptr ds:[409907]
00401683	72 E7	jb keygenme.40166C
00401685	68 00 04 00 00	push 400

PROCEDIMIENTO EN EL CUAL CREA UNA CONSTE DE VALORES ASCII Y QUE SE CONCATENA CON API_GetUserNameA

Salimos a **00401601 PUSH keygenme.40AD07** y si miramos lo que sigue, vemos que es un procedimiento muy parecido, prácticamente igual. Si recordamos nosotros concatenábamos nuestro nombre con "**CrackS-Latinos**"; aquí lo haremos con el nombre del administrador de la cuenta y que si recordamos lo halló hace rato con la **API_GetUserNameA** y con una nueva constante, que retornada de este **00401609 CALL keygenme.406210**, así que entremos a ese **CALL**.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

00406210	55		push ebp
00406211	8B EC		mov ebp,esp
00406213	57		push edi
00406214	56		push esi
00406215	53		push ebx
00406216	8B 5D 0C		mov ebx,dword ptr ss:[ebp+C]
00406219	8B 7D 10		mov edi,dword ptr ss:[ebp+10]
0040621C	85 DB		test ebx,ebx
0040621E	8B 75 08		mov esi,dword ptr ss:[ebp+8]
00406221	74 36		je keygenme.406259
00406223	0F B6 06		movzx eax,byte ptr ds:[esi]
00406226	8B C8		mov ecx,eax
00406228	83 C7 02		add edi,2
0040622B	C1 E9 04		shr ecx,4
0040622E	83 E0 0F		and eax,F
00406231	83 E1 0F		and ecx,F
00406234	83 F8 0A		cmp eax,A
00406237	1B D2		sbb edx,edx
00406239	83 D0 00		adc eax,0
0040623C	8D 44 D0 37		lea eax,dword ptr ds:[eax+edx*8+37]
00406240	83 F9 0A		cmp ecx,A
00406243	1B D2		sbb edx,edx
00406245	83 D1 00		adc ecx,0
00406248	C1 E0 08		shl eax,8
0040624B	8D 4C D1 37		lea ecx,dword ptr ds:[ecx+edx*8+37]
0040624F	0B C1		or eax,ecx
00406251	46		inc esi
00406252	66 89 47 FE		mov word ptr ds:[edi-2],ax
00406256	4B		dec ebx
00406257	75 CA		jne keygenme.406223
00406259	8B C7		mov eax,edi

Este procedimiento también lo programé y coge **BYTE** a **BYTE** las cuatro constantes resultantes del procedimiento **Pesadilla** para tener como resultado la nueva constante a concatenar. Ustedes pueden ver cómo trabaja, yo saltaré de una vez al final del procedimiento donde crea una nueva "consUseErHEXA".

00401676	8B 88 07 A1 40 0	mov byte ptr ds:[eax+40A107],cl	
0040167C	40	inc eax	
0040167D	3B 05 07 99 40 0	cmp eax,dword ptr ds:[409907]	
00401683	72 E7	jb keygenme.40166C	
00401685	68 00 04 00 00	push 400	
0040168A	68 07 AD 40 00	push keygenme.40AD07	
0040168F	E8 4E 07 00 00	call <keygenme.RtlZeroMemory>	
00401694	E8 7F 4A 00 00	call keygenme.406118	
00401699	68 00 04 00 00	push 400	
0040169E	68 07 A1 40 00	push keygenme.40A107	
004016A3	E8 80 4A 00 00	call keygenme.406158	
004016A8	E8 0B 4B 00 00	call keygenme.406188	
004016AD	68 07 AD 40 00	push keygenme.40AD07	
004016B2	6A 10	push 10	
004016B4	50	push eax	
004016B5	E8 56 4B 00 00	call keygenme.406210	
004016BA	68 00 04 00 00	push 400	

→ CARGAMOS DE NUEVO LAS 4 CONSTANTES INICIALES

→ COMPARA 0x40 CON 0x400. AQUÍ SI EJECUTAMOS EL LOOP

→ ESTE ES DONDE SE COMPARA CON 0x38 Y TENEMOS MÁS DE PESADILLA

→ CREA LA NUEVA CONSTANTE ASCII PERO YA ES PRÁCTICAMENTE NUESTRO SERIAL

Diagram showing arrows from the assembly code to the explanatory text boxes on the right.

Estamos parados en **00401685** **PUSH 400**. Podemos ver que hacemos lo mismo que en la primera parte, solo debemos hacer que nuestro KeyGen haga los procedimientos las veces que nos piden y que ya por fortuna tenemos programados.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

0040615E	8B 5D 0C	mov ebx,dword ptr ss:[ebp+C]	EAX	00000000	
00406161	8B 75 08	mov esi,dword ptr ss:[ebp+8]	EBX	00000400	L'É'
00406164	01 1D D0 B9 40 0	add dword ptr ds:[40B9D0],ebx	ECX	00000040	'@'
0040616A	EB 40	jmp keygenme.4061AC	EDX	00000000	
0040616C	A1 D4 B9 40 00	mov eax,dword ptr ds:[40B9D4]	EBP	0012FC24	
00406171	B9 40 00 00 00	mov ecx,40	ESP	0012FC18	
00406176	2B C8	sub ecx,eax	ESI	0040A107	keygenme.0040A107
00406178	8D B8 80 B9 40 0	lea edi,dword ptr ds:[eax+40B980]	EDI	0040B980	keygenme.0040B980
0040617E	3B CB	cmp ecx,ebx			
00406180	77 1E	ja keygenme.4061A0			
00406182	2B D9	sub ebx,ecx			
00406184	F3 A4	repe movsb			
00406186	E8 B5 F9 FF FF	call keygenme.405B40			
00406188	33 C0	xor eax,eax			
0040618D	A3 D4 B9 40 00	mov dword ptr ds:[40B9D4],eax			
00406192	BF 80 B9 40 00	mov edi,keygenme.40B980			
00406197	B9 10 00 00 00	mov ecx,10			
0040619C	F3 AB	repe stosd			
0040619E	EB 0C	jmp keygenme.4061AC			
004061A0	8B CB	mov ecx,ebx			
004061A2	F3 A4	repe movsb			
004061A4	01 1D D4 B9 40 0	add dword ptr ds:[40B9D4],ebx			
004061AA	EB 04	jmp keygenme.406180			
004061AC	0B DB	or ebx,ebx			
004061AE	75 BC	jne keygenme.40616C			
004061B0	5B	non ehx			

Ahí vemos que ya no se compara con **0x18** que era nuestra longitud, si no que ahora es **0x400** y es solo para repetir **10** veces a **Pesadilla**. Después de terminar el **LOOP**.

00406188	56	push esi			
00406189	57	push edi			
0040618A	8B 0D D4 B9 40 0	mov ecx,dword ptr ds:[40B9D4]			
004061C0	C6 81 80 B9 40 0	mov byte ptr ds:[ecx+40B980],80			
004061C7	83 F9 38	cmp ecx,38			
004061CA	72 18	jb keygenme.4061E4			
004061CC	E8 6F F9 FF FF	call keygenme.405B40			
004061D1	33 C0	xor eax,eax			
004061D3	A3 D4 B9 40 00	mov dword ptr ds:[40B9D4],eax			
004061D8	BF 80 B9 40 00	mov edi,keygenme.40B980			
004061DD	B9 10 00 00 00	mov ecx,10			
004061E2	F3 AB	repe stosd			
004061E4	A1 D0 B9 40 00	mov eax,dword ptr ds:[40B9D0]			
004061E9	33 D2	xor edx,edx			
004061EB	0F A4 C2 03	shld edx,eax,3			
004061EF	C1 E0 03	shl eax,3			
004061F2	A3 B8 B9 40 00	mov dword ptr ds:[40B988],eax			
004061F7	89 15 BC B9 40 0	mov dword ptr ds:[40B98C],edx			
004061FD	E8 3E F9 FF FF	call keygenme.405B40			
00406202	B8 C0 B9 40 00	mov eax,keygenme.40B9C0			
00406207	5F	pop edi			
00406208	5E	pop esi			
00406209	C3	ret			

Aquí lo mis mismo, y repetimos **Pesadilla** para que al salir de esa rutina, vallamos al **CALL** que nos da el serial.

00406210	55	push ebp			
00406211	8B EC	mov ebp,esp			
00406213	57	push edi			
00406214	56	push esi			
00406215	53	push ebx			
00406216	8B 5D 0C	mov ebx,dword ptr ss:[ebp+C]			
00406219	8B 7D 10	mov edi,dword ptr ss:[ebp+10]			
0040621C	85 DB	test ebx,ebx			
0040621E	8B 75 08	mov esi,dword ptr ss:[ebp+8]			
00406221	74 36	je keygenme.406259			
00406223	0F B6 06	movzx eax,byte ptr ds:[esi]			
00406226	8B C8	mov ecx,eax			
00406228	83 C7 02	add edi,2			
0040622B	C1 E9 04	shr ecx,4			
0040622E	83 E0 0F	and eax,F			
00406231	83 E1 0F	and ecx,F			
00406234	83 F8 0A	cmp eax,A			
00406237	1B D2	sbb edx,edx			
00406239	83 D0 00	adc eax,0			
0040623C	8D 44 D0 37	lea eax,dword ptr ds:[eax+edx*8+37]			
00406240	83 F9 0A	cmp ecx,A			
00406243	1B D2	sbb edx,edx			
00406245	83 D1 00	adc ecx,0			
00406248	C1 E0 08	shl eax,8			
0040624B	8D 4C D1 37	lea ecx,dword ptr ds:[ecx+edx*8+37]			
0040624F	0B C1	or eax,ecx			
00406251	46	inc esi			
00406252	66 89 47 FE	mov word ptr ds:[edi-2],ax			
00406256	4B	dec ebx			
00406257	75 CA	jne keygenme.406223			
00406259	8B C7	mov eax,edi			
0040625B	C6 07 00	mov byte ptr ds:[edi+1],0			

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

Como pueden ver todos ustedes, todo ha sido lo mismo. Arriba podemos observar la rutina que nos da el serial. Solo nos faltan unos cambios mínimos o adornos.

004016BA	68 00 04 00 00	push 400
004016BF	68 07 A5 40 00	push keygenme.40A507
004016C4	E8 19 07 00 00	call <keygenme.RtlZeroMemory>
004016C9	68 07 AD 40 00	push keygenme.40AD07
004016CE	E8 39 07 00 00	call <keygenme.lstrlenA>
004016D3	A3 07 A5 40 00	mov dword ptr ds:[40A507],eax
004016D8	33 C0	xor eax,eax
004016DA	80 B8 07 AD 40 0	cmp byte ptr ds:[eax+40AD07],46
004016E1	75 07	jne keygenme.4016EA
004016E3	C6 80 07 AD 40 0	mov byte ptr ds:[eax+40AD07],54
004016EA	40	inc eax
004016EB	3B 05 07 A5 40 0	cmp eax,dword ptr ds:[40A507]
004016F1	72 E7	jb keygenme.4016DA
004016F3	68 00 04 00 00	push 400
004016F8	68 07 A5 40 00	push keygenme.40A507
004016FD	E8 E0 06 00 00	call <keygenme.RtlZeroMemory>
00401702	68 07 AD 40 00	push keygenme.40AD07
00401707	68 07 A5 40 00	push keygenme.40A507
0040170C	E8 F5 06 00 00	call <keygenme.lstrcpy>
00401711	C6 05 0E A5 40 0	mov byte ptr ds:[40A50E],43
00401718	C6 05 16 A5 40 0	mov byte ptr ds:[40A516],4C
0040171F	C6 05 1E A5 40 0	mov byte ptr ds:[40A51E],53
00401726	C9	leave
00401727	C2 04 00	ret 4

Listo, ahora a nuestro serial que sale de 004016B5 **CALL** keygenme.406210 en el procedimiento resaltado en **VERDE** reemplaza todas las letras "F"(0x46) por "T"(0x54) y lo resaltado en **ROJO** reemplaza los caracteres de la posiciones así:

Posición 8 = "C"(0x43)

Posición 16 = "L"(0x4C)

Posición 24 = "S"(0x53)

```
'Reemplazamos todos los caracteres "F" por "T".
UsEr = Replace(UsEr, "F", "T")

'Cambiamos los caracteres de las siguientes posiciones.
'Posición 8 = Caracter "C"
'Posición 16 = Caracter "L"
'Posición 24 = Caracter "S"
UsEr = Mid(UsEr, 1, 7) + "C" + Mid(UsEr, 9, 7) + "L" + Mid(UsEr, 17, 7) + "S" + Mid(UsEr, 25, 8)
```

Ahí amigos les muestro otro adelanto de cómo lo hice en VB.NET.

Ya todo se ha dicho y explicado lo mejor posible. Ahora les digo que solo queda programarlo, y eso ya depende de cada uno de nosotros y de cómo lo queremos hacer, claro está, si ustedes encuentran otras formas de resolver este reto hecho por ZELT@, no seas **FLOJO** y escribe un tutorial, y compártelo con la comunidad.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

```
Sub Pesadilla()  
  
    calSumROLEAX = 0  
    calSumROLEBX = 0  
    calSumROLECX = 0  
    calSumROLEDX = 0  
  
    '00405B58 | MOV EDI,ECX -> Mueve consHEXA3 a calXORconsEDI  
    calXORconsEDI = ConverterNum(consHEXA3,16)  
  
    '00405B5A | XOR EDI,EDX -> Xoreamos consHEXA3 con consHEXA4  
    calXORconsEDI = calXORconsEDI Xor "&H" + consHEXA4  
  
    '00405B5C | AND EDI,EBX -> AND calXORconsEDI con consHEXA2  
    calXORconsEDI = calXORconsEDI And "&H" + consHEXA2  
  
    '00405B5E | XOR EDI,EDX -> Xoreamos calXORconsEDI con consHEXA4  
    calXORconsEDI = calXORconsEDI Xor "&H" + consHEXA4
```

El **CALL Keygenme.405B40** lo programé como un procedimiento llamado **Sub Pesadilla()** es que me sacó canas verdes. Para llamarlo cuando sea necesario y lo mismo hice con el **CALL keygenme.406210**.

```
Sub NoPesadilla()  
    Dim FLAG_C As Int16  
  
    consAfterPesadilla = ""  
    calSumROLEDX = 0  
  
    'Trabajamos con esos nuevos valores de las constantes. Toma de a BYTE y realiza un nuevo cálculo.  
    For j = 1 To 4  
        For i = 1 To 8 Step 2  
            '00406223 | MOVZX EAX,BYTE PTR DS:[esi]  
            calSumROLEAX = ConverterNum(Mid(CargarHexDelDUMP(RedondearHEXA(Hex(consHEXA1EAX))) & Redon  
            '00406226 | MOV ECX,EAX  
            calSumROLECX = calSumROLEAX
```

Y llamé a ese procedimiento **Sub NoPesadilla()**, es que apenas terminé el **Sub Pesadilla()** ya todo se volvió breve.

[Tuto008 - KeyGenMe v1.0 por ZLT (KeyGen)(x64DBG)(ASM)]

```
Module Module1
    64 referencias
    Public Function Hexbi(ByVal hex As String) As String
        Dim bin As String = ""

        bin = Convert.ToString(Convert.ToInt32(hex, 16), 2)

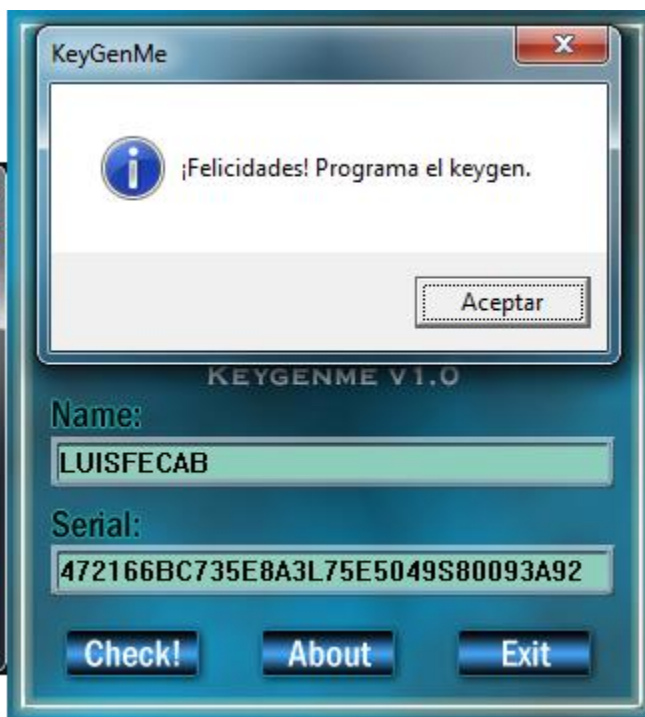
        Return bin
    End Function

    64 referencias
    Public Function MyFuncionROL(ByRef numBin As String, ByRef MovsPos As Integer) As String
        Dim myNumROLeado As String
        Dim i As Int16

        If Len(numBin) < 32 Then
            Dim relleno As String = ""
            For i = 1 To 32 - Len(numBin)
                relleno = relleno + "0"
            Next
            numBin = relleno + numBin
        End If

        myNumROLeado = Mid(numBin, MovsPos + 1, Len(numBin) - MovsPos + 1) + Mid(numBin, 1, MovsPos)
```

En un Módulo coloqué unas funciones necesarias que me sirven para trabajar correctamente. Bueno, todo todito lo encuentran el **SRC-KeyGen** y que lo dejé con comentarios explicando todo lo mejor posible. Veamos una captura de nuestro Keygen venciendo el reto <KeyGenMe v1.0>.



PARA TERMINAR

Tremendo reto, que como yo lo abordé y fue mi solución, terminé programando cada instrucción que hacia el <KeyGenMe v1.0> y eso me tomó mucho tiempo, y hubo muchas ocasiones que se me iba mal el dedo y me quedaba mal y no obtenía lo mismo que arrojaba el <KeyGenMe v1.0>, y debía buscar mi error y eso era doble trabajo y esfuerzo.

Les comenté por encima de la instrucción **ROL** que me hizo mella, ya que VB.NET no tiene una función para hacerla, busqué por Internet y explicaban de buena manera qué hacía, entonces hice mi propia función y el resultado arrojado no era el correcto, así que seguía buscando por internet el motivo y no lo encontraba; así que no tuve más opción que pedir ayuda en el grupo de **CracksLatinoS** y por fortuna me la resolvieron y para mi alegría fue el propio maestro Ricardo. No voy a explicar cómo funciona la función **ROL** en el tuto, ya estoy cansado para extenderme mucho más pero si buscan en el grupo "**función ROL**", seguro encontrarán mi consulta con la solución y explicación de esta.

Creo que este tutorial, el número 8 que hago, me enseñó muchísimo, empezando por esa instrucción **ROL** y luego porque hice unas funciones que puedo utilizar para futuros KeyGen. Ya solid me recomendó aprender ASM y así poder reutilizar el código para crear los KeyGen, y si muy cierto, pero algo que noté, es que mejor para mi trabajar con lo que conozco, que es VB.NET; de esa forma voy viendo y conociendo las instrucciones, vistas y analizadas desde el enfoque de VB.NET.

Espero, que les agrade el tutorial y que si tienen una observación o un consejo, no duden el hacerlo saber, ya que para mí es muy importante aprender de este arte y corregir mis errores por falta de conocimiento. Yo probé mi KeyGen con varios nombres y longitudes para ver si fallaba y todo bien; si ejecutan mi KeyGen y encuentran una falla, recuerden que ustedes tienen acceso el **SRC-KeyGen** y corregir el o los errores.

Como dato curioso, el <KeyGenMe v1.0> acepta un nombre vacío, como comenté en al análisis el no verifica si has ingresado tu nombre.

Saludos a todos,

@LUISFECAB