

[Complemento Teoría 1663. Utilizando HASH MD5 (KeyGen)(x64DBG)]



Software	KeyGenMe v1.0 por ZLT
Protección	Serial.
Herramientas	Windows 7 Home Premium SP1 x32 Bits (S.O donde trabajamos.) X64DBG (Feb 14 2018) Microsoft Visual Studio 2017 DESCARGAR HERRAMIENTAS DESCARGAR TUTO+ARCHIVOS
SOLUCIÓN	KEYGEN
AUTOR	LUISFECAB
RELEASE	Septiembre 18 2018 [TUTORIAL 009]

INTRODUCCIÓN

Toda esta historia empieza con en el tutorial anterior, [1663](#); en donde resolví el reto de **Zelt@**, el <**KeyGenMe v1.0 por ZLT**>, que dio como resultado el mejor tutorial que pude hacer para mi propio beneficio. Yo tenía entendido de la existencia de los algoritmos de encriptación, pero jamás en mi vida me había topado con uno y sin darme cuenta este reto utilizaba el **HASH MD5**, y yo lo resolví programando cada uno de los pasos que hacía el <**KeyGenMe v1.0 por ZLT**>, y eso me resultó extremadamente largo y laborioso, y aun así lo pude resolver y ni me di por enterado la presencia del señor doctor **HASH MD5**.

Resultó entonces, que terminé programando todas las funciones del **HASH MD5** y todo por falta de conocimientos, si hubiera sabido de su existencia, o mejor aún, su existencia y saber cómo utilizarlo en **VB.NET**, porque **Thunder** cuando leyó mi solución anterior me explicó que ahí se utilizaba el **HASH MD5** y me explicó cómo reconocerlo, entonces me puse a reemplazar todo ese código que hacía parte del **HASH MD5**, por la función del **HASH MD5** de **VB.NET** y nada, no pude replicar eso en **VB.NET** y el motivo no fue otro, más que la falta de conocimientos y fundamentos en todo, en programación y saber cómo adecuar las strings para convertirla en nuestro **HASH MD5**. Resumiendo, por **FALTA DE EXPERIENCIA**.

No me quedaba de otra que pedir de nuevo ayuda a la lista **CracksLatinoS**. Eso hice, consulté por una ayudita, y afortunadamente **Thunder** vino de nuevo en mi ayuda; así que muchas gracias **Thunder** por toda tu colaboración. Gracias a eso, pude comparar y ver en dónde fallada y por qué no me cuajaba el **HASH MD5**, y la respuesta fue la misma, **FALTA DE NIVEL**.

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 ayude a quienes, como yo, abordan el **HASH MD5**.

COMPLEMENTO

Aquí no vamos hacer nuestro habitual **ANALIS** y **AL ATAQUE**, todo eso ya lo hicimos en nuestro anterior tutorial, [1663](#); que les recomiendo lo lean primero, también les dejo estos tutoriales de las teorías numeras [547](#), [1299](#), [1568](#); que nos dan una muy buena teoría y práctica con nuestro **HASH MD5**, y claro está, buscar por Internet.

Bueno, vayamos directamente al **x64DBG** y cargamos nuestro <**KeyGenMe v1.0**> en donde pararemos directamente en la zona de interés, en la cual haremos puro análisis estático porque el análisis a profundidad ya lo hicimos.

004015C0	A3 07 99 40 00	mov dword ptr ds:[409907],eax	eax:BaseThreadInitThunk
004015C5	33 C0	xor eax,eax	eax:BaseThreadInitThunk
004015C7	33 C9	xor ecx,ecx	
004015C9	0F BE 88 07 9D 4	movsx ecx,byte ptr ds:[eax+409D07]	
004015D0	80 F1 1E	xor cl,1E	
004015D3	88 88 07 A1 40 0	mov byte ptr ds:[eax+40A107],cl	
004015D9	40	inc eax	eax:BaseThreadInitThunk
004015DA	3B 05 07 99 40 0	cmp eax,dword ptr ds:[409907]	eax:BaseThreadInitThunk
004015E0	72 E7	jnb keygenme v1.0.4015C9	
004015E2	A3 07 A5 40 00	mov dword ptr ds:[40A507],eax	eax:BaseThreadInitThunk
004015E7	E8 2C 4B 00 00	call keygenme v1.0.406118	MD5_INI
004015EC	FF 35 07 A5 40 0	push dword ptr ds:[40A507]	
004015F2	68 07 A1 40 00	push keygenme v1.0.40A107	
004015F7	E8 5C 4B 00 00	call keygenme v1.0.406158	MD5_Longitud y calcular
004015FC	E8 B7 4B 00 00	call keygenme v1.0.406188	MD5_Longitud y calcular
00401601	68 07 AD 40 00	push keygenme v1.0.40AD07	
00401606	6A 10	push 10	
00401608	50	push eax	eax:BaseThreadInitThunk
00401609	E8 02 4C 00 00	call keygenme v1.0.406210	MD5_HASH
0040160E	68 07 B1 40 00	push keygenme v1.0.40B107	
00401613	68 07 AD 40 00	push keygenme v1.0.40AD07	

Entonces, recapitulando un poco. La parte **VERDE** es un **LOOP** donde creamos una string a la cual le aplicamos el **HASH MD5** y eso lo hacemos con esos **4 CALL** que tenemos en la parte resaltada en **ROJO**, pues esos son el **HASH MD5**. Como pueden ver por las **FLECHAS AZULES**, les he puesto a cada **CALL** un comentario para identificar cada procedimiento. Si observamos, tengo **2 CALL** que los llamé "**MD5_Longitud y calcular**" y es porque de acuerdo a la longitud de la string entran a calcular, más adelante trataré de explicar eso como yo lo entiendo. Así que entremos al primer **CALL Keygenme v1.0.406118** que sería "**MD5_INI**".

00406118	57	push edi
00406119	33 C0	xor eax,eax
0040611B	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 v1.0.40B980
0040612C	B9 10 00 00 00	mov ecx,10
00406131	F3 AB	repe stosd
00406133	B8 C0 B9 40 00	mov eax,keygenme v1.0.40B9C0
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],EFCDA8B9
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

Según la teoría, el **HASH MD5** se inicializa cargando esas cuatro constantes resaltadas en **AMARILLO**, y precisamente son esas cuatro constantes las que dan a entender que se trata de un **HASH MD5**, porque siempre serán esas mismas. Entonces, si tú estás crackeando algo y te encuentras con estas cuatro constantes, es una prueba inequívoca que estas en frente de un **HASH MD5**. Bueno, pasemos a nuestro segundo **CALL keygenme v1.0.406158**, que sería mi "**MD5_Longitud y calcular**".

[Complemento Teoría 1663. Utilizando HASH MD5 (KeyGen)(x64DBG)]

00406158	55	push ebp
00406159	8B EC	mov ebp,esp
0040615B	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 v1.0.4061AC
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 v1.0.4061A0
00406182	2B D9	sub ebx,ecx
00406184	F3 A4	repe movsb
00406186	E8 B5 F9 FF FF	call keygenme v1.0.405B40
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 v1.0.40B980
00406197	B9 10 00 00 00	mov ecx,10
0040619C	F3 AB	repe stosd
0040619E	EB 0C	jmp keygenme v1.0.4061AC
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 v1.0.4061B0
004061AC	0B DB	or ebx,ebx
004061AE	75 BC	jne keygenme v1.0.40616C
00406180	5B	pop ebx
004061B1	5F	pop edi
004061B2	5E	pop esi
004061B3	C9	leave
004061B4	C2 08 00	ret 8

CALCULAR

Esto procedimiento lo explicamos muy bien, y lo resaltado en VERDE es el procedimiento **Pesadilla**. Resulta que antes de entrar a ese **CALL keygenme v1.0.405B40** debe el cuadrar su longitud y por eso hace comparaciones con nuestra longitud. Algo de teoría.

Paso 1. Adición de bits

El mensaje será extendido hasta que su longitud en bits sea congruente con 448, módulo 512. Esto es, si se le resta 448 a la longitud del mensaje tras este paso, se obtiene un múltiplo de 512. Esta extensión se realiza siempre, incluso si la longitud del mensaje es ya congruente con 448, módulo 512.

La extensión se realiza como sigue: un solo bit "1" se añade al mensaje, y después se añaden bits "0" hasta que la longitud en bits del mensaje extendido se haga congruente con 448, módulo 512. En todos los mensajes se añade al menos un bit y como máximo 512.

Paso 2. Longitud del mensaje

Un entero de 64 bits que represente la longitud 'b' del mensaje (longitud antes de añadir los bits) se concatena al resultado del paso anterior. En el supuesto no deseado de que 'b' sea mayor que 2^{64} , entonces sólo los 64 bits de menor peso de 'b' se usarán.

En este punto el mensaje resultante (después de rellenar con los bits y con 'b') se tiene una longitud que es un múltiplo exacto de 512 bits. A su vez, la longitud del mensaje es múltiplo de 16 palabras (32 bits por palabra). Con $M[0 \dots N-1]$ denotaremos las palabras del mensaje resultante, donde N es múltiplo de 16.

Bueno, como yo entiendo lo anterior es que, ese es el motivo de que porqué completaba los **16 DWORDS**, recordemos que en el tuto anterior yo completaba mi constante hasta los **64 BYTES = 16 DWORDS**. También analizábamos la condición para que ocurriera el **LOOP** del **CALL keygenme v1.0.406158**, y eso era cuando la longitud de la string fuera mayor a **0x40 = 64**. Entonces para strings mayores de **0x40 = 64**, se realiza ese **LOOP**

[Complemento Teoría 1663. Utilizando HASH MD5 (KeyGen)(x64DBG)]

para ir calculando el **HASH MD5** de esa longitud que no abarcó en los primeros **0x40 = 64** y el tercer **CALL keygenme v1.0.4061B8** termina de cumplir esa tarea y lo que hace es completar nuestra longitud, según la teoría de arriba.

```
004061B8 56 push esi
004061B9 57 push edi
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 v 72 18 jb keygenme v1.0.4061E4
004061CC E8 6F F9 FF FF call keygenme v1.0.405B40
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 v1.0.40B980
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 v1.0.405B40
00406202 B8 C0 B9 40 00 mov eax,keygenme v1.0.40B9C0
00406207 5F pop edi
00406208 5E pop esi
00406209 C3 ret
```

Ese viene siendo nuestro tercer **CALL keygenme v1.0.4061B8**, o sea, nuestro segundo “MD5 Longitud y calcular” y en cualquiera de ellos tenemos el **CALL keygenme v1.0.405B40**.

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	8B 06	mov eax,dword ptr ds:[esi]	Pasa primer constante EAX
00405B4D	8B 5E 04	mov ebx,dword ptr ds:[esi+4]	Pasa segunda constante a EBX
00405B50	8B 4E 08	mov ecx,dword ptr ds:[esi+8]	Pasa tercera constante a ECX
00405B53	8B EF	mov ebp,edi	Pasa nuestra constante a EBP
00405B55	8B 56 0C	mov edx,dword ptr ds:[esi+C]	Pasa cuarta constante a EDX
00405B58	8B 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+nuestracons
00405B63	8D 84 38 78 A4 6	lea eax,dword ptr ds:[eax+edi]	
00406108	01 06	add dword ptr ds:[esi],eax	
0040610A	01 5E 04	add dword ptr ds:[esi+4],ebx	
0040610D	01 4E 08	add dword ptr ds:[esi+8],ecx	
00406110	01 56 0C	add dword ptr ds:[esi+C],edx	
00406113	61	popad	
00406114	C3	ret	

Register	Value	Address
EAX	4A445309	
EBX	3F3C6DA5	
ECX	3040F827	
EDX	A2D44F66	
EBP	0040B980	keygenme.0040B980
ESP	0012FC04	
ESI	0040B9C0	keygenme.0040B9C0
EDI	172FB391	

Address	Hex
0040B980	52 48 57 4D 58 58 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 C0 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

La imagen de arriba es del tuto anterior pero muestra muy bien el **CALL keygenme v1.0.405B40** donde se va originando el **HASH MD5**. Por fortuna esto ya lo explicamos también. Después de calcular todo, pasamos a nuestro cuarto **CALL keygenme v1.0.406210**, el que yo llamé “MD5_HASH”.

[Complemento Teoría 1663. Utilizando HASH MD5 (KeyGen)(x64DBG)]

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
00406228	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
00406248	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

Ya con este obtenemos nuestro **HASH MD5**.

Como podemos ver, todo eso me lo pude haber ahorrado si hubiera sabido lo que ahora aprendí, por eso es que digo que siento que el tutorial anterior y este, su complemento, es lo mejor que he escrito.

Después de tener esto bien claro y con la ayuda **Thunder**, ya pude programar mi KeyGen en **VB.NET** utilizando la función del **HASH MD5** que trae.

```
Dim UsEr As String = txtNombre.Text
Dim consUsErHEXAMD5 As String = ""

'El KeyGenMe v1.0 trabaja con 64 BYTES (1 BYTE = 1 CHARACTER) para generar el serial, que serían &H40=64 caracteres de
'longitud máxima, los cuales son utilizados por el KeyGenMe v1.0 para generar el serial. Usted puede ingresar una mayor
'longitud pero no serán tomados en cuenta. Tenemos la constante " CrackS-Latinos" que tiene &HF=15 de longitud, entonces
'nuestra máxima longitud real sería un serial de longitud &H31=49 caracteres porque debemos descontar la longitud de la
'constante " CrackS-Latinos".

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 concatenado con "CrackS-Latinos"
For i = 1 To Len(UsEr)
    consUsErHEXAMD5 += Convert.ToChar(Convert.ToUInt32(Hex(CLng("&H" + Hex(Asc(Mid(UsEr, i, 1)))) Xor CLng("&H1E")), 16))
Next

'Probar MD5
consUsErHEXAMD5 = CrearMD5(consUsErHEXAMD5)
```

Arriba vemos nuestro código hasta la primer parte, donde hallábamos nuestro primer **HASH MD5**, ahí podemos ver mi función **CrearMD5()**. Mi problema en realidad no era utilizar la función **CrearMD5()**, si no, que no hacía bien la String, recordemos que en el tuto anterior yo trabajaba con puros valores **HEXADECIMALES**.

[Complemento Teoría 1663. Utilizando HASH MD5 (KeyGen)(x64DBG)]

0040166A	33 C9		xor ecx,ecx
0040166C	0F BE 88 07 9D 4		movsx ecx,byte ptr ds:[eax+409D07]
00401673	80 F1 3C		xor cl,3C
00401676	88 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 v1.0.40166C
00401685	68 00 04 00 00		push 400
0040168A	68 07 AD 40 00		push keygenme v1.0.40AD07
0040168F	E8 4E 07 00 00		call <keygenme v1.0.RtlZeroMemory>
00401694	E8 7F 4A 00 00		call keygenme v1.0.406118
00401699	68 00 04 00 00		push 400
0040169E	68 07 A1 40 00		push keygenme v1.0.40A107
004016A3	E8 80 4A 00 00		call keygenme v1.0.406158
004016A8	E8 08 4B 00 00		call keygenme v1.0.406188
004016AD	68 07 AD 40 00		push keygenme v1.0.40AD07
004016B2	6A 10		push 10
004016B4	50		push eax
004016B5	E8 56 4B 00 00		call keygenme v1.0.406210
004016BA	68 00 04 00 00		push 400

Ahí tenemos la segunda parte del KeyGen y en donde volvemos a calcular un nuevo **HASH MD5**, y era aquí donde no podía hacer la String de forma correcta para calcular mi **HASH MD5**; si recordamos, yo sin saberlo decía en el otro tuto, que donde era nuestra longitud ahora teníamos **0x400**, y resulta que ese valor da como entrada una String de **0x400** de largo, entonces debemos completar nuestra longitud a **0x400**.

```
'////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////'
'Probar MD5
consUsErHEXAMD5 = consUsErHEXAMD5.PadRight(&H400, Convert.ToChar(0))
UsEr = CreaerMD5(consUsErHEXAMD5)
'////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////'
```

COMPLETAMOS LA LONGITUD A 0x400

Bueno, eso era todo. Ahora tenemos nuevo conocimiento dentro de nuestra Mochila y seguro ya hemos dado un pequeño pasito hacia adelante para próximos casos como este reto.

PARA TERMINAR

No hay mucho que decir, solo que ahí vamos aprendiendo nuevas cosas, aquí nunca terminamos de aprender.

Creo que voy a reconocer y agradecer a personas puntuales que de alguna forma me ayudaron y dejaron sus opiniones. Como dice la canción "GRACIAS TOTALES" para **Thunder, sequeyo, MCKSys Argentina, J.J, ZELT@** y por supuesto a **Ricardo Narvaja**.

Saludos a todos,

@LUISFECAB