

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]



<b>Software</b>	Xylitol Crypto-KeyGenMe 1: BarbecueMe
<b>Protección</b>	Serial.
<b>Herramientas</b>	Windows 7 Home Premium SP1 x32 Bits (S.O donde trabajamos.) OllyDBG v1.10 RDG Packer Detector v0.7.6.2017 PEiD v0.95 (Maquetado QwErTy) Greatis WinDowse v5.3 Microsoft Visual Studio 2017  <a href="#">DESCARGAR HERRAMIENTAS</a>  <a href="#">DESCARGAR TUTO+ARCHIVOS</a>
<b>SOLUCIÓN</b>	KEYGEN
<b>AUTOR</b>	LUISFECAB
<b>RELEASE</b>	Octubre 1 2018 [TUTORIAL 010]

# INTRODUCCIÓN

Hoy es 19 de Septiembre de 2018, y he querido colocar esta fecha como punto de partida para ver cuánto tiempo me tomará en lograr este reto, claro está, si logro terminarlo, si no estas palabras han de quedar aquí guardadas como testigos mudos de uno más de mis muchos intentos en cracking.

Ya han pasado varios días desde que escribí el párrafo de arriba, ya pude vencer el reto <[Xylitol Crypto-KeyGenMe 1: BarbecueMe](#)> pero esto no termina y el tiempo sigue corriendo. Todavía me falta terminar este tutorial y una vez terminado si podré decir que este reto lo completé, porque qué gracia tiene hacer el **KeyGen** y no compartir la solución, eso me parece muy mezquino. Así como he podido aprender gracias a los tutoriales que otros muchos han compartido, yo quiero hacer lo mismo y retribuir en algo por todas esas enseñanzas.

Este reto lo tenía guardado a la espera de que me sintiera con conocimientos para enfrentarlo. El primer vistazo de este **KeyGenMe** se remonta al tutorial hecho por [QwErTy](#), [1649](#). Luego hice mis dos tutoriales anteriores que vienen siendo uno solo, [1663](#) y [1664](#), ya con eso conocí un poco de las maneras de encriptación y los métodos utilizados, y sobre todo de aplicaciones para averiguar si alguna aplicación hace uso de estos métodos, evitándonos seguir caminos más largos y complicados para hallar una solución.

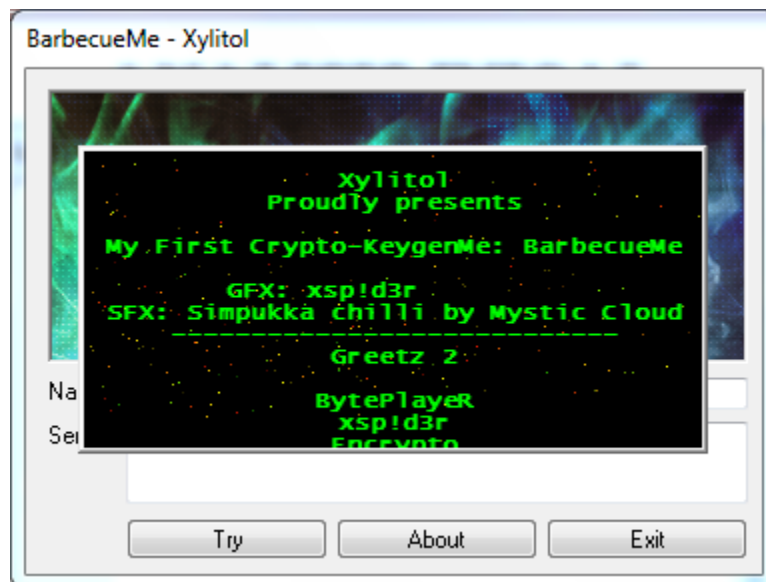
En este nuevo reto conozco el **BLOWFISH** y el **HASH 256**, empezando por el segundo; el **HASH 256** que no ofreció dificultad porque tiene un principio igual al **HASH MD5** y que por fortuna ya conocía, en cambio el **BLOWFISH** no se comporta como los **HASH**, es más, mientras buscaba información para este método noté que lo clasificaban como un encriptador pero que no hace parte de la familia de los **HASH**; y precisamente el **BLOWFISH** fue el que me demoró para hacer el reto porque la teoría que encontraba y los ejemplos de este no se aplicaban de la misma manera que lo hacía en el <[Xylitol Crypto-KeyGenMe 1: BarbecueMe](#)>, así que terminé haciéndolo de frente y sin dolor.

Ya contamos un poco de historia de este reto y cómo finalmente lo hicimos, solo queda saludar a toda la lista de **CracksLatinoS** y **PeruCrackerS** que siempre me han ayudado para seguir adelante y completar estos retos.

## ANALISIS INICAL



Empecemos con una captura inicial, ahí vemos el botón **"Try"** que lógicamente es para probar nuestro serial, pero aquí nos enfocamos en el botón **"About"** para saber más acerca de este reto, que **QwErTy** lo convirtió de un feroz tiburón a una indefensa sardina.



Ahí **Xylitol** orgullosamente nos presenta su primer **Crypto-KeyGenMe**. Ya con eso nos está diciendo que muy seguramente utiliza métodos criptográficos. Podrán recordar que en un tuto anterior hice un **Patch**, y que ahí utilicé el icono y música que saqué del SRC de este reto. Ahora miremos el **ReadMe** a ver qué nos dice.

=> Xylitol Crypto-KeyGenMe 1: BarbecueMe

- No Patching
- No self keygenning (Forbidden!)
- Make a Keygen with a cool chiptune :)
- Write a tutorial and send it :)
- How to win FreeCell Game #11982 ?

hope you will enjoy it :p

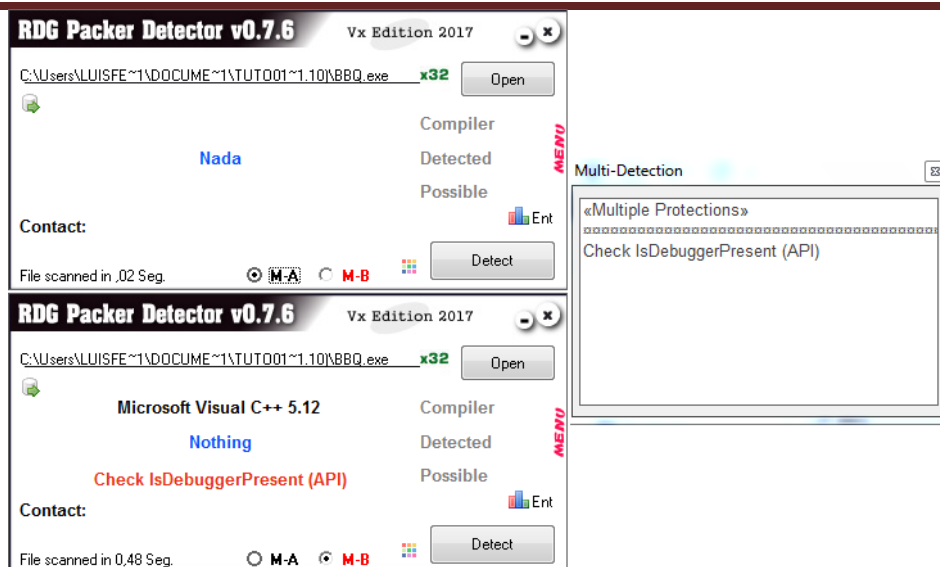
Happy keygening & thanks for attention

Muy claras las reglas, esto es un reto **KeyGenMe**, así que nada de **Parchear** y mucho menos hacer un **Self-KeyGen**, solo hacer el **KeyGen** y de paso con un buen tuneo. El **KeyGen** te lo hago **Xylitol** pero te quedo debiendo el tuneo. También escribir el tutorial, que ya estoy en eso ahora mismo; luego miro cómo te lo envió. El último punto, supongo que es para tu grupo que sabrán a qué juego te refieres, claro cabe precisar que este reto ya lleva años de presentado, así que ese punto ya no tendrá relevancia. Sigamos analizándolo.

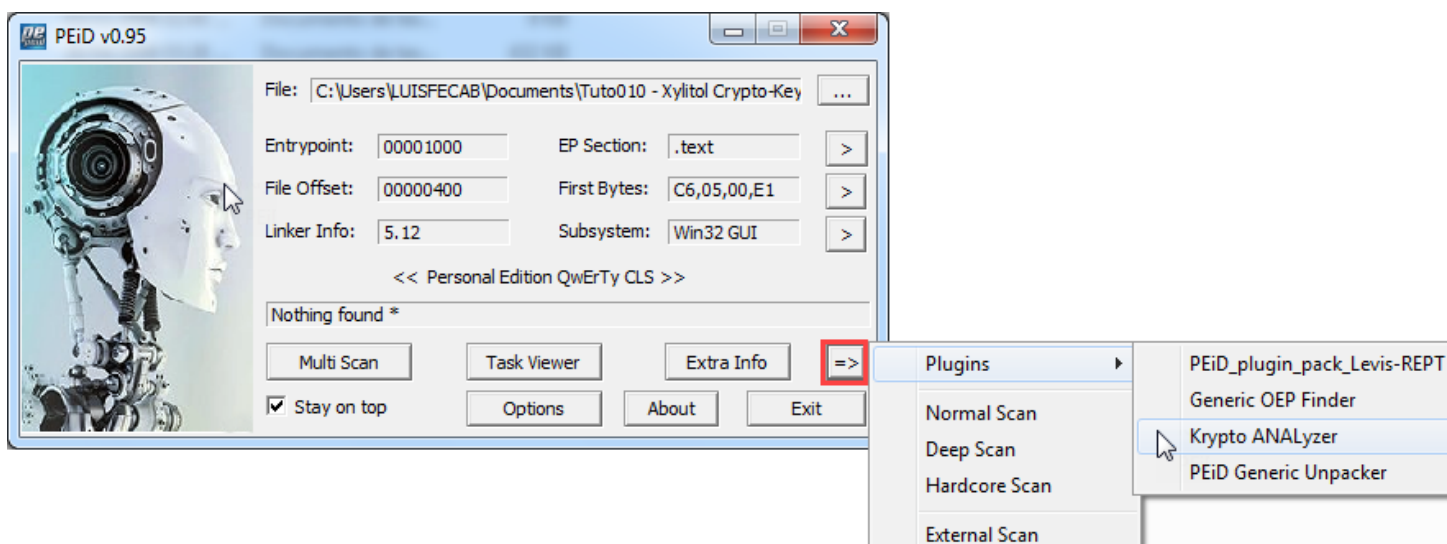


Vemos al "**CHICO MALO**" y también que no hace ninguna validación en "**Name**" o "**Serial**" vacíos. Supongo que habrá algún serial para un usuario sin nombre, entonces mi **KeyGen** hará lo mismo que saque un serial para el usuario sin nombre. Ahora vamos a revisarlo con el <**RDG Packer Detector v0.7.6.2017**> para ver con qué lo hicieron y si trae sorpresas.

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]



**M-A** no muestra nada y el escaneo en **M-B** dice que fue hecho en **Microsoft Visual C++ 5.12** y que tiene la **API\_IsDebuggerPresent**. Aquí la información verdadera es que efectivamente tiene la **API\_IsDebuggerPresent** y lo falso es que sea un **MVC++** porque en realidad es un **ASM**, este error ya lo vi en mi anterior tuto. Y para ver si tiene rutinas de encriptación utilizaremos el Plugin **Krypto ANALyzer** del <**PEiD v0.95 (Maquetado QwErTy)**>, conocido regularmente como **KANAL**. Agradecer a **QwErTy** que me compartió su <**PEiD**> personalizado, gracias **QwErTy** por ese detalle.

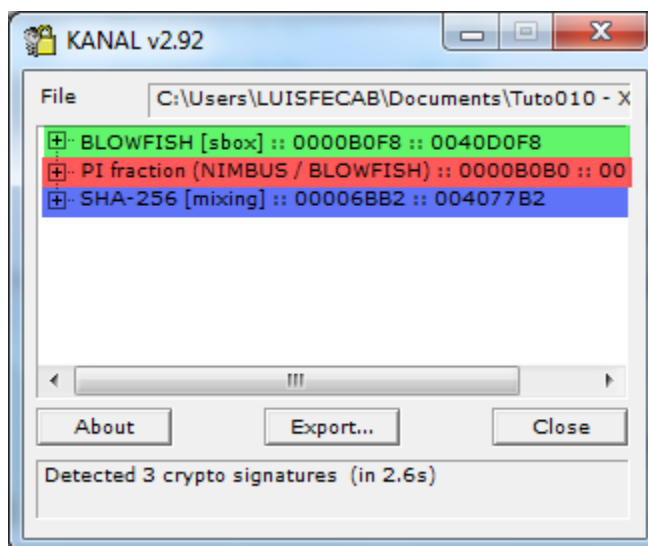


Agradecer también a **Nextco** de **PeruCrackers** que me compartió unas cuantas herramientas para cuando trabajemos con criptografía.

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

Crypto Checker v1.3 alpha 11	25/09/2018 09:20 a...	Carpeta de archivos	
Keygener Assistant v2.1.0	25/09/2018 11:35 a...	Carpeta de archivos	
Krypto ANALyzer 2.92	25/09/2018 10:56 a...	Carpeta de archivos	
SND Reverser Tool 1.5b1	25/09/2018 10:50 a...	Carpeta de archivos	
Crypto Checker v1.3 alpha 11	12/04/2018 01:28 ...	WinRAR archive	308 KB
Hash & Crypto Detector v1.4	12/04/2018 01:12 ...	WinRAR archive	369 KB
IDA.Pro.findcrypt2	20/02/2018 12:48 a...	WinRAR ZIP archive	373 KB
Keygener Assistant v2.1.0	12/04/2018 01:02 ...	WinRAR archive	1.404 KB
Krypto ANALyzer 2.92	12/04/2018 01:04 ...	WinRAR archive	118 KB
md5w	12/04/2018 12:36 ...	WinRAR archive	824 KB
RSA-Tool by tE!	12/04/2018 01:23 ...	WinRAR archive	57 KB
SND Reverser Tool 1.5b1	12/04/2018 01:03 ...	WinRAR archive	588 KB

Me surtió de muy buenas herramientas que fueron directamente a la mochila del cracking. Podemos ver que también está el Plugging **KANAL** que lo podemos usar independientemente sin el <PEid>, así que ahí tenemos dos maneras de usar nuestro **KANAL**. Miremos ahora qué nos arrojó el Plugin.



Probé las otras herramientas como detectores de Cripto-Análisis pero para cosas como esta me pareció mejor el **KANAL**. Resaltado en **VERDE** y **ROJO** tenemos detectado dos referencias a **BLOWFISH**. Voy a expresar lo que pienso se refieren esas dos referencias, con lo que pude entender cuando averigüé de este mentado **BLOWFISH** y qué hace en él reto. El primer **BLOWFISH [sbox]** es utilizado en el <BarbecueMe>, pero primero un poco de teoría.

### Teoría:

**BLOWFISH** es un codificador de 16 rondas Feistel y usa llaves que dependen de las Cajas-S. Tiene una estructura similar a CAST-128, el cual usa Cajas-S fijas. Son 4 cajas-S de 256 entradas.

La teoría de este algoritmo de encriptación es más extensa, en donde explican los otros procedimientos de encriptación pero por lo que yo pude ver en este reto no se hacen, aquí pareciese que solo se aplica el procedimiento de la teoría de arriba que sería el **BLOWFISH [sbox]**. Resulta que el <BarbecueMe> mueve una sección de **0x1000 BYTES** antes de entrar a la rutina de 16 rondas Feistel; pues esos **0x1000 BYTES** son las 4 cajas-S. Hagamos los cálculos, cada entrada equivale a **0x4 BYTES**

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

que darían **0x400** entradas y como son **4 cajas-S**, entonces cada caja tendrá **0x100** entradas que es igual a **256** entradas. Según tengo entendido esas entradas son constantes, puedo estar equivocado con respecto a ser constantes pero hasta aquí lo que pude entender es que sí. Esto lo veremos más claro cuando lo estemos reversando.

Luego tenemos el **PI fraction (NINBUS/BLOWFISH)** pero que no lo ejecuta el **<BarbecueMe>** para generar el serial. Y por último el **SHA-256**, que lo replicamos fácilmente.

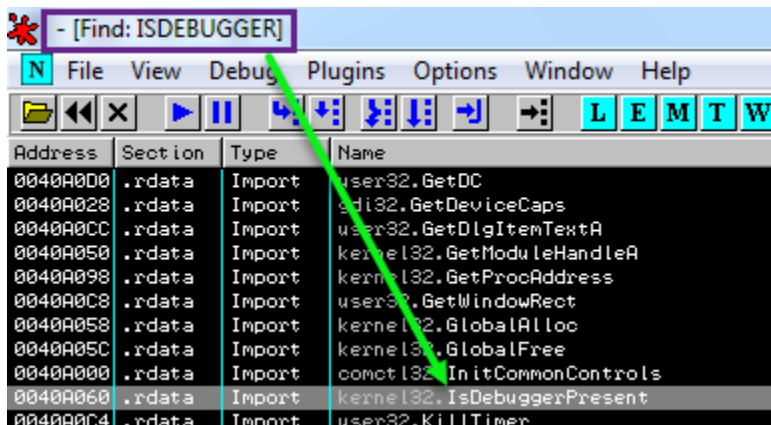
Con este análisis ya tenemos una muy buena idea de lo que debemos hacer para crear nuestro **KeyGen**, así que lo cargamos en nuestro querido **<OllyDBG v1.10>** para empezar el ataque.



## AL ATAQUE

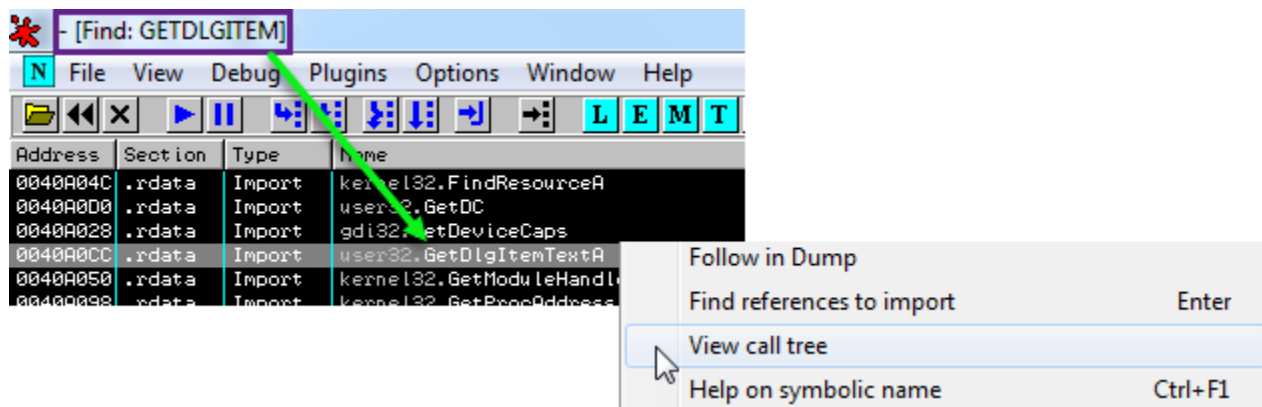
Vamos ir escribiendo este tutorial obviando algunas capturas de pasos que ya debemos saber. Si tú has realizado el curso de **Maestro Ricardo**, **OLLY DESDE CERO** y si has seguido mis primeros tutos como refuerzo, no tendrás inconveniente en seguirme en esta explicación.

Cargamos el <BarbecueMe> en nuestro Olly e inmediatamente buscamos las **API's** que usa el programa, <Clic derecho->Search for->Name (label) in current Module> o <CTRL+N>



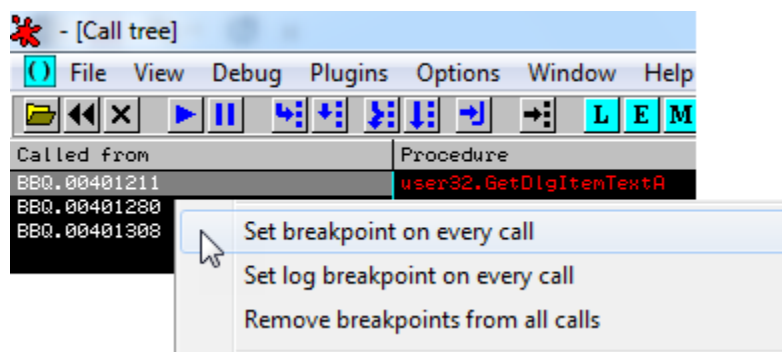
Ahí buscamos la **API\_IsDebuggerPresent**. Efectivamente el <RDG Packer Detector v0.7.6.2017> no mentía, así que debemos evitar este método antidebugging, y yo voy a utilizar los Plugins ya conocidos por todos nosotros para evitar esto, más sin embargo vuelvo a recomendarles el tutorial de **QwErTy**, [1649](#), donde nos explica muy claramente cómo vencer este método sin uso de los Plugins.


Ahora a buscar la "**ZONA CALIENTE**". Ya que estamos con las **API's**, buscaremos si hace uso de algunas de ellas para recuperar **Strings**. Busco la **API\_GetWindowTextA** y no me aparece; pruebo con otra muy conocida la **API\_GetDlgItemTextA** y esa sí me apareció.



Busquemos los **CALL** que llaman esta **API** con <View call tree> y les ponemos un <BREAKPOINT> para ver si uno de esos nos deja en la "**ZONA CALIENTE**".





Listo, ejecutemos con <F9> o  he ingresamos un **Name** y **Serial**, yo ingresaré los de combate que siempre uso, **LUISFECAB** y **MUYDIFICIL**.



Probemos con "Try" para ver dónde paramos con nuestros <BREAKPOINTS>.

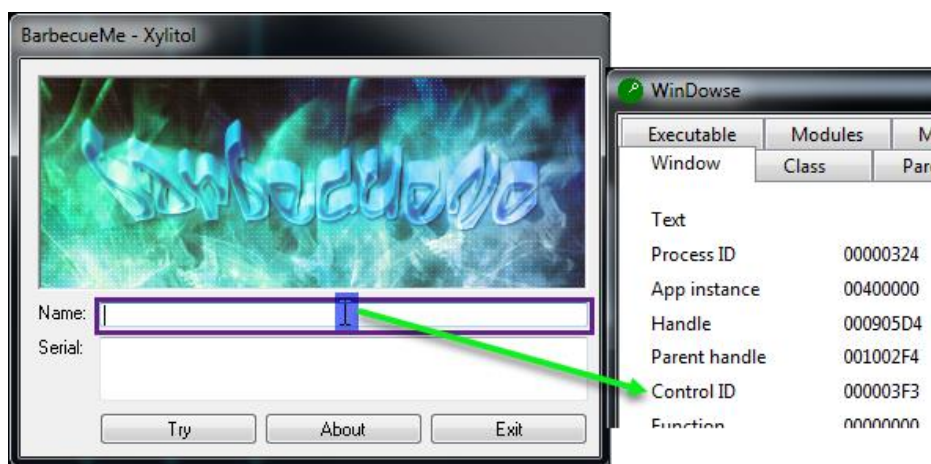
<pre> 00401211 . E8 40080000 CALL &lt;JMP.&amp;user32.GetDlgItemTextA&gt; 00401216 . 68 1DE14000 PUSH BBQ.0040E11D 0040121B . E8 DE080000 CALL &lt;JMP.&amp;kernel32.lstrlenA&gt; 00401220 . A3 1DED4000 MOV DWORD PTR DS:[40ED1D],EAX 00401225 . 69C0 39050000 IMUL EAX,EAX,539 0040122B . 50 PUSH EAX 0040122C . 68 09C94000 PUSH BBQ.0040C909 00401231 . 68 1DE54000 PUSH BBQ.0040E51D 00401236 . E8 F1070000 CALL &lt;JMP.&amp;user32.wsprintfA&gt; 0040123B . 68 1DE34000 PUSH BBQ.0040E31D 00401240 . FF35 1DED4000 PUSH DWORD PTR DS:[40ED1D] 00401246 . 68 1DE14000 PUSH BBQ.0040E11D 0040124B . E8 90800000 CALL BBQ.004092E0 00401250 . 68 1DE34000 PUSH BBQ.0040E31D 00401255 . 68 1DE74000 PUSH BBQ.0040E71D 0040125A . E8 99080000 CALL &lt;JMP.&amp;kernel32.lstrcpyA&gt; 0040125F . 68 1DE54000 PUSH BBQ.0040E51D 00401264 . 68 1DE74000 PUSH BBQ.0040E71D 00401269 . E8 7E080000 CALL &lt;JMP.&amp;kernel32.lstrcpyA&gt; 0040126E . 68 00020000 PUSH 200 00401273 . 68 21F14000 PUSH BBQ.0040F121 00401278 . 68 00C94000 PUSH BBQ.0040C90C 0040127D . FF75 08 PUSH DWORD PTR SS:[EBP+8] 00401280 . E8 D1070000 CALL &lt;JMP.&amp;user32.GetDlgItemTextA&gt; 00401285 . 50 PUSH EAX 00401286 . 68 21F14000 PUSH BBQ.0040F121 0040128B . E8 80620000 CALL BBQ.00407510 00401290 . 68 21ED4000 PUSH BBQ.0040ED21 00401295 . 68 1DE74000 PUSH BBQ.0040E71D 0040129A . E8 49630000 CALL BBQ.004075E8 </pre>	<pre> L Carga nuestro Name String = "" lstrlenA Mueve longitud de nuestro Name.  [&lt;%X&gt; Format = "%X" s = BBQ.0040E51D wsprintfA  Rutina crea String(longitud, Name). Convierte BYTES a Strings String2 = "" String1 = BBQ.0040E71D lstrcpyA StringToAdd = "" ConcatString = "" Concatena String + Cons(longitud*539) Count = 200 (512.) Buffer = BBQ.0040F121 ControlID = 40C90D (4245773.) hWnd GetDlgItemTextA  Mueve una seccion de memoria de longitud 400 (BLOWFISH [SB0X])  Rutina BLOWFISH </pre>
--	--

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

Ahí nos detuvimos en la dirección **00401211**. Podemos notar que he colocado mis comentarios para tener más claro todo, y sí, estamos en la "ZONA CALIENTE". Voy a extenderme un poco con la **API\_GetDlgItemTextA** para reforzar conocimientos. Esta **API** retorna el título o texto asociado a un control en un dialog box (cajas de texto). Miremos los parámetros de la API en el **STACK** o **PILA**.

0012FAD0	00120244	hWnd = 00120244 ('BarbecueMe - Xylitol',class='#32770')
0012FAD4	000003F3	ControlID = 3F3 (1011.)
0012FAD8	0040E11D	Buffer = BBQ.0040E11D
0012FADC	00000200	Count = 200 (512.)

Empecemos con el parámetro **ControlID = 3F3 (1011.)**. Ese es el control al que le buscará su título o texto asociado. Averigüemos cuál es ese control en el **<BarbecueMe>**, para eso le echamos mano a la herramienta **<Greatis WinDowse v5.3>** que conocimos en el curso **OLLY DESDE CERO**.



Al posicionarnos sobre el **TextBox** de nuestro **Name** obtenemos el **Control ID** que es **3F3**. Ya con eso sabemos que va a leer nuestro **Name** ingresado. El siguiente parámetro es el **Buffer = BBQ.0040E11D**, y es donde guardará nuestro **Name** en memoria, y por último tenemos el **Count = 200 (512.)**, que es la longitud a leer y guardar en el **Buffer = BBQ.0040E11D**; así que nos podemos dar cuenta que tenemos la opción de ingresar un **Name** con una longitud máxima de **0x200=512** caracteres, bueno podemos ingresar una longitud mayor pero no serán tomados en cuenta. Como estamos detenidos antes de llamar la **API\_GetDlgItemTextA**, pasémosla con **<F8>** y miramos que nos queda en el **Buffer = BBQ.0040E11D**.

0040120E	. FF75	PUSH	DWORD PTR SS:[EBP+8]	hWnd
00401211	. E8 48	CALL	<JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
00401216	. 68 10	PUSH	BBQ.0040E11D	String = "LUISFECAB"
0040121B	. E8 0E	CALL	<JMP.&kernel32.lstrlenA>	lstrlenA
00401220	. A3 10	MOV	DWORD PTR DS:[40E01D],EAX	Mueve longitud de nuestro Name.
00401225	. 69C0	IMUL	EAX,EAX,539	
0040122B	. 50	PUSH	EAX	<%X>

0040E11D=BBQ.0040E11D (ASCII "LUISFECAB")

Address	Hex dump	ASCII
0040E11D	4C 55 49 53 46 45 43 41 42 00 00 00 00 00 00 00	LUISFECAB.....
0040E120	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0040E130	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0040E140	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Ahí pasó nuestra **API** a la que inmediatamente le sacará su longitud para hacer una multiplicación en la dirección **00401225 IMUL EAX,EAX,539**; que luego a ese valor de la multiplicación lo pasa por la **API\_wsprintfA**.

# [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

0040122C	. 68 09	PUSH BBQ.0040C909	Format = "%.X"
00401231	. 68 10	PUSH BBQ.0040E510	s = BBQ.0040E510
00401236	. E8 F1	CALL <JMP.&user32.wsprintfA>	wsprintfA
0040123B	. 68 10	PUSH BBQ.0040E310	
00401240	. 55	PUSH EBP	

Esa **API** lo que hace es almacenar en un **Buffer** valores o caracteres que los convierte en una **String** con un formato definido, que en este caso fue determinado por el programador, y el formato que le dio fue "%.X", que es convertirlo a mayúsculas. Este valor más adelante lo concatenará con una **String** que se obtiene a partir de nuestro **Name**.

00401246	. 68 10	PUSH BBQ.0040E110	ASCII "LUISFECAB"
0040124B	. E8 98	CALL BBQ.004092E0	Rutina crea String(longitud, Name). Convierte BYTES a Strings
00401250	. 68 10	PUSH BBQ.0040E310	String2 = ""
00401255	. 68 10	PUSH BBQ.0040E510	String1 = BBQ.0040E510

Si seguimos traceando, llegamos a **0040124B CALL BBQ.004092E0**. Vamos a recordar mi tuto [1663](#), pues este procedimiento no es otro más que mi **Sub NoPesadilla()**.

004092E0	55	PUSH EBP	
004092E1	8BEC	MOV EBP,ESP	
004092E3	57	PUSH EDI	
004092E4	56	PUSH ESI	
004092E5	53	PUSH EBX	
004092E6	8B5D 00	MOV EBX,DWORD PTR SS:[EBP+C]	
004092E9	8B7D 10	MOV EDI,DWORD PTR SS:[EBP+10]	
004092EC	85D8	TEST EBX,EBX	
004092EE	8B75 08	MOV ESI,DWORD PTR SS:[EBP+8]	
004092F1	74 36	JE SHORT BBQ.00409329	
004092F3	0FB606	MOVZX EAX,BYTE PTR DS:[ESI]	
004092F6	8BC8	MOV ECX,EAX	
004092F8	83C7 02	ADD EDI,2	
004092FB	C1E9 04	SHR ECX,4	
004092FE	83E0 0F	AND EAX,0F	
00409301	83E1 0F	AND ECX,0F	
00409304	83F8 0A	CMP EAX,0A	EAX < 0A ACTIVA FLAG-C = 1
00409307	1BD2	SBB EDX,EDX	
00409309	83D0 00	ADC EAX,0	
0040930C	8D44D0	LEA EAX,DWORD PTR DS:[EAX+EDX*8+37]	
00409310	83F9 0A	CMP ECX,0A	ECX < 0A ACTIVA FLAG-C = 1
00409313	1BD2	SBB EDX,EDX	
00409315	83D1 00	ADC ECX,0	
00409318	C1E0 08	SHL EAX,8	
0040931B	8D4CD1	LEA ECX,DWORD PTR DS:[ECX+EDX*8+37]	
0040931F	0BC1	OR EAX,ECX	
00409321	46	INC ESI	
00409322	66:8947	MOV WORD PTR DS:[EDI-2],AX	
00409326	4B	DEC EBX	
00409327	75 CA	JNZ SHORT BBQ.004092F3	
00409329	8BC7	MOV EAX,EDI	
0040932B	C607 00	MOV BYTE PTR DS:[EDI],0	
0040932E	2B45 10	SUB EAX,DWORD PTR SS:[EBP+10]	
00409331	5B	POP EBX	
00409332	5E	POP ESI	
00409333	5F	POP EDI	
00409334	C9	LEAVE	
00409335	C2 0C00	RET 0C	

Y lo único que hace es tomar los valores almacenados en un **Buffer** (Valores **HEXA** del **DUMP**) y convertirlos en una **String**. Recordemos que los **HASH** y el **BLOWFISH** devuelven una cadena de **BYTES** (Los **BYTES** son valores **HEXA**) que en el **DUMP** serían valores **HEXA**, y precisamente esos **BYTES** son convertidos a una **String**, la cual es nuestra cadena encriptada. Un poco redundante la explicación pero quería dejarlo claro. Hagamos un ejemplo, tomemos mi primer carácter de mi **Name**, **LUISFECAB**, que sería la **L** y su valor **HEXA** es **4C**, pues ese sería el primer valor de la **String** a partir de nuestro **Name**. Les dejo adjunto la tablita "ASCII.jpg" para que se den cuenta.

En el tuto anterior programé esa rutina tal cual se carga en el Debugger pero ahora hice mi propio procedimiento para este **KeyGen**.

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

0040124B | CALL BBQ.004092E0 -> Convierte los BYTES de nuestro UsEr en una String, la cual es retornada en la variable consUsErHEXA pero en BYTES y no como ASCII.

```
For i = 1 To Len(UsEr)
    consUsErHEXA += Hex(Asc(Mid(UsEr, i, 1)))
Next

Dim array As Byte() = System.Text.Encoding.ASCII.GetBytes(consUsErHEXA + consLong)

consUsErHEXA = ""

For Each By In array
    consUsErHEXA += Hex(By.ToString)
Next
```

0040124B	. E8 90	CALL	BBQ.004092E0	Rutina crea String(longitud, Name). Convierte BYTES a Strings
00401250	. 68 10	PUSH	BBQ.0040E310	String2 = "4C5549534645434142"
00401255	. 68 10	PUSH	BBQ.0040E710	String1 = BBQ.0040E710
0040125A	. E8 99	CALL	<JMP.&kernel32.lstrcpyA>	lstrcpyA
0040125F	. 68 10	PUSH	BBQ.0040E510	StringToAdd = "2F01"
00401264	. 68 10	PUSH	BBQ.0040E710	ConcatString = ""
00401269	. E8 78	CALL	<JMP.&kernel32.lstrcatA>	Concatena String + Cons(longitud*539)
0040126E	. 68 00	PUSH	200	Count = 200 (512.)
00401273	. 68 21	PUSH	BBQ.0040F121	Buffer = BBQ.0040F121
00401278	. 68 00	PUSH	BBQ.0040C900	ControlID = 40C900 (4245773.)
0040127D	. FF 75	PUSH	DWORD PTR SS:[EBP+8]	hWnd
00401280	. E8 D1	CALL	<JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA

Ahí pasamos el **CALL** BBQ.004092E0 y si vemos la **String** en 00401250 **PUSH** BBQ.0040E31D tenemos **4C** como caracteres iniciales. Aquí lo relevante es que más abajo concatenamos nuestra **String** con el valor de la multiplicación con la **API\_lstrcatA**. Por último, más abajo tenemos en 00401280 otra vez la **API\_GetDlgItemTextA** pero no retorna nada, pueden revisarla para que vean que no hace nada. Sigamos avanzando hasta llegar a 0040128B **CALL** BBQ.00407510.

00401286	. 68 21	PUSH	BBQ.0040F121	
0040128B	. E8 80	CALL	BBQ.00407510	Mueve una seccion de memoria de longitud 400 (BLOWFISH [SBOX])
00401290	. 68 21	PUSH	BBQ.0040E021	
00401295	. 68 10	PUSH	BBQ.0040E710	ASCII "4C55495346454341422F01"
0040129A	. E8 49	CALL	BBQ.004075E8	Rutina BLOWFISH
0040129F	. 68 21	PUSH	BBQ.0040EF21	
004012A4	. FF 35	PUSH	DWORD PTR DS:[40E010]	
004012AA	. 68 21	PUSH	BBQ.0040ED21	
004012AF	. E8 20	CALL	BBQ.004092E0	

Por fin llegamos a lo bueno, recordando nuestro **ANALISIS INICAL** el **CALL** BBQ.00407510 moverá, más bien copiará esas **0x400** entradas de las **4 cajas-S**. Pueden entrar a ese **CALL** para que lo vean, no lo muestro porque no hay nada extraño ahí, solo hace un **REP MOV** en 00407525 **REP MOV** **DWORD PTR ES:[EDI],DWORD PTR DS:[ESI]** para mover las **0x400** entradas. El otro 0040129A **CALL** BBQ.004075E8 hará las **16 rondas Feistel** con nuestra **String** que sería "4C55495346454341422F01". Entremos a ese **CALL**.

# [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

```

004075E8 55 PUSH EBP
004075E9 8BEC MOV EBP,ESP
004075EB 56 PUSH ESI
004075EC 57 PUSH EDI
004075ED 53 PUSH EBX
004075EE 8B75 MOV ESI,DWORD PTR SS:[EBP+8]
004075F1 33C9 XOR ECX,ECX
004075F3 8B06 MOV EAX,DWORD PTR DS:[ESI]
004075F5 8B56 MOV EDX,DWORD PTR DS:[ESI+4]
004075F8 0FC8 BSWAP EAX
004075FA 0FCA BSWAP EDX
004075FC 33048 XOR EAX,DWORD PTR DS:[ECX*4+416630]
00407603 8BF8 MOV EDI,EAX
00407605 8BF0 MOV ESI,EAX
00407607 C1EF SHR EDI,18
0040760A C1EE SHR ESI,10
0040760D 81E6 AND ESI,0FF
00407613 8B1C MOV EBX,DWORD PTR DS:[EDI*4+416678]
0040761A 031C ADD EBX,DWORD PTR DS:[ESI*4+416A78]
00407621 0FB6 MOVZX EDI,AL
00407624 0FB6 MOVZX ESI,AL
00407627 331C XOR EBX,DWORD PTR DS:[EDI*4+416E78]
0040762E 031C ADD EBX,DWORD PTR DS:[ESI*4+417278]
00407635 3304 XOR EBX,EDX
00407637 41 INC ECX
00407638 8BD0 MOV EDX,EAX
0040763A 83F9 CMP ECX,10
0040763D 8BC3 MOV EHX,EBX
0040763F 75 BE JNZ SHORT BBQ.004075FC
00407641 92 XCHG EAX,EDX
00407642 3315 XOR EDX,DWORD PTR DS:[416670]
00407648 3305 XOR EAX,DWORD PTR DS:[416674]
0040764E 8B75 MOV ESI,DWORD PTR SS:[EBP+C]
00407651 0FC8 BSWAP EAX
00407653 0FCA BSWAP EDX
00407655 8906 MOV DWORD PTR DS:[ESI],EAX
00407657 8956 MOV DWORD PTR DS:[ESI+4],EDX
0040765A 5B POP EBX
0040765B 5F POP EDI
0040765C 5E POP ESI
0040765D C9 LEAVE
0040765E C2 08 RETN 8

```

TOMA LOS PRIMEROS 8 BYTES DE NUESTRA STRING Y LOS INVIERTE.

LEE ENTRADAS DE NUESTRAS cajas-S.

LEE ENTRADAS DE NUESTRAS cajas-S.

LAS 16 RONDAS.

GUARDO NUESTROS BYTES ENCRIPTADOS EN EL DUMP PARA LUEGO CONVERTIRLOS A UNA CADENA STRING.

Aquí ocurre la magia del **BLOWFISH**. Coge los primeros **8 BYTES** de nuestra **String** y los encripta con los valores constantes que toma de las **cajas-S** en función de aplicar unos cálculos a nuestros **BYTES**. Después de hacer sus **16 rondas** sale en **EAX** y **EDX** nuestros **8 BYTES** encriptados. Aquí podemos analizar algo más, si nosotros ingresamos un **Name** con una longitud que ocupara los **8 BYTES** iniciales, que sería **4** caracteres, el valor de la multiplicación **00401225 IMUL EAX,EAX,539** no será tomada en consideración, así mismo el resto del **Name** ingresado, pero solo para el encriptado **BLOWFISH** porque más adelante la longitud si es tomada en cuenta para calcular el **HASH 256**. De la misma forma, si ingresamos un **Name** con una longitud menor que no complete los otros **8 BYTES**, aún concatenado el resultado de **00401225 IMUL EAX,EAX,539**, el **BLOWFISH** tomará los **8 BYTES** los cuales están completados con **0x00**. Debemos de tomar eso en cuenta a la hora de programar el **KeyGen**, claro yo lo hice así porque no encontré una mejor solución para mi **BLOWFISH** y me tocó programar toda esta rutina (**CALL BBQ.004075E8**). Voy a reiniciar todo y meteré como **Name** solo la letra **"L"** para observar mejor todo este cuento. Lleguemos de nuevo a este **CALL**.

# [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

004075E8	55	PUSH EBP
004075E9	8BEC	MOV EBP,ESP
004075EB	56	PUSH ESI
004075ED	57	PUSH EDI
004075ED	53	PUSH EBX
004075EE	8B75	MOV ESI,DWORD PTR SS:[EBP+8]
004075F1	33C9	XOR ECX,ECX
004075F3	8B06	MOV EAX,DWORD PTR DS:[ESI]
004075F5	8B56	MOV EDX,DWORD PTR DS:[ESI+4]
004075F8	0FC8	BSWAP EAX
004075FA	0FCA	BSWAP EDX
DS:[0040E710]=33354334		
EAX=00000000		
Address	Hex dump	ASCII
0040E710	34 43 35 33 39 00 00 00 00 00 00 00 00 00 00 00	4C539...
0040E720	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Ahí está bien claro, la **Srting** es "**4C539**", **4C** por nuestro carácter "**L**" y **539** por **00401225 IMUL EAX,EAX,539**. Aquí sigue habiendo algo más que puede ser una "cascarita" o trampita, o bien puede ser un Bug pero que lo analizaremos más adelante. En el **KeyGen** explico esto también.

Como ven en la línea 225 completamos **consUsErHEXA** con "0" hasta una longitud de 16. La explicación a esto es que, para entrar a calcular nuestro encriptado **BLOWFISH** este toma los primeros 8 BYTES de nuestro **consUsErHEXA**, y recordemos que el BYTE lo representamos con dos caracteres, y ese es el motivo de colocar longitud 16 ( $8*2=16$ ). Analizando un poco, si nuestro user tiene una longitud igual o mayor a 4, estos ocuparán los 8 BYTES iniciales y por consiguiente la **consLong** no será tomada en consideración, así mismo el resto del user ingresado pero solo para el encriptado **BLOWFISH** porque más adelante la longitud si es tomada en cuenta para calcular el **HASH 256**.

```
consUsErHEXA = consUsErHEXA.PadRight(16, Convert.ToChar("0"))
```

```
'0040129A | CALL BBQ.004075E8 -> Rutina BLOWFISH
```

```
calCaracterEAX = CLng("&H" + Mid(consUsErHEXA, 1, 8))
```

```
calCaracterEDX = CInt("&H" + Mid(consUsErHEXA, 9, 8))
```

Bueno, volvamos a donde estábamos con mi **Name**, **LUISFECAB**. Al salir de nuestra rutina **BLOWFISH** haremos el tercer **004012AF CALL BBQ.004092E0**.

Address	Hex dump	ASCII
0040ED21	AE 21 DA 01 EE 0D 5A 8F 00 00 00 00 00 00 00 00	<+r0-.ZA.....
0040ED31	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0040ED41	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
0040ED51	00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....
Address	Hex dump	ASCII
0040EF21	41 45 32 31 44 41 30 31 45 45 30 44 35 41 38 46	AE21DA01EE0D5A8F
0040EF31	30 30 00 00 00 00 00 00 00 00 00 00 00 00 00 00	00.....
0040EF41	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Lo que los comentaba hace rato, solo convierte en una **String** nuestros **BYTES**. Podemos notar que convirtió **9 BYTES** y eso es porque lo hace en función de nuestra longitud del **Name** que es **9**. Entonces, nuestra nueva **String** encriptada tendrá la longitud de nuestro **Name** que se completará con ceros para longitudes mayores. Para nuestro caso de **Name** **LUISFECAB**, la **string** es "**AE21DA01EE0D5A8F00**". Pero si aplicamos el ejemplo de nuestro **Name** = "**L**". Reiniciemos todo ingresando la "**L**".

# [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

SALIDA DE LA RUTINA **BLOWFISH**.

Address	Hex dump	ASCII
0040ED21	B3 85 0E 6F D8 26 77 1E 00 00 00 00 00 00 00 00	àRoï&wA... ....
0040ED31	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

CONVERTIMOS A UNA CADENA STRING,  
Y SOLO ES EL PRIMER BYTE PORQUE  
NUESTRA LONGITUD ES DE 1 (LA **L**)

Address	Hex dump	ASCII
0040EF21	42 33 01 00 00 00 00 00 00 00 00 00 00 00 00 00	B3.....
0040EF31	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

La imagen de arriba es muy clara, como nuestra longitud es **1**, nuestra **String** a aplicarle el **HASH 256** sería "**B3**". Todos esos pequeños detalles los debemos tener en cuenta para nuestro **KeyGen**.

La rutina **BLOWFISH** devuelve una cadena de 8 BYTES la cual es convertida a una String para aplicarle el **HASH 256**. Aquí ya lo tenemos como una String, solo completamos la String en función de la longitud del user.

BYTES retornados por la rutina **BLOWFISH**.

consUsErHEXA = RedondearHEXA(Hex(calCaracterEAX)) & RedondearHEXA(Hex(calCaracterEDX))

Ajustamos nuestra String en función de la longitud.

consUsErHEXA = Mid(consUsErHEXA, 1, Len(UsEr) \* 2)

Bueno regresemos a nuestro **Name**, "**LUISFECAB**", que la **string** es "**AE21DA01EE0D5A8F00**" a la cual le aplicamos el **HASH 256**.

004012B4	. 8BF8	MOV EDI,EAX	
004012B6	. E8 B9	CALL BBQ.00409174	INI_SHA256
004012BB	. 57	PUSH EDI	
004012BC	. 68 21	PUSH BBQ.0040EF21	ASCII "AE21DA01EE0D5A8F00"
004012C1	. E8 0A	CALL BBQ.004091D0	CALC_SHA256
004012C6	. E8 79	CALL BBQ.00409244	CALC_SHA256
004012CB	. 68 10	PUSH BBQ.0040EB10	
004012D0	. 6A 20	PUSH 20	
004012D2	. 50	PUSH EAX	
004012D3	. E8 08	CALL BBQ.004092E0	SALIDA_SHA256. Convierte BYTES a Strings
004012D8	. 68 10	PUSH BBQ.0040EB10	String2 = ""

Gracias a nuestro tuto anterior donde conocimos el **HASH MD5**, no habrá problema con este **HASH 256** porque tienen el mismo principio, así que el primer **004012B6** **CALL BBQ.00409174**; ahí lo llamé **INI\_SHA256**, se cargaran las constantes.



# [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

```

00409174 . 57 PUSH EDI
00409175 . 33C0 XOR EAX,EAX
00409177 . A3 C0 MOV DWORD PTR DS:[4176C0],EAX
0040917C . 33C0 XOR EAX,EAX
0040917E . A3 C4 MOV DWORD PTR DS:[4176C4],EAX
00409183 . BF 80 MOV EDI,BBQ.00417680
00409188 . B9 10 MOV ECX,10
0040918D . F3AB REP STOS DWORD PTR ES:[EDI]
0040918F . B8 C8 MOV EBX,ARR.004176C8
00409194 . C700 MOV DWORD PTR DS:[EAX],6A09E667
0040919A . C740 MOV DWORD PTR DS:[EAX+4],BB67AE85
004091A1 . C740 MOV DWORD PTR DS:[EAX+8],3C6EF372
004091A8 . C740 MOV DWORD PTR DS:[EAX+C],A54FF53A
004091AF . C740 MOV DWORD PTR DS:[EAX+10],510E527F
004091B6 . C740 MOV DWORD PTR DS:[EAX+14],9B05688C
004091BD . C740 MOV DWORD PTR DS:[EAX+18],1F83D9AB
004091C4 . C740 MOV DWORD PTR DS:[EAX+1C],5BE0CD19
004091C8 . 5F POP EDI
004091CC . C3 RETN

```

Luego tenemos estos dos **004012C1 CALL BBQ.004091D0** y **004012C6 CALL BBQ.00409244**, que encriptan nuestro **HASH 256** y que será convertido a una cadena **String** con la rutina que ya conocimos y que es llamada desde **004012D3 CALL BBQ.004092E0**. Pasemos todo esto hasta llegar a **004012F1 CALL <JMP.&kernel32.lstrcatA>**.

004012E7 . 68 17 PUSH BBQ.0040C917	StringToAdd = "-OMGWTFBBQ"
004012EC . 68 10 PUSH BBQ.0040E71D	ConcatString = "6F0A4F72226076BB27021DD3360BE2CD272FC6C4A851AB3581D4635FE9ADCE4B"
004012F1 . E8 F6 CALL <JMP.&kernel32.lstrcatA>	lstrcatA
004012F6 . 68 00 PUSH 200	Count = 200 (512.)
004012FB . 68 23 PUSH BBQ.0040F323	Buffer = BBQ.0040F323
00401300 . 68 F7 PUSH 3F7	ControlID = 3F7 (1015.)
00401305 . FF75 PUSH DWORD PTR SS:[EBP+8]	hWnd
00401308 . E8 49 CALL <JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
0040130D . 68 10 PUSH BBQ.0040E71D	String2 = "6F0A4F72226076BB27021DD3360BE2CD272FC6C4A851AB3581D4635FE9ADCE4B"
00401312 . 68 23 PUSH BBQ.0040F323	String1 = ""
00401317 . E8 D6 CALL <JMP.&kernel32.lstrcmpiA>	lstrcmpiA
0040131C . 0BC0 OR EAX,EAX	
0040131E . 74 16 JE SHORT BBQ.00401336	
00401320 . 6A 18 PUSH 18	Style = MB_OK MB_ICONHAND MB_APPLMODAL
00401322 . 68 94 PUSH BBQ.0040CF94	Title = "BarbecueMe"
00401327 . 68 60 PUSH BBQ.0040CF60	Text = "OMG Sausages burning! Your serial is not correct"
0040132C . FF75 PUSH DWORD PTR SS:[EBP+8]	hOwner
0040132F . E8 3A CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401334 . EB 5F JMP SHORT BBQ.00401395	
00401336 . 6A 40 PUSH 40	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
00401338 . 68 C4 PUSH BBQ.0040CFC4	Title = "BarbecueMe"
0040133D . 68 A0 PUSH BBQ.0040CFA0	Text = "w00t, w00t! Your serial is correct"
00401342 . FF75 PUSH DWORD PTR SS:[EBP+8]	hOwner
00401345 . E8 24 CALL <JMP.&user32.MessageBoxA>	MessageBoxA

Vamos a concatenar nuestro **HASH 256** con la **String "-OMGWTFBBQ"**. En el **KeyGen**, el **HASH 256** y la **String "-OMGWTFBBQ"** lo hacemos en una sola línea.

```

Este es nuestro serial
consUsErHEXA = CrearHASH256(consUsErHEXA) & "-OMGWTFBBQ"

```

Luego con el **00401308 CALL <JMP.&kernel32.GetDlgItemTextA>**, cargaremos nuestro **Serial** que será comparado con nuestra **String** concatenada en **004012F1 CALL <JMP.&kernel32.lstrcmpiA>**. Lleguemos al salto, **0040131E JE BBQ.00401336**.

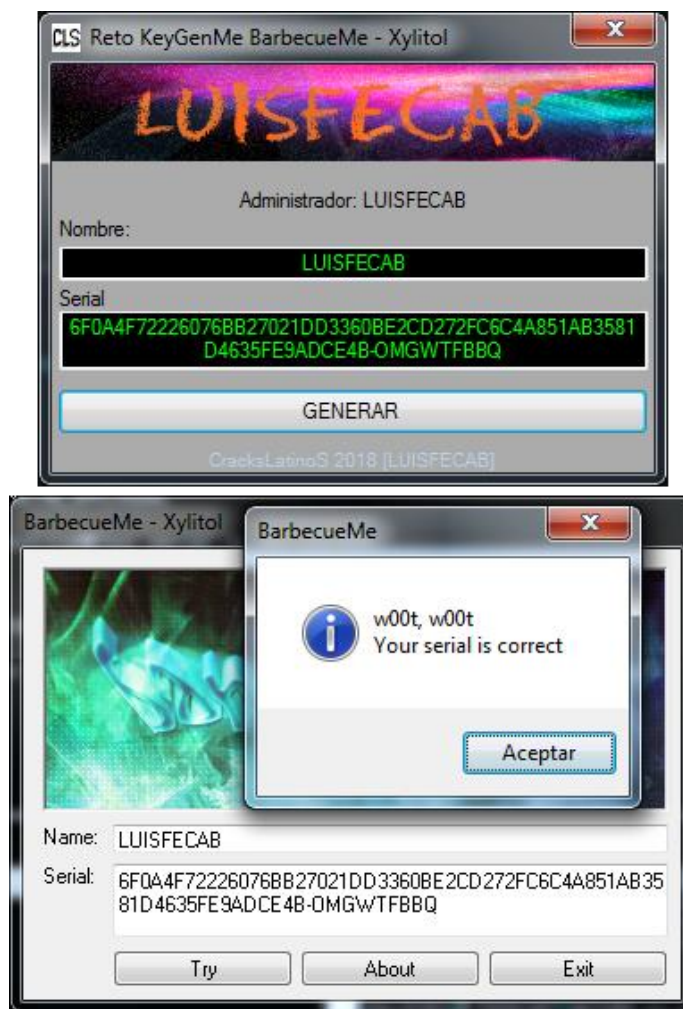
0040130D . 68 10 PUSH BBQ.0040E71D	String2 = "6F0A4F72226076BB27021DD3360BE2CD272FC6C4A851AB3581D4635FE9ADCE4B-OMGWTFBBQ"
00401312 . 68 23 PUSH BBQ.0040F323	String1 = "MUVDIFICIL"
00401317 . E8 D6 CALL <JMP.&kernel32.lstrcmpiA>	lstrcmpiA
0040131E . 74 16 JE SHORT BBQ.00401336	
00401320 . 6A 18 PUSH 18	Style = MB_OK MB_ICONHAND MB_APPLMODAL
00401322 . 68 94 PUSH BBQ.0040CF94	Title = "BarbecueMe"
00401327 . 68 60 PUSH BBQ.0040CF60	Text = "OMG Sausages burning! Your serial is not correct"
0040132C . FF75 PUSH DWORD PTR SS:[EBP+8]	hOwner
0040132F . E8 3A CALL <JMP.&user32.MessageBoxA>	MessageBoxA
00401334 . EB 5F JMP SHORT BBQ.00401395	
00401336 . 6A 40 PUSH 40	Style = MB_OK MB_ICONASTERISK MB_APPLMODAL
00401338 . 68 C4 PUSH BBQ.0040CFC4	Title = "BarbecueMe"
0040133D . 68 A0 PUSH BBQ.0040CFA0	Text = "w00t, w00t! Your serial is correct"
00401342 . FF75 PUSH DWORD PTR SS:[EBP+8]	hOwner
00401345 . E8 24 CALL <JMP.&user32.MessageBoxA>	MessageBoxA

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

En el recuadro **VERDE** esta la comparación y como vemos no son iguales, entonces no tomamos el salto y seguimos derecho al recuadro **ROJO** donde estará esperándonos el **<CHICO MALO>**.

Bueno, ya explicamos lo mejor posible todo lo que hace el **<BarbecueMe>** para generar el **Serial**, solo nos queda hacer el **KeyGen** con lo que sabemos. Ahí les adjunto el **SRC** de **KeyGen** para que lo estudien y si le encuentran algún **Bug**, sean libres de corregirlo y de paso me lo dicen porque si no corrijo mis errores los seguiré haciendo.

Pongamos a prueba nuestro **KeyGen** y generemos nuestro serial, y lo metemos al **<BarbecueMe>**.



Listo, funciona muy bien. Como dijo **QwErTy**, este tiburón ya no es tan feroz, aquí lo terminé pescando....Jejeje. Con esto damos por terminado nuestro **AL ATAQUE**.

## PARA TERMINAR

Esta sección va a ser más extensa de lo normal porque me quedaron un par de cosas pendientes y no quería dejarlas si tratarlas. Voy a empezar con el **BLOWFISH**, es algo de teoría básica nada técnico.

### Teoría:

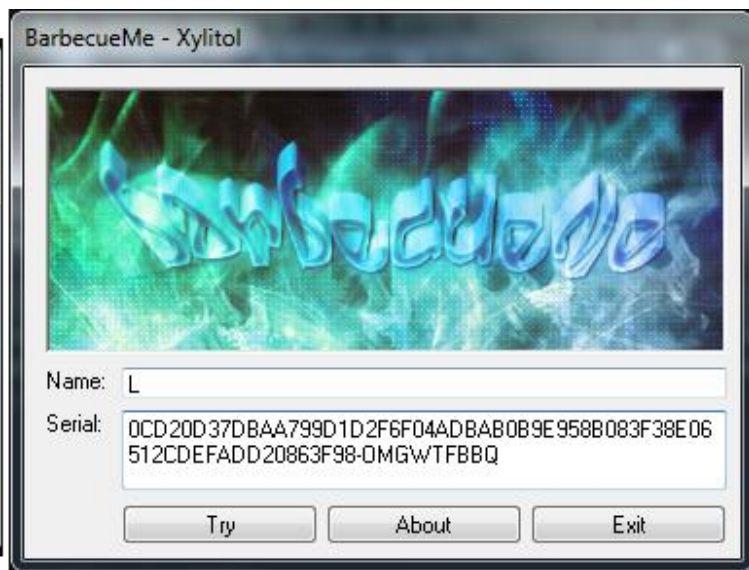
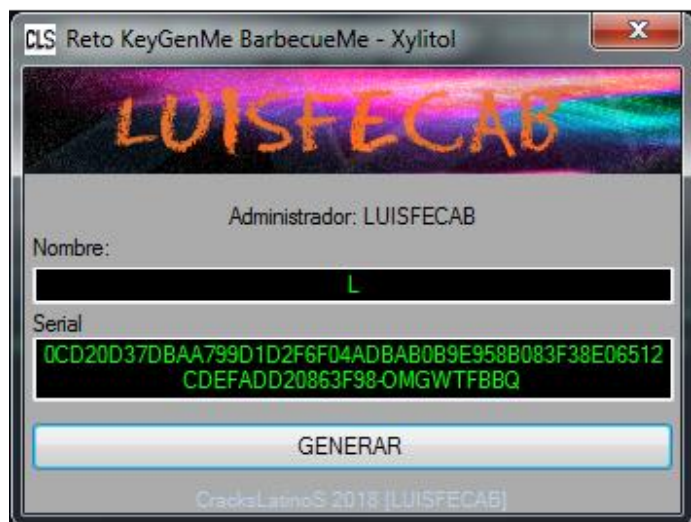
**BLOWFISH** es un codificador que utiliza una **KEY** la cual es ingresada por el usuario para encriptar el mensaje, el mensaje encriptado se puede desencriptar gracias a que usa Feistel para encriptar, solo se debe invertir el procedimiento y conocer la **KEY**; podemos decir que es un método criptográfico de doble sentido.

Bueno, esa teoría la escribí yo, no como si descubriera algo nuevo, si no como resumen de lo que he entendido sobre el **BLOWFISH**. Aquí se habla de una **KEY** que nosotros no utilizamos en el <BarbecueMe> por eso dije que aquí utilizábamos una parte del **BLOWFISH**.

No habrán pensado que se me había olvidado la "cascarita" o trampita, o posible Bug que trae el <BarbecueMe>. Para esto voy a pegar algo de teoría de este tuto que escribimos previamente.

Si ingresamos un **Name** con una longitud menor que no complete los otros 8 **BYTES**, aún concatenado el resultado de `00401225 IMUL EAX,EAX,539`, el **BLOWFISH** tomará los 8 **BYTES** los cuales están completados con `0x00`.

El <BarbecueMe> guarda esa **String** en una sección de memoria donde carga los primeros 8 **BYTES** y después de hacer su encriptado **BLOWFISH** hará un **HASH 256** y lo guardará en esa misma sección, pero después de mostrar al <CHICO BUENO> o <CHICO MALO> nunca libera o llena esa sección con "0", así que si tu ingresas un **Name** + `IMUL EAX,EAX,539` que no ocupe de nuevo esos 8 **BYTES**, entonces al cargar los 8 **BYTES** se cargarán una parte de **HASH 256** y eso conlleva a generar un **BLOWFISH** diferente y por ende un serial distinto. Hagamos un ejemplo con la "L".



# [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

Ingresamos la "L". Ahora iremos donde se carga la **String** para el **BLOWFISH**.

00401295	. 68 10E74000	PUSH	BBQ.0040E71D	ASCII "4C539"
0040129A	. E8 49630000	CALL	BBQ.004075E8	Rutina BLOWFISH
0040129F	. 68 21EF4000	PUSH	BBQ.0040EF21	
0040EF21=BBQ.0040EF21 (ASCII "4C539")				
Address	Hex dump	ASCII		
0040E71D	34 43 35 33 39 00 00 00 00 00 00 00 00 00 00 00	4C539.....		
0040E72D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			

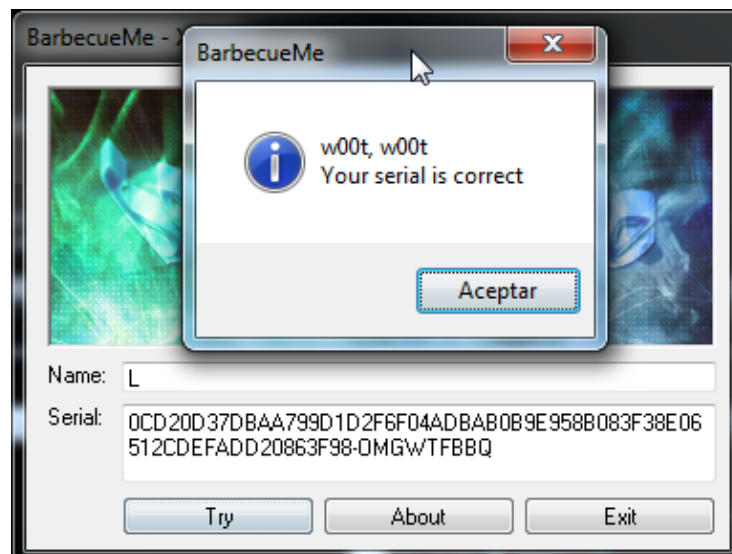
En la dirección **0040E71D** del **DUMP** tenemos la **String** para pasarle el **BLOWFISH**. Pasemos el **BLOWFISH** y miremos los **BYTES** que nos arroja.

00401295	. 68 10E74000	PUSH	BBQ.0040E71D	ASCII "4C539"
0040129A	. E8 49630000	CALL	BBQ.004075E8	Rutina BLOWFISH
0040129F	. 68 21EF4000	PUSH	BBQ.0040EF21	
004012A4	. FF35 10ED4000	PUSH	BBQ.0040ED21	
004012AA	. 68 21ED4000	PUSH	BBQ.0040ED21	
0040ED21=BBQ.0040ED21				
Address	Hex dump	ASCII		
0040ED21	B3 85 0E 6F D8 26 77 1E 00 00 00 00 00 00 00 00	!a#oi&wA.....		
0040ED31	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....		
0040ED41	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....		

Eso son los **BYTES** que serán convertidos a una **String** a la cual le aplicaremos el **HASH 256**, recordemos que es en función de nuestra longitud, y como la longitud es **1**, la **String** viene siendo el primer **BYTE** que sería "B3". Ahora lleguemos a donde el **HASH 256** ya es convertido en una **String**.

004012D3	. E8 08800000	CALL	BBQ.004092F8	SALIDA SHA256. Convierte BYTES a Strings
004012D8	. 68 10EB4000	PUSH	BBQ.0040EB1D	String2 = "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98"
004012DD	. 68 10E74000	PUSH	BBQ.0040E71D	String1 = BBQ.0040E71D
004012E2	. E8 11080000	CALL	<JMP.&kernel32.1strcpyA>	1strcpyA
0040C917=BBQ.0040C917 (ASCII "--OMGWTFBQQ")				
Address	Hex dump	ASCII		
0040E71D	30 43 44 32 30 44 33 37 44 42 41 41 37 39 39 44	0CD20D37DBAA799D		
0040E72D	31 44 32 46 36 46 30 34 41 44 42 41 42 30 42 39	1D2F6F04ADBAB0B9		
0040E73D	45 39 35 38 42 30 38 33 46 33 38 45 30 36 35 31	E958B083F38E0651		
0040E74D	32 43 44 45 46 41 44 44 32 30 38 36 33 46 39 38	2CDEFADD20863F98		
0040E75D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			
0040E76D	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00			

Miremos que copia el **HASH 256** en la dirección **0040E71D** del **DUMP** sobrescribiendo lo que teníamos antes. Hasta aquí todo parece bien, así que mostremos a nuestro <**CHICO BUENO**>.



## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

Pero qué pasa si volvemos a probar este mismo **Name** y **Serial**, en este mismo **<BarbecueMe>** que nos mostró el **<CHICO BUENO>**. Entonces paremos en la rutina del **BLOWFISH**.

00401295	. 68 1DE74000	PUSH	BBQ.0040E71D	ASCII "4C539"
0040129A	. E8 49630000	CALL	BBQ.004075E8	Rutina BLOWFISH
0040129F	. 68 21EF4000	PUSH	BBQ.0040EF21	ASCII "B3"

Address	Hex dump	ASCII
0040E71D	34 43 35 33 39 00 33 37 44 42 41 41 37 39 39 44	4C539.370BAA799D
0040E72D	31 44 32 46 36 46 30 34 41 44 42 41 42 30 42 39	1D2F6F04ADBAB0B9
0040E73D	45 39 35 38 42 30 38 33 46 33 38 45 30 36 35 31	E958B083F38E0651
0040E74D	32 43 44 45 46 41 44 44 32 30 38 36 33 46 39 38	2CDEFADD20863F98
0040E75D	2D 4F 4D 47 57 54 46 42 42 51 00 00 00 00 00 00	-OMGWTFBQQ.....

Aquí la cascarita, como no liberó memoria o no puso todo a "0", o que por lo menos los primeros 8 BYTES que son tomados por el **BLOWFISH** no fueron reemplazados por nuestro **Name + IMUL EAX,EAX,539** debido a que no ocupan esos 8 BYTES; y por ese motivo tendremos un **BLOWFISH** completamente diferente y por ende un **HASH 256** distintito al volver a probar el **Serial** para el **Name "L"**.

00401295	. 68 1DE74000	PUSH	BBQ.0040E71D	ASCII "4C539"
0040129A	. E8 49630000	CALL	BBQ.004075E8	Rutina BLOWFISH
0040129F	. 68 21EF4000	PUSH	BBQ.0040EF21	ASCII "B3"
004012A4	. FF35 1DED4000	PUSH	OMGWTFBQQ	
004012AA	. 68 21ED4000	PUSH	BBQ.0040ED21	

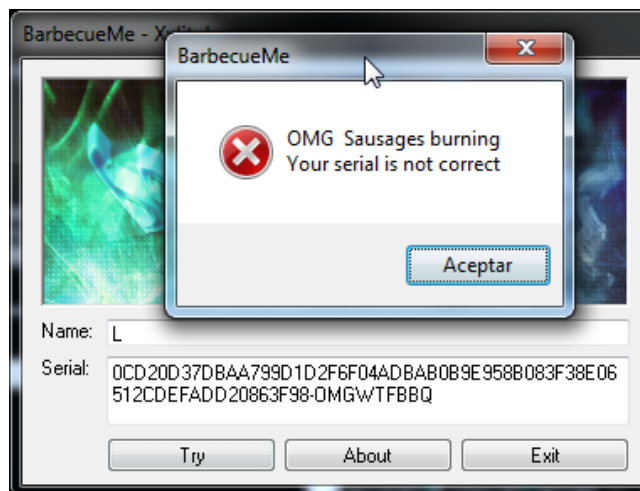
0040EF21=BBQ.0040EF21 (ASCII "B3")

Address	Hex dump	ASCII
0040ED21	EA CE 21 47 9F 0E 45 71 00 00 00 00 00 00 00 00	0p!Gf#Eq.....
0040ED31	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Ahí podemos ver que el **BLOWFISH** es diferente y que al convertirlo a una **String** y tomando la longitud de 1, tendremos es un **"EA"** y no el **"B3"** para aplicarle el **HASH 256**. Pasemos todo hasta llegar a la comparación para ver que ya no hay igualdad.

0040130D	. PUSH	BBQ.0040E71D	String2 = "FB2A4EDD17F56DD07D1C9AEF6B98E9094C35846F260BDC765C87BD6F2E8A7921-OMGWTFBQQ"
00401312	. PUSH	BBQ.0040F329	String1 = "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98-OMGWTFBQQ"
00401317	. CALL	<JMP.&kernel32.lstrcmpiA>	lstrcmpiA

No hay igualdad, así que nos manda al **<CHICO MALO>**. Y lo que parecía que era el **Serial** correcto en una segunda prueba ya no lo es.



Ahora, ¿cómo corregir esto?, si recordamos un poco. Cuando saca el **HASH 256** lo copiaba a **0040E71D** del **DUMP**, entonces podemos **NOPEAR** esa zona para que no copie el **HASH 256** en esa dirección, y ahí mismo podemos colocar unas instrucciones para reemplazar los 8 primeros **BYTES** por **0x00**, y también debemos cambiar esa dirección



## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

por la que originalmente está el **HASH 256** cuando es requerido para concatenarlo y compararlo.

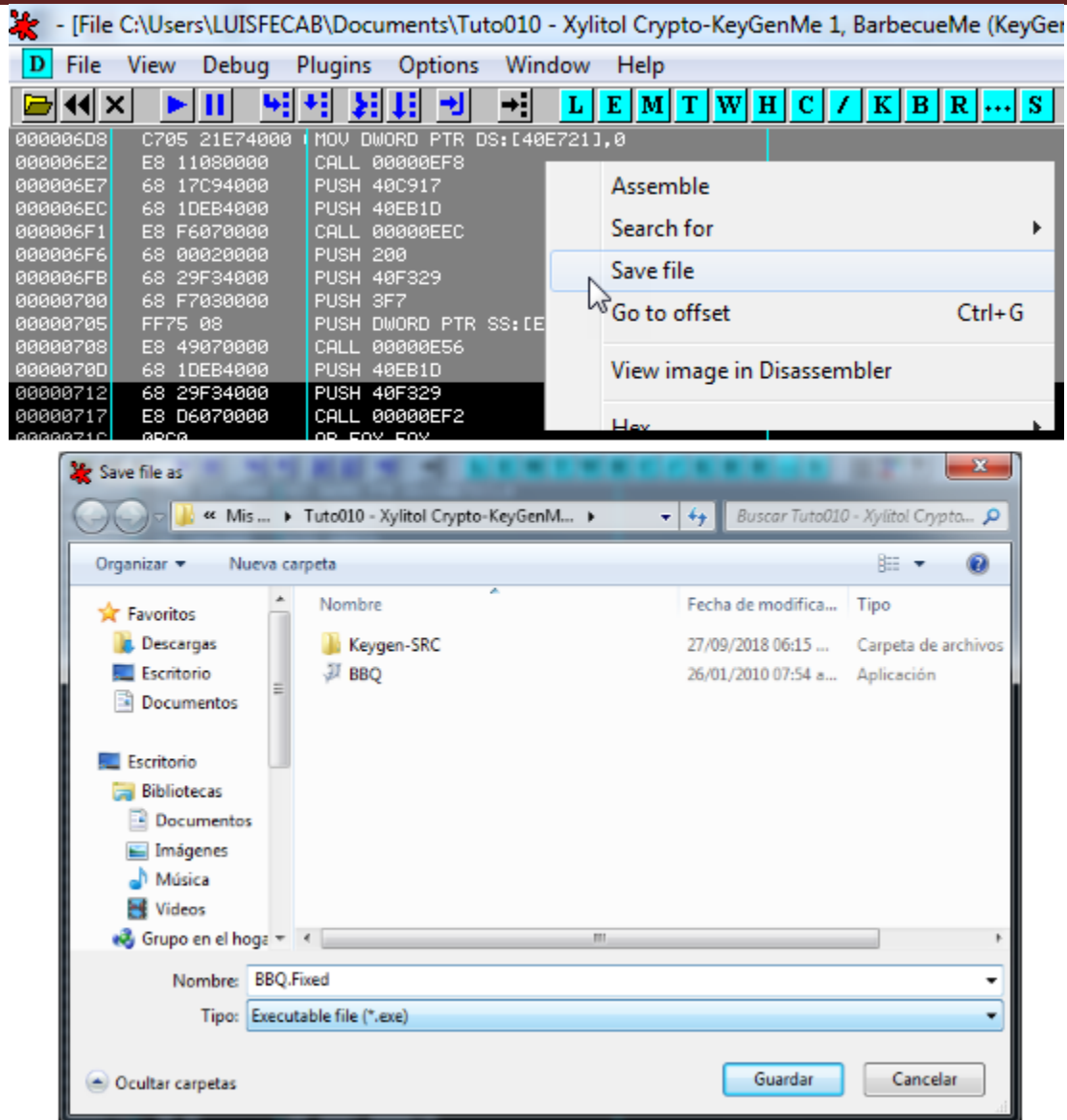
Address	Hex	Disassembly	Comment
004012C1	CALL	BBQ.004091D0	CALC_SHA256
004012C6	CALL	BBQ.00409244	CALC_SHA256
004012CB	PUSH	BBQ.0040EB1D	ASCII "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98-OMGWTFBBO"
004012D0	PUSH	20	
004012D2	PUSH	EAX	
004012D3	CALL	BBQ.004092E0	SALIDA_SHA256. Convierte BYTES a Strings
004012D8	MOV	DWORD PTR DS:[40E7211],0	
004012E2	CALL	<JMP.&kernel32.lstrcpyA>	
004012E7	PUSH	BBQ.0040C917	ASCII "-OMGWTFBBO"
004012EC	PUSH	BBQ.0040EB1D	ASCII "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98-OMGWTFBBO"
004012F1	CALL	<JMP.&kernel32.lstrcatA>	
004012F6	PUSH	200	Count = 200 (512.)
004012FB	PUSH	BBQ.0040F329	Buffer = BBQ.0040F329
00401300	PUSH	3F7	ControlID = 3F7 (1015.)
00401305	PUSH	DWORD PTR SS:[EBP+8]	hWnd
00401308	CALL	<JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
0040130D	PUSH	BBQ.0040EB1D	ASCII "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98-OMGWTFBBO"
00401312	PUSH	BBQ.0040F329	String1 = "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98-OMGWTFBBO"
00401317	CALL	<JMP.&kernel32.lstrcmpiA>	lstrcmpiA
0040131C	OR	EAX,EAX	
0040131E	JE	SHORT BBQ.00401336	
00401320	PUSH	10	Style = MB_OK MB_ICONHAND MB_APPLMODAL
00401322	PUSH	BBQ.0040CF94	Title = "BarbecueMe"
00401327	PUSH	BBQ.0040CF60	Text = "OMG Sausages burning! Your serial is not correct"
0040132C	PUSH	DWORD PTR SS:[EBP+8]	hOwner
0040132F	CALL	<JMP.&user32.MessageBoxA>	MessageBoxA

Una maravilla, me funcionó muy bien. Resaltado en los recuadros **VERDES** tenemos los cambios que hicimos. Yo hablaba de reemplazar los primeros **8 BYTES** pero me di cuenta que no era necesario, además no tenía suficiente espacio para colocar las dos instrucciones para reemplazar los **8 BYTES**; solo podía colocar una sola y coloqué la instrucción para reemplazar los últimos **4 BYTES** de esos **8 BYTES** iniciales porque con un **Name** de longitud **1** es suficiente para copar los primeros **5 BYTES**. Solo nos queda guardar nuestro **<BarbecueMe>** corregido.

004012D2	PUSH	EAX	
004012D3	CALL	BBQ.004092E0	SALIDA_SHA256. Convierte BYTES a Strings
004012D8	MOV	DWORD PTR DS:[40E7211],0	
004012E2	CALL	<JMP.&kernel32.lstrcpyA>	
004012E7	PUSH	BBQ.0040C917	
004012EC	PUSH	BBQ.0040EB1D	
004012F1	CALL	<JMP.&kernel32.lstrcatA>	
004012F6	PUSH	200	
004012FB	PUSH	BBQ.0040F329	
00401300	PUSH	3F7	
00401305	PUSH	DWORD PTR SS:[EBP+8]	
00401308	CALL	<JMP.&user32.GetDlgItemTextA>	GetDlgItemTextA
0040130D	PUSH	BBQ.0040EB1D	ASCII "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98-OMGWTFBBO"
00401312	PUSH	BBQ.0040F329	String1 = "0CD20D37DBAA799D1D2F6F04ADBAB0B9E958B083F38E06512CDEFADD20863F98-OMGWTFBBO"
00401317	CALL	<JMP.&kernel32.lstrcmpiA>	lstrcmpiA
0040131C	OR	EAX,EAX	

Todavía sigo sin entender por qué en ocasiones no me aparece la opción de toda las modificaciones, así que me toca seleccionar una sección que me abarque los cambios que hice y ahí si **<Clic derecho>Copy to executable>Selection>**.

## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]



Se nos abre la ventana del **DUMP**, <Clic derecho->**Save File**> y guardamos nuestros cambios; yo lo guardé como <**BBQ.Fixed**> y con eso damos por terminado las temas pendientes.

Ahora si en serio para terminar, muy contento de poder vencer al <**BarbecueMe**>, y gracias a los últimos conocimientos de criptografía adquiridos en la lista pude mejorar mi nivel y salir a pescar un tiburón.

Como siempre agradecer a todos aquellos que me han ayudado a lograr esto, en ocasiones nombro a algunos que han estado más directamente relacionados con el tuto pero créanme cuando les digo que los recuerdo a todos con mucho afecto mientras escribo mis tutoriales. He tenido la fortuna de entablar contacto con ustedes para compartir nuestras experiencias en esto que nos une a todos. También deseo enviarles un saludo a todos aquellos que leen este tutorial pero que no tengo la fortuna de conocer y motivarlos a aprender este arte del **Cracking**, y que compartan sus conocimientos y logros con todos nosotros, que hacemos parte de la lista **CracksLatinoS**.

Empecé colocando la fecha de inicio de este tutorial en la **INTRODUCCIÓN**. Personalmente yo pensaba para mis adentros que no podría completar este **KeyGenMe** y que sería devorado por el <**BarbecueMe**>, pero para mi satisfacción lo pude lograr,



## [Tuto010 - Xylitol Crypto-KeyGenMe 1: BarbecueMe (ASM)(KeyGen)(OllyDBG v1.10)]

---

así que como inicié he de terminar. Hoy, 1 de Octubre de 2018 escribo mis últimas palabras de este tutorial dedicado con todo el cariño a todos ustedes. Ahora sí puedo decirles **MISIÓN CUMPLIDA**.

Saludos a todos,

@LUISFECAB