

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]



Software	RandomWeird-Single by nextco
Protección	Serial.
HERRAMIENTAS	<p>Windows 7 Home Premium SP1 x32 Bits (SO donde trabajamos) X64DBG (Feb 14 2018) Keygener Assistant v2.1.0 Microsoft Visual Studio 2017</p> <p>DESCARGAR HERRAMIENTAS</p> <p>DESCARGAR TUTO+ARCHIVOS</p>
SOLUCIÓN	SERIAL. (FUERZA BRUTA-KEYGEN)
AUTOR	LUISFECAB
RELEASE	Mayo 8 2019 [TUTORIAL 014]

INTRODUCCIÓN

Tenía pendiente hacer este tutorial que sin las explicaciones de **nextco** creo que no hubiera podido completar el reto o me hubiera tomado años en resolverlo porque fijo lo hubiera abandonado y echado en el olvido.

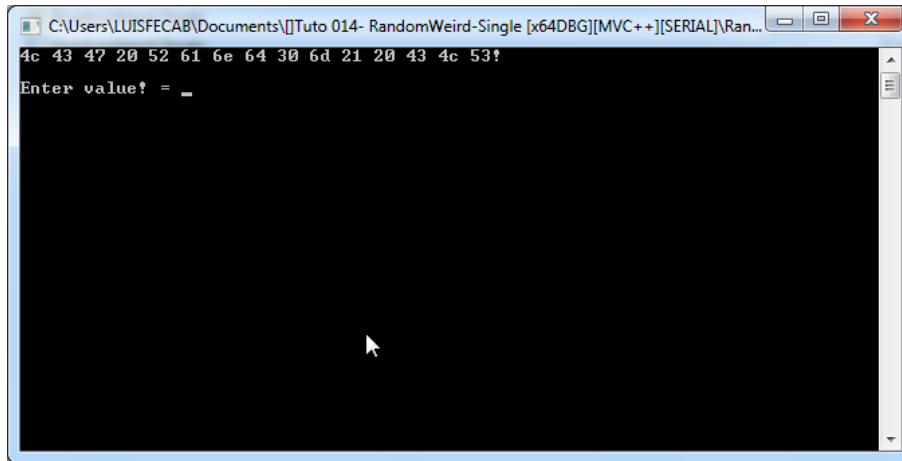
Este reto es la versión sencilla que **nextco** me compartió de forma exclusiva para que pudiera entender mejor cómo resolver el **Crackme**, por eso lo llamé <**RandomWeird-Single**> porque la versión original aunque es muy similar no funciona con el serial que hallamos aquí y que no he resuelto aún; prefiero mejor escribir este tutorial y disfrutar de esta pequeña victoria que me ha enseñado mucho.

Ya lo he dicho en anteriores tutoriales, que por falta de conocimiento es que termino siendo derrotado por estos retos, pero que cuando supe por dónde era la cosa pude resolverlo. Aquí aprendí algo llamado **Generador lineal congruencial (GLC)** y no es más que la aplicación de la teoría de **Números Pseudoaleatorios**.

Para terminar esta pequeña introducción, quiero como siempre saludar a la lista de **CracksLatinoS** y por supuesto a mis amigos de **PeruCrackerS**.

ANALISIS INICAL

Aquí no hay mucho que analizarle al <RandomWeird-Single>. El Crackme está hecho en MVC++ y nos pide que ingresemos un valor.



Lo que haremos en esta parte es profundizar en la teoría de **Números Pseudoaleatorios** enfocado en programación, claro resumido con mis palabras.

Teoría:

Si nos referimos a **Números Aleatorios** en programación caemos en un término mal utilizado porque en realidad los números no se generan aleatoriamente, si no, que se originan a partir de algoritmos que dan valores predeterminados, conociendo ciertos valores iniciales, y es por eso que nos referimos a **Números Pseudoaleatorios**.

Estos algoritmos se clasifican en algoritmos no congruenciales y congruenciales. Los algoritmos no congruenciales que podemos nombrar son: cuadrados medios, productos medios y multiplicador constante. Entre los algoritmos congruenciales se encuentran los algoritmos lineales y no lineales, como algoritmos lineales podemos referirnos al **algoritmo congruencial lineal**, multiplicativo y aditivo; y para los algoritmos no lineales, como el algoritmo de Blum, Blum y Shub, y el congruencial cuadrático¹.

Los diferentes lenguajes de programación han optado por implementar el **algoritmo congruencial lineal** para generar sus valores aleatorios.

Este algoritmo congruencial fue propuesto por D. H. Lehmer en 1951. El **algoritmo congruencial lineal** genera una secuencia de números enteros por medio de la siguiente ecuación recursiva:

$$x_{i+1} = (ax_i + c) \bmod(m) \quad i = 0, 1, 2, 3, \dots, n$$

¹ Eduardo García Dunna, H. G. (2013). Simulación y análisis con ProModel (Segunda ed.). México: PEARSON. Página 24.

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]

donde x_0 es la semilla, a es la constante multiplicativa, c es una constante aditiva, y m es el módulo. $x_0 > 0$, $a > 0$, $c > 0$ y $m > 0$ deben ser números enteros².

Una explicación más adecuada con un énfasis en programación lo puedes leer en Wikipedia buscando por **Generador lineal congruencial**.

En la siguiente imagen podemos ver los diferentes valores iniciales del algoritmo para diferentes lenguajes y entre esos el que nos interesa el de **MVC++**.

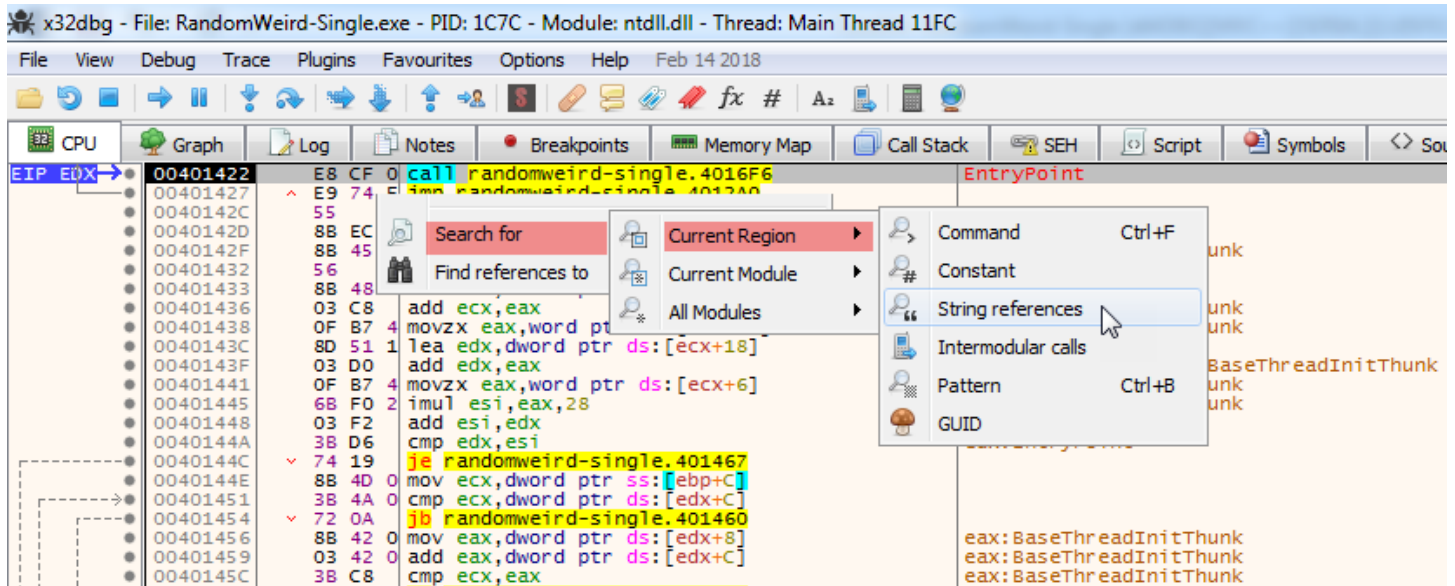
Fuente	m	(multiplicador) a	(incremento) c	Bits de salida en $rand()$ o $Random(L)$
<i>Numerical Recipes</i>	2^{32}	1664525	1013904223	
Borland C/C++	2^{32}	22695477	1	bits 30...16 en $rand()$, 30...0 en $lrand()$
glibc (usado por GCC) ⁷	$2^{31} - 1$	1103515245	12345	bits 30...0
ANSI C: Watcom, Digital Mars, CodeWarrior, IBM VisualAge C/C++ ⁸ C99, C11: Suggestion in the ISO/IEC 9899 ⁹	2^{31}	1103515245	12345	bits 30...16
Borland Delphi, Virtual Pascal	2^{32}	134775813	1	bits 63...32 de (<i>semilla</i> * L)
Turbo Pascal	2^{32}	134775813 (0x8088405 ₁₆)	1	
Microsoft Visual/Quick C/C++	2^{32}	214013 (343FD ₁₆)	2531011 (269EC3 ₁₆)	bits 30...16
Microsoft Visual Basic (6 and earlier) ¹⁰	2^{24}	1140671485 (43FD43FD ₁₆)	12820163 (C39EC3 ₁₆)	
RtlUniform from Native API ¹¹	$2^{31} - 1$	2147483629 (7FFFFFFF ₁₆)	2147483587 (7FFFFFFC ₁₆)	
Apple CarbonLib, C++11's <code>minstd_rand</code> ¹²	$2^{31} - 1$	16807	0	ver MINSTD
C++11's <code>minstd_rand</code> ¹²	$2^{31} - 1$	48271	0	ver MINSTD
MMIX by Donald Knuth	2^{64}	6364136223846793005	1442695040888963407	
Newlib, Musl	2^{64}	6364136223846793005	1	bits 63...32
VMS's MTHSRANDOM , ¹³ old versions of glibc	2^{32}	69069 (10DCD ₁₆)	1	
.Random en Java, POSIX <code>[ln]rand48</code> , glibc <code>[ln]rand48_r</code>	$2^{48} - 1$	25214903917 (5DEECE66D ₁₆)	11	bits 47...16
<code>random</code> ^{14 15 16 17 18} Si X_n es par, entonces X_{n+1} será impar, y vice versa—el bit más bajo oscila a cada paso.	$134456 = 2^{37}5$	8121	28411	$\frac{X_n}{134456}$
POSIX ¹⁹ <code>[jm]rand48</code> , glibc <code>[mj]rand48_r</code>	2^{48}	25214903917 (5DEECE66D ₁₆)	11	bits 47...15
POSIX <code>[de]rand48</code> , glibc <code>[de]rand48_r</code>	2^{48}	25214903917 (5DEECE66D ₁₆)	11	bits 47...0
cc65 ²⁰	2^{23}	65793 (10101 ₁₆)	4282663 (415927 ₁₆)	bits 22...8
cc65	2^{32}	16843009 (1010101 ₁₆)	826366247 (31415927 ₁₆)	bits 31...16
Anteriormente de uso común: RANDU ⁵	2^{31}	65539	0	

Ya con esta teoría podemos empezar nuestro ataque y ver cómo se implementa en **MVC++** la generación de números aleatorios (Pseudoaleatorios) con su función **rand()**.

² Eduardo García Dunna, H. G. (2013). Simulación y análisis con ProModel (Segunda ed.). México: PEARSON. Página 27.

AL ATAQUE

Carguemos el <RandomWeird-Single> en el **x64DBG** y busquemos en <String references> para ver si desde ahí podemos saltar a nuestra <ZONA CALIENTE>.



Perfecto, podemos ver que hay strings que nos cargarán buenas o malas noticias según le peguemos al serial correcto.

Address	Disassembly	String
00401006	push randomweird-single.403000	"4c 43 47 20 52 61 6e 64 30 6d 21 20 43 4c 53!\n"
00401013	push randomweird-single.403030	"\nEnter value! = "
00401028	push randomweird-single.403044	"%i"
00401099	push randomweird-single.403048	"You win!"
004010A8	push randomweird-single.403054	"You lose!"
004010BC	push randomweird-single.403060	"Only numbers!"

Me voy por el <CHICO BUENO> que dice "You win!" y desde ahí revisaré subiendo para ver desde dónde empieza a realizar cálculos.

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]

00401048	51	0	push ecx		
00401049	E8 E1 0	0	call <randomweird-single.srand>	→	SEMILLA INICIAL (X ₀)
0040104E	83 C4 0	0	add esp,4		
00401051	C7 45 F	0	mov dword ptr ss:[ebp-C],0		
00401058	C7 45 F	0	mov dword ptr ss:[ebp-4],0		
0040105F	EB 09	0	jmp randomweird-single.40106A		
00401061	8B 55 F	0	mov edx,dword ptr ss:[ebp-4]		edx:EntryPoint
00401064	83 C2 0	0	add edx,1		edx:EntryPoint
00401067	89 55 F	0	mov dword ptr ss:[ebp-4],edx		edx:EntryPoint
0040106A	83 7D F	0	cmp dword ptr ss:[ebp-4],8		
0040106E	7D 23	0	jge randomweird-single.401093		
00401070	E8 C0 0	0	call <randomweird-single.rand>	→	Generador Lineal Congruencial (GLC) $x_{i+1} = (ax_i + c) \bmod(m)$
00401075	99	0	cdq		
00401076	B9 0A 0	0	mov ecx,A		
0040107B	F7 F9	0	idiv ecx		edx:EntryPoint
0040107D	89 55 F	0	mov dword ptr ss:[ebp-C],edx		
00401080	8B 55 F	0	mov edx,dword ptr ss:[ebp-C]		
00401083	3B 55 F	0	cmp edx,dword ptr ss:[ebp-4]		
00401086	74 09	0	je randomweird-single.401091		
00401088	C7 45 F	0	mov dword ptr ss:[ebp-10],0		
0040108F	EB 02	0	jmp randomweird-single.401093		
00401091	EB CE	0	jmp randomweird-single.401061		
00401093	83 7D F	0	cmp dword ptr ss:[ebp-10],0		
00401097	74 0F	0	je randomweird-single.4010A8	→	AQUÍ ES EL SALTO DECISIVO
00401099	68 48 3	0	push randomweird-single.403048		403048:"You win!"
0040109E	E8 BD 0	0	call randomweird-single.401160		
004010A3	83 C4 0	0	add esp,4		
004010A6	EB 12	0	jmp randomweird-single.4010BA		
004010A8	68 54 3	0	push randomweird-single.403054		403054:"You lose!"

Tenemos el salto decisivo en la dirección **00401097** y lo **RESALTADO EN VERDE** es donde haré los cálculos con nuestro valor ingresado que viene siendo la **Semilla inicial (X₀)** y que en el programa es almacenada por la función **srand()** que se ejecuta en la dirección **00401049**. Luego se origina nuestro número aleatorio con la función **rand()** en la dirección **00401070**, ahí es que entra en acción el **Generador Lineal Congruencial**. Revisando lo que sucede en esa **ZONA**, pasa lo siguiente: nuestro valor ingresado (**Semilla inicial**) debe generarnos números terminados en 0,1,2,3,...7.

Primer número	----	XXXXXXXXX0	00401070	E8 C0 0	0	call <randomweird-single.rand>	
			00401075	99	0	cdq	
			00401076	B9 0A 0	0	mov ecx,A	
			0040107B	F7 F9	0	idiv ecx	
Segundo número	----	XXXXXXXXX1	0040107D	89 55 F	0	mov dword ptr ss:[ebp-C],edx	
			00401080	8B 55 F	0	mov edx,dword ptr ss:[ebp-C]	
			00401083	3B 55 F	0	cmp edx,dword ptr ss:[ebp-4]	
Tercer número	----	XXXXXXXXX2	00401086	74 09	0	je randomweird-single.401091	

TOMA EL RESTO DE LA DIVISIÓN

Después hace una división en **0040107B IDIV ECX**. Podemos ver en la instrucción anterior **00401076 MOV ECX,A** que vamos a dividir por **0xA** que es **10**, y con eso el resto de la división será el número en que termina el valor generado, y aquí esta lo más importante porque va comparando nuestros restos con un contador seteado a 0. Nuestro primer resto debe ser **0**, y para que eso suceda nuestro número aleatorio debe terminar en **0**, y como el contador es **0**, se cumple la comprobación, entonces el contador aumenta en **1** y volvemos a generar otro número aleatorio que debe terminar en **1** para poder obtener el resto igual a **1**, y así de esa forma cumplir la condición y no saltar al **<CHICO MALO>**. Lo anterior se repite hasta que el contador llegue a **8**. Todo eso ocurre en un **LOOP**. Ahora puede sonar un poco enredado pero cuando ingresemos un valor y lo probemos entenderemos mejor.

Lo que acabamos de explicar en el párrafo anterior es lo que debemos programar en nuestro **KeyGen** para hallar la **Semilla inicial**, que viene siendo el **Valor correcto** que nos pide el **<RandomWeird-Single>** que ingresemos.

Ya voy conociendo cómo se comporta estas aplicaciones de consola hechas de forma sencilla, a las que no le han puesto nada malicioso para confundirnos. Podemos ir

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISEFCAB]

donde se carga el valor ingresado a puro ojo, ya de memoria. Si subimos un poco más, al inicio de ese procedimiento.

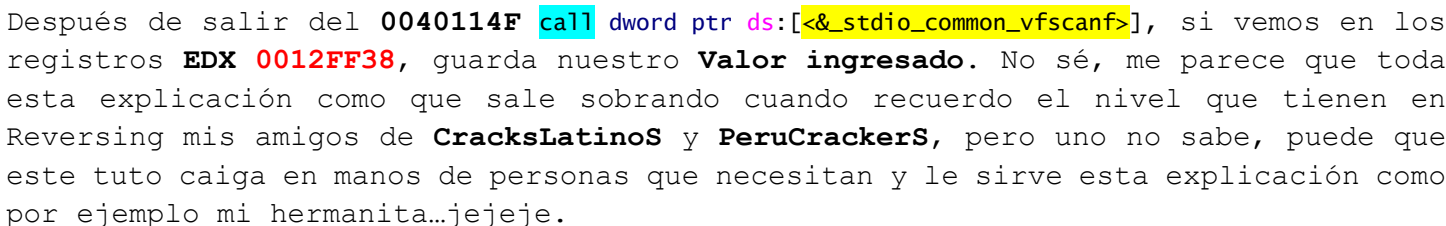
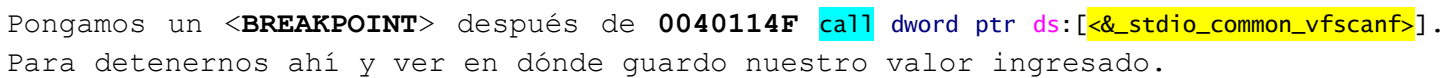
```
00401000 55 push ebp
00401001 8B mov ebp,esp
00401003 83 sub esp,10
00401006 68 push randomweird-single.403000 403000:"4c 43 47 20 52 61 6e 64 30 6d 21 20 43 4c 53!\n"
00401008 E8 call randomweird-single.401160
00401010 83 add esp,4
00401013 68 push randomweird-single.403030 403030:"\nEnter value! = "
00401018 E8 call randomweird-single.401160
0040101D 83 add esp,4
00401020 C7 mov dword ptr ss:[ebp-8],0
00401027 8D lea eax,dword ptr ss:[ebp-8] eax:BaseThreadInitThunk
0040102A 50 push eax eax:BaseThreadInitThunk
0040102B 68 push randomweird-single.403044 403044:"xi"
00401030 E8 call randomweird-single.4011A0
00401035 83 add esp,8
00401038 C7 mov dword ptr ss:[ebp-10],1
0040103F 83 cmp dword ptr ss:[ebp-8],0
00401043 74 je randomweird-single.4010BC
00401045 8B mov ecx,dword ptr ss:[ebp-8]
00401048 51 push ecx
00401049 E8 call <randomweird-single.srand>
0040104E 83 add esp,4
00401051 C7 mov dword ptr ss:[ebp-C],0
```

Podemos ver que va a mostrar o cargar texto en la consola, y sabemos que **MVC++** tiene la función **printf()** o similares. Miremos las direcciones **0040100B** y **00401018**, esas dos van al mismo **call randomweird-single.401160**, y desde ese se inicia el procedimiento para mostrar el texto en consola. Si entramos y lo revisamos.

```
00401160 55 push ebp
00401161 8B mov ebp,esp
00401163 83 sub esp,8
00401166 8D lea eax,dword ptr ss:[ebp+C]
00401169 8B mov dword ptr ss:[ebp-4],eax
0040116C 8B mov ecx,dword ptr ss:[ebp-4]
0040116F 51 push ecx
00401170 6A push 0
00401172 8B mov edx,dword ptr ss:[ebp+8]
00401175 52 push edx
00401176 6A push 1
00401178 FF call dword ptr ds:[&_acrt_iob]
0040117E 83 add esp,4
00401181 50 push eax
00401182 E8 call randomweird-single.401100
00401187 83 add esp,10
0040118A 8B mov dword ptr ss:[ebp-8],eax
0040118D C7 mov dword ptr ss:[ebp-4],0
00401194 8B mov eax,dword ptr ss:[ebp-8]
00401197 8B mov esp,ebp
00401199 5D pop ebp
0040119A CC ret
00401198 CC int3
0040119C CC int3
0040119D CC int3
0040119E CC int3
0040119F CC int3
004011A0 55 push ebp
004011A1 8B mov ebp,esp
004011A3 83 sub esp,8
004011A6 8D lea eax,dword ptr ss:[ebp+C]
004011A9 8B mov dword ptr ss:[ebp-4],eax
004011AC 8B mov ecx,dword ptr ss:[ebp-4]
004011AF 51 push ecx
004011B0 6A push 0
004011B2 8B mov edx,dword ptr ss:[ebp+8]
004011B5 52 push edx
004011B6 8B mov ebx,dword ptr ss:[ebp+14]
004011B8 50 push ebx
004011BA 8B mov ecx,dword ptr ss:[ebp+10]
004011BC 50 push ecx
004011BE 8B mov edx,dword ptr ss:[ebp+C]
004011C0 50 push edx
004011C2 8B mov eax,dword ptr ss:[ebp+8]
004011C4 50 push eax
004011C6 E8 call randomweird-single.4010E0
004011C9 8B mov ecx,dword ptr ds:[eax+4]
004011CB 50 push ecx
004011CD 8B mov edx,dword ptr ds:[eax]
004011CF 50 push edx
004011D1 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004011D3 50 push ecx
004011D5 83 add esp,18
004011D7 5D pop ebp
004011D9 CC ret
004011DB 52 push edx
004011DD 8B mov ebx,dword ptr ss:[ebp+14]
004011DF 50 push ebx
004011E1 8B mov ecx,dword ptr ss:[ebp+10]
004011E3 50 push ecx
004011E5 8B mov edx,dword ptr ss:[ebp+C]
004011E7 50 push edx
004011E9 8B mov eax,dword ptr ss:[ebp+8]
004011EB 50 push eax
004011ED E8 call randomweird-single.4010E0
004011F0 8B mov ecx,dword ptr ds:[eax+4]
004011F2 50 push ecx
004011F4 8B mov edx,dword ptr ds:[eax]
004011F6 50 push edx
004011F8 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004011FA 50 push ecx
004011FC 83 add esp,18
004011FE 5D pop ebp
00401200 CC ret
00401202 52 push edx
00401204 8B mov ebx,dword ptr ss:[ebp+14]
00401206 50 push ebx
00401208 8B mov ecx,dword ptr ss:[ebp+10]
0040120A 50 push ecx
0040120C 8B mov edx,dword ptr ss:[ebp+C]
0040120E 50 push edx
00401210 8B mov eax,dword ptr ss:[ebp+8]
00401212 50 push eax
00401214 E8 call randomweird-single.4010E0
00401217 8B mov ecx,dword ptr ds:[eax+4]
00401219 50 push ecx
0040121B 8B mov edx,dword ptr ds:[eax]
0040121D 50 push edx
0040121F 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401221 50 push ecx
00401223 83 add esp,18
00401225 5D pop ebp
00401227 CC ret
00401229 52 push edx
0040122B 8B mov ebx,dword ptr ss:[ebp+14]
0040122D 50 push ebx
0040122F 8B mov ecx,dword ptr ss:[ebp+10]
00401231 50 push ecx
00401233 8B mov edx,dword ptr ss:[ebp+C]
00401235 50 push edx
00401237 8B mov eax,dword ptr ss:[ebp+8]
00401239 50 push eax
0040123B E8 call randomweird-single.4010E0
0040123E 8B mov ecx,dword ptr ds:[eax+4]
00401240 50 push ecx
00401242 8B mov edx,dword ptr ds:[eax]
00401244 50 push edx
00401246 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401248 50 push ecx
0040124A 83 add esp,18
0040124C 5D pop ebp
0040124E CC ret
00401250 52 push edx
00401252 8B mov ebx,dword ptr ss:[ebp+14]
00401254 50 push ebx
00401256 8B mov ecx,dword ptr ss:[ebp+10]
00401258 50 push ecx
0040125A 8B mov edx,dword ptr ss:[ebp+C]
0040125C 50 push edx
0040125E 8B mov eax,dword ptr ss:[ebp+8]
00401260 50 push eax
00401262 E8 call randomweird-single.4010E0
00401265 8B mov ecx,dword ptr ds:[eax+4]
00401267 50 push ecx
00401269 8B mov edx,dword ptr ds:[eax]
0040126B 50 push edx
0040126D 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
0040126F 50 push ecx
00401271 83 add esp,18
00401273 5D pop ebp
00401275 CC ret
00401277 52 push edx
00401279 8B mov ebx,dword ptr ss:[ebp+14]
0040127B 50 push ebx
0040127D 8B mov ecx,dword ptr ss:[ebp+10]
0040127F 50 push ecx
00401281 8B mov edx,dword ptr ss:[ebp+C]
00401283 50 push edx
00401285 8B mov eax,dword ptr ss:[ebp+8]
00401287 50 push eax
00401289 E8 call randomweird-single.4010E0
0040128C 8B mov ecx,dword ptr ds:[eax+4]
0040128E 50 push ecx
00401290 8B mov edx,dword ptr ds:[eax]
00401292 50 push edx
00401294 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401296 50 push ecx
00401298 83 add esp,18
0040129A 5D pop ebp
0040129C CC ret
0040129E 52 push edx
004012A0 8B mov ebx,dword ptr ss:[ebp+14]
004012A2 50 push ebx
004012A4 8B mov ecx,dword ptr ss:[ebp+10]
004012A6 50 push ecx
004012A8 8B mov edx,dword ptr ss:[ebp+C]
004012AA 50 push edx
004012AC 8B mov eax,dword ptr ss:[ebp+8]
004012AE 50 push eax
004012B0 E8 call randomweird-single.4010E0
004012B3 8B mov ecx,dword ptr ds:[eax+4]
004012B5 50 push ecx
004012B7 8B mov edx,dword ptr ds:[eax]
004012B9 50 push edx
004012BB 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004012BD 50 push ecx
004012BF 83 add esp,18
004012C1 5D pop ebp
004012C3 CC ret
004012C5 52 push edx
004012C7 8B mov ebx,dword ptr ss:[ebp+14]
004012C9 50 push ebx
004012CB 8B mov ecx,dword ptr ss:[ebp+10]
004012CD 50 push ecx
004012CF 8B mov edx,dword ptr ss:[ebp+C]
004012D1 50 push edx
004012D3 8B mov eax,dword ptr ss:[ebp+8]
004012D5 50 push eax
004012D7 E8 call randomweird-single.4010E0
004012DA 8B mov ecx,dword ptr ds:[eax+4]
004012DC 50 push ecx
004012DE 8B mov edx,dword ptr ds:[eax]
004012E0 50 push edx
004012E2 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004012E4 50 push ecx
004012E6 83 add esp,18
004012E8 5D pop ebp
004012EA CC ret
004012EC 52 push edx
004012EE 8B mov ebx,dword ptr ss:[ebp+14]
004012F0 50 push ebx
004012F2 8B mov ecx,dword ptr ss:[ebp+10]
004012F4 50 push ecx
004012F6 8B mov edx,dword ptr ss:[ebp+C]
004012F8 50 push edx
004012FA 8B mov eax,dword ptr ss:[ebp+8]
004012FC 50 push eax
004012FE E8 call randomweird-single.4010E0
00401301 8B mov ecx,dword ptr ds:[eax+4]
00401303 50 push ecx
00401305 8B mov edx,dword ptr ds:[eax]
00401307 50 push edx
00401309 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
0040130B 50 push ecx
0040130D 83 add esp,18
0040130F 5D pop ebp
00401311 CC ret
00401313 52 push edx
00401315 8B mov ebx,dword ptr ss:[ebp+14]
00401317 50 push ebx
00401319 8B mov ecx,dword ptr ss:[ebp+10]
0040131B 50 push ecx
0040131D 8B mov edx,dword ptr ss:[ebp+C]
0040131F 50 push edx
00401321 8B mov eax,dword ptr ss:[ebp+8]
00401323 50 push eax
00401325 E8 call randomweird-single.4010E0
00401328 8B mov ecx,dword ptr ds:[eax+4]
0040132A 50 push ecx
0040132C 8B mov edx,dword ptr ds:[eax]
0040132E 50 push edx
00401330 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401332 50 push ecx
00401334 83 add esp,18
00401336 5D pop ebp
00401338 CC ret
0040133A 52 push edx
0040133C 8B mov ebx,dword ptr ss:[ebp+14]
0040133E 50 push ebx
00401340 8B mov ecx,dword ptr ss:[ebp+10]
00401342 50 push ecx
00401344 8B mov edx,dword ptr ss:[ebp+C]
00401346 50 push edx
00401348 8B mov eax,dword ptr ss:[ebp+8]
0040134A 50 push eax
0040134C E8 call randomweird-single.4010E0
0040134F 8B mov ecx,dword ptr ds:[eax+4]
00401351 50 push ecx
00401353 8B mov edx,dword ptr ds:[eax]
00401355 50 push edx
00401357 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401359 50 push ecx
0040135B 83 add esp,18
0040135D 5D pop ebp
0040135F CC ret
00401361 52 push edx
00401363 8B mov ebx,dword ptr ss:[ebp+14]
00401365 50 push ebx
00401367 8B mov ecx,dword ptr ss:[ebp+10]
00401369 50 push ecx
0040136B 8B mov edx,dword ptr ss:[ebp+C]
0040136D 50 push edx
0040136F 8B mov eax,dword ptr ss:[ebp+8]
00401371 50 push eax
00401373 E8 call randomweird-single.4010E0
00401376 8B mov ecx,dword ptr ds:[eax+4]
00401378 50 push ecx
0040137A 8B mov edx,dword ptr ds:[eax]
0040137C 50 push edx
0040137E 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401380 50 push ecx
00401382 83 add esp,18
00401384 5D pop ebp
00401386 CC ret
00401388 52 push edx
0040138A 8B mov ebx,dword ptr ss:[ebp+14]
0040138C 50 push ebx
0040138E 8B mov ecx,dword ptr ss:[ebp+10]
00401390 50 push ecx
00401392 8B mov edx,dword ptr ss:[ebp+C]
00401394 50 push edx
00401396 8B mov eax,dword ptr ss:[ebp+8]
00401398 50 push eax
0040139A E8 call randomweird-single.4010E0
0040139D 8B mov ecx,dword ptr ds:[eax+4]
0040139F 50 push ecx
004013A1 8B mov edx,dword ptr ds:[eax]
004013A3 50 push edx
004013A5 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004013A7 50 push ecx
004013A9 83 add esp,18
004013AB 5D pop ebp
004013AD CC ret
004013AF 52 push edx
004013B1 8B mov ebx,dword ptr ss:[ebp+14]
004013B3 50 push ebx
004013B5 8B mov ecx,dword ptr ss:[ebp+10]
004013B7 50 push ecx
004013B9 8B mov edx,dword ptr ss:[ebp+C]
004013BB 50 push edx
004013BD 8B mov eax,dword ptr ss:[ebp+8]
004013BF 50 push eax
004013C1 E8 call randomweird-single.4010E0
004013C4 8B mov ecx,dword ptr ds:[eax+4]
004013C6 50 push ecx
004013C8 8B mov edx,dword ptr ds:[eax]
004013CA 50 push edx
004013CC 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004013CE 50 push ecx
004013D0 83 add esp,18
004013D2 5D pop ebp
004013D4 CC ret
004013D6 52 push edx
004013D8 8B mov ebx,dword ptr ss:[ebp+14]
004013DA 50 push ebx
004013DC 8B mov ecx,dword ptr ss:[ebp+10]
004013DE 50 push ecx
004013E0 8B mov edx,dword ptr ss:[ebp+C]
004013E2 50 push edx
004013E4 8B mov eax,dword ptr ss:[ebp+8]
004013E6 50 push eax
004013E8 E8 call randomweird-single.4010E0
004013EB 8B mov ecx,dword ptr ds:[eax+4]
004013ED 50 push ecx
004013EF 8B mov edx,dword ptr ds:[eax]
004013F1 50 push edx
004013F3 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004013F5 50 push ecx
004013F7 83 add esp,18
004013F9 5D pop ebp
004013FB CC ret
004013FD 52 push edx
004013FF 8B mov ebx,dword ptr ss:[ebp+14]
00401401 50 push ebx
00401403 8B mov ecx,dword ptr ss:[ebp+10]
00401405 50 push ecx
00401407 8B mov edx,dword ptr ss:[ebp+C]
00401409 50 push edx
0040140B 8B mov eax,dword ptr ss:[ebp+8]
0040140D 50 push eax
0040140F E8 call randomweird-single.4010E0
00401412 8B mov ecx,dword ptr ds:[eax+4]
00401414 50 push ecx
00401416 8B mov edx,dword ptr ds:[eax]
00401418 50 push edx
0040141A 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
0040141C 50 push ecx
0040141E 83 add esp,18
00401420 5D pop ebp
00401422 CC ret
00401424 52 push edx
00401426 8B mov ebx,dword ptr ss:[ebp+14]
00401428 50 push ebx
0040142A 8B mov ecx,dword ptr ss:[ebp+10]
0040142C 50 push ecx
0040142E 8B mov edx,dword ptr ss:[ebp+C]
00401430 50 push edx
00401432 8B mov eax,dword ptr ss:[ebp+8]
00401434 50 push eax
00401436 E8 call randomweird-single.4010E0
00401439 8B mov ecx,dword ptr ds:[eax+4]
0040143B 50 push ecx
0040143D 8B mov edx,dword ptr ds:[eax]
0040143F 50 push edx
00401441 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401443 50 push ecx
00401445 83 add esp,18
00401447 5D pop ebp
00401449 CC ret
0040144B 52 push edx
0040144D 8B mov ebx,dword ptr ss:[ebp+14]
0040144F 50 push ebx
00401451 8B mov ecx,dword ptr ss:[ebp+10]
00401453 50 push ecx
00401455 8B mov edx,dword ptr ss:[ebp+C]
00401457 50 push edx
00401459 8B mov eax,dword ptr ss:[ebp+8]
0040145B 50 push eax
0040145D E8 call randomweird-single.4010E0
00401460 8B mov ecx,dword ptr ds:[eax+4]
00401462 50 push ecx
00401464 8B mov edx,dword ptr ds:[eax]
00401466 50 push edx
00401468 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
0040146A 50 push ecx
0040146C 83 add esp,18
0040146E 5D pop ebp
00401470 CC ret
00401472 52 push edx
00401474 8B mov ebx,dword ptr ss:[ebp+14]
00401476 50 push ebx
00401478 8B mov ecx,dword ptr ss:[ebp+10]
0040147A 50 push ecx
0040147C 8B mov edx,dword ptr ss:[ebp+C]
0040147E 50 push edx
00401480 8B mov eax,dword ptr ss:[ebp+8]
00401482 50 push eax
00401484 E8 call randomweird-single.4010E0
00401487 8B mov ecx,dword ptr ds:[eax+4]
00401489 50 push ecx
0040148B 8B mov edx,dword ptr ds:[eax]
0040148D 50 push edx
0040148F 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401491 50 push ecx
00401493 83 add esp,18
00401495 5D pop ebp
00401497 CC ret
00401499 52 push edx
0040149B 8B mov ebx,dword ptr ss:[ebp+14]
0040149D 50 push ebx
0040149F 8B mov ecx,dword ptr ss:[ebp+10]
004014A1 50 push ecx
004014A3 8B mov edx,dword ptr ss:[ebp+C]
004014A5 50 push edx
004014A7 8B mov eax,dword ptr ss:[ebp+8]
004014A9 50 push eax
004014AB E8 call randomweird-single.4010E0
004014AE 8B mov ecx,dword ptr ds:[eax+4]
004014B0 50 push ecx
004014B2 8B mov edx,dword ptr ds:[eax]
004014B4 50 push edx
004014B6 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004014B8 50 push ecx
004014BA 83 add esp,18
004014BC 5D pop ebp
004014BE CC ret
004014C0 52 push edx
004014C2 8B mov ebx,dword ptr ss:[ebp+14]
004014C4 50 push ebx
004014C6 8B mov ecx,dword ptr ss:[ebp+10]
004014C8 50 push ecx
004014CA 8B mov edx,dword ptr ss:[ebp+C]
004014CC 50 push edx
004014CE 8B mov eax,dword ptr ss:[ebp+8]
004014D0 50 push eax
004014D2 E8 call randomweird-single.4010E0
004014D5 8B mov ecx,dword ptr ds:[eax+4]
004014D7 50 push ecx
004014D9 8B mov edx,dword ptr ds:[eax]
004014DB 50 push edx
004014DD 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004014DF 50 push ecx
004014E1 83 add esp,18
004014E3 5D pop ebp
004014E5 CC ret
004014E7 52 push edx
004014E9 8B mov ebx,dword ptr ss:[ebp+14]
004014EB 50 push ebx
004014ED 8B mov ecx,dword ptr ss:[ebp+10]
004014EF 50 push ecx
004014F1 8B mov edx,dword ptr ss:[ebp+C]
004014F3 50 push edx
004014F5 8B mov eax,dword ptr ss:[ebp+8]
004014F7 50 push eax
004014F9 E8 call randomweird-single.4010E0
004014FC 8B mov ecx,dword ptr ds:[eax+4]
004014FE 50 push ecx
00401500 8B mov edx,dword ptr ds:[eax]
00401502 50 push edx
00401504 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401506 50 push ecx
00401508 83 add esp,18
0040150A 5D pop ebp
0040150C CC ret
0040150E 52 push edx
00401510 8B mov ebx,dword ptr ss:[ebp+14]
00401512 50 push ebx
00401514 8B mov ecx,dword ptr ss:[ebp+10]
00401516 50 push ecx
00401518 8B mov edx,dword ptr ss:[ebp+C]
0040151A 50 push edx
0040151C 8B mov eax,dword ptr ss:[ebp+8]
0040151E 50 push eax
00401520 E8 call randomweird-single.4010E0
00401523 8B mov ecx,dword ptr ds:[eax+4]
00401525 50 push ecx
00401527 8B mov edx,dword ptr ds:[eax]
00401529 50 push edx
0040152B 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
0040152D 50 push ecx
0040152F 83 add esp,18
00401531 5D pop ebp
00401533 CC ret
00401537 52 push edx
00401539 8B mov ebx,dword ptr ss:[ebp+14]
0040153B 50 push ebx
0040153D 8B mov ecx,dword ptr ss:[ebp+10]
0040153F 50 push ecx
00401541 8B mov edx,dword ptr ss:[ebp+C]
00401543 50 push edx
00401545 8B mov eax,dword ptr ss:[ebp+8]
00401547 50 push eax
00401549 E8 call randomweird-single.4010E0
0040154C 8B mov ecx,dword ptr ds:[eax+4]
0040154E 50 push ecx
00401550 8B mov edx,dword ptr ds:[eax]
00401552 50 push edx
00401554 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401556 50 push ecx
00401558 83 add esp,18
0040155A 5D pop ebp
0040155C CC ret
00401560 52 push edx
00401562 8B mov ebx,dword ptr ss:[ebp+14]
00401564 50 push ebx
00401566 8B mov ecx,dword ptr ss:[ebp+10]
00401568 50 push ecx
0040156A 8B mov edx,dword ptr ss:[ebp+C]
0040156C 50 push edx
0040156E 8B mov eax,dword ptr ss:[ebp+8]
00401570 50 push eax
00401572 E8 call randomweird-single.4010E0
00401575 8B mov ecx,dword ptr ds:[eax+4]
00401577 50 push ecx
00401579 8B mov edx,dword ptr ds:[eax]
0040157B 50 push edx
0040157D 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
0040157F 50 push ecx
00401581 83 add esp,18
00401583 5D pop ebp
00401585 CC ret
00401589 52 push edx
0040158B 8B mov ebx,dword ptr ss:[ebp+14]
0040158D 50 push ebx
0040158F 8B mov ecx,dword ptr ss:[ebp+10]
00401591 50 push ecx
00401593 8B mov edx,dword ptr ss:[ebp+C]
00401595 50 push edx
00401597 8B mov eax,dword ptr ss:[ebp+8]
00401599 50 push eax
0040159B E8 call randomweird-single.4010E0
0040159E 8B mov ecx,dword ptr ds:[eax+4]
004015A0 50 push ecx
004015A2 8B mov edx,dword ptr ds:[eax]
004015A4 50 push edx
004015A6 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004015A8 50 push ecx
004015AA 83 add esp,18
004015AC 5D pop ebp
004015AE CC ret
004015B3 52 push edx
004015B5 8B mov ebx,dword ptr ss:[ebp+14]
004015B7 50 push ebx
004015B9 8B mov ecx,dword ptr ss:[ebp+10]
004015BB 50 push ecx
004015BD 8B mov edx,dword ptr ss:[ebp+C]
004015BF 50 push edx
004015C1 8B mov eax,dword ptr ss:[ebp+8]
004015C3 50 push eax
004015C5 E8 call randomweird-single.4010E0
004015C8 8B mov ecx,dword ptr ds:[eax+4]
004015CA 50 push ecx
004015CC 8B mov edx,dword ptr ds:[eax]
004015CE 50 push edx
004015D0 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004015D2 50 push ecx
004015D4 83 add esp,18
004015D6 5D pop ebp
004015D8 CC ret
004015DD 52 push edx
004015DF 8B mov ebx,dword ptr ss:[ebp+14]
004015E1 50 push ebx
004015E3 8B mov ecx,dword ptr ss:[ebp+10]
004015E5 50 push ecx
004015E7 8B mov edx,dword ptr ss:[ebp+C]
004015E9 50 push edx
004015EB 8B mov eax,dword ptr ss:[ebp+8]
004015ED 50 push eax
004015EF E8 call randomweird-single.4010E0
004015F2 8B mov ecx,dword ptr ds:[eax+4]
004015F4 50 push ecx
004015F6 8B mov edx,dword ptr ds:[eax]
004015F8 50 push edx
004015FA 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004015FC 50 push ecx
004015FE 83 add esp,18
00401600 5D pop ebp
00401602 CC ret
00401608 52 push edx
0040160A 8B mov ebx,dword ptr ss:[ebp+14]
0040160C 50 push ebx
0040160E 8B mov ecx,dword ptr ss:[ebp+10]
00401610 50 push ecx
00401612 8B mov edx,dword ptr ss:[ebp+C]
00401614 50 push edx
00401616 8B mov eax,dword ptr ss:[ebp+8]
00401618 50 push eax
0040161A E8 call randomweird-single.4010E0
0040161D 8B mov ecx,dword ptr ds:[eax+4]
0040161F 50 push ecx
00401621 8B mov edx,dword ptr ds:[eax]
00401623 50 push edx
00401625 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401627 50 push ecx
00401629 83 add esp,18
0040162B 5D pop ebp
0040162D CC ret
00401633 52 push edx
00401635 8B mov ebx,dword ptr ss:[ebp+14]
00401637 50 push ebx
00401639 8B mov ecx,dword ptr ss:[ebp+10]
0040163B 50 push ecx
0040163D 8B mov edx,dword ptr ss:[ebp+C]
0040163F 50 push edx
00401641 8B mov eax,dword ptr ss:[ebp+8]
00401643 50 push eax
00401645 E8 call randomweird-single.4010E0
00401648 8B mov ecx,dword ptr ds:[eax+4]
0040164A 50 push ecx
0040164C 8B mov edx,dword ptr ds:[eax]
0040164E 50 push edx
00401650 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
00401652 50 push ecx
00401654 83 add esp,18
00401656 5D pop ebp
00401658 CC ret
0040165F 52 push edx
00401661 8B mov ebx,dword ptr ss:[ebp+14]
00401663 50 push ebx
00401665 8B mov ecx,dword ptr ss:[ebp+10]
00401667 50 push ecx
00401669 8B mov edx,dword ptr ss:[ebp+C]
0040166B 50 push edx
0040166D 8B mov eax,dword ptr ss:[ebp+8]
0040166F 50 push eax
00401671 E8 call randomweird-single.4010E0
00401674 8B mov ecx,dword ptr ds:[eax+4]
00401676 50 push ecx
00401678 8B mov edx,dword ptr ds:[eax]
0040167A 50 push edx
0040167C 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
0040167E 50 push ecx
00401680 83 add esp,18
00401682 5D pop ebp
00401684 CC ret
00401689 52 push edx
0040168B 8B mov ebx,dword ptr ss:[ebp+14]
0040168D 50 push ebx
0040168F 8B mov ecx,dword ptr ss:[ebp+10]
00401691 50 push ecx
00401693 8B mov edx,dword ptr ss:[ebp+C]
00401695 50 push edx
00401697 8B mov eax,dword ptr ss:[ebp+8]
00401699 50 push eax
0040169B E8 call randomweird-single.4010E0
0040169E 8B mov ecx,dword ptr ds:[eax+4]
004016A0 50 push ecx
004016A2 8B mov edx,dword ptr ds:[eax]
004016A4 50 push edx
004016A6 8B mov ecx,dword ptr ds:[&_stdio_common_vfprintf]
004016A8 50 push ecx
004016AA 83 add esp,18
004016AC 5D pop ebp
004016AE CC ret
004016B3 52 push edx
004016B5 8B mov ebx
```

Entrando a ese **CALL** que es prácticamente el mismo procedimiento, solo que utilizará la función **scanf()**.

Entrando a ese **CALL** que es prácticamente el mismo procedimiento, solo que utilizará la función **scanf()**.



Continuemos traceando con <F8> hasta salir de todos esos procedimientos y llegar a la <PARTE CALIENTE> y detenemos el trazo en 0040103E.

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]

00401030	E8	call randomweird-single.4011A0	
00401035	83	add esp,8	
00401038	C7	mov dword ptr ss:[ebp-10],1	
0040103F	83	cmp dword ptr ss:[ebp-8],0	
00401043	74	je randomweird-single.4010BC	
00401045	8B	mov ecx,dword ptr ss:[ebp-8]	
00401048	51	push ecx	
00401049	E8	call <randomweird-single.004010BC 403060:"Only numbers!"	
0040104E	83	add esp,4	
00401051	C7	mov dword ptr ss:[ebp-10],1	
00401058	C7	mov dword ptr ss:[ebp-8],0	
0040105F	EB	jmp randomweird-single.4010BC	
00401061	8B	mov ecx,dword ptr ss:[ebp-8]	
00401064	83	add edx,1	
00401067	89	mov dword ptr ss:[ebp-4],ecx	
0040106A	83	cmp dword ptr ss:[ebp-4],0	

Si ingresamos un valor que no sea un número nos manda para afuera, así que solo números, no olvidarlo. Revisemos las siguientes instrucciones.

00401045 mov ecx,dword ptr ss:[ebp-8] Mueve a ECX el valor ingresado.
 00401048 push ecx Mueve al STACK el valor ingresado.
 00401049 call <randomweird-single.srand> Guarda valor ingresado como nuestra **Semilla inicial**.

El 00401049 call <randomweird-single.srand> no hace otra cosa que guardar nuestro valor ingresado (**Semilla inicial**) en memoria para ser utilizada cuando se ejecute la función **rand()**. Recordar toda la teoría antes planteada. Pasemos con <F8> el call <randomweird-single.srand> y podemos observar que en **EAX 00235AA0** queda una dirección que se comporta como un **Offset** o **Puntero** desde el cual permite llegar a nuestra **Semilla inicial**.

00401043	74	je randomweird-single.4010BC	
00401045	8B	mov ecx,dword ptr ss:[ebp-8]	
00401048	51	push ecx	
00401049	E8	call <randomweird-single.srand>	
0040104E	83	add esp,4	
00401051	C7	mov dword ptr ss:[ebp-10],1	

Address	Hex	ASCII
00235AA0	88 6D 92 6E	.m.n.
00235AB0	00 00 00 00
00235AC0	00 00 00 00
00235AD0	00 00 00 00

Nuestra **Semilla inicial** está en [00235AA0+0x18] o [EAX+0x18]. Recordemos esto porque así es como la función **rand()** coge nuestra **Semilla inicial**. Lleguemos a 00101070 call <randomweird-single.rand> y entremos con <F7> y sigamos hasta donde se ejecuta instrucción por instrucción función **rand()**.

$$x_{i+1} = (ax_i + c) \bmod(m)$$

6E95D520	E8	call ucrtbase.6E975E32	
6E95D525	69	imul ecx,dword ptr ds:[eax+18],343FD	
6E95D52C	81	add ecx,269EC3	
6E95D532	89	mov dword ptr ds:[eax+18],ecx	
6E95D535	C1	shr ecx,10	
6E95D538	81	and ecx,7FFF	
6E95D53E	8B	mov eax,ecx	
6E95D540	C3	ret	

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]

Algo relevante, es que lo primero que se hace en un `call ucrtbase.6C755E32` y es solo para obtener nuestro **Offset** en **EAX** para cargar nuestra **Semilla inicial**. Sigamos traceando con <F8> para pasar ese **CALL** y lleguemos hasta `mov dword ptr ds:[eax+18],ecx`.

6C73D520	E8	call ucrtbase.6C755E32	rand
6C73D525	69	imul ecx,dword ptr ds:[eax+18],343FD	
6C73D52C	81	add ecx,269EC3	
6C73D532	89	mov dword ptr ds:[eax+18],ecx	
6C73D535	C1	shr ecx,10	
6C73D538	81	and ecx,7FFF	
6C73D53E	8B	mov eax,ecx	
6C73D540	C3	ret	

ANTES DE GUARDAR LA NUEVA SEMILLA

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	
Address	Hex				ASC
00235AA0	98 6D 92 6E	00 00 00 00	00 00 00 00	00 00 00 00	.m.
00235AB0	00 00 00 00	00 00 00 00	0A 00 00 00	00 00 00 00	...
00235AC0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...
00235AD0	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...

SE GUARDA LA NUEVA SEMILLA

Dump 1	Dump 2	Dump 3	Dump 4	Dump 5	
Address	Hex				ASC
00215AC8	98 6D 70 6C	00 00 00 00	00 00 00 00	00 00 00 00	.mp
00215AD8	00 00 00 00	00 00 00 00	A5 46 47 00	00 00 00 00	...
00215AE8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...
00215AF8	00 00 00 00	00 00 00 00	00 00 00 00	00 00 00 00	...

Podemos ver que se origina una **Nueva semilla** y con esa se hallará un nuevo número aleatorio cuando se llame de nuevo la función `rand()`, y así sucesivamente. Pues esa es la **Semilla** la que debemos hallar y que sabemos nos debe dar números que nos finalicen desde 0 hasta 7. Según lo que pude leer y pude entender es que no hay un formula fija para hallar una **Semilla inicial** pero podemos programar un procedimiento que nos halle una mediante fuerza bruta, conociendo las condiciones de inicio. Tracemos hasta el `RET`.

Graph	Log	Notes	Breakpoints	Memory Map	Call Stack	SEH
6C73D520	E8	call ucrtbase.6C755E32	rand			
6C73D525	69	imul ecx,dword ptr ds:[eax+18],343FD				
6C73D52C	81	add ecx,269EC3				
6C73D532	89	mov dword ptr ds:[eax+18],ecx				
6C73D535	C1	shr ecx,10				
6C73D538	81	and ecx,7FFF				
6C73D53E	8B	mov eax,ecx				
6C73D540	C3	ret				

Hide FPU	
EAX	00000047
EBX	7FFDF000
ECX	00000047
EDX	00000000
EBP	0012FF40
ESI	00000000

Listo, ya tenemos nuestro primer número aleatorio en **EAX 00000047**, $0x47=71$. Como vemos el número termina en 1 y ya con eso sabemos que no sirve porque $71 \bmod 10 = 1$ y nosotros sabemos que el primer resto debe ser 0.

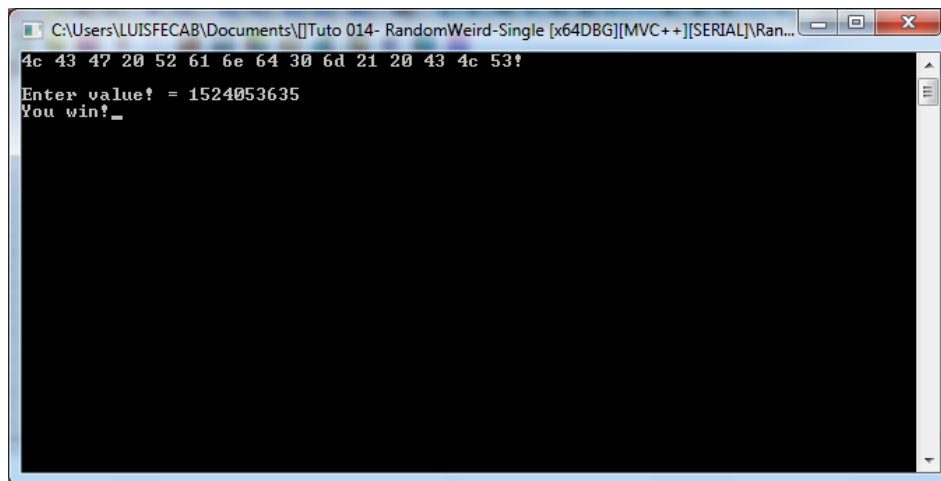
Supongo que deben haber diferentes formas de programar para hallar la **Semilla Inicial**. Yo opté por iniciar al cálculo siempre con **Semilla inicial=1** y si la **Semilla inicial** no me sirve, entonces se tome como **Nueva Semilla** la que se origina en la función `rand()`. Esto tiene una gran desventaja, y es que mi **KeyGen** es poco flexible ya que siempre tendré el mismo resultado al ejecutarlo y si sigo calculando nuevos valores, serán en el mismo sentido porque no he permitido ingresar una **Semilla inicial** si no que la va aumentando en 1.

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]

Ya hemos analizado todo y sabemos las condiciones iniciales de cómo se generan los números aleatorios en **MVC++** y también sabemos la condición del **Crackme** de **nextco** para poderlo vencer. Queda solo hacer mi **KeyGen** y que me halle un valor correcto.

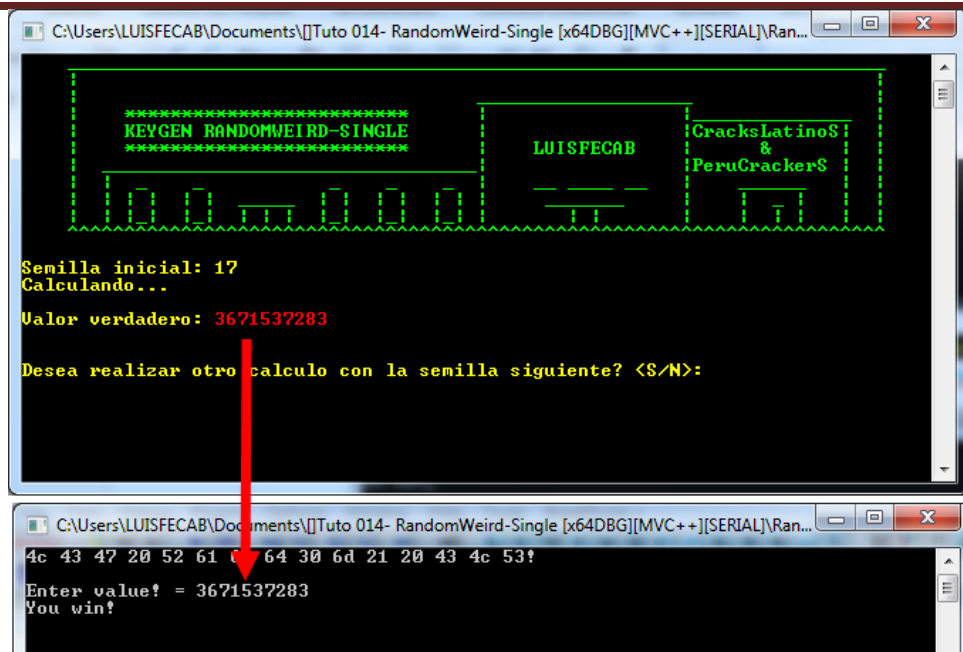


Como ven mi **Semilla inicial: 1**, y por eso siempre tendré **Valor Verdadero: 1524053635**, pero lo importante era hallar un valor que venciera el **Crackme** **<RandomWeird-Single>**.



Perfecto, ya pudimos hallar un valor correcto. Si quisiera calcular un **Valor verdadero** para **Semilla inicial: 17** debo realizar más cálculos hasta llegar a 17. No es cómodo pero sirve para su cometido. Otra cosa es que se nos repetirán Valores verdaderos, no es que hallemos un **Valor verdadero** por cada **Semilla inicial**; recordemos que es aquel número que nos permite tener números finalizados desde 0 a 7 y como es por **FUERZA BRUTA** pues se repetirán.

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]

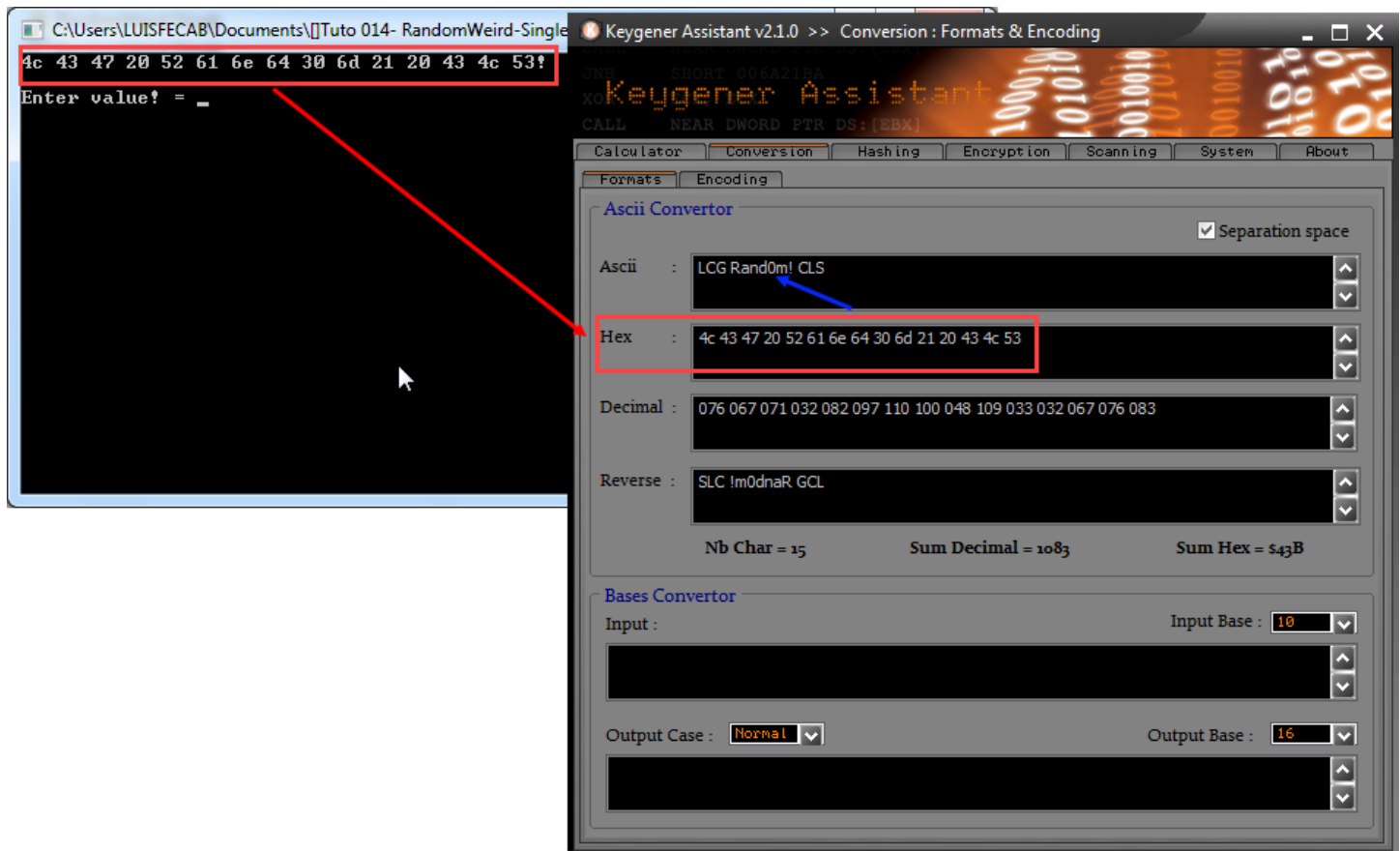


PARA TERMINAR

Ya vamos terminando, creo que podemos vencer el **Crackme** original <RandomWeird> con todo lo aprendido en este tutorial pero que lo enfrentaré después, porque con este he tenido suficiente, créanme cuando le digo eso. Cuando hagamos ese **Crackme** también trataremos de mejorar el **KeyGen** para que podamos ingresar una **Semilla inicial**, o un rango, o talvez otras opciones; luego veremos eso.

Este **Crackme** tiene lo suyo, recuerdo que **nextco** me dijo - *Has este reto, te divertirás haciéndolo* -. Yo venía muy vanidoso y creído porque había hecho otros y pensé que iba ser pan cómodo y resultó que el pan era yo, qué aterrizada me pegó este **Crackme** y me hizo recordar que en esto del **Reversing** uno debe tener humildad como muchas veces lo ha dicho el **Maestro Ricardo**.

Pistas que tiene el **crackme** que descubrí desde un inicio pero en mi ignorancia no supe aprovechar. Recordemos que se cargan unos valores hexadecimales que fijo traen un mensaje; para saber qué decía hice uso de la tool <Keygener Assistant v2.1.0>.



"LCG Rand0m! CLS", más claro no canta un gallo, **Lineal Congruential Generator (LCG)**.

Si hay algo incorrecto que he escrito o alguna observación para mejorar lo expuesto aquí, son bienvenidas porque entre más aprendemos, mucho mejor para mí y me ayudará

RandomWeird-Single [x64DBG][MVC++][SERIAL][LUISFECAB]

a hacer mejores tutoriales que contengan mejor información que en algún momento servirá a otros.

Amigos, esto es todo, "*hasta aquí me trajo el río*". Saludos a todos los que en cierta forma me conocen y para los que me leen y no me han tratado también un saludo afectuoso.

Y qué dijeron que no me despido de mis grupos, no señor, no puedo irme sin despedirme de mi **PeruCrackers** y **CracksLatinos**.

@LUISFECAB