



CRACKME MATEMATICA BY JHON

Crackme Solution by ThunderClS

Abstract

Un sencillo escrito de cómo resolver este crackme propuesto en la web de apuromafo

ThunderClS
reverse0de.wordpress.com

INFORMACIÓN:

Según los datos del autor tenemos la siguiente:

<http://crackmes.apuromafo.net/index.php?topic=90.0>

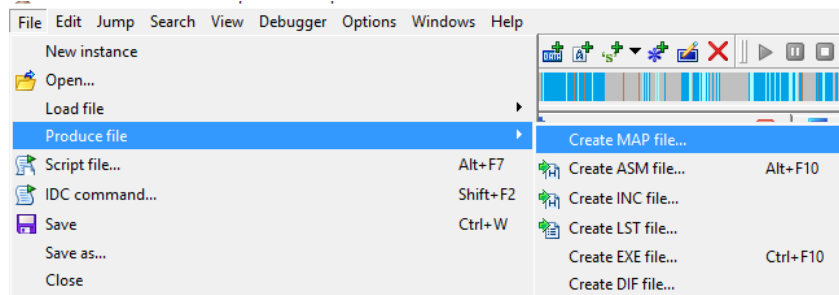
- 1) Dificultad : 2 Facil
- 2) Lenguaje :Borland Delphi
- 3)Platforma : Windows
- 4) Version del Sistema Operativo :Todos
- 5)Packer / Protector : Ninguno
- 6) Autor /Fuente: Jhon(Jhonjhon_123)

Objetivos:

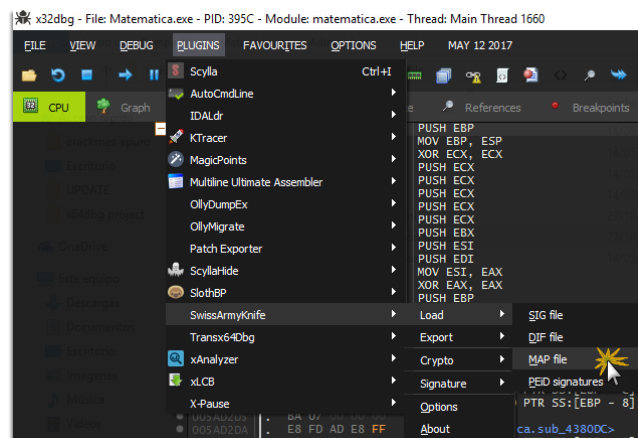
1. Encontrar la ecuación matemática que comprueba el serial.
2. Resolver la ecuación.
3. Resolver la cascara al ingresar el serial.
4. Listo!

ANÁLISIS:

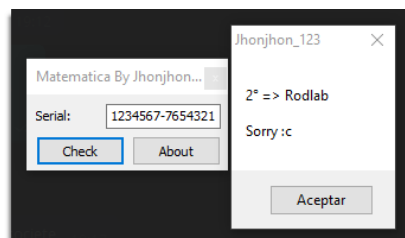
Como ya sabemos que estamos frente a un Borland Delphi, vamos a exportar un poco de símbolos y así no perdernos entre tantas llamadas de la VCL. Abrimos el crackme en IDA y exportamos un .map



Ahora nos vamos al depurador que deseemos e importamos el .map para tener todos los símbolos y demás en su lugar, yo estaré usando en este escrito el x64dbg, así que me voy al plugin "SwissArmyKnife".



Una vez todo se ha importado ejecutamos el crackme, llenamos el numero de serie con cualquier combinación numérica que deseemos y damos al botón "Check", entonces nos saca un mensaje.



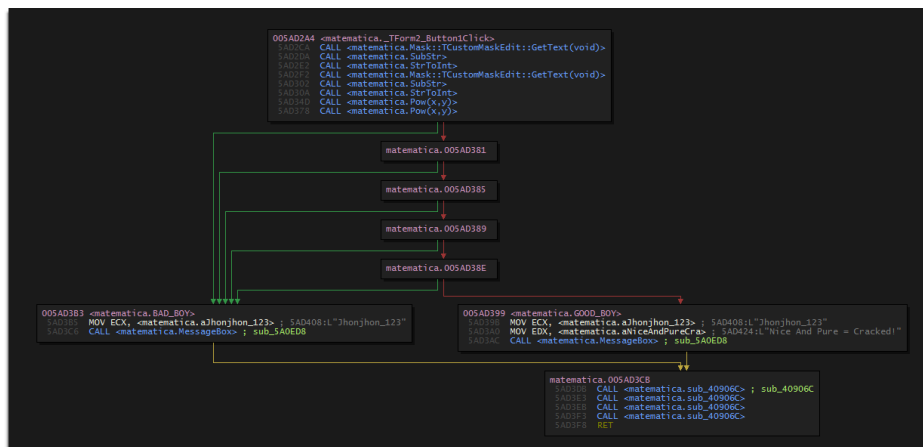
Sin aceptar el mensaje pausamos el debugger y nos vamos a la pestaña **Call Stack** ahí localizamos solo los retornos al "user code" y para ir más rápido nos vamos al tercero que es el que más nos acerca al propio código del crackme que estamos buscando.

To	Address	From	Size	Comment	Party
7429C1B0	001F004	7461254C	30	win32.7461254C	System
7429C1D7	001F004	7429C1B0	CC	user32.7429C1B0	System
74308E3B	001F100	743082D7	164	user32.743082D7	System
74308BAE	001F264	74308E38	80	user32.74308E38	System
74308BA8	001F264	74308BAE	20	user32.74308BAE	System
74308E98	001F304	74308BA8	1C	user32.74308BA8	System
005A0FD6	001F320	74308E98	88	user32.74308E98	User
005A0F3B	001F3A8	005A0FD6	38	matematica.005A0FD6	User
005A1F71	001F3E0	005A0F3B	144	matematica.loc.5AD9CB	User
005199AE	001F524	00515474	4C	matematica.loc.515474	User
00528E79	001F570	005199AE	2C	matematica.005199AE	User
00519B14	001F59C	00528E79	158	matematica.loc.528E79	User
005199AE	001F6F4	00519B14	4C	matematica.00519B14	User
005977E2	001F740	005199AE	2C	matematica.005199AE	User
00597EF	001F766	005977E2	30	matematica.loc.5977E2	User
004C0C5E	001F79C	005197EF	18	matematica.005197EF	User

Al dar doble clic caemos en la única función que nos interesa de este crackme.

Address	Disassembly	Comment
005AD39E	OP AF FF	INUL EDI, EAX
005AD39F	81 FF 00 01 00 00	CMP EDI, 100
005AD3A7	75 1A	JNE <i>matematica.SAD3B3</i>
005AD3A9	66 0A	PUSH 0
005AD3AB	B9 08 D4 5A 00	MOV ECK, <matematica.a3hjhjhjh_123>
005AD3AD	BA D4 5A 00	MOV EDI, <matematica.a1nIceAndPureCrack>
005AD3AE	A1 DC AD 5B 00	MOV EAX, DWORD PTR DS:[off_SBADDC]
005AD3AF	8B 00	MOV EAX, DWORD PTR DS:[EAX]
005AD3B0	E8 27 3B FF FF	CALL <matematica.sub_5A0ED8>
005AD3B1	EB 18	JMP <i>matematica.SAD3CB</i>
005AD3B3	66 0A	PUSH 0
005AD3B5	B9 08 D4 5A 00	MOV ECK, <matematica.a3hjhjhjh_123>
005AD3B7	BA 58 D4 5A 00	MOV EDI, <matematica.a2>
005AD3B9	A1 DC AD 5B 00	MOV EAX, DWORD PTR DS:[off_SBADDC]
005AD3BA	8B 00	MOV EAX, DWORD PTR DS:[EAX]
005AD3BB	E8 0D 3B FF FF	CALL <matematica.sub_5A0ED8>
005AD3BC	33 C0	XOR EAX, EAX
005AD3BD	5A	POP EDI
005AD3BE	59	POP ECK
005AD3BF	59	POP ECK
005AD3D0	64 89 10	MOV DWORD PTR FS:[EAX], EDI

Podemos ver que tenemos los dos mensajes a la vista, por lo que podemos ir al principio de la función y empezar a analizar a partir de ahí hasta abajo, yo iré poniendo labels para tener las próximas capturas con una mejor vista grafica resumen de lo que sucede aquí, veamos. Presionamos "G" para cambiar a vista gráfica y luego "U" para cambiar a vista resumen, tendríamos algo como esto frente a nosotros



Como lo vemos aquí el código no es muy complicado o enrevesado en sí. Va tomando el serial a partes (sabiendo de antemano que el serial es de la forma **XXXXXXXX-XXXXXXXX** suponemos que tomara una parte a la izquierda del guion y luego la otra), y también podemos notar que pasara ambos valores de cadenas de texto a enteros con la función de la VCL de Borland **“StrToInt”**, más abajo vemos algunos **“Pow”**, pero ahora veremos en más detalle de que se trata. Notamos además para terminar cuatro condiciones que nos lanzan al mensaje de éxito o de error, sencillo hasta ahora. Entonces veamos ahora más de cerca el código para ir sacando las cosas en claro. Ponemos BP en el inicio de la función, reiniciamos el crackme, ponemos como serial de prueba **“1234567-7654321”**, aceptamos y ahora vamos a ir traceando para ver cómo se van comportando las cosas.

```

005AD2C4 8B 86 A0 03 00 00 MOV EAX, DWORD PTR DS:[ESI + 3A0]
005AD2CA E8 49 E0 FF FF CALL <matematica.Mask::TCustomMaskEdit::GetText(void)>
005AD2CF 8B 45 F4 MOV EAX, DWORD PTR SS:[EBP - C]
005AD2D3 8D 4D F8 LEA ECX, DWORD PTR SS:[EBP - 8]
005AD2D5 BA 07 00 00 00 MOV EDI, 7
005AD2DA E8 FD AD E8 FF CALL <matematica.SubStr>
005AD2DF 8B 45 F8 MOV EAX, DWORD PTR SS:[EBP - 8] [ebp-8]:L"1234567"
005AD2E2 E8 35 55 E7 FF CALL <matematica.StrToInt>
005AD2E7 8B D8 MOV EBX, EAX
005AD2E9 8D 55 EC LEA EDI, DWORD PTR SS:[EBP - 14]

```

Como decía al principio vemos que este código es para obtener la primera parte del serial y pasarlo a entero, justo el próximo es idéntico

```

005AD300 E8 F1 AD E8 FF CALL <matematica.SubStr>
005AD307 8B 45 F0 MOV EAX, DWORD PTR SS:[EBP - 10] [ebp-10]:L"7654321"
005AD30C E8 0D 55 E7 FF CALL <matematica.StrToInt>
005AD30F 8B F0 MOV ESI, EAX

```

Ya tenemos las dos partes del serial listas y partir de aquí empieza lo que nos interesa, a partir de **005AD30F**, pero antes vamos a ver el porqué de la función que etiquete antes como **"Pow"** (para los no programadores, esta función *"devuelve la base elevada al exponente de potencia indicado"*, o lo que es lo mismo eleva un numero dado a una potencia específica). Ahora en este punto vamos a hacer un combo entre IDA y x64dbg. Nos vamos a IDA a dicha función y presionamos **"F5"** para obtener un pseudocódigo de la misma, veamos de que se trata

```

1 float __fastcall pow(float base, int exponent)
2 {
3     float pwr_value; // ecx@1
4     int index; // edx@1
5
6     pwr_value = base;
7     index = exponent - 1;
8     if ( index > 0 )
9     {
10        do
11        {
12            pwr_value *= base;
13            --index;
14        }
15        while ( index );
16    }
17    return pwr_value;
18 }

```

Luego de cambiar los nombres de variables podemos ver más claro el propósito del código. Pues en este punto y ya que estamos dentro de IDA una vez más nos regresamos a la función anterior, pero no queremos empezar a dar F8 por todo ese código sin más, así que antes de volvernos locos, usamos **"F5"** para descompilar el código principal que estamos analizando y poder avanzar mucho más rápido en la búsqueda de un serial correcto. Ok, luego que procesamos todo ese rollo solo tomamos el extracto que más necesitamos y entonces tendríamos algo como esto para trabajar.

```

1 Mask::TCustomMaskEdit::GetText(*(_DWORD *) (a1 + 928), &v20);
2 SubStr(v20, 7, (int)&v21);
3 s1 = StrToInt(v21);
4
5 Mask::TCustomMaskEdit::GetText(*(_DWORD *) (v4 + 928), &v18);
6 SubStr(v18, 7, &v19);
7 s2 = StrToInt(v19);
8
9 dvar = s1 / 675829;
10 d1 = s1 / 8777 / 77;
11 d2 = s2 / 8777 / 77;
12 e1 = (d1 * (d1 - 1) - 5 * (d1 - 2)) * pow(d1, 3);
13 e2 = pow(d2, d2 * (d2 - 1) - 5 * (d2 - 2));
14
15 if ( e2 == e1 && e1 && e2 && d1 == dvar && e2 * e1 == 256 )
16     MessageBox(*off_5BADDC, L"Nice And Pure = Cracked!", L"Jhonjhon_123", 0);
17 else
18     MessageBox(*off_5BADDC, "2", L"Jhonjhon_123", 0);

```

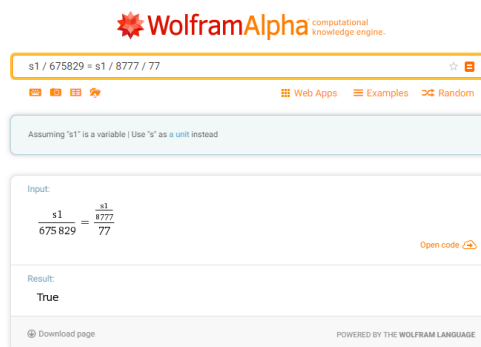
En solo esas líneas esta todo lo que necesitamos para sacar un serial correcto, así que empezamos desde el principio. Siguiendo la misma practica de renombrar variables hemos obtenido un código bastante legible por sí mismo, ahora solo nos falta analizar el algo. Partimos sacando las premisas de la comparación final y por lo que vemos podemos sacar las siguientes:

```
// Premisas
// -----
e1 == e2 // resultado de "ecuacion 1" tiene que ser igual a resultado de "ecuacion 2"
d1 == dvar // "division 1" tiene que ser igual a "division var"
e2 * e1 == 256 // la multiplicacion entre los resultados de ambas ecuaciones igual a 256
```

Por transitividad obtenemos que el resultado de la primera y la segunda ecuación tienen que ser iguales a 16 (aunque esta información no es esencial para la solución). Luego tenemos que **d1** necesita ser igual a **dvar**, conformando la ecuación correspondiente tenemos:

$$s1 / 675829 = s1 / 8777 / 77$$

Comprobamos matemáticamente y este es el resultado:



WolframAlpha computational knowledge engine.

Input: $\frac{s1}{675829} = \frac{s1}{8777 / 77}$

Result: True

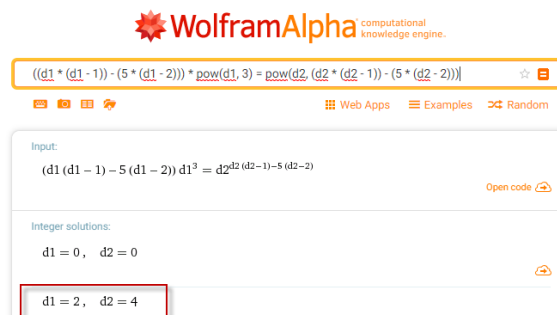
Como vemos esta premisa siempre se cumplirá, así que solo queda de relleno y no debemos prestarle más atención, pasando al otro par de ecuaciones que podemos formar tenemos:

$$16 = ((d1 * (d1 - 1)) - (5 * (d1 - 2))) * \text{pow}(d1, 3)$$

$$16 = \text{pow}(d2, (d2 * (d2 - 1)) - (5 * (d2 - 2)))$$

$$\text{Premisa 1: } ((d1 * (d1 - 1)) - (5 * (d1 - 2))) * \text{pow}(d1, 3) = \text{pow}(d2, (d2 * (d2 - 1)) - (5 * (d2 - 2)))$$

En este caso el par de valores obtenidos son los siguientes:



WolframAlpha computational knowledge engine.

Input: $((d1 * (d1 - 1)) - (5 * (d1 - 2))) * \text{pow}(d1, 3) = \text{pow}(d2, (d2 * (d2 - 1)) - (5 * (d2 - 2)))$

Integer solutions:

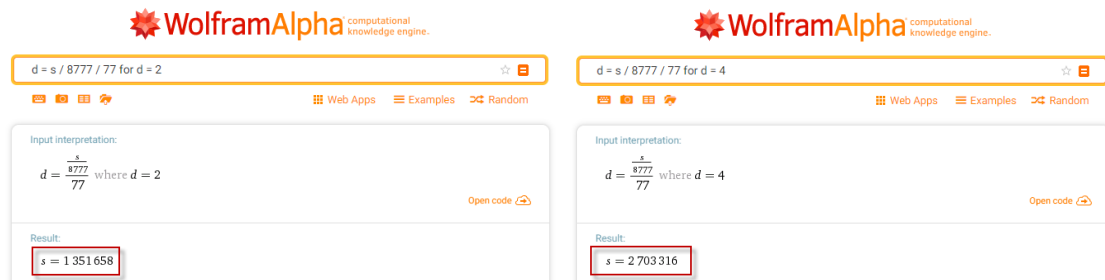
d1 = 0, d2 = 0

d1 = 2, d2 = 4

Como vemos hemos obtenido un par valido, si regresamos al pseudocódigo de arriba ahora conformamos las últimas ecuaciones para obtener s1(Serial Part1) y s2(Serial Part2) así que serían las siguientes:

$$d1 = s1 / 8777 / 77; \text{ para } d1 = 2$$

$$d2 = s2 / 8777 / 77; \text{ para } d2 = 4$$



Pues ahí están nuestros dos seriales, solo nos queda probarlos a ver si van

