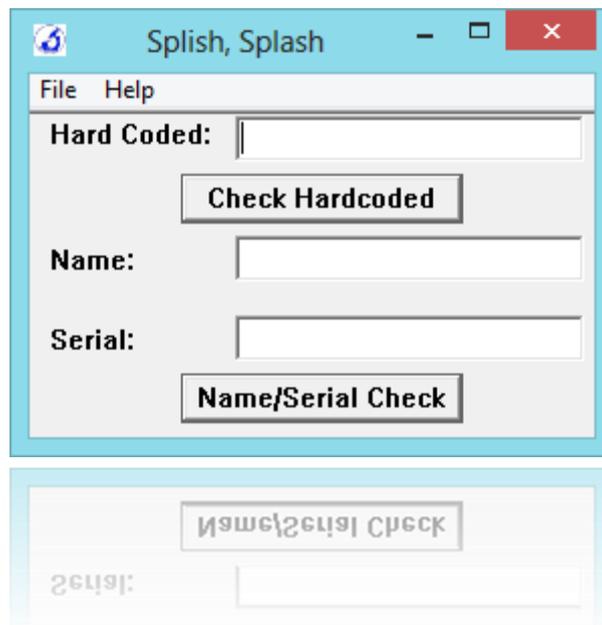


Keygen para el CrackMe Splish Splash de Crudd

Tres niveles



By deurus
14/09/2014

ÍNDICE

1.	Primeras impresiones.....	2
2.	Nopeando la Splash Screen	2
3.	Serial Hardcodeado	3
4.	El nombre y número de serie	4
5.	Notas finales.....	7
6.	Enlaces	7

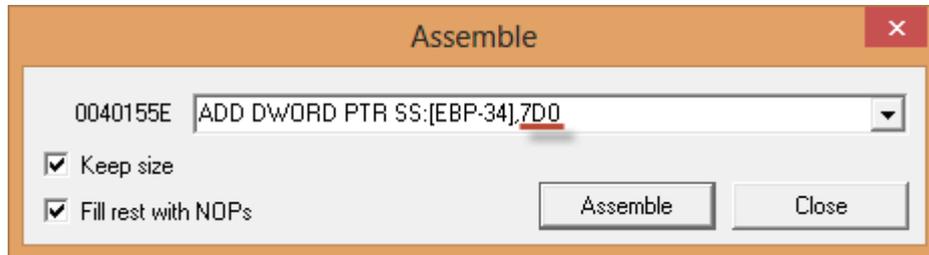
Equipo utilizado:

S.O: Windows 7 x32 / Windows 8 x64

Depurador: Ollydbg 1.10 (32bits) con plugins

Analizador: PEiD 0.95

Vemos la llamada a **CreateWindowExA** que podríamos parchear pero vamos a pensar un poco. Vemos la función **GetTickCount** y que carga el valor **7D0**. 7D0 es **2000 en decimal**, que perfectamente pueden ser milisegundos, por lo tanto el parcheo más elegante sería poner la función **GetTickCount** a **0**. En la imagen inferior se puede ver como queda parcheado el valor 7D0.



<pre> 0040154B . FF35 11324000 00401551 . E8 22020000 00401556 . E8 35020000 0040155B . 8945 CC 0040155E . 8345 CC 00 00401562 . 90 00401563 . 90 00401564 . 90 00401565 > . E8 26020000 0040156A . 8945 CC </pre>	<pre> PUSH DWORD PTR DS:[403211] CALL <JMP.&USER32.UpdateWindow> CALL <JMP.&KERNEL32.GetTickCount> MOV DWORD PTR SS:[LOCAL.13],EAX ADD DWORD PTR SS:[EBP-34],0 NOP NOP NOP CALL <JMP.&KERNEL32.GetTickCount> CMP DWORD PTR SS:[LOCAL.13],EAX </pre>	<pre> hWnd = 000D0EBA USER32.UpdateWindow KERNEL32.GetTickCount Jump to KERNEL32.GetTickCount </pre>
---	---	---

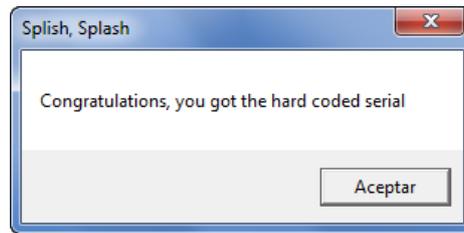
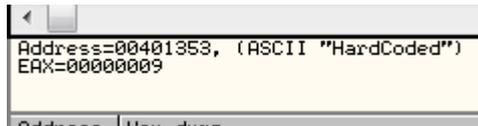
Probamos y funciona, pasamos a lo siguiente.

3. Serial Hardcodeado

El mensaje de error del serial hardcodeado dice "Sorry, please try again". Lo buscamos en las **string references** y vamos a parar aquí.

<pre> 0040135D > 6A 20 0040135F . 68 15324000 00401364 . FF35 90344000 0040136A . E8 B0300000 0040136F . 8D1D 15324000 00401375 > 8038 00 0040137B > 74 0C 0040137E . 8A08 00401382 . 9A13 00401384 . 38D1 00401386 > 75 4A 00401388 . 40 00401389 . 43 0040138A ^ EB EF 0040138C > EB 2F 0040138E . 43 6F 6E 67 72 61 74 75 00401392 . 20 79 6F 75 20 67 6F 74 0040139E . 64 20 63 6F 64 65 64 20 004013BD > 6A 00 004013BF . 68 0A304000 004013D4 . 68 8E134000 004013D9 . 6A 00 004013DB . E8 78030000 004013D8 > EB 13 004013D2 > 6A 00 004013D4 . 68 0A304000 004013D9 . 68 67304000 004013DE . 6A 00 004013E0 . E8 63030000 </pre>	<pre> PUSH 20 PUSH Splish.00403215 PUSH DWORD PTR DS:[403490] CALL <JMP.&USER32.GetWindowTextA> LEA EBX,DWORD PTR DS:[403215] CMP BYTE PTR DS:[EAX],0 JE SHORT Splish.0040138C MOV CL,BYTE PTR DS:[EAX] MOV DL,BYTE PTR DS:[EBX] CMP CL,DL JNZ SHORT Splish.004013D2 INC EAX INC EBX JMP SHORT Splish.0040137B JMP SHORT Splish.004013BD ASCII "Congratulations," ASCII " you got the hat" ASCII "d coded serial",0 PUSH 0 PUSH Splish.0040300A PUSH Splish.0040138E PUSH 0 CALL <JMP.&USER32.MessageBoxA> JMP SHORT Splish.004013E5 PUSH 0 PUSH Splish.0040300A PUSH Splish.00403067 PUSH 0 CALL <JMP.&USER32.MessageBoxA> </pre>	<pre> Count = 20 (32.) Buffer = Splish.00403215 hWnd = 0075079E (class="Edit",parent=0010083C) GetWindowTextA Style = MB_OK!MB_APPLMODAL Title = "Splish, Splash" Text = "Congratulations, you got the hard coded serial!" hOwner = NULL MessageBoxA Style = MB_OK!MB_APPLMODAL Title = "Splish, Splash" Text = "Sorry, please try again." hOwner = NULL MessageBoxA </pre>
--	--	---

Vemos un bucle de comparación que carga unos bytes de la memoria, los bytes dicen "HardCoded", probamos y prueba superada.



4. El nombre y número de serie

Con el mismo método de las **string references** localizamos el código que nos interesa. Metemos deurus como nombre y 12345 como serial y empezamos a tracear. Lo primero que hace es una serie de operaciones con nuestro nombre a las que podemos llamar aritmética modular. Aunque en la imagen viene bastante detallado se ve mejor con un ejemplo.

```

00401621 | . 8D35 36324000 | LEA ESI,[403236] | ASCII "deurus"
00401627 | . 8D3D 58324000 | LEA EDI,[403258] | ; Los mod del Serial los guarda a partit de 403258
0040162D | . B9 0A000000 | MOV ECX,0A | ; <-- (EAX = digito del nombre que toque)
00401632 | > 0FBE041E | MOVSX EAX,BYTE PTR DS:[EBX+ESI] | ; EAX / ECX = EAX y resto a EDX
00401636 | . 99 | CDQ | ; Resto(EDX) XOR 0,1,2,3...
00401637 | . F7F9 | IDIV ECX | ; EDX + 2
00401639 | . 3D03 | XOR EDX,EBX | ; Compara EDX con 10
0040163B | . 83C2 02 | ADD EDX,2 | ; Si DL <10 salta
0040163E | . 80FA 0A | CMP DL,0A | ; Si DL es >=10 le resta 10
00401641 | . 7C 03 | JL SHORT 00401646 | ; Guarda DL en el DUMP 403258 y sucesivos
00401643 | . 80EA 0A | SUB DL,0A | ; Compara EBX con Len(Nombre)
00401646 | > 88141F | MOV BYTE PTR DS:[EBX+EDI],DL | ; Bucle -->
00401649 | . 43 | INC EBX |
0040164A | . 3B1D 63344000 | CMP EBX,DWORD PTR DS:[403463] |
00401650 | . 75 E0 | JNE SHORT 00401632 |

```

Ejemplo para Nombre: **deurus**

```

1 d e u r u s
2 64 65 75 72 75 73 -hex
3 100 101 117 114 117 115 -dec
4
5 1ºByte = ((Nombre[0] % 10)^0)+2
6 2ºByte = ((Nombre[1] % 10)^1)+2
7 3ºByte = ((Nombre[2] % 10)^2)+2
8 4ºByte = ((Nombre[3] % 10)^3)+2
9 5ºByte = ((Nombre[4] % 10)^4)+2
10 6ºByte = ((Nombre[5] % 10)^5)+2
11
12 1ºByte = ((100 Mod 10) Xor 0) + 2
13 2ºByte = ((101 Mod 10) Xor 1) + 2
14 3ºByte = ((117 Mod 10) Xor 2) + 2
15 4ºByte = ((114 Mod 10) Xor 3) + 2
16 5ºByte = ((117 Mod 10) Xor 4) + 2
17 6ºByte = ((115 Mod 10) Xor 5) + 2
18
19 Si el byte > 10 --> Byte = byte - 10
20
21 1ºByte = 2
22 2ºByte = 2
23 3ºByte = 7
24 4ºByte = 9
25 5ºByte = 5
26 6ºByte = 2

```

Lo que nos deja que los **Bytes mágicos** para deurus son: **227952**.

Debido a la naturaleza de la operación **IDIV** y el bucle en general, llegamos a la conclusión de que para cada letra es un solo byte mágico y que este está comprendido entre 0 y 9.

A continuación realiza las siguientes **operaciones** con el **serial introducido**.

```

00401658 | . 8D35 4D324000 LEA ESI,[403240]          ASCII "12345"
0040165E | . 8D3D 4D324000 LEA EDI,[403240]          ; Los mod del Serial los guarda a partir de 403240
00401664 | . B9 0A000000    MOV ECX,0A
00401669 | > 0FBE041E      MOVSX EAX,BYTE PTR DS:[EBX+ESI] ; <-- (EAX = el dígito del serial que toque)
0040166D | . 99           CDQ
0040166E | . F7F9        IDIV ECX
00401670 | . 88141F      MOV BYTE PTR DS:[EBX+EDI],DL ; EAX / ECX (Dígito ascii serial / 10, el resto e EDX)
00401673 | . 43         INC EBX ; Guarda el resto de la división en 403240 y siguientes
00401674 | . 3B1D 67344000 CMP EBX,DWORD PTR DS:[403467]
0040167A | . ^ 75 ED      JNE SHORT 00401669 ; Bucle -->
0040167C | . v EB 2A     JMP SHORT 004016A8

```

Ejemplo para serial: **12345**

1	1	2	3	4	5	
2	31	32	33	34	35	-hex
3	49	50	51	52	53	-dec
4						
5	49	mod 10	=	9		
6	50	mod 10	=	0		
7	51	mod 10	=	1		
8	52	mod 10	=	2		
9	53	mod 10	=	3		

Los bytes mágicos del serial son: **90123**, que difieren bastante de los conseguidos con el nombre.

A continuación **compara byte a byte 227952 con 90123**.

```

004016A8 | > 8D35 4D324000 LEA ESI,[403240]          ; Contiene los bytes del serial
004016AE | . 8D3D 58324000 LEA EDI,[403258]          ; Contiene los bytes del HashNombre
004016B4 | . 33DB        XOR EBX,EBX
004016B6 | > 3B1D 63344000 CMP EBX,DWORD PTR DS:[403463]
004016BC | . v 74 0F      JE SHORT 004016C0 ; Si todo ha salido bien salta a Good job
004016BE | . 0FBE041F   MOVSX EAX,BYTE PTR DS:[EBX+EDI]
004016C2 | . 0FBE0C1E   MOVSX ECX,BYTE PTR DS:[EBX+ESI]
004016C6 | . 3BC1       CMP EAX,ECX ; Compara byte a byte 403240 y sucesivas con 403258 y sucesivas
004016C8 | . v 75 18     JNE SHORT 004016E2 ; Si no coincide algun byte salta a ERROR
004016CA | . 43        INC EBX
004016CB | . ^ EB E9     JMP SHORT 004016B6

```

En resumen, para cada nombre genera un código por cada letra y luego la comprobación del serial la realiza usando el módulo 10 del dígito ascii. Lo primero que se me ocurre es que necesitamos cotejar algún dígito del 0 al 9 para tener cubiertas todas las posibilidades. Realizamos manualmente **mod 10** a los números del 0 al 9 y obtenemos sus valores.

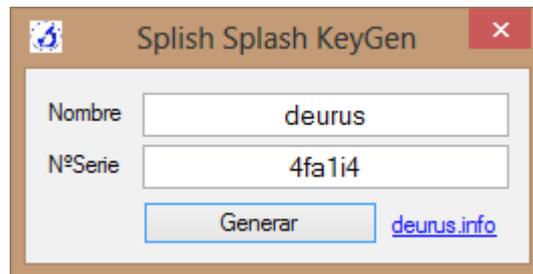
1	(0)	48 mod 10 = 8
2	(1)	49 mod 10 = 9
3	(2)	50 mod 10 = 0
4	(3)	51 mod 10 = 1
5	(4)	52 mod 10 = 2
6	(5)	53 mod 10 = 3
7	(6)	54 mod 10 = 4
8	(7)	55 mod 10 = 5
9	(8)	56 mod 10 = 6
10	(9)	57 mod 10 = 7

Con esto ya podríamos generar un serial válido.

1	0123456789	- Nuestro alfabeto numérico
2		
3	8901234567	- Su valor Mod 10

Por lo que para **deurus** un **serial válido** sería: **449174**. Recordemos que los bytes mágicos para **deurus** eran "227952", solo hay que sustituir.

Para realizar un **KeyGen** más interesante, he sacado los valores de un alfabeto mayor y le he añadido una rutina aleatoria para que genere seriales diferentes para un mismo nombre.



```

1 'abcdefghijklmnopqrstuvwxyz0123456789ABCDEFHGHIJKLMNOPQRSTUVWXYZ - Alfabeto
2 '7890123456778901234567789018901234567567890123455678901234556880 - Valor
3 Dim suma As Integer = 0
4 'Para hacer el serial más divertido
5 Dim brute() As String = {"2", "3", "4", "5", "6", "7", "8", "9", "0", "1"}
6 Dim brute2() As String = {"d", "e", "f", "g", "h", "i", "j", "a", "b", "c"}
7 Dim brute3() As String = {"P", "Q", "R", "S", "T", "U", "j", "a", "D", "E"}
8 Dim alea As New Random()
9 txtserial.Text = ""
10 'Evito nombres mayores de 11 para evitar el BUG comentado en le manual
11 If Len(txtnombre.Text) > 0 And Len(txtnombre.Text) < 12 Then
12     For i = 1 To Len(txtnombre.Text)
13         Dim aleatorio As Integer = alea.Next(0, 9)
14         suma = (((Asc(Mid(txtnombre.Text, i, 1))) Mod 10) Xor i - 1) + 2
15         If suma > 9 Then
16             suma = suma - 10
17         End If
18         If (aleatorio) >= 0 And (aleatorio) <= 4 Then
19             txtserial.Text = txtserial.Text & brute(suma)
20         ElseIf (aleatorio) > 4 And (aleatorio) <= 7 Then
21             txtserial.Text = txtserial.Text & brute2(suma)
22         ElseIf (aleatorio) > 7 And (aleatorio) <= 10 Then
23             txtserial.Text = txtserial.Text & brute3(suma)
24         End If
25         suma = 0
26     Next
27 Else
28     txtserial.Text = "El Nombre..."
29 End If

```

5. Notas finales

Hay un pequeño **bug** en el almacenaje del nombre y serial y en el guardado de bytes mágicos del serial. Si nos fijamos en los bucles del nombre y el serial, vemos que los bytes mágicos del nombre los guarda a partir de la **dirección** de memoria **403258** y los bytes mágicos del serial a partir de **40324D**. En la siguiente imagen podemos ver seleccionados los 11 primeros bytes donde se almacenan los bytes mágicos del serial. Vemos que hay seleccionados 11 bytes y que el siguiente sería ya **403258**, precisamente donde están los bytes mágicos del nombre. Como puedes imaginar si escribes un serial >11 dígitos se solapan bytes y es una chapuza, de modo que el keygen lo he limitado a nombres de 11 dígitos.

6. Enlaces

<https://deurus.info/archivos/manuales/>