



Estudiando el sistema de registrado de EF StartUp Manager 6.40

Fecha	30 de octubre de 2016
Victima	EF StartUp Manager 6.40 x32 PORTABLE
URL de descarga	http://www.efsoftware.com/dw/wzp.cgi?su
MD5 del instalador	7E40A31A6D5F59B0F3D478EB5520A746
Protección	Asprotect 2.xx, registrado con archivo de licencia
Herramientas	RDG Packer Detector, Ollydbg, C++Builder 6
Objetivo	Estudiar la verificación de datos del archivo de licencia poder crear un keygen
Dificultad	Media
Cracker	Aguml

Lo primero es analizarlo:



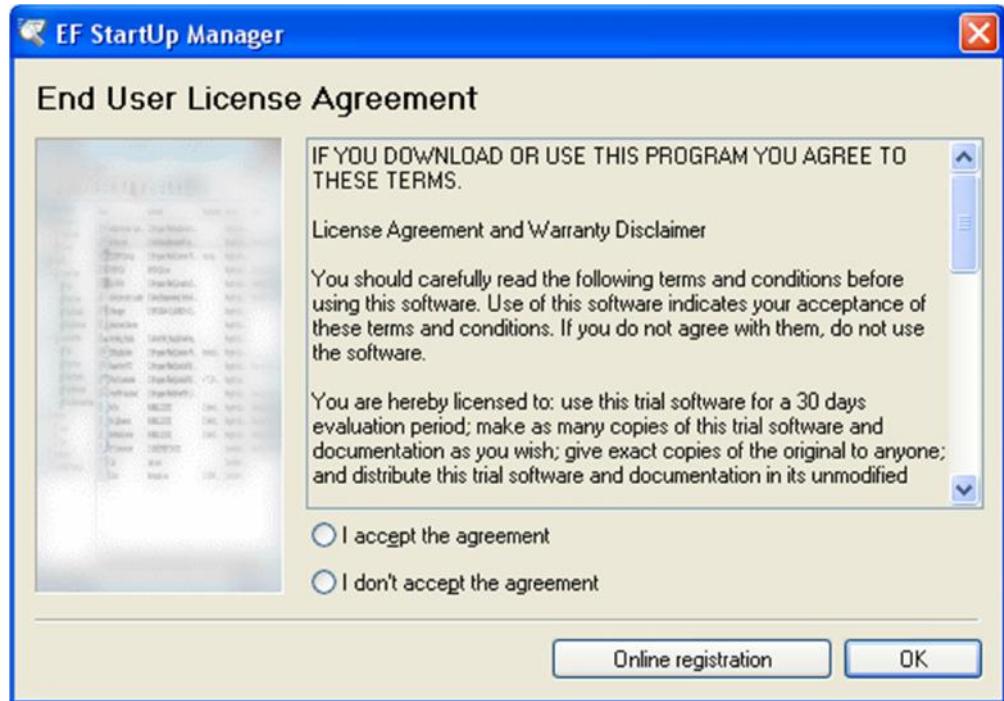
Y si lo abro y voy al menú “Ayuda”:



Si doy a “Informacion del producto...”:



Y a los pocos segundos aparece esto:



Acepto y al cerrar el programa sale esto:



Uso un archivo de licencia de otra versión que me pasó Apuromafo y lo coloco en el directorio de la aplicación. Abro el ejecutable en Olly, doy a F9, hago clic derecho y elijo "Search For->All constants" y busco la constante "80000003" en la parte de hexadecimal ya que se usa en la función desencriptadora tanto del EF Find como del EF Checksum Manager.

Salen estas:

R References in EFSUM: to constant 80000003			
Address	Disassembly	Comment	
00401000	mov eax, ds:[ecx]		(Initial CPU selection)
00414B5F	and edx, 80000003		
00434471	cmp dword ptr ds:[esi], 80000003		

Y la que me interesa es esta:

```
00414B5F | . 81E2 03000080 |and      edx, 80000003
```

Si voy a donde está esa línea ya estoy en la función que desencripta:

00414B40	\$ 56	push	esi	
00414B41	. 8B35 04134500	mov	esi, ds:[451304]	
00414B47	. 81C6 B8010000	add	esi, 1B8	
00414B4D	.~ 74 40	je	short 00414B8F	EFSUM.00414B8F
00414B4F	. 53	push	ebx	
00414B50	. 33C9	xor	ecx, ecx	
00414B52	. 57	push	edi	
00414B53	. 8D79 39	lea	edi, ds:[ecx+39]	
00414B56	> 8A0431	mov	al, ds:[ecx+esi]	
00414B59	. 3C 40	cmp	al, 40	
00414B5B	.~ 73 16	jnb	short 00414B73	EFSUM.00414B73
00414B5D	. 8BD1	mov	edx, ecx	
00414B5F	. 81E2 03000080	and	edx, 80000003	
00414B65	.~ 79 05	jns	short 00414B6C	EFSUM.00414B6C
00414B67	. 4A	dec	edx	
00414B68	. 83CA FC	or	edx, FFFFFFFC	
00414B6B	. 42	inc	edx	
00414B6C	> 2AC2	sub	al, dl	
00414B6E	. 880431	mov	ds:[ecx+esi], al	
00414B71	.~ EB 14	jmp	short 00414B87	EFSUM.00414B87
00414B73	> 0FB6C0	movzx	eax, al	
00414B76	. 83E8 41	sub	eax, 41	
00414B79	. 99	cdq		
00414B7A	. BB 0A000000	mov	ebx, 0A	
00414B7F	. F7FB	idiv	ebx	
00414B81	. 80C2 30	add	dl, 30	
00414B84	. 881431	mov	ds:[ecx+esi], dl	
00414B87	> 41	inc	ecx	
00414B88	. 83EF 01	sub	edi, 1	
00414B8B	.^ 75 C9	jnz	short 00414B56	EFSUM.00414B56
00414B8D	. 5F	pop	edi	

Pongo un HBP on Execution en la línea 414B8D, que es la que está justo después del bucle desencriptador, reinicio Olly y, cuando pare en ese HBP, miro en ESI y ya tengo ahí el puntero hacia el ID desencriptado. Voy a esa dirección en el Dump y pongo un Memory BP on Access en todo el ID desencriptado. Doy a F9 y la primera vez que para es aquí:

00431D1D	> 8A10	mov dl, ds:[eax]	
00431D1F	. 83C0 04	add eax, 4	
00431D22	. 83C1 04	add ecx, 4	
00431D25	. 84D2	test dl, dl	
00431D27	.~ 74 52	je short 00431D7B	EFSUM.00431D7B
00431D29	. 3A51 FC	cmp dl, ds:[ecx-4]	
00431D2C	.~ 75 4D	jnz short 00431D7B	EFSUM.00431D7B
00431D2E	. 8A50 FD	mov dl, ds:[eax-3]	
00431D31	. 84D2	test dl, dl	
00431D33	.~ 74 3C	je short 00431D71	EFSUM.00431D71
00431D35	. 3A51 FD	cmp dl, ds:[ecx-3]	
00431D38	.~ 75 37	jnz short 00431D71	EFSUM.00431D71
00431D3A	. 8A50 FE	mov dl, ds:[eax-2]	
00431D3D	. 84D2	test dl, dl	
00431D3F	.~ 74 26	je short 00431D67	EFSUM.00431D67
00431D41	. 3A51 FE	cmp dl, ds:[ecx-2]	
00431D44	.~ 75 21	jnz short 00431D67	EFSUM.00431D67
00431D46	. 8A50 FF	mov dl, ds:[eax-1]	
00431D49	. 84D2	test dl, dl	
00431D4B	.~ 74 10	je short 00431D5D	EFSUM.00431D5D
00431D4D	. 3A51 FF	cmp dl, ds:[ecx-1]	

Registers (FPU)	<	<	<	<
EAX 00A34BCD ASCII "9534530110190761231110907023"				
ECX 00126F9C ASCII "953453,0110190761231"				
EDX 00126F9C ASCII "953453,0110190761231"				
EBX 00000013				
ESP 00126F70				
EBP 00126F7C				
ESI 00A34BCD ASCII "9534530110190761231110907023"				
EDI 0000000F				

Esta función es la que es como el strncmp de C con lo que miro en EAX y veo que apunta al carácter 30º del ID desencriptado y en ECX y EDX veo el serial sin guiones pero con una peculiaridad, introduce una coma justo después del 6º carácter.

Voy en el Dump a la dirección que apunta EAX y me aseguro que coincidan todos con lo que hay en ECX. Traceo hasta salir de la función y llego aquí:

00411BFF	. 6A 13	push 13		
00411C01	. 8D5424 10	lea edx, ss:[esp+10]		
00411C05	. 52	push edx		
00411C06	. 81C6 D5010000	add esi, 1DS		
00411C0C	. 56	push esi		
00411C0D	. E8 DF000200	call 00431CF1	EFSUM.00431CF1	
00411C12	. 83C4 14	add esp, 14		
00411C15	. 85C0	test eax, eax		
00411C17	.~ 74 1F	je short 00411C38	EFSUM.00411C38	
00411C19	. 8B0D 04134500	mov ecx, ds:[451304]		
00411C1F	. 33C0	xor eax, eax		
00411C21	. 81C1 06010000	add ecx, 106		
00411C27	. 8901	mov ds:[ecx], eax		
00411C29	. 8941 04	mov ds:[ecx+4], eax		
00411C2C	. 8941 08	mov ds:[ecx+8], eax		
00411C2F	. 8941 0C	mov ds:[ecx+C], eax		
00411C32	. 8941 10	mov ds:[ecx+10], eax		
00411C35	. 8941 14	mov ds:[ecx+14], eax		
00411C38	> 8B4C24 38	mov ecx, ss:[esp+38]		

Pongo un HBP on Execution en el inicio de la función, reinicio y repito el proceso anterior y cuando paro en este último HBP voy traceando y aquí:

00411BF2	. 50	push eax		
00411BF3	. 8D8E 06010000	lea ecx, ds:[esi+106]		
00411BF9	. 51	push ecx		
< []				
Address=00A34AFE, (ASCII "953453,01101-90761231")				
ecx=00000001				

O sea que ya ha puesto la coma antes de entrar a la función con lo que sigo traceando y cuando llego a este call:

00411BF9	. 51	push ecx		
00411BFA	. E8 A14CFFFF	call 004068A0	EFSUM.004068A0	
00411BFF	. 6A 13	push 13		

Entro y veo esto:

004068A0	56	push	esi	
004068A1	8B7424 08	mov	esi, ss:[esp+8]	
004068A5	33C9	xor	ecx, ecx	
004068A7	33D2	xor	edx, edx	
004068A9	380E	cmp	ds:[esi], cl	
004068AB	. 74 22	je	short 004068CF	EFSUM.004068CF
004068AD	. 53	push	ebx	
004068AE	. 57	push	edi	
004068AF	. 8B7C24 14	mov	edi, ss:[esp+14]	
004068B3	. 8BC6	mov	eax, esi	
004068B5	> 8A00	mov	al, ds:[eax]	
004068B7	. 3C 2D	cmp	al, 2D	
004068B9	.~ 74 07	je	short 004068C2	EFSUM.004068C2
004068BB	. 0FB7DA	movzx	ebx, dx	
004068BE	. 88043B	mov	ds:[ebx+edi], al	
004068C1	. 42	inc	edx	
004068C2	> 41	inc	ecx	
004068C3	. 0FB7C1	movzx	eax, cx	
004068C6	. 03C6	add	eax, esi	
004068C8	. 8038 00	cmp	byte ptr ds:[eax], 0	
004068CB	.^ 75 E8	jnz	short 004068B5	EFSUM.004068B5
004068CD	. 5F	pop	edi	
004068CE	. 5B	pop	ebx	
004068CF	> 5E	pop	esi	
004068D0	. C3	retn		

Es la función que quita los guiones del serial y es interesante quedarse con esta dirección ya que entra aquí para más de una comprobación con el serial.

Sigo traceando hasta el siguiente call que es el que se parece a strncmp:

00411BFF	. 6A 13	push	13	
00411C01	. 8D5424 10	lea	edx, ss:[esp+10]	
00411C05	. 52	push	edx	
00411C06	. 81C6 D5010000	add	esi, 1D5	
00411C0C	. 56	push	esi	
00411C0D	. E8 DF000200	call	00431CF1	EFSUM.00431CF1
00411C12	. 83C4 14	add	esp, 14	
00411C15	. 85C0	test	eax, eax	
00411C17	.~ 74 1F	je	short 00411C38	EFSUM.00411C38
00411C19	. 8B0D 04134500	mov	ecx, ds:[451304]	

00126F84	00A34BCD	ASCII "9534530110190761231110907023"
00126F88	00126F9C	
00126F8C	00000013	
00126F90	00A34AFE	ASCII "953453,01101-90761231"

Me aseguro por si acaso de que mi ID desencriptado pase la comparación yendo en el Dump a la dirección apuntada por la dirección que se ve en el Stack que contiene el Serial con la coma y modificándolo:

Address	Hex dump	ASCII
00A34AFE	39 35 33 34 35 33 30 31 31 30 31 2D 39 30 37 36	9534530110190761231110907023
00A34B0E	31 32 33 31 00 00 0A 00 20 00 4F 54 20 52 45 47	1231.... .OT REG

Sigo traceando hasta salir de la función y caigo en otra función así que pongo un HBP on Execution al inicio, reinicio y repito todo el proceso y, cuando pare, voy traceando y, al llegar a esta call:

00419AFD	. 66:8B7424 14	mov	si, ss:[esp+14]	
00419B02	. 83C4 04	add	esp, 4	
00419B05	.~ EB 02	jmp	short 00419B09	EFSUM.00419B09
00419B07	> 33F6	xor	esi, esi	
00419B09	> E8 C280FFFF	call	00411BDO	EFSUM.00411BDO
00419B0E	. 66:85F6	test	si, si	
00419B11	.~ 74 16	je	short 00419B29	EFSUM.00419B29
00419B13	. 33C0	xor	eax, eax	
00419B15	. 33F6	xor	esi, esi	
00419B17	. 394424 1B	cmp	ss:[esp+1B], eax	
00419B1E	. 74 02	je	short 00419B1E	EFSUM.00419B1E

Entro y veo esto:

00411BDO	\$ 83EC 38	sub	esp, 38	
00411BD3	. A1 F0FD4400	mov	eax, ds:[44FDF0]	
00411BD8	. 33C4	xor	eax, esp	
00411BDA	. 894424 34	mov	ss:[esp+34], eax	
00411BDE	. 56	push	esi	
00411BDF	. 8B35 04134500	mov	esi, ds:[451304]	
00411BE5	. 83BE 08020000	cmp	dword ptr ds:[esi+208], 0	
00411BEC	.~ 74 4A	je	short 00411C38	EFSUM.00411C38
00411BEE	. 8D4424 04	lea	eax, ss:[esp+4]	
00411BF2	. 50	push	eax	
00411BF3	. 8D8E 06010000	lea	ecx, ds:[esi+106]	
00411BF9	. 51	push	ecx	
00411BFA	. E8 A14CFFFF	call	004068A0	EFSUM.004068A0

Si traceo hasta el salto veo que no se cumple con lo que no va a llamar a la función que quita los guiones.
Si doy a F9 caigo en el HBP que está justo después del desencriptado del ID y si vuelvo a dar a F9 caigo nuevamente en la llamada a esta función.

Si vuelvo a tracear hasta la comparación justo antes del salto veo esto:

00411BDF	. 8B35 04134500	mov	esi, ds:[451304]	
00411BE5	. 83BE 08020000	cmp	dword ptr ds:[esi+208], 0	
00411BEC	.~ 74 4A	je	short 00411C38	EFSUM.00411C38
00411BEE	. 8D4424 04	lea	eax, ss:[esp+4]	
< >				
Serial No.: 9"....)				
ds:[00A34C00]=00A3D4D0, (ASCII "EF StartUp Manager distribution license",CR,LF,CR,I				

O sea que ahora sí que va a entrar y [esi+208] apunta al buffer con el ID sustituido por una sucesión de serials algo peculiar como pasaba en el EF Find y en el EF Checksum Manager.

Traceo unas líneas y veo esto:

00411BEC	.~ 74 4A	je	short 00411C38	EFSUM.00411C38
00411BEE	. 8D4424 04	lea	eax, ss:[esp+4]	
00411BF2	. 50	push	eax	
00411BF3	. 8D8E 06010000	lea	ecx, ds:[esi+106]	
00411BF9	. 51	push	ecx	
00411BFA	. E8 A14CFFFF	call	004068A0	EFSUM.004068A0
< >				
Address=00A34AFE, (ASCII "953453,01101-90761231")				
ecx=FFFFFFFF				

Como no veo la manera de dar con el sitio donde coloca la coma opto por lo que hice en el caso del EF Find, así que reinicio, coloco un BP en CreateFileW y voy dando a F9 hasta que vaya a abrir el archivo de licencia:

0012A200	0042D370	CALL to CreateFileW from EFSUM.0042D36A
0012A204	0012DAD4	FileName = "C:\Documents and Settings\BlueDeep\Escritorio\EF Startup Manager x32 PORTABLE\EFSUM.LIC"
0012A208	80000000	Access = GENERIC_READ
0012A20C	00000003	ShareMode = FILE_SHARE_READ FILE_SHARE_WRITE
0012A210	00000000	pSecurity = NULL
0012A214	00000003	Mode = OPEN_EXISTING
0012A218	00000080	Attributes = NORMAL
0012A21C	00000000	hTemplateFile = NULL

Una vez parado ahí, doy a Ctrl+F9, quito el BP, coloco otro en ReadFile y doy a F9.

Una vez ahí, voy al buffer en el Dump y doy a Ctrl+F9 y ya veo el buffer lleno con la información del archivo de licencia.

Me voy a la parte dónde está mi serial, quito todos los HBP que tengo y pongo un HBP on Access en el primer carácter del serial. Para varias veces para copiarlo en otras partes y voy poniendo un HBP en el primer carácter del serial en todas las copias y al final llego aquí:

004314E8	. F3:A5	rep	movs dword ptr es:[edi], dword	1
004314EA	. FF2495 A415A4	jmp	ds:[edx*4+4315A4]	
004314F1	8D49 00	lea	ecx, ds:[ecx]	
004314F4	> 23D1	and	edx, ecx	
004314F6	. 8A06	mov	al, ds:[esi]	
004314F8	. 8807	mov	ds:[edi], al	
004314FA	. 8A46 01	mov	al, ds:[esi+1]	
004314FD	. C1E9 02	shr	ecx, 2	
00431500	. 8847 01	mov	ds:[edi+1], al	
00431503	. 83C6 02	add	esi, 2	
00431506	. 83C7 02	add	edi, 2	
00431509	. 83F9 08	cmp	ecx, 8	
0043150C	.^ 72 A6	jb	short 004314B4	EFSUM.004314B4
0043150E	. F3:A5	rep	movs dword ptr es:[edi], dword	1
00431510	. FF2495 A415A4	jmp	ds:[edx*4+4315A4]	

Si miro adonde apunta ESI es justo al primer carácter del serial en el buffer original y si traceo un poco veo como a partir de una serie de operaciones lo que hace es otra copia más así que pongo otro HBP on Access al primer carácter de esta copia y vuelvo a dar a F9 y caigo aquí:

0041384F	. 33C0	xor	eax, eax	
00413851	> 0FB7D0	movzx	edx, ax	
00413854	. 80BC0A 060100	cmp	byte ptr ds:[edx+ecx+106], 0D	
0041385C	.~ 74 OA	je	short 00413868	EFSUM.00413868
0041385E	. 40	inc	eax	
0041385F	. 66:83F8 18	cmp	ax, 18	
00413863	.^ 72 EC	jb	short 00413851	EFSUM.00413851
00413865	. 43	inc	ebx	
00413866	.~ EB 6E	jmp	short 004138D6	EFSUM.004138D6
00413868	> 0FB7C0	movzx	eax, ax	
0041386B	. C68408 060100	mov	byte ptr ds:[eax+ecx+106], 0	
00413873	. 8B0D 04134500	mov	ecx, ds:[451304]	

Lo que va a hacer ahí es poner un carácter de fin de cadena al final de la última copia del serial.

Voy pasando los lugares no interesantes con F9 hasta que caigo aquí:

004068A0	\$ 56	push	esi	
004068A1	. 8B7424 08	mov	esi, ss:[esp+8]	
004068A5	. 33C9	xor	ecx, ecx	
004068A7	. 33D2	xor	edx, edx	
004068A9	. 380E	cmp	ds:[esi], cl	
004068AB	.. 74 22	je	short 004068CF	EFSUM.004068CF
004068AD	. 53	push	ebx	
004068AE	. 57	push	edi	
004068AF	. 8B7C24 14	mov	edi, ss:[esp+14]	
004068B3	. 8BC6	mov	eax, esi	
004068B5	> 8A00	mov	al, ds:[eax]	
004068B7	. 3C 2D	cmp	al, 2D	
004068B9	.. 74 07	je	short 004068C2	EFSUM.004068C2
004068BB	. 0FB7DA	movzx	ebx, dx	
004068BE	. 88043B	mov	ds:[ebx+edi], al	
004068C1	. 42	inc	edx	
004068C2	> 41	inc	ecx	
004068C3	. 0FB7C1	movzx	eax, cx	
004068C6	. 03C6	add	eax, esi	
004068C8	. 8038 00	cmp	byte ptr ds:[eax], 0	
004068CB	.^ 75 E8	jnz	short 004068B5	EFSUM.004068B5
004068CD	. 5F	pop	edi	
004068CE	. 5B	pop	ebx	
004068CF	> 5E	pop	esi	
004068D0	. C3	ret	n	

Es la función que quita los guiones del serial así que pongo un BP en el retn y doy a F9 tantas veces como sea necesario hasta que pare en él, quito el BP, salgo de la función con F8 y sigo traceando un poco hasta llegar aquí:

004138A5	. E8 F62FFFFF	call	004068A0	EFSUM.004068A0
004138AA	. 6A 13	push	13	
004138AC	. 8D4424 20	lea	eax, ss:[esp+20]	
004138B0	. 55	push	ebp	
004138B1	. 50	push	eax	
004138B2	. E8 1994FFFF	call	0040CCD0	EFSUM.0040CCD0
004138B7	. 83C4 14	add	esp, 14	
004138BA	. 8D9B 00000000	lea	ebx, ds:[ebx]	

Esta función parece prometedora porque sus parámetros son el tamaño del serial sin guiones, un puntero hacia el ID encriptado, y un puntero al serial sin guiones así que entro con F7:

0040CCD0	\$ 55	push	ebp		
0040CCD1	. 0FB76C24 10	movzx	ebp, word ptr ss:[esp+10]		
0040CCD6	. 56	push	esi		
0040CCD7	. 8B7424 10	mov	esi, ss:[esp+10]		
0040CCDB	. 57	push	edi		
0040CCDC	. 8B7C24 10	mov	edi, ss:[esp+10]		
0040CCE0	. 33C9	xor	ecx, ecx		
0040CCE2	. B8 01000000	mov	eax, 1		
0040CCE7	> 3BC5	[cmp	eax, ebp		
0040CCE9	.~ 7C 08	jb	short 0040CCF3	EFSUM.0040CCF3	
0040CCEB	. 8BC1	mov	eax, ecx		
0040CCED	. 83E0 03	and	eax, 3		
0040CCF0	. 83C0 04	add	eax, 4		
0040CCF3	> 8A1438	mov	dl, ds:[eax+edi]		
0040CCF6	. 881431	mov	ds:[ecx+esi], dl		
0040CCF9	. 41	inc	ecx		
0040CCFA	. 40	inc	eax		
0040CCFB	. 83F9 39	cmp	ecx, 39		
0040CCFE	.^ 72 E7	jb	short 0040CCE7	EFSUM.0040CCE7	
0040CD00	. 5F	pop	edi	00A34BE9	
0040CD01	. 5E	pop	esi		
0040CD02	. 5D	pop	ebp		
0040CD03	. C3	ret			
0040CD04	. CC				

Después de tracear la función, veo cómo va rellenando el ID encriptado con sucesiones algo peculiares del serial sin guiones:

Address	Hex dump	ASCII
00A3D59D	33 31 0D 0A 52 65 67 2E 49 44 2E 20 20 20 3A 20	31..Reg.ID. :
00A3D5AD	35 33 34 35 33 30 31 31 30 31 39 30 37 36 31 32	5345301101907612
00A3D5BD	33 31 30 31 31 30 31 39 30 37 36 31 32 33 31 31	3101101907612311
00A3D5CD	31 30 31 39 30 37 36 31 32 33 31 31 31 30 31 39	1019076123111019
00A3D5DD	30 37 36 31 32 33 31 31 31 0D 0A 0D 0A 57 61 72	076123111....War
00A3D5ED	6E 69 6E 67 3A 20 54 68 69 73 20 70 72 6F 64 75	ning: This produ
00A3D5FD	63 74 20 69 73 20 6C 69 63 65 6E 73 65 64 20 74	ct is licensed t

Como ya se vio en los tutos del EF Checksum Manager y en el del EF Find, ese cambio lo hace para poder hacer los cálculos con el buffer del archivo de licencia y obtener una cadena con la que verificará si parte del ID desencriptado coincide con parte de esa cadena.

Doy a F9 y caigo aquí:

0040F416	. 81C2 280C0000	add edx, 0C28	
0040F41C	. 52	push edx	
0040F41D	. 68 74CE4400	push 44CE74	ASCII "%06lu-%05lu-%08lu"
0040F422	. 8D4424 1C	lea eax, ss:[esp+1C]	
0040F426	. 6A 3F	push 3F	
0040F428	. 50	push eax	
0040F429	. E8 7D270200	call 00431BAB	EFSUM.00431BAB
0040F42E	. 83C4 18	add esp, 18	
0040F431	. 8BCB	mov ecx, ebx	
0040F433	. 8D4424 0C	lea eax, ss:[esp+C]	
0040F437	> 8A10	mov dl, ds:[eax]	
0040F439	. 3A11	cmp dl, ds:[ecx]	
0040F43B	✓ 75 1A	jnz short 0040F457	EFSUM.0040F457
0040F43D	. 84D2	test dl, dl	
0040F43F	✓ 74 12	je short 0040F453	EFSUM.0040F453
0040F441	. 8A50 01	mov dl, ds:[eax+1]	
0040F444	. 3A51 01	cmp dl, ds:[ecx+1]	
0040F447	✓ 75 0E	jnz short 0040F457	EFSUM.0040F457
0040F449	. 83C0 02	add eax, 2	
0040F44C	. 83C1 02	add ecx, 2	
0040F44F	. 84D2	test dl, dl	
0040F451	✓ 75 E4	jnz short 0040F437	EFSUM.0040F437
0040F453	> 33C0	xor eax, eax	
0040F455	✓ EB 05	jmp short 0040F45C	EFSUM.0040F45C
0040F457	> 1BC0	sbb eax, eax	

Es la zona de la función de lista negra. La reconozco enseguida por la cadena que se ve arriba para dar formato y el call que hay más abajo es seguro el que llama a _snprintf.

Me aseguro de pasar esa zona de comprobación cambiando el salto:

```
0040F43B | . /75 1A           || jnz      short 0040F457 ; EFSUM.0040F457
```

Por:

```
0040F43B     /EB 1A           jmp      short 0040F457 ; EFSUM.0040F457
```

Y el salto:

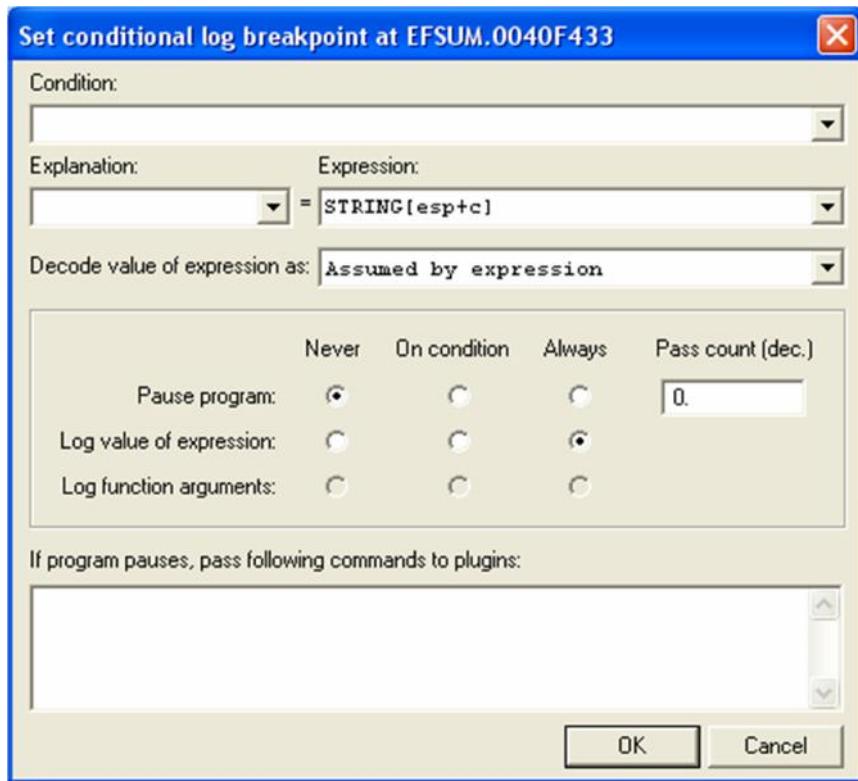
```
0040F4B4 | . /0F85 BA000000 | jnz      0040F574 ; EFSUM.0040F574
```

Por:

```
0040F4B4     /E9 BB000000   jmp      0040F574 ; EFSUM.0040F574
0040F4B9     | 90          nop      
```

Pongo un BP conditional Log en estas direcciones:

```
0040F433 | . 8D4424 0C      | lea      eax, ss:[esp+C]
0040F4A7 | . 8D4424 0C      lea      eax, ss:[esp+C]
```



Doy a "New origin here" en la línea:

```
0040F431 | . 8BCB          |mov      ecx, ebx
```

Limpio el log y pongo un BP en la linea:

```
0040F589 | . 5F           pop      edi
```

Doy a F9 tantas veces como sea necesario hasta llegar al BP anterior y después de borrar las líneas donde indica que ha parado en los HBPs queda así:

```
0040F433 COND: 803207-21086-90751233
0040F433 COND: 963740-22130-90860237
0040F433 COND: 093215-23287-90770242
0040F433 COND: 833974-91093-90750235
0040F433 COND: 913994-41175-98760235
0040F433 COND: 903904-51176-98760235
0040F433 COND: 003706-71286-90770237
0040F433 COND: 803201-02076-98751213
0040F433 COND: 873232-92089-90751233
0040F433 COND: 163841-82370-98780236
0040F433 COND: 963944-91170-98760235
0040F433 COND: 993914-61177-98760235
0040F433 COND: 973934-81179-98760235
0040F4A7 COND: 923385-12194-90760231
0040F4A7 COND: 963045-23100-90760244
0040F4A7 COND: 863240-12090-90751233
0040F4A7 COND: 933179-23103-90760243
0040F4A7 COND: 953453-01101-90761231
0040F4A7 COND: 953150-03191-90760233
```

Ahí tengo ya lista la lista negra.

Quito el BP y doy a F9 y para en la función que quita los guiones así que traceo y al salir veo que caigo en la zona donde va a comparar mi serial sin guiones con la parte del ID desencriptado y esta vez no hay coma. ¿Podría ser que la coma la ponga la función de lista negra?

Reinicio y repito todo el proceso hasta que llegue a la función de lista negra. Una vez allí pongo un BP en la salida de ambos bucles de búsqueda:

```
0040F453 |> \33C0          xor    eax, eax  
0040F4D4 |> \33C0          xor    eax, eax
```

¿Por qué en esas direcciones? Porque mi serial está en la lista negra con lo que saldré de uno de los dos bucles por esa parte y quiero ver que hace cuando me detecta.

Doy a F9 varias veces y para en el segundo así que traceo un poco y llego hasta aquí:

0040F579	> 85C0	test eax, eax	
0040F57B	v 74 24	je short 0040F5A1	EFSUM.0040F5A1
0040F57D	. 83C7 0C	add edi, 0C	
0040F580	. 833F 00	cmp dword ptr ds:[edi], 0	
0040F583	.^ 0F85 ECFFFE jnz 0040F475		EFSUM.0040F475
0040F589	. 5F	pop edi	
0040F58A	. 5E	pop esi	
0040F58B	. 5B	pop ebx	
0040F58C	. 8B8C24 400800 mov ecx, ss:[esp+840]		
0040F593	. 33CC	xor ecx, esp	
0040F595	. E8 861E0200 call 00431420		EFSUM.00431420
0040F59A	. 81C4 44080000 add esp, 844		
0040F5A0	. C3	retn	
0040F5A1	> A1 04134500 mov eax, ds:[451304]		
0040F5A6	. FE88 0C010000 dec byte ptr ds:[eax+10C]		
0040F5AC	. 68 00040000 push 400		
0040F5B1	. 8D5424 50 lea edx, ss:[esp+50]		
0040F5B5	. 52	push edx	
0040F5B6	. 68 74DF4400 push 44DF74		ASCII "You try to start a ilegal copy!"
0040F5BB	. E8 702B0100 call 00422130		EFSUM.00422130
0040F5C0	A1 D0124500 mov eax, ds:[4512D001]		

El salto me lleva a la zona donde me muestra el mensaje de que es una copia ilegal así que ese salto no tendría que cumplirse jamás. Si sigo traceando veo esto:

0040F5A1	> A1 04134500	mov eax, ds:[451304]	
0040F5A6	. FE88 0C010000	dec byte ptr ds:[eax+10C]	
0040F5AC	. 68 00040000	push 400	
0040F5B1	. 8D5424 50	lea edx, ss:[esp+50]	
0040F5B5	. 52	push edx	
0040F5B6	. 68 74DF4400	push 44DF74	ASCII "You try to start a ilegal copy!"
0040F5BB	. E8 702B0100	call 00422130	EFSUM.00422130

Apunta justo al primer guion del serial y al pasar esa línea con F8 me convierte el guión en una coma. Ese es el culpable jejeje.

Una vez pasado hago que el carácter vuelva a ser un guion cambiándolo en el Dump y doy a F9 y llego a la función que quita los guiones y, obviamente, ahora si podrá quitar ambos guiones con lo que pongo un BP en el retn y doy a F9 tantas veces como sea necesario hasta caer en el BP, lo quito y salgo de la función con F7.

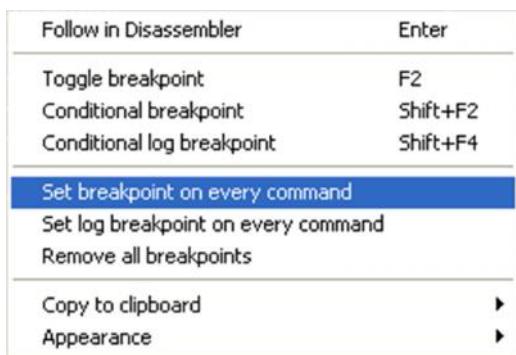
Traceo unas líneas y llego a la parte donde va a comparar mi serial sin guiones con una parte del ID con lo que lo de la coma es algo que hace en la función de lista negra para asegurarse de que no pasamos esta comparación. Ahora si paso la comparación sin problemas así que decido continuar con F9 y para nuevamente en la función que quita los guiones y al salir veo que estoy en el mismo sitio otra vez. Esto se repite muchas veces y, de repente, para y Olly se queda a la espera de algún nuevo evento así que minimizo y veo la ventana de chico malo que me indica que es una copia ilegal y al aceptarla se cierra el programa.

Como ya se cómo saltarme la comprobación de lista negra, reinicio Olly y pongo un HBP on Execution en la función descriptadora justo después del bucle que descripta y, cuando pare ahí, me voy a la función de lista negra y busco el call que llama a la función _snprintf y busco todas las referencias:

0040F41C	. 52	push	eax	
0040F41D	. 68 74CE4400	push	44CE74	ASCII "%06lu-%05lu-%08lu"
0040F422	. 8D4424 1C	lea	eax, ss:[esp+1C]	
0040F426	. 6A 3F	push	3F	
0040F428	. 50	push	eax	
0040F429	. E8 7D270200	call	00431BAB	EFSUM.00431BAB
0040F42E	. 83C4 18	add	esp, 18	



Y pongo un BP en todas ellas:



Con eso parará cada vez que intente crear una cadena con este método.

Ahora me voy a la línea 411C0D que es donde está el call que llama a strcmp y hago lo mismo, o sea, busco todas las referencias y pongo un BP en todas.

Me aseguro de que pase la comprobación de lista negra cambiando los dos saltos y doy a F9.

La primera vez que para es en la comparación entre el serial con guiones y parte del ID descriptado así que quito ese BP ya que lo paso sin problemas así que doy a F9 y ahora caigo aquí:

0040CAA6	. 51	push	ecx	
0040CAA7	. 68 28CE4400	push	44CE28	ASCII "%02lu"
0040CAAC	. 8D5424 4C	lea	edx, ss:[esp+4C]	
0040CAB0	. 6A 3F	push	3F	
0040CAB2	. 52	push	edx	
0040CAB3	. E8 F3500200	call	00431BAB	EFSUM.00431BAB
0040CAB8	. A1 04134500	mov	eax, ds:[4513041]	

No veo por ningún sitio a partir de que cadena realiza las operaciones para crear el valor que usará esta función así que pongo un BP al inicio de la función, reinicio Olly, doy a F9 y, cuando pare en el HBP que tengo en la función desencriptadora, habilito todos los BPs, cambio los saltos de la función de lista negra y doy a F9. Para en las dos llamadas a `_snprintf` que están en la función de lista negra así que esos dos BPs los quito y continuo con F9 y ahora me para aquí:

00402460	\$ 83EC 54	sub esp, 54		
00402463	. A1 F0FD4400	mov eax, ds:[44FDF0]		
00402468	. 33C4	xor eax, esp		
0040246A	. 894424 50	mov ss:[esp+50], eax		
0040246E	. 8D0424	lea eax, ss:[esp]		
00402471	. 50	push eax		
00402472	. FF15 78514400	call ds:[445178]		
00402478	. 8B4C24 58	mov ecx, ss:[esp+58]		
0040247C	. 51	push ecx		
0040247D	. 68 38CE4400	push 44CE38	ASCII "%04lu"	
00402482	. 8D5424 18	lea edx, ss:[esp+18]		
00402486	. 6A 3F	push 3F		
00402488	. 52	push edx		
00402489	. E8 1DF70200	call 00431BAB	EFSUM.00431BAB	
0040248E	. A1 04134500	mov eax, ds:[451304]		
00402493	. 8B88 D1010000	mov ecx, ds:[eax+1D1]		
00402499	. 83C4 10	add esp, 10		
0040249C	. 3B4C24 10	cmp ecx, ss:[esp+10]		
004024A0	.~ 74 59	je short 004024FB	EFSUM.004024FB	
004024A2	. OFB75424 02	movzx edx, word ptr ss:[esp+2]		

Es otro diferente al de antes y tampoco sé a partir de que cadena hace los cálculos así que pongo otro BP al inicio de la función, reinicio y repito el proceso y ahora para al inicio de esta última función así que traceo un poco y veo esto:

00402471	. 50	push eax		
00402472	. FF15 78514400	call ds:[445178]		
00402478	. 8B4C24 58	mov ecx, ss:[esp+58]		
0040247C	. 51	push ecx		
0040247D	. 68 38CE4400	push 44CE38	ASCII "%04lu"	
00402482	. 8D5424 18	lea edx, ss:[esp+18]		
00402486	. 6A 3F	push 3F		
00402488	. 52	push edx		
00402489	. E8 1DF70200	call 00431BAB	EFSUM.00431BAB	
0040248E	. A1 04134500	mov eax, ds:[451304]		

Stack ss:[0012DABC]=00000266
ecx=0000009C

Si entro en el call que acabo de pasar veo esto:

00A906D8	- E9 97A1D77B	jmp 7C80A874	kernel32.GetLocalTime
00A906DD	45	inc ebp	

No tengo ni idea de para qué hace eso ya que el valor que va a usar como parámetro en `_snprintf` no lo saca de ahí.

Sigo traceando hasta pasar la llamada a `_snprintf` y veo esto:

0012DA54	0012DA74	ASCII "8806"
0012DA58	0000003F	
0012DA5C	0044CE38	ASCII "%04lu"

O sea que es donde obtiene la cadena "8806" y que no cambia en ninguno de los programas que he visto de este desarrollador.

Si llego hasta el retn y salgo de la función veo esto:

0041425C	.	68 66220000	push	2266		
00414261	.	83CE FE	or	esi, FFFFFFFE		
00414264	.	E8 F7E1FEFF	call	00402460		EFSUM.00402460
00414269	.	8B5424 0C	mov	edx, ss:[esp+C]		
00414274	00	83C4 04	add	esp, 4		

O sea que el valor 0x2266, que es el que usa para crear la cadena "8806", es constante y la llamada a GetLocalTime no influye en nada ya que en la MSDN veo que solo usa un parámetro el cual es una estructura de tipo SYSTEMTIME:

```
typedef struct _SYSTEMTIME {
    WORD wYear;
    WORD wMonth;
    WORD wDayOfWeek;
    WORD wDay;
    WORD wHour;
    WORD wMinute;
    WORD wSecond;
    WORD wMilliseconds;
} SYSTEMTIME, *PSYSTEMTIME;
```

Y su tamaño total es de 16 bytes con lo que si reinicio y repito todo y miro la dirección del parámetro que introduce a la llamada a GetLocalTime me voy al Dump veo esto:

Address	Hex dump	ASCII
0012DA64	E9 4B A3 00 05 00 00 00 E6 D5 A3 00 71 37 41 00	éK£.0...æñ.ç7A.
0012DA74	1C DB 44 00 04 D8 A3 00 90 DA 12 00 D0 D4 A3 00	□ÜD.□ø£.□Ú.▫Ü£.
0012DA84	1C 0A 04 00 1C 0A 04 00 1E 39 41 00 00 00 00 00	□.□.□.□.□9A.....
0012DA94	39 35 33 34 35 33 30 31 31 30 31 39 30 37 36 31	9534530110190761
0012DAA4	32 33 31 00 26 E5 3A 77 53 8E 3A 7E 62 AF 3A 7E	231.å:wsž:~b^-:~
0012DAB4	09 24 C8 E4 69 42 41 00 66 22 00 00 1C 0A 04 00	.\$.ÈäiBA.f"..□.□.

Paso el call que llama a la función y veo esto:

Address	Hex dump	ASCII
0012DA64	E0 07 0A 00 01 00 1F 00 0D 00 0B 00 22 00 D4 01	à□..□.□...□.".ô□
0012DA74	1C DB 44 00 04 D8 A3 00 90 DA 12 00 D0 D4 A3 00	□ÜD.□ø£.□Ú.▫Ü£.
0012DA84	1C 0A 04 00 1C 0A 04 00 1E 39 41 00 00 00 00 00	□.□.□.□.□9A.....
0012DA94	39 35 33 34 35 33 30 31 31 30 31 39 30 37 36 31	9534530110190761
0012DAA4	32 33 31 00 26 E5 3A 77 53 8E 3A 7E 62 AF 3A 7E	231.å:wsž:~b^-:~
0012DAB4	09 24 C8 E4 69 42 41 00 66 22 00 00 1C 0A 04 00	.\$.ÈäiBA.f"..□.□.

Lo que está seleccionado es el tamaño que ocupa la estructura que usa GetLocalTime y lo que está rodeado de rojo es donde está la constante 2266 con lo que nunca podrá pisarla y no influye para nada en la obtención de la

cadena. No tengo ni idea de para que llama a GetLocalTime aquí ya que no veo que haga luego nada con los datos obtenidos con esta.

Si paso la parte donde crea la cadena “8806” y avanzo un poco veo esto:

0040247C	. 54	push	ecx	
0040247D	. 68 38CE4400	push	44CE38	ASCII "%04lu"
00402482	. 8D5424 18	lea	edx, ss:[esp+18]	
00402486	. 6A 3F	push	3F	
00402488	. 52	push	edx	
00402489	. E8 1DF70200	call	00431BAB	EFSUM.00431BAB
0040248E	. A1 04134500	mov	eax, ds:[451304]	
00402493	. 8B88 D1010000	mov	ecx, ds:[eax+ID1]	
00402499	. 83C4 10	add	esp, 10	
0040249C	. 3B4C24 10	cmp	ecx, ss:[esp+10]	
004024A0	.~ 74 59	je	short 004024FB	EFSUM.004024FB
004024A2	. 0FB75424 02	movzx	edx, word ptr ss:[esp+2]	
004024A7	. 0FB70424	movzx	eax, word ptr ss:[esp+1]	

ds:[00A34BC9]=36303838
ecx=36303838

Address	Hex dump	ASCII
00A34BA9	00 00 00 00 00 00 00 37 31 35 34 32 35 34 34 38715425448
00A34BB9	37 31 30 32 30 30 37 30 30 39 38 36 34 32 33 37	7102007009864237
00A34BC9	38 38 30 36 39 35 33 34 35 33 30 31 31 30 31 39	8806953453011019
00A34BD9	30 37 36 31 32 33 31 31 31 30 39 30 37 30 32 33	0761231110907023
00A34BE9	00 00 00 A4 FE 12 00 6C FE 12 00 88 FE 12 00 50	...xP.1p.^.p.P

O sea que va a comparar la cadena obtenida con esa parte de mi ID desencriptado y que empieza en la posición 26º del ID.

En mi caso coincide con lo que quito ese BP de arriba y doy a F9 y ahora para aquí:

0040CA20	\$ 81EC 80000000	sub	esp, 80	
0040CA26	. A1 F0FD4400	mov	eax, ds:[44FDF0]	
0040CA2B	. 33C4	xor	eax, esp	
0040CA2D	. 894424 7C	mov	ss:[esp+7C], eax	
0040CA31	. A1 04134500	mov	eax, ds:[451304]	
0040CA36	. 83B8 08020000	cmp	dword ptr ds:[eax+208],	
0040CA3D	.~ 0F84 91000000	je	0040CAD4	EFSUM.0040CAD4
0040CA43	. 56	push	esi	
0040CA44	. 57	push	edi	
0040CA45	. 05 B8010000	add	eax, 1B8	
0040CA4A	. 6A 39	push	39	
0040CA4C	. 50	push	eax	
0040CA4D	. 8D4424 10	lea	eax, ss:[esp+10]	
0040CA51	. 50	push	eax	
0040CA52	. E8 D9490200	call	00431430	EFSUM.00431430
0040CA57	. 8A0D 40F44400	mov	cl, ds:[44F440]	
0040CA5D	. 83C4 0C	add	esp, 0C	
0040CA60	. 884C24 08	mov	ss:[esp+8], cl	

Sigo traceando hasta que llego aquí:

0040CA57	. 8A0D 40F4440	mov cl, ds:[44F440]
0040CA5D	. 83C4 0C	add esp, 0C
0040CA60	. 884C24 08	mov ss:[esp+8], cl
0040CA64	. 33C0	xor eax, eax
0040CA66	. 33F6	xor esi, esi



```
cl=20 (' ')
Stack ss:[0012CBE4]=37 ('7')
```

Address	Hex dump	ASCII
0012CBE4	37 31 35 34 32 35 34 34 38 37 31 30 32 30 30 37	7154254487102007
0012CBEC	30 30 39 38 36 34 32 33 37 38 38 30 36 39 35 33	0098642378806953
0012CC0C	34 35 33 30 31 31 30 31 39 30 37 36 31 32 33 31	4530110190761231
0012CC1C	31 31 30 39 30 37 30 32 33 00 00 C0 4C A1 7C 00	110907023..ÀL;I.

Se puede ver que va a sustituir el primer carácter del ID desencriptado por un espacio.

Sigo traceando y veo como hace una serie de operaciones con todo el ID modificado con el espacio al inicio:

0040CA64	. 33C0	xor eax, eax	
0040CA66	. 33F6	xor esi, esi	
0040CA68	. B9 0E000000	mov ecx, 0E	
0040CA6D	. 33D2	xor edx, edx	
0040CA6F	. 90	nop	
0040CA70	> 0FB67C04 08	movzx edi, byte ptr ss:[esp+e]	
0040CA75	. 8D4C79 04	lea ecx, ds:[ecx+edi*2+4]	
0040CA79	. 0FB67C04 09	movzx edi, byte ptr ss:[esp+e]	
0040CA7E	. 8D547A 04	lea edx, ds:[edx+edi*2+4]	
0040CA82	. 0FB67C04 0A	movzx edi, byte ptr ss:[esp+e]	
0040CA87	. 83C0 03	add eax, 3	
0040CA8A	. 83F8 39	cmp eax, 39	
0040CA8D	. 8D747E 04	lea esi, ds:[esi+edi*2+4]	
0040CA91	.^ 7C DD	j1 short 0040CA70	EFSUM.0040CA70
0040CA93	. 03F2	add esi, edx	
0040CA95	. 03CE	add ecx, esi	
0040CA97	. B8 1F85EB51	mov eax, 51EB851F	
0040CA9C	. F7E1	mul ecx	
0040CA9E	. C1EA 05	shr edx, 5	
0040CAA1	. 6BD2 64	imul edx, edx, 64	
0040CAA4	. 2BCA	sub ecx, edx	
0040CAA6	. 51	push ecx	

Registers (MMX)

EAX	0A3D7748
ECX	00000048
EDX	00001770
EBX	0000003C
ESP	0012CBE4
EBP	00040A1C
ESI	00000FC8
EDI	00000033

En ECX tendrá el resultado de las operaciones el cual usará para crear la cadena.

Sigo traceando y llego a la comparacion:

0040CAB3	. E8 F3500200	call	00431BAB	
0040CAB8	. A1 04134500	mov	eax, ds:[451304]	
0040CABD	. 8A4C24 54	mov	cl, ss:[esp+54]	
0040CAC1	. B3C4 10	add	esp, 10	
0040CAC4	. 5F	pop	edi	
0040CAC5	. 5E	pop	esi	
0040CAC6	. 3A88 B8010000	cmp	cl, ds:[eax+1B8]	
0040CACC	.~ 74 06	je	short 0040CAD4	EFSUM.0040CAD4
0040CAE	. FE80 06010000	inc	byte ptr ds:[eax+106]	
0040CAD4	> 8B4C24 7C	mov	ecx, ss:[esp+7C]	



ds:[00A34BB0]=37 ('?')

cl=37 ('?')

Address	Hex dump	ASCII
00A34BB0	37 31 35 34 32 35 34 34 38 37 31 30 32 30 30 37	7154254487102007
00A34BC0	30 30 39 38 36 34 32 33 37 38 38 30 36 39 35 33	0098642378806953
00A34BD0	34 35 33 30 31 31 30 31 39 30 37 36 31 32 33 31	4530110190761231
00A34BE0	31 31 30 39 30 37 30 32 33 00 00 00 A4 FE 12 00	110907023...xp0.

Compara el primer carácter de la cadena obtenida con el primer carácter del ID desencriptado así que, como en mi caso coincide, quito el BP de arriba y el del inicio de la función y doy a F9.

Ahora para aquí:

00405349	> 51	push	ecx	
0040534A	. 68 38CE4400	push	44CE38	ASCII "%04lu"
0040534F	. 8D4424 0C	lea	eax, ss:[esp+C]	
00405353	. 6A 3F	push	3F	
00405355	. 50	push	eax	
00405356	. E8 50C80200	call	00431BAB	EFSUM.00431BAB
0040535B	. A1 04134500	mov	eax, ds:[451304]	

Y en los registros tengo esto:

Registers (MMX)	
EAX	0012AC9C ASCII "953453-01101-90761231"
ECX	0E23B920
EDX	0711DCD4
EBX	00000015
ESP	0012AC88
EBP	0015E9F0
ESI	0015E9F0
EDI	001309E6 UNICODE "6a7"

Y en el Stack tengo esto:

0012AC88	0012AC9C	ASCII "953453-01101-90761231"
0012AC8C	0000003F	
0012AC90	0044CE38	ASCII "%04lu"
0012AC94	0E23B920	
0012AC98	0012B514	UNICODE "Menu Inicio\Usuario Actual"

Está claro que la cadena que va a obtener es a partir de algunos cálculos sobre el serial así que miro arriba para ver qué cálculos realiza:

004052F0	\$ 83EC 44	sub	esp, 44	
004052F3	. A1 F0FD4400	mov	eax, ds:[44FDF0]	
004052F8	. 33C4	xor	eax, esp	
004052FA	. 894424 40	mov	ss:[esp+40], eax	
004052FE	. 8B0D 04134500	mov	ecx, ds:[451304]	
00405304	. 6A 40	push	40	
00405306	. 8D4424 04	lea	eax, ss:[esp+4]	
0040530A	. 50	push	eax	
0040530B	. 81C1 06010000	add	ecx, 106	
00405311	. 51	push	ecx	
00405312	. E8 D9D00100	call	004223F0	EFSUM.004223F0
00405317	. 8D4424 0C	lea	eax, ss:[esp+C]	
0040531B	. 83C4 0C	add	esp, 0C	
0040531E	. 8D50 01	lea	edx, ds:[eax+1]	
00405321	> 8A08	mov	cl, ds:[eax]	
00405323	. 40	inc	eax	
00405324	. 84C9	test	cl, cl	
00405326	.^ 75 F9	jnz	short 00405321	EFSUM.00405321
00405328	. 2BC2	sub	eax, edx	

Ahí lo que hace es obtener el serial y obtener su tamaño.
Si miro algo más abajo:

0040532A	. 53	push	ebx	
0040532B	. BB 00000000	mov	ebx, 0	
00405330	. 8BCB	mov	ecx, ebx	
00405332	.~ 74 15	je	short 00405349	EFSUM.00405349
00405334	> 0FB6541C 04	movzx	edx, byte ptr ss:[esp+el]	
00405339	. 8D1451	lea	edx, ds:[ecx+edx*2]	
0040533C	. 03D3	add	edx, ebx	
0040533E	. 03D3	add	edx, ebx	
00405340	. 43	inc	ebx	
00405341	. 8D4C11 02	lea	ecx, ds:[ecx+edx+2]	
00405345	. 3BD8	cmp	ebx, eax	
00405347	.^ 72 EB	jb	short 00405334	EFSUM.00405334
00405349	> 51	push	ecx	

Ahí están los cálculos que hace con la cadena del serial y si sigo traceando hasta llegar a la comparacion veo esto:

00405364	. 83C4 10	add esp, 10		
00405367	. 3A88 CE010000	cmp cl, ds:[eax+1CE]		
0040536D	.~ 75 18	jnz short 00405387	EFSUM.00405387	
0040536F	. 8A5424 05	mov dl, ss:[esp+5]		
00405373	. 3A90 CF010000	cmp dl, ds:[eax+1CF]		
00405379	.~ 75 0C	jnz short 00405387	EFSUM.00405387	
0040537B	. 8A4C24 06	mov cl, ss:[esp+6]		
0040537F	. 3A88 D0010000	cmp cl, ds:[eax+1D0]		
00405385	.~ 74 0F	je short 00405396	EFSUM.00405396	

ds:[00A34BC6]=32 ('2')
cl=32 ('2')

Address	Hex dump	ASCII
00A34BA6	00 00 00 00 00 00 00 00 00 00 37 31 35 34 32 35715425
00A34BB6	34 34 38 37 31 30 32 30 30 37 30 30 39 38 36 34	4487102007009864
00A34BC6	32 33 37 38 38 30 36 39 35 33 34 35 33 30 31 31	2378806953453011
00A34BD6	30 31 39 30 37 36 31 32 33 31 31 31 30 39 30 37	0190761231110907
00A34BE6	30 32 33 00 00 00 A4 FE 12 00 6C FE 12 00 88 FE	023...xp0.1p0.^p

O sea que compara los 3 primeros caracteres de la cadena obtenida con los caracteres 23º, 24º y 25º del ID desencriptado. En mi caso coinciden así que quito el BP de arriba y doy a F9 y ahora para aquí:

0040D67D	. 56	push esi		
0040D67E	. 68 30CE4400	push 44CE30	ASCII "%3lu"	
0040D683	. 8D4424 14	lea eax, ss:[esp+14]		
0040D687	. 6A 7F	push 7F		
0040D689	. 50	push eax		
0040D68A	. E8 1C450200	call 00431BAB	EFSUM.00431BAB	
0040D68F	. A1 04134500	mov eax, ds:[451304]		

No hay ni rastro de ninguna cadena a partir de la que haya podido hacer los cálculos para obtener el valor así que miro más arriba y veo esto:

0040D624	> 8A08	mov cl, ds:[eax]		
0040D626	. 40	inc eax		
0040D627	. 84C9	test cl, cl		
0040D629	.^ 75 F9	jnz short 0040D624	EFSUM.0040D624	
0040D62B	. 2BC2	sub eax, edx		
0040D62D	. 8BC8	mov ecx, eax		

Esa parte es un bucle que mide el largo de la cadena y a continuación veo esto:

0040D62F	. 33C0	xor	eax, eax	
0040D631	. 83F9 02	cmp	ecx, 2	
0040D634	.~ 7C 24 j1	short 0040D65A		EFSUM.0040D65A
0040D636	. 8D51 FF	lea	edx, ds:[ecx-1]	
0040D639	. 55	push	ebp	
0040D63A	. 8D9B 00000000	lea	ebx, ds:[ebx]	
0040D640	> 0FB66C04 10	movzx	ebp, byte ptr ss:[esp+e]	
0040D645	. 03E9	add	ebp, ecx	
0040D647	. 03FD	add	edi, ebp	
0040D649	. 0FB66C04 11	movzx	ebp, byte ptr ss:[esp+e]	
0040D64E	. 03E9	add	ebp, ecx	
0040D650	. 83C0 02	add	eax, 2	
0040D653	. 03DD	add	ebx, ebp	
0040D655	. 3BC2	cmp	eax, edx	
0040D657	.^ 72 E7 jb	short 0040D640		EFSUM.0040D640
0040D659	. 5D	pop	ebp	
0040D65A	> 3BC1	cmp	eax, ecx	
0040D65C	.~ 73 09 jnb	short 0040D667		EFSUM.0040D667
0040D65E	. 0FB65404 0C	movzx	edx, byte ptr ss:[esp+e]	
0040D663	. 8D740A 17	lea	esi, ds:[edx+ecx+17]	
0040D667	> 03DF	add	ebx, edi	
0040D669	. 03F3	add	esi, ebx	
0040D66B	. B8 D34D6210	mov	eax, 10624DD3	
0040D670	. F7E6	mul	esi	
0040D672	. C1EA 06	shr	edx, 6	
0040D675	. 69D2 E8030000	imul	edx, edx, 3E8	
0040D67B	. 2BF2	sub	esi, edx	
0040D67D	. 5F	push	esi	

Comprueba si la cadena es mayor o igual a 2 y si no es así salta la mayor parte de los cálculos. Eso ya lo he visto antes en los demás productos de este desarrollador y lo usa para los cálculos con la cadena de la compañía así que quito el BP y voy traceando hasta la comparación:

0040D69E	. 3A88 EE010000	cmp	cl, ds:[eax+1EE]	
0040D6A4	.~ 75 18 jnz	short 0040D6BE		EFSUM.0040D6BE
0040D6A6	. 8A5424 01	mov	dl, ss:[esp+1]	
0040D6AA	. 3A90 EF010000	cmp	dl, ds:[eax+1EF]	
0040D6B0	.~ 75 0C jnz	short 0040D6BE		EFSUM.0040D6BE
0040D6B2	. 8A4C24 02	mov	cl, ss:[esp+2]	
0040D6B6	. 3A88 F0010000	cmp	cl, ds:[eax+1F0]	
0040D6BC	.~ 74 OF je	short 0040D6CD		EFSUM.0040D6CD
0040D6BE	> 80B8 F0010000	cmp	byte ptr ds:[eax+1F0], 0	
0040D6C5	.~ 74 06 je	short 0040D6CD		EFSUM.0040D6CD
0040D6C7	.FF80 00010000 inc	byte ptr ds:[eax+1F0]		

ds:[00A34BE6]=30 ('0')

cl=30 ('0')

Address	Hex dump	ASCII
00A34BA6	00 00 00 00 00 00 00 00 00 00 37 31 35 34 32 35715425
00A34BB6	34 34 38 37 31 30 32 30 30 37 30 30 39 38 36 34	4487102007009864
00A34BC6	32 33 37 38 38 30 36 39 35 33 34 35 33 30 31 31	2378806953453011
00A34BD6	30 31 39 30 37 36 31 32 33 31 31 31 30 39 30 37	0190761231110907
00A34BE6	30 32 33 00 00 00 A4 FE 12 00 6C FE 12 00 88 FE	023...xp.1p.^p

Va a comparar los tres primeros caracteres de la cadena obtenida con los caracteres 55°, 56° y 57° del ID desencriptado. En mi caso coinciden así que quito el BP, doy a F9 y para aquí:

00408790	. 57	push	edi	
00408791	. 68 38CE4400	push	44CE38	ASCII "%04lu"
00408796	. 8D4424 18	lea	eax, ss:[esp+18]	
0040879A	. 68 FF030000	push	3FF	
0040879F	. 50	push	eax	
004087A0	. E8 06940200	call	00431BAB	EFSUM.00431BAB
004087A5	A1 04134500	mov	eax, ds:[4513041]	

Y en el registro:

Registers (MMX)	<	<	<	<
EAX 00126830 ASCII "Roland Geister953453-01101-90761231"				
ECX 00000133				
EDX 000000C0				
EBX 0000003C				
ESP 00126810				
EBP 00100360				
ESI 000010D0				
EDI 00002371				

Y en el Stack:

00126810	00126830	ASCII "Roland Geister953453-01101-90761231"
00126814	000003FF	
00126818	0044CE38	ASCII "%04lu"

Según lo que vi en otros productos del desarrollador, usa la concatenación “nombre+serial+compañía” para realizar los cálculos y esta es la zona donde los realiza:

00408732	. OFB64C24 14	movzx	ecx, byte ptr ss:[esp+14]	
00408737	. 53	push	ebx	
00408738	. 4A	dec	edx	
00408739	. 55	push	ebp	
0040873A	. 8D9B 00000000	lea	ebx, ds:[ebx]	
00408740	> 0FB65C04 18	movzx	ebx, byte ptr ss:[esp+e	
00408745	. 8BE8	mov	ebp, eax	
00408747	. D1ED	shr	ebp, 1	
00408749	. 03E9	add	ebp, ecx	
0040874B	. 8D5C5D 00	lea	ebx, ss:[ebp+ebx*2]	
0040874F	. 03F3	add	esi, ebx	
00408751	. OFB65C04 19	movzx	ebx, byte ptr ss:[esp+e	
00408756	. 8D68 01	lea	ebp, ds:[eax+1]	
00408759	. D1ED	shr	ebp, 1	
0040875B	. 03E9	add	ebp, ecx	
0040875D	. 8D5C5D 00	lea	ebx, ss:[ebp+ebx*2]	
00408761	. 83C0 02	add	eax, 2	
00408764	. 03FB	add	edi, ebx	
00408766	. 3BC2	cmp	eax, edx	
00408768	.^ 72 D6	jb	short 00408740	EFSUM.00408740
0040876A	. 8B4C24 14	mov	ecx, ss:[esp+14]	
0040876E	. 8B5424 10	mov	edx, ss:[esp+10]	
00408772	. 5D	pop	ebp	
00408773	. 5B	pop	ebx	
00408774	> 3BC2	cmp	eax, edx	
00408776	.~ 73 14	jnb	short 0040878C	EFSUM.0040878C
00408778	. OFB65404 10	movzx	edx, byte ptr ss:[esp+ea	
0040877D	. D1E8	shr	eax, 1	
0040877F	. 8D0450	lea	eax, ds:[eax+edx*2]	
00408782	. OFB65424 14	movzx	edx, byte ptr ss:[esp+14]	
00408787	. 03D1	add	edx, ecx	
00408789	. 8D0C02	lea	ecx, ds:[edx+eax]	
0040878C	> 03FE	add	edi, esi	
0040878E	. 03F9	add	edi, ecx	

Si llego a la comparación veo que va a comparar los 3 primeros caracteres de la cadena obtenida con los caracteres 52º, 53º y 54º del ID desencriptado. En mi caso coinciden así que quito el BP de arriba, doy a F9 y para aquí:

004135F3	. 83B8 08020000	cmp	dword ptr ds:[eax+208],	
004135FA	.~ 74 47	je	short 00413643	EFSUM.00413643
004135FC	. 8B80 08020000	mov	eax, ds:[eax+208]	
00413602	. 50	push	eax	
00413603	. E8 786CFFFF	call	0040A280	EFSUM.0040A280
00413608	. 50	push	eax	
00413609	. 68 60CE4400	push	44CE60	ASCII "%09lu"
0041360E	. 8D4C24 0C	lea	ecx, ss:[esp+C]	
00413612	. 6A 3F	push	3F	
00413614	. 51	push	ecx	
00413615	. E8 91E50100	call	00431BAB	EFSUM.00431BAB
0041361A	. A1 04134500	mov	eax, ds:[451304]	
0041361F	. 6A 09	push	9	
00413621	. 8D5424 18	lea	edx, ss:[esp+18]	
00413625	. 52	push	edx	
00413626	. 05 B9010000	add	eax, 1B9	
0041362B	. 50	push	eax	
0041362C	. E8 C0E60100	call	00431CF1	EFSUM.00431CF1

No veo que cadena ha usado así que pongo un BP al inicio de la función y reinicio y repito todo el proceso hasta que pare en el BP que acabo de poner y voy traceando hasta que llego aquí:

004135FC	. 8B80 08020000	mov	eax, ds:[eax+208]	
00413602	. 50	push	eax	
00413603	. E8 786CFFFF	call	0040A280	EFSUM.0040A280
00413608	. 50	push	eax	
00413609	. 68 60CE4400	push	44CE60	ASCII "%09lu"
0041360E	. 8D4C24 0C	lea	ecx, ss:[esp+C]	
00413612	. 6A 3F	push	3F	

Si miro a donde apunta EAX veo esto:

Address	Hex dump	ASCII
00A3D4D0	45 46 20 53 74 61 72 74 55 70 20 4D 61 6E 61 67	EF StartUp Manag
00A3D4E0	65 72 20 64 69 73 74 72 69 62 75 74 69 6F 6E 20	er distribution
00A3D4F0	6C 69 63 65 6E 73 65 0D 0A 0D 0A 43 6F 70 79 72	license....Copyr
00A3D500	69 67 68 74 20 28 63 29 20 31 39 39 34 2D 32 30	ight (c) 1994-20
00A3D510	30 37 2C 20 45 6D 69 6C 20 46 69 63 6B 65 6C 0D	07, Emil Fickel.
00A3D520	0A 41 6C 6C 20 72 69 67 68 74 73 20 72 65 73 65	.All rights rese
00A3D530	72 76 65 64 2E 0D 0A 0D 0A 54 68 69 73 20 63 6F	rved.....This co
00A3D540	70 79 20 69 73 20 6C 69 63 65 6E 73 65 64 20 74	py is licensed t
00A3D550	6F 3A 0D 0A 4E 61 6D 65 20 20 20 20 20 3A 20	o:..Name :
00A3D560	52 6F 6C 61 6E 64 20 47 65 69 73 74 65 72 0D 0A	Roland Geister..
00A3D570	43 6F 6D 70 61 6E 79 20 20 20 3A 20 0D 0A 53 65	Company : ..Se
00A3D580	72 69 61 6C 20 4E 6F 2E 3A 20 39 35 33 34 35 33	rial No.: 953453
00A3D590	2D 30 31 31 30 31 2D 39 30 37 36 31 32 33 31 0D	-01101-90761231.
00A3D5A0	0A 52 65 67 2E 49 44 2E 20 20 20 3A 20 35 33 34	.Reg.ID. : 534
00A3D5B0	35 33 30 31 31 30 31 39 30 37 36 31 32 33 31 30	5301101907612310
00A3D5C0	31 31 30 31 39 30 37 36 31 32 33 31 31 31 30 31	1101907612311101
00A3D5D0	39 30 37 36 31 32 33 31 31 31 30 31 39 30 37 36	9076123111019076
00A3D5E0	31 32 33 31 31 31 0D 0A 0D 0A 57 61 72 6E 69 6E	123111....Warnin
00A3D5F0	62 31 20 54 68 60 72 30 32 65 64 35 62 31 20	g: This product

Es el buffer del archivo de licencia con el ID sustituido por las sucesiones de seriales así que entro en el call y veo esto:

0040A280	\$ 56	push	esi	
0040A281	. 8B7424 08	mov	esi, ss:[esp+8]	
0040A285	. 85F6	test	esi, esi	
0040A287	.~ 74 69	je	short 0040A2F2	EFSUM. 0040A2F2
0040A289	. 8A06	mov	al, ds:[esi]	
0040A28B	. 57	push	edi	
0040A28C	. 33FF	xor	edi, edi	
0040A28E	. 84C0	test	al, al	
0040A290	.~ 74 0D	je	short 0040A29F	EFSUM. 0040A29F
0040A292	> 3C 1A	cmp	al, 1A	
0040A294	.~ 74 09	je	short 0040A29F	EFSUM. 0040A29F
0040A296	. 8A4437 01	mov	al, ds:[edi+esi+1]	
0040A29A	. 47	inc	edi	
0040A29B	. 84C0	test	al, al	
0040A29D	.^ 75 F3	jnz	short 0040A292	EFSUM. 0040A292

Ahí lo que hace es ir recorriendo el buffer hasta llegar al final o hasta que encuentre el carácter 0x1A el cual no lo va a encontrar ya que mi archivo solo tiene caracteres imprimibles con lo que de ahí saldrá con el largo del archivo.

Si miro debajo veo esto:

0040A2A0	. 33C9	xor	ecx, ecx	
0040A2A2	. 8BC7	mov	eax, edi	
0040A2A4	. 33DB	xor	ebx, ebx	
0040A2A6	. 894424 10	mov	ss:[esp+10], eax	
0040A2AA	. 85FF	test	edi, edi	
0040A2AC	.~ 76 40	jbe	short 0040A2EE	EFSUM. 0040A2EE
0040A2AE	. 55	push	ebp	
0040A2AF	. 8D6F 01	lea	ebp, ds:[edi+1]	
0040A2B2	> 0FB60431	movzx	eax, byte ptr ds:[ecx+e]	
0040A2B6	. 8D5408 07	lea	edx, ds:[eax+ecx+7]	
0040A2BA	. 33DA	xor	ebx, edx	
0040A2BC	. 33D2	xor	edx, edx	
0040A2BE	. 8BC3	mov	eax, ebx	
0040A2C0	. F7F5	div	ebp	
0040A2C2	. 42	inc	edx	
0040A2C3	. 3BD7	cmp	edx, edi	
0040A2C5	.~ 76 05	jbe	short 0040A2CC	EFSUM. 0040A2CC
0040A2C7	. BA 0C000000	mov	edx, 0C	
0040A2CC	> 0B4424 14	mov	eax, ss:[esp+14]	
0040A2D0	. 0FB61432	movzx	edx, byte ptr ds:[edx+e]	
0040A2D4	. 83C0 03	add	eax, 3	
0040A2D7	. OFAFC3	imul	eax, ebx	
0040A2DA	. 03C2	add	eax, edx	
0040A2DC	. 0FB61431	movzx	edx, byte ptr ds:[ecx+e]	
0040A2E0	. 41	inc	ecx	
0040A2E1	. 8D4450 0B	lea	eax, ds:[eax+edx*2+B]	
0040A2E5	. 894424 14	mov	ss:[esp+14], eax	
0040A2E9	. 3BCF	cmp	ecx, edi	
0040A2EB	.^ 72 C5	jb	short 0040A2B2	EFSUM. 0040A2B2
0040A2ED	. SD	non	ehn	

Son los cálculos que realizará con este buffer para obtener el valor con el que creará la cadena.
Sigo traceando y llego aquí:

00413614	. 51	push	ecx	
00413615	. E8 91E50100	call	00431BAB	EFSUM.00431BAB
0041361A	. A1 04134500	mov	eax, ds:[451304]	
0041361F	. 6A 09	push	9	
00413621	. 8D5424 18	lea	edx, ss:[esp+18]	
00413625	. 52	push	edx	
00413626	. 05 B9010000	add	eax, 1B9	
0041362B	. 50	push	eax	
0041362C	. E8 COE60100	call	00431CF1	EFSUM.00431CF1
00413631	. 83C4 20	add	esp, 20	
00413634	. 85C0	test	eax, eax	
00413636	.~ 74 0B	je	short 00413643	EFSUM.00413643
00413638	. A1 04134500	mov	eax, ds:[451304]	

0012CCD8	00A34BB1	ASCII "1542544871020070098642378806953453011019
0012CCDC	0012CCF8	ASCII "1542544877"
0012CCEO	00000009	

Si miro en el Dump:

Address	Hex dump	ASCII
00A34BA1	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 377
00A34BB1	31 35 34 32 35 34 34 38 37 31 30 32 30 30 37 30	1542544871020070
00A34BC1	30 39 38 36 34 32 33 37 38 38 30 36 39 35 33 34	0986423788069534
00A34BD1	35 33 30 31 31 30 31 39 30 37 36 31 32 33 31 31	5301101907612311
00A34BE1	31 30 39 30 37 30 32 33 00 00 00 A4 FE 12 00 6C	10907023...Mp0.1

O sea que va a comparar los primeros nueve caracteres de la cadena obtenida con los nueve que hay a partir del segundo del ID desencriptado. En mi caso coinciden así que quito los dos BPs, doy a F9 y para aquí:

00402657	> 8A08	mov cl, ds:[eax]		
00402659	. 40	inc eax		
0040265A	. 84C9	test cl, cl		
0040265C	.^ 75 F9	jnz short 00402657		EFSUM.00402657
0040265E	. 2BC2	sub eax, edx		
00402660	. 33C9	xor ecx, ecx		
00402662	. 8D51 0C	lea edx, ds:[ecx+C]		
00402665	> 3BC8	cmp ecx, eax		
00402667	.~ 73 0F	jnb short 00402678		EFSUM.00402678
00402669	. 0FB6740D 00	movzx esi, byte ptr ss:[ebp+e		
0040266E	. 03F1	add esi, ecx		
00402670	. 8D1456	lea edx, ds:[esi+edx*2]		
00402673	. 03D0	add edx, eax		
00402675	. 41	inc ecx		
00402676	.^ EB ED	jmp short 00402665		EFSUM.00402665
00402678	> 52	push edx		
00402679	. 68 40CE4400	push 44CE40		ASCII "%05lu"
0040267E	. 8D55 00	lea edx, ss:[ebp]		
00402681	. 6A 7F	push 7F		
00402683	. 52	push edx		
00402684	. E8 22F50200	call 00431BAB		EFSUM.00431BAB
00402689	. A1 04134500	mov eax, ds:[451304]		

Registers (MMX)					
EAX	0000000E				
ECX	0000000E				
EDX	0012CA7C	ASCII "Roland Geister"			
EBX	0000003C				
ESP	0012C84C				
EBP	0012CA7C	ASCII "Roland Geister"			
ESI	0000007F				
		0012C84C	0012CA7C	ASCII "Roland Geister"	
		0012C850	0000007F		
		0012C854	0044CE40	ASCII "%05lu"	
		0012C858	001E4F77		

O sea que va a usar el nombre de registro para crear el valor y si sigo traceando hasta la comparación veo esto:

Address	Hex dump	ASCII
00A34BA2	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 37 3171
00A34BB2	35 34 32 35 34 34 38 37 31 30 32 30 30 37 30 30	5425448710200700
00A34BC2	39 38 36 34 32 33 37 38 38 30 36 39 35 33 34 35	9864237880695345
00A34BD2	33 30 31 31 30 31 39 30 37 36 31 32 33 31 31 31	3011019076123111
00A34BE2	30 39 30 37 30 32 33 00 00 A4 FE 12 00 6C FE	0907023...mp0.lp

Va a comparar los caracteres 2º, 3º y 4º de la cadena obtenida con los caracteres 19º, 20º y 21º. Como en mi caso coinciden, quito el BP de arriba, doy a F9 y ya no para en ninguno que sea importante con lo que pruebo a dar en todos los botones y opciones pero no pasa nada más con lo que doy por terminada la fase de búsqueda de comprobaciones.

Así queda el ID desencriptado:

Parte obtenida con los cálculos con el ID iniciado con espacio.
 Parte obtenida con los cálculos con el ID relleno de seriales concatenados.
 Parte obtenida con los cálculos con el nombre.
 Parte obtenida con los cálculos con el serial.
 La parte del 8806.
 Es el serial sin guiones.
 Parte obtenida con los cálculos con la cadena obtenida de la concatenación "nombre+serial+compañía".
 Parte obtenida con los cálculos con el nombre de compañía.

715425448710200700986423788069534530110190761231110907023

Con esto ya tengo todo lo necesario para crear mi keygen y me valgo para ello del proyecto del EF Find el cual se parece mucho pero solo cambian los algoritmos que usa para calcular los valores.

Y así quedó el código del keygen:

```
//-----
#include <vcl.h>
#include <stdio.h>
#include <fstream.h>
#pragma hdrstop

#include "Unit1.h"
//-----
#pragma package(smart_init)
#pragma resource "* .dfm"
TForm1 *Form1;

const char cadenaEncabezado[] = "EF StartUp Manager distribution license\x0D\x0A\x0D\x0A"
                                "Copyright (c) 1994-2007, Emil Fickel\x0D\x0A"
                                "All rights reserved.\x0D\x0A\x0D\x0A"
                                "This copy is licensed to:\x0D\x0A";
const char cadenaNombre[] = "Name      : ";
const char cadenaCompania[] = "Company   : ";
const char cadenaSerial[] = "Serial No.: ";
const char cadenaID[] = "Reg.ID.   : ";
const char cadenaPie[] = "\x0D\x0AWarning: This product is licensed to you pursuant to
the terms of the\x0D\x0A"
                                "author license agreement included with the original
software.\x0D\x0A"
                                "It is protected by copyright law and international
treaties.\x0D\x0A"
                                "Unauthorized reproduction or distribution may result in severe
civil and\x0D\x0A"
                                "criminal penalties, and will be prosecuted to the maximum
extent possible\x0D\x0A"
                                "under the law.\x0D\x0A\x0D\x0A"
                                "One registered copy of this software may be dedicated to a
single\x0D\x0A"
                                "person who uses the software on one or more computers or to a
single\x0D\x0A"
                                "workstation used by multiple people.\x0D\x0A";
const char salto[] = "\x0D\x0A";
//-----

_fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----

AnsiString Encriptar(AnsiString cadena)
{
    DWORD valor;
    DWORD contador;
    DWORD aux;

    contador = 0;

    while(contador < cadena.Length()){


```

```

        valor = cadena[contador+1];
        if (valor >= 0x30) {
            valor += 0x41 - 0x30;
            cadena[contador+1]=(char)valor;
        } else {
            aux = contador;
            aux &= 0x80000003;
            if(aux < 0){
                aux--;
                aux |= 0xfffffffffc;
                aux++;
            }
            valor += aux;
            cadena[contador+1] = (char)valor;
        }
        contador++;
    }
    return cadena;
}
//-----
AnsiString Desencriptar(AnsiString cadena)
{
    DWORD valor;
    DWORD contador;
    DWORD aux;

    contador = 0;

    while(contador < cadena.Length()){
        valor = cadena[contador+1];
        if (valor >= 0x40) {
            valor -= 0x41;
            valor = valor % 0xA;
            valor += 0x30;
            cadena[contador+1] = (char)valor;
        } else {
            aux = contador;
            aux &= 0x80000003;
            if(aux < 0){
                aux--;
                aux |= 0xfffffffffc;
                aux++;
            }
            valor -= aux;
            cadena[contador+1] = (char)valor;
        }
        contador++;
    }
    return cadena;
}
//-----
AnsiString CalcularValorSerial(AnsiString serial)
{
    unsigned long retval = 0;
    int contador;

    for(contador = 0; contador < serial.Length(); contador++){
        retval += serial[contador + 1] * 2 + retval + (contador * 2) + 2;
    }
}

```

```

        return AnsiString().sprintf("%04lu", retval);
    }
//-----

AnsiString CalcularValorNombre(AnsiString nombre)
{
    unsigned long total = 0xC;
    int contador = 0;

    while ( contador < nombre.Length() )
    {
        total = nombre.Length() + contador + nombre[contador + 1] + 2 * total;
        ++contador;
    }
    return AnsiString().sprintf("%05lu", total);
}
//-----
```

```

AnsiString ObtenerValorCompania(AnsiString compania)
{
    unsigned long valor1=0, valor2=0, retval=0x17;
    int pos=0;

    if(compania.Length() >= 2){
        while((compania.Length() - pos) > 1){
            valor1 += compania[pos + 1] + compania.Length();
            pos++;
            valor2 += compania[pos + 1] + compania.Length();
            pos++;
        }
    }

    if(compania.Length() - pos == 1)
        retval += (compania[pos + 1] + compania.Length());

    retval += (valor1 + valor2);

    //Obtengo la parte alta de la multiplicación
    valor1 = ((__int64)retval * 0x10624DD3 >> 32);

    valor1 = valor1 >> 6;
    retval -= (valor1 * 0x3E8);
    return AnsiString().sprintf("%03lu", retval);
}
//-----
```

```

AnsiString CalcularValorLIC(char *cadena,int sizebuffer)
{
    unsigned long valor1 = 0, valor2, valor3, valor4 = sizebuffer;
    unsigned long retval;
    int pos;
    int contador = 0;

    do
    {
        valor1 ^= cadena[contador] + contador + 7;
        pos = valor1 % (sizebuffer + 1) + 1;
        if ( pos >= sizebuffer )
            pos = 12;
        valor2 = cadena[pos] + valor1 * (valor4 + 3);
```

```

        valor3 = cadena[contador++];
        retval = valor2 + 2 * valor3 + 11;
        valor4 = retval;
    }
    while ( contador < sizebuffer );

    return AnsiString().sprintf("%09lu", retval);
}
//-----

AnsiString CalcularValorID(AnsiString id)
{
    unsigned long valor1 = 0xE, valor2=0, valor3=0, valor4, total;
    int contador = 0;

    id[1] = 0x20;

    do
    {
        valor1 += 2 * id[contador+1] + 4;
        valor2 += 2 * id[contador+2] + 4;
        valor4 = id[contador+3];
        contador += 3;
        valor3 += 2 * valor4 + 4;
    }while ( contador < 57 );

    total = valor1 + valor2 + valor3;
    valor4 = (((__int64)total * 0x51EB851F) >> 32);
    valor4 = valor4 >> 5;
    valor4 *= 0x64;
    total -= valor4;

    return AnsiString().sprintf("%02lu", total);
}
//-----

//Limpiar buffer
void LimpiarBuffer(char *buffer,int sizeBuffer)
{
    for(int i = 0; i < sizeBuffer;buffer[i] = '\0', i++);
}
//-----
```



```

AnsiString ValorDeConcatenados(AnsiString cadena)
{
    unsigned long retval = cadena[1];
    unsigned long valor1 = 0, valor2=0;
    int contador = 0;

    do{
        valor1 += cadena[5] + (contador >> 1) + 2 * cadena[contador + 1];
        valor2 += cadena[5] + ((contador + 1) >> 1) + 2 * cadena[contador + 2];
        contador += 2;
    }while(contador < cadena.Length() - 1);

    if(contador < cadena.Length() )
        retval += cadena[5] + (contador >> 1) + 2 * cadena[contador + 1];
    retval += valor1 + valor2;
    return AnsiString().sprintf("%03lu", retval);
}
//-----
```

```

void __fastcall TForm1::ButtonCalcularClick(TObject *Sender)
{
    AnsiString id, aux;
    unsigned long serialP1, serialP2, serialP3;

    //Si ya hay memoria asignada al buffer la libero antes de nada
    if(bufferLleno){
        delete bufferArchivo;
        bufferArchivo = NULL;
        bufferLleno = false;
    }

    do{
        //Obtengo las 3 partes del serial con valores aleatorios
        serialP1 = random(999999);
        serialP2 = random(99999);
        serialP3 = random(99999999);

        EditSerial->Text = AnsiString().sprintf("%06lu-%05lu-%08lu", serialP1, serialP2,
        serialP3);
        }while(listaNegra->IndexOf(EditSerial->Text) != -1); //Si el serial está en la lista
negra creo otro diferente

        //concateno las partes del serial sin guiones
        aux = AnsiString().sprintf("%06lu%05lu%08lu", serialP1, serialP2, serialP3);

        //Relleno el ID con una sucesión del serial pero empezando siempre por el segundo
        //carácter hasta llegar a 57 caracteres
        id="";
        for(int x=0, pos=1; x < 57; x++, pos++){
            if(pos >= aux.Length())
                pos = (x & 3) + 4;
            id += (char)aux[pos+1];
        }

        //Obtengo el tamaño que ocupará todo el texto
        sizeBuffer = snprintf(NULL, 0, "%s%s%s%s%s%s%s%s%s%s%s",
                               cadenaEncabezado,
                               cadenaNombre, EditNombre->Text.c_str(), salto,
                               cadenaCompania, EditCompania->Text.c_str(), salto,
                               cadenaSerial, EditSerial->Text.c_str(), salto,
                               cadenaID, id.c_str(), salto,
                               cadenaPie);
        sizeBuffer++;

        //Reservo la memoria necesaria para el buffer
        bufferArchivo = new char[sizeBuffer];

        //Limpio el buffer
        LimpiarBuffer(bufferArchivo, sizeBuffer);

        //Concateno todo como lo requiere la operación para obtener el valor sobre el texto
        //de la licencia. Viene a ser como el original pero usando el ID con la concatenación
        //de seriales sin guiones
        snprintf(bufferArchivo, sizeBuffer-1, "%s%s%s%s%s%s%s%s%s%s",
                 cadenaEncabezado,
                 cadenaNombre, EditNombre->Text.c_str(), salto,
                 cadenaCompania, EditCompania->Text.c_str(), salto,
                 cadenaSerial, EditSerial->Text.c_str(), salto,
                 cadenaID, id.c_str(), salto,

```

```

cadenaPie);

//Creo un ID con caracteres aleatorios desde el carácter 0x21 que es '!' hasta
//el carácter 0x39 que es '9' y como 0x39 - 0x21 = 0x18 uso ese valor para crear
//el valor aleatorio y luego le sumo el valor del primero para obtener un carácter
//dentro de ese rango. Tiene que ser ese rango porque, al desencriptar el ID,
//cualquier ID no se sale de este rango y si ponemos un rango mayor no funcionará
id = "";
for(int i = 0; i < 57; i++){
    id += (char)(random(0x18) + 0x21);
}

//Desencripto el ID
id = Desencriptar(id);

//Concateno las partes del serial sin guiones
aux = AnsiString().sprintf("%06lu%05lu%08lu", serialP1, serialP2, serialP3);

//Coloco el serial sin guiones en el ID desencriptado
id = id.SubString(1,29) + aux + id.SubString(49, id.Length() - 48);

//Obtengo la cadena con el valor del nombre introducido
aux = CalcularValorNombre(EditNombre->Text);

//Coloco los caracteres 2, 3, y 4 de la cadena obtenida en el ID desencriptado
id = id.SubString(1, 18) + aux.SubString(2, 3) + id.SubString(22, id.Length() - 21);

//Obtengo la parte del "8806"
aux = AnsiString().sprintf("%04lu", 0x2266);

//La coloco en el ID
id = id.SubString(1, 25) + aux.SubString(1, 4) + id.SubString(30, id.Length() - 29);

//Obtengo el valor a partir del serial
aux = CalcularValorSerial(EditSerial->Text);

//Coloco los 3 primeros caracteres del valor obtenido en el ID
id = id.SubString(1, 22) + aux.SubString(1, 3) + id.SubString(26, id.Length() - 25);

//Obtengo el valor a partir de la compañía
aux = ObtenerValorCompania(EditCompania->Text);

//La coloco en su posición
id = id.SubString(1, 54) + aux.SubString(1, 3);

//Obtengo la cadena con el valor que se obtiene a partir del buffer de licencia
aux= CalcularValorLIC(bufferArchivo, sizeBuffer - 1);

//Coloco los 9 primeros caracteres de la cadena en el ID
id = id.SubString(1, 1) + aux.SubString(1, 9) + id.SubString(11, id.Length() - 10);

//Obtengo el valor del nombre, serial y compañía concatenados
aux = ValorDeConcatenados(EditNombre->Text + EditSerial->Text + EditCompania->Text);
id = id.SubString(1,51) + aux.SubString(1,3) + id.SubString(55, id.Length() - 54);

//Obtengo la cadena que se genera a partir del ID
aux = CalcularValorID(id);

//Coloco el que necesito del resultado en su posición en el ID
id = aux.SubString(1,1) + id.SubString(2, id.Length() - 1);

```

```

//Encripto el ID y lo muestro
id = Encriptar(id);

//Como al desencriptar, si es mayor o igual a 0x40, va a dividir su valor entre 0xA
//que es 10 y quedarse con el resto, cualquier valor que sea mayor o igual a 0x40
//podemos multiplicarlo por 10, 20, o 30 sin temor de salirnos de los valores de la
//tabla ASCII y así tener un ID que contenga más caracteres ya que si no solo
//saldrían los menores de 0x40 y el rango de la A a la J
for(int i=1; i<=57; i++)
    if(id[i] >= 0x40)
        id[i] = id[i] + (random(3) * 10);

//Limpio el buffer
LimpiarBuffer(bufferArchivo, sizeBuffer);

//Monto el buffer con la cadena de licencia final
snprintf(bufferArchivo, sizeBuffer - 1, "%s%s%s%s%s%s%s%s%s%s%s",
          cadenaEncabezado,
          cadenaNombre, EditNombre->Text.c_str(), salto,
          cadenaCompania, EditCompania->Text.c_str(), salto,
          cadenaSerial, EditSerial->Text.c_str(), salto,
          cadenaID, id.c_str(), salto,
          cadenaPie);
bufferLleno = true;
EditID->Text = id;
ButtonCrearLIC->Enabled = true;
}
//-----
AnsiString GetLastErrorAsString()
{
    //Get the error message, if any.
    DWORD errorMessageID = GetLastError();
    if(errorMessageID == 0)
        return NULL; //No error message has been recorded

    LPSTR messageBuffer = NULL;
    FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM |
FORMAT_MESSAGE_IGNORE_INSERTS, NULL, errorMessageID, MAKELANGID(LANG_NEUTRAL,
SUBLANG_DEFAULT), (LPSTR)&messageBuffer, 0, NULL);

    AnsiString message = messageBuffer;

    //Free the buffer.
    LocalFree(messageBuffer);

    return message;
}
//-----
void __fastcall TForm1::ButtonCrearLICClick(TObject *Sender)
{
    ofstream archivoLIC;
    AnsiString error;

    //Abro el archivo de licencia
    archivoLIC.open("EFSUM.LIC", ios::binary | ios::out);

    if(archivoLIC.fail()){

```

```

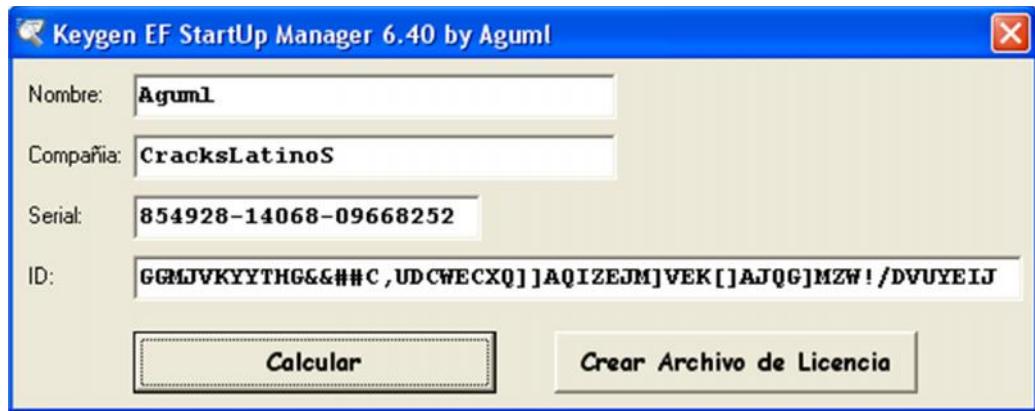
        Application->MessageBoxA(AnsiString("No se pudo crear el archivo de licencia.\n"
+ GetLastErrorAsString()).c_str(), "Error", MB_ICONERROR);
        return;
    }
    //Guardo la información en el
    archivoLIC.write(bufferArchivo, sizeBuffer - 1);

    //Cierro el archivo de licencia
    archivoLIC.close();
    Application->MessageBoxA("El archivo de licencia se ha creado satisfactoriamente.\nNo
olvide colocarla en el directorio del programa.", "Enhорабуна", MB_ICONINFORMATION);
}
//-----
void __fastcall TForm1::FormClose(TObject *Sender, TCloseAction &Action)
{
    if(bufferLleno)
        delete bufferArchivo;
    bufferArchivo != NULL;
    bufferLleno = false;
    delete listaNegra;
}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
    bufferLleno = false;
    bufferArchivo = NULL;
    listaNegra = new TStringList;

    listaNegra->Add("803207-21086-90751233");
    listaNegra->Add("963740-22130-90860237");
    listaNegra->Add("093215-23287-90770242");
    listaNegra->Add("833974-91093-90750235");
    listaNegra->Add("913994-41175-98760235");
    listaNegra->Add("903904-51176-98760235");
    listaNegra->Add("003706-71286-90770237");
    listaNegra->Add("803201-02076-98751213");
    listaNegra->Add("873232-92089-90751233");
    listaNegra->Add("163841-82370-98780236");
    listaNegra->Add("963944-91170-98760235");
    listaNegra->Add("993914-61177-98760235");
    listaNegra->Add("973934-81179-98760235");
    listaNegra->Add("923385-12194-90760231");
    listaNegra->Add("963045-23100-90760244");
    listaNegra->Add("863240-12090-90751233");
    listaNegra->Add("933179-23103-90760243");
    listaNegra->Add("953453-01101-90761231");
    listaNegra->Add("953150-03191-90760233");

    randomize();
}
//-----

```



Y pruebo el nuevo archivo de licencia y veo esto:

