KeyGen sin nombre de usuario
Cuando el serial cumple ciertas condiciones

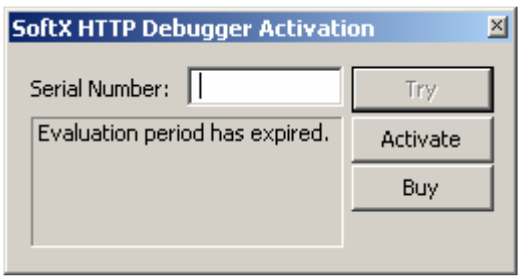
Programa:	SoftX HTTP Debugger 3.2
Protección:	ninguna
Descripción:	Captura, analiza y depura todas las comunicaciones HTTP
Dificultad:	Algo mayor de pequeña
Herramientas:	OllyDBG y RadAsm
Objetivos:	Hacer un keygen
Cracker:	Orniaco

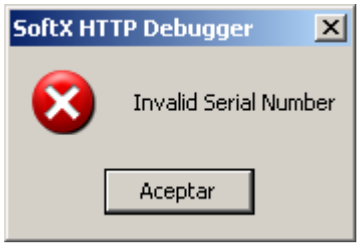
Introducción

En el capítulo de hoy vamos a enfrentarnos a un serial que nos está vinculado a ningún nombre de usuario. El serial debe cumplir algunos requisitos para que sea admitido como válido.

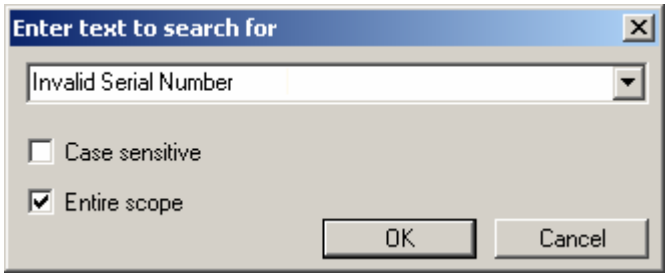
Encontrando a chico bueno-chico malo

Al ejecutar el programa observamos la siguiente ventana, en especial si ha caducado el período de evaluación:





Y si introducimos un serial al azar nos devuelve el mensaje "Invalid Serial Number". Cómo el código no está empaquetado, abrimos a Olly y apretando el botón derecho del ratón elegimos Search for → All referenced text strings y buscamos, con alcance completo, el mensaje:



Y la encontramos en la dirección 00415CEB:

```
00415CEB| PUSH HTTP.0045DF08                    | ASCII "Invalid Serial Number"
```

Si analizamos los alrededores podemos darnos cuenta que el serial tiene que tener exactamente 16 caracteres:

```
00415D0B| .  83F8 10      |CMP EAX,10
```

Supongamos que introducimos el serial siguiente “1122334455667788”, si paramos la ejecución en 00415D6D, observaremos varias cosas:

```
00415D6A| .  83C4 10      | ADD ESP,10
00415D6D| .  3C 7A        | CMP AL,7A
00415D6F| .  894424 20    | MOV DWORD PTR SS:[ESP+20],EAX
00415D73| .  894C24 24    | MOV DWORD PTR SS:[ESP+24],ECX
00415D77| .v 0F85 A500000 | JNZ HTTP.00415E22
00415D7D| .  8D5424 20    | LEA EDX,DWORD PTR SS:[ESP+20]
00415D81| .  52           | PUSH EDX
00415D82| .  8D4424 18    | LEA EAX,DWORD PTR SS:[ESP+18]
00415D86| .  50           | PUSH EAX
00415D87| .  E8 84FEFFFF  | CALL HTTP.00415C10
00415D8C| .  8B4C24 1C    | MOV ECX,DWORD PTR SS:[ESP+1C]
```

```
Registers (FPU)
EAX 11223344
ECX 55667788
EDX 0012EE5C
EBX 00000000
ESP 0012EEAC
EBP 0012EF14
ESI 00A76430 ASCII "1122334455667788"
EDI 0012EEF8

EIP 00415D6D HTTP.00415D6D
```

```
AL=44 ('D')
```

- El registro ESI apunta a nuestro serial.
- El registro EAX contiene los 8 primeros dígitos interpretados como hexadecimal.
- El registro ECX contiene los 8 últimos dígitos interpretados como hexadecimal.
- El serial debería ser “11223337A55667788” en vez de “1122334455667788”.
- En la rutina 00415D87 se “corta todo el bacalao”.

Nuestro serial se descompone en dos números hexadecimales de 8 dígitos, es decir, en dos Double words (8 bytes) que vamos a llamar Serial1 y Serial2. En memoria se disponen poniendo el byte más significativo antes.

Address	Hex dump	
0012EECC	7A 33 22 11	88 77 66 55
	Serial1	Serial2

El análisis de la rutina 00415D87 muestra que además del byte 7A fijo:

7A							
Serial1				**Serial2**			

Otros tres bytes pueden tomar valores aleatorios (&1, $2 y $1):

7A			&1			$2	$1
Serial1=&				**Serial2 = $**			

Y los cuatro restantes se obtienen operando con los anteriores (&3, &2,$4 y $3):

7A	&3	&2	&1	$4	$3	$2	$1
Serial1=&				**Serial2 = $**			

Deduzcamos los valores de los cuatros bytes:

&3 = not &1

```
00415C22 |.  8A48 03      |MOV CL,BYTE PTR DS:[EAX+3]          | cl = &1
00415C25 |.  8AD1         |MOV DL,CL                           | dl = cl
00415C27 |.  F6D2         |NOT DL                              | dl = not dl
00415C29 |.  8850 01      |MOV BYTE PTR DS:[EAX+1],DL          | &3 = dl
```

&2 = 7A xor $1

```
00415C2C |.  0FB650 07    |MOVZX EDX,BYTE PTR DS:[EAX+7]       | edx = $1
00415C30 |.  3210         |XOR DL,BYTE PTR DS:[EAX]            | dl = dl xor 7A
00415C32 |.  53           |PUSH EBX                            |
00415C33 |.  8850 02      |MOV BYTE PTR DS:[EAX+2],DL          | &2 = dl
```

$4 = (not $2) xor &1

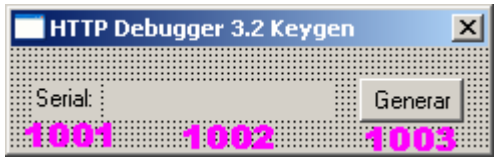
```
00415C33 |.  8850 02      |MOV BYTE PTR DS:[EAX+2],DL          | &2 = dl
00415C36 |.  8A50 06      |MOV DL,BYTE PTR DS:[EAX+6]          | dl = $2
00415C39 |.  8ADA         |MOV BL,DL                           | bl = dl
00415C3B |.  F6D3         |NOT BL                              | bl = not bl
00415C3D |.  32D9         |XOR BL,CL                           | bl = bl xor &1
00415C3F |.  33C9         |XOR ECX,ECX                         |
00415C41 |.  80F2 07      |XOR DL,7                            |
00415C44 |.  56           |PUSH ESI                            |
00415C45 |.  8B7424 28    |MOV ESI,DWORD PTR SS:[ESP+28]       |
00415C49 |.  8850 05      |MOV BYTE PTR DS:[EAX+5],DL          |
00415C4C |.  8B10         |MOV EDX,DWORD PTR DS:[EAX]          |
00415C4E |.  8858 04      |MOV BYTE PTR DS:[EAX+4],BL          | $4 = bl
```

$3 = $2 xor 07

```
00415C33 |.  8850 02      |MOV BYTE PTR DS:[EAX+2],DL          | &2 = dl
00415C36 |.  8A50 06      |MOV DL,BYTE PTR DS:[EAX+6]          | dl = $2
00415C39 |.  8ADA         |MOV BL,DL                           |
00415C3B |.  F6D3         |NOT BL                              |
00415C3D |.  32D9         |XOR BL,CL                           |
00415C3F |.  33C9         |XOR ECX,ECX                         |
00415C41 |.  80F2 07      |XOR DL,7                            | dl = dl xor 07
00415C44 |.  56           |PUSH ESI                            |
00415C45 |.  8B7424 28    |MOV ESI,DWORD PTR SS:[ESP+28]       |
00415C49 |.  8850 05      |MOV BYTE PTR DS:[EAX+5],DL          | $3= dl
```

Haciendo el Keygen

Vamos a hacerlo en RASDAM, arrancamos el programa y generamos el proyecto **HTTPDebuggerKeygen** con una form como la siguiente con una etiqueta, una caja de texto y un botón (en color rosa los números que los identifican: ID)



En la sección **.data** de HTTPDebuggerKeygen.inc añadimos:

```
szSerial    db 16 dup (0)    ; va a contener la cadena caracteres serial
Serial1     dd 0             ; va a contener el hexadecimal de Serial1
Serial1     dd 0             ; va a contener el hexadecimal de Serial2
Semilla     dd 0             ; va a contener la semilla de rutina aleatorios
```

En la sección **.const** de la pestaña HTTPDebuggerKeygen.inc añadimos:

```
szFormato           db "%08lX%08lX",0 ; formato para wsprintf
```

Añadimos entre las líneas **.elseif eax==WM_COMMAND** y **.elseif eax==WM_CLOSE** el siguiente código:

```
.if (wParam==1003)                    ; si es el botón identificado con 1005
  Invoke Generar, hWin                ; genera el serial
.endif
```

Añadimos entre las líneas: "invoke ExitProcess, 0" y "DlgProc proc hWin:HWND, uMsg:UINT, wParam:WPARAM, lParam:LPARAM" el siguiente código (para formar la rutina Generar):

```
Generar proc hWnd:DWORD
    invoke Rand, 16, 255                ; nuestro serial no empieza por cero
    mov byte ptr ds:[Serial1+3], al     ; [  ][  ][  ][&1] [  ][  ][  ][  ]
    mov byte ptr ds:[Serial1], 7Ah      ; [7A][  ][  ][**] [  ][  ][  ][  ]
    not al
    mov byte ptr ds:[Serial1+1], al     ; [**][&3][  ][**] [  ][  ][  ][  ]
    invoke Rand, 0, 255
    mov byte ptr ds:[Serial2+3], al     ; [**][**][  ][**] [  ][  ][  ][$1]
    xor al, 7Ah
    mov byte ptr ds:[Serial1+2], al     ; [**][**][&2][**] [  ][  ][  ][**]
    invoke Rand, 0, 255
    mov byte ptr ds:[Serial2+2], al     ; [**][**][**][**] [  ][  ][$2][**]
    not al
    xor al, byte ptr ds:[Serial1+3]
    mov byte ptr ds:[Serial2],al        ; [**][**][**][**] [$4][  ][**][**]
    mov al, byte ptr ds:[Serial2+2]
    xor al, 7
    mov byte ptr ds:[Serial2+1], al     ; [**][**][**][**] [**][$3][**][**]

    invoke wsprintf, addr szSerial, addr szFormato, Serial1, Serial2
    invoke SetDlgItemText, hWnd, 1002, addr szSerial
    invoke SendDlgItemMessage,hWnd,1002,EM_SETSEL,0,-1
    invoke SendDlgItemMessage,hWnd,1002,WM_COPY,0,0
    ret
Generar endp
```

La rutina Rand está explicada en el minitutorial "Generación de números aleatorios en RadAsm". Compilamos, linkamos y ejecutamos nuestro keygen.

Agradecimientos

A Ricardo Narvaja, a todos los CracksLatinos y muy especialmente a Carlos y stzwei.