```
////////////////////////////////////////////////////////////////////////////////
//    - P L S - Q L  D E V E L O P E R  8 . 0 4 -           //
////////////////////////////////////////////////////////////////////////////////
//  Protection type:  Trial                                                   //
//  Secure methods: AntiDebugger, upx                                         //
//   Crack type:   Change jnz to jmp                                          //
//  Time:         7 hours                                                     //
//      Muzic:        Handel                                                 //
//      Dificultad:      Facil                                               //
//      Herramientas:   Ida, ollydgb, ultraestudio, fuu     //
////////////////////////////////////////////////////////////////////////////////
```

## -Introduccion

Ide SQL exclusivo para DB de Oracle.
Editor Plsql, debugger, code folder, search object, etc...
Actualizado periódicamente y para mi gusto el mas usable de todos(sql navigator,toad, sql-developer ufff).

-Desensamblado

Primero de todo, abrimos Ida, para hechar un vistazo al código, que analice el flujo para que genere el bonito "flow graph" y así hacernos una idea general del bichito que nos está esperando.

Como soy un noob y estoy aprendiendo, mis retos en este binario son:

-Saltar la trial, lo mas fácil.
-Quitar el nag del trial.

Primero de todo vemos que está comprimido con upx, el propio ida nos lo muestra, en las direcciones de memoria del desensamblado asi:

*UPX0:00C28CC8  push  ebp  ----->Este es el punto de entrada*

Lo desempaquetamos con fuu (googlecode), generamos el exe sin upx.

Lo ejecuto para ver si todo ha ido bien, y vualá tengo plsqldeveloper funcionando correctamente.

## -Debug

Antes de nada pruebo lo más fácil, adelanto la hora del sistema, y lo ejecuto...

Obtengo un bonito mensaje diciéndome que se ha expirado la demo aissssh, pero si hace 5 minutos que me lo he bajado...ahora ya sabemos que la fecha la coje del sistema y para ello tenemos varias funciones, entre ellas getLocalTime, importada por nuestro ejecutable.

Enciendo Olly dispuesto a buscar las referencias a getLocalTime y poner bp por aquí y por allá, inocente de mi...

Aquí empiezan mis tormentos uffff, me topo con un sistema de protección anti-debug.

Miro las referencias importadas en ida, y veo que tiene una referencia a OutputDebugString, creo que hay un método anti-debug llamando a esta función.

Me dispongo a debuggear para ver como se detecta y me pierdo en el codigo durante horas.

Pruebo varios plugins para olly, así como versiones modificadas para esconderlo y nada.

Finalmente lo ejecuto con el dbg immunity, para ver si es cosa del olly y nada, despues con el debugger del Ida y tampoco chuta, exceptions y mas exceptions, punto muerto.

Al final decido atachar el debugger al proceso ya ejecutado, busco strings y encuentro cositas como expired, register, trial, etc, miro en el código con el olly y veo algo asi:

```
00A203B4  BA F805A200      MOV EDX,plsqlunp.00A205F8       ; ASCII "DoRegister "
00A203B9  E8 D6609EFF      CALL plsqlunp.00406494
00A203BE  8B45 F0          MOV EAX,DWORD PTR SS:[EBP-10]
00A203C1  E8 96731F00      CALL plsqlunp.00C1775C
00A203C6  8D45 F8          LEA EAX,DWORD PTR SS:[EBP-8]
00A203C9  8B15 F07DC600    MOV EDX,DWORD PTR DS:[C67DF0]     ; plsqlunp.00C74E9C
00A203CF  8B12             MOV EDX,DWORD PTR DS:[EDX]
00A203D1  8B92 00040000    MOV EDX,DWORD PTR DS:[EDX+400]
00A203D7  8B52 40          MOV EDX,DWORD PTR DS:[EDX+40]
00A203DA  E8 095E9EFF      CALL plsqlunp.004061E8
00A203DF  33DB             XOR EBX,EBX
00A203E1  B8 0C06A200      MOV EAX,plsqlunp.00A2060C       ; ASCII "CreateForm"
00A203E6  E8 71731F00      CALL plsqlunp.00C1775C
00A203EB  8D4D F4          LEA ECX,DWORD PTR SS:[EBP-C]
00A203EE  A1 C48AC600      MOV EAX,DWORD PTR DS:[C68AC4]
00A203F3  8B00             MOV EAX,DWORD PTR DS:[EAX]
00A203F5  8B15 F4FCA100    MOV EDX,DWORD PTR DS:[A1FCF4]     ; plsqlunp.00A1FD40
00A203FB  E8 B829ABFF      CALL plsqlunp.004D2DB8
00A20400  8B55 FC          MOV EDX,DWORD PTR SS:[EBP-4]
00A20403  8B45 F4          MOV EAX,DWORD PTR SS:[EBP-C]
00A20406  E8 B9020000      CALL plsqlunp.00A206C4
00A2040B  B8 2006A200      MOV EAX,plsqlunp.00A20620       ; ASCII "RegisterForm.ShowModal"
00A20410  E8 47731F00      CALL plsqlunp.00C1775C
00A20415  8B45 F4          MOV EAX,DWORD PTR SS:[EBP-C]
00A20418  8B10             MOV EDX,DWORD PTR DS:[EAX]
00A2041A  FF92 FC000000    CALL DWORD PTR DS:[EDX+FC]
00A20420  48               DEC EAX
00A20421  0F85 34010000    JNZ plsqlunp.00A2055B
00A20427  B8 4006A200      MOV EAX,plsqlunp.00A20640       ; ASCII "RegisterForm OK"
```

Mmmm aqui tenemos sitios interesantes para poder meter un bp miremos por ejempo en:

```
00A2041A  FF92 FC000000  CALL DWORD PTR DS:[EDX+FC]
00A20420  48        DEC EAX   ------------------->BP
```

Parece que en esta función se creará y llamara a la ventana modal que nos servirá para registrar el programa, después dejará un resultado y le hace un dec.

Hay 2 maneras de acceder a este código:

-En periodo de prueba, a través de Help->Register

-Cuando detecta que ha expirado el tiempo accede directamente aquí, sin inicializar las diferentes secciones del programa.

Recordamos que he cambiado la hora y por lo tanto accedo directamente a una pantalla modal para registrar la aplicación.

Bueno, vamos a probar todo esto, ejecuto la aplicación, salta el modal, enciendo el olly, attach al proceso, y cambio el modulo a ver, ya que por defecto nos deja en librerias internas de sistema.

*View->Executable Modules*

Selecciono el plsqldev.exe y botón derecho seleccionamos "Follow entry", con esto ya estamos en el entry point del ejecutable, ponemos un bp en:

*00A20420  48  DEC EAX*

Doy al boton register, y efectivamente se para en 0x00A20420, empiezo a debugger para adelante miro un poco y veo que ahí mismo estan casi todas las soluciones, key-gen, y bypass, de la protección.

Pero si he llegado aquí es porque antes se ha verificado que hubiera expirado no? Creo yo que sí, vamos a ver de donde venimos, otro gran dilema del ser humano.

Para ello tenemos un bonito historial de funciones de nuestro querido binario, éste lo podremos encontrar en el stack, allí quedan guardadas todas las direcciones de retorno de las call que se han hecho anteriormente y nos da una pista mas que considerable de lo que buscamos.

```
0018F74C |009D49F9 RETURN to plsqlunp.009D49F9 from plsqlunp.009C7CCC ------>Interesante no?
0018F750 |0018FDC0 Pointer to next SEH record
0018F754 |009D601C SE handler
0018F758 |0018FDB8
0018F75C |00000000
0018F760 |009B8258 plsqlunp.009B8258
0018F764 |0138E8B0
```

Vamos a ver que hay por 0x009D49F9 simple curiosidad eeeh!

Para mirar código estático prefiero utilizar Ida, ya que el analisis de código te divide las funciones y para mi es mas intuitivo de seguir,cuando no se esta debuggeando, esto es lo que obtengo.

```
UPX0:009D49F1 loc_9D49F1:              ; CODE XREF: UPX0:009D49A0
UPX0:009D49F1        mov   eax, [ebp-4]
UPX0:009D49F4        call  sub_9C7CCC
UPX0:009D49F9        test  al, al
UPX0:009D49FB        jnz   short loc_9D4A06
UPX0:009D49FD        mov   byte_C5F434, 1
UPX0:009D4A04        jmp   short loc_9D4A08
UPX0:009D4A06 ; ---------------------------------------------------------------------------
```

Siguiendo el codigo hacia atras vemos que esta función se llama desde:

```
UPX0:009D48BD loc_9D48BD:              ; CODE XREF: UPX0:009D48A6
```

Y mirando mas todavía nos encontramos que esta función ha sido llamada desde:

```
UPX0:009D487A loc_9D487A:              ; CODE XREF: UPX0:009D486B
```

Y por lo que se ve en el código, hasta aquí ya se ha verificado que:

-La licencia es incorrecta.

-La licencia es de otro producto.

-El Trial ha expirado.

Por lo tanto, si hemos entrado en esta función se acabo nuestro disfrute, porque es la encargada de mostrar el modal, y otras cosillas ;)

No ha sido muy dificil trazar las llamadas hacia atras, con Ida, ya que aún sin poder ver las funciones en modo graph flow, claramente se ven todos los jmp, y calls relacionados con lineas rojas y es muy intuitivo.

Veamos otra vez de donde venimos y quien es el culpable de llevarnos a esta función.

```
UPX0:009D4857 loc_9D4857:              ; CODE XREF: UPX0:009D47E7 j
UPX0:009D4857         cmp   byte_C5F434, 0
UPX0:009D485E         jnz   loc_9D4EE4
UPX0:009D4864         call  sub_664B9C
UPX0:009D4869         test  al, al
UPX0:009D486B         jnz   short loc_9D487A
UPX0:009D486D         cmp   byte_C5F438, 0
UPX0:009D4874         jz   loc_9D4A8B
UPX0:009D487A
```

La función "UPX0:009D4857 loc_9D4857" , es la encargada de chequear si entramos en licencia invalida, o trial expired.

Si nos fijamos en la instrucción:

```
UPX0:009D485E         jnz   loc_9D4EE
```

Hace un salto hacía otra función, y si miramos el código se esta saltando todo el tema de validaciones extras, e inicialización de la pantalla modal.

El código de UPX0:009D485E se ecargá de inicializar la segunda pantalla modal, con la configuración de la conexión a la DB. De oracle.

Es decir si hacemos un salto incondicional hacia UPX0:009D4EE, puede ser que nos saltemos la pantalla modal y....pues no se ya veremos no?

Dejaríamos la instrucción así "*JMP 009D4EE4*"

Abrimos nuestro editor hex favorito buscamos primero los opcodes del jnz:

```
0F8580060000
```

En el olly buscamos la dirección y con doble click en jnz lo podremos cambiar fácilmente, damos assemble, y vemos como los opcodes del olly han cambiado, y por cierto han añadido un nop porque la mueva instrucción tiene un byte menos.

El nuevo opcode es:

```
E98106000090
```

Cambiamos los valores, grabamos el hex editor, y....tachan.

Tenemos el plsqldeveloper funcional, y sin nag de aviso ni na de na.

Ciaooo