```
////////////////////////////////////////////////////////////////////////////
//        - U L T R A E D I T  L I N U X   2.1.0.3 -          //
////////////////////////////////////////////////////////////////////////////
//     Protection type:   Trial and  nag                        //
//     Secure methods:                                          //
//     Crack type:        Parche   bytes                        //
//     Time:              2 horas                               //
//     Muzic:             bach ,brandimburgo , +bach //
//     Dificultad:        Fácil                                 //
//     Herramientas:      Ida, Gdb, Ultraedit           //
////////////////////////////////////////////////////////////////////////////
```

# -Introducción

Ultraedit es un editor de texto, y se podría incluso catalogar como ide ya que soporta un montón de lenguajes de programación y se puede configurar para añadir compiladores.
Code folder, syntax highlight y editor hexadecimal, también tiene una característica que todavía no he visto en ningún editor, selección por columna, que para trabajar con db en ficheros va de lujo ;)

# 2-Protección

Bueno parece ser que para esta versión de linux la seguridad no se ha tenido en cuenta, (en la de win no lo sé :P), y cuando expira podremos reiniciar la trial borrando el fichero de configuración creado en el home del user de linux:

*/home/user/.idm/uex/uex.conf*

Decir que no hace falta borrar el fichero, con solo "toquetear" la variable value podríamos ir saltándonos la protección del trial...

```
[Registration]
Value=14980
Version=1
```

Nuestro invitado (aka uex) no dejará que lo utilicemos mas de 30 días,  veamos si adelantando la hora del sistema engañamos y hacemos creer a uex que ya pasaron...sip, salta la ventanita conforme a expirado, se acabó el disfrute...  o no, quizás está empezando XD.

# 3-Desensamblado

Voy a utilizar algunas tools de linux que vienen instaladas por defecto en cualquier distribución creo, tampoco son indispensables y simplemente dan información interesante.

Para desensamblar tenemos objdump, que aparte nos puede dar información sobre el tipo de binario, secciones y symbolos.

Para información de las secciones y tipo de binario con el flag h:

*objdump -h /usr/bin/uex*

```
 ***************************************************************************
/usr/bin/uex:     file format elf64-x86-64

Sections:
 Idx Name          Size      VMA               LMA               File off  Algn
  0 .interp       0000001c  0000000000400238  0000000000400238  00000238  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  1 .note.ABI-tag 00000020  0000000000400254  0000000000400254  00000254  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .note.gnu.build-id 00000024  0000000000400274  0000000000400274  00000274  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  3 .hash         00002c18  0000000000400298  0000000000400298  00000298  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  4 .gnu.hash     00000c48  0000000000402eb0  0000000000402eb0  00002eb0  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  5 .dynsym       0000a7b8  0000000000403af8  0000000000403af8  00003af8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  6 .dynstr       00011359  000000000040e2b0  000000000040e2b0  0000e2b0  2**0
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  7 .gnu.version  00000dfa  000000000041f60a  000000000041f60a  0001f60a  2**1
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  8 .gnu.version_r 000001c0  0000000000420408  0000000000420408  00020408  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  9 .rela.dyn     000008d0  00000000004205c8  00000000004205c8  000205c8  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 10 .rela.plt     000086a0  0000000000420e98  0000000000420e98  00020e98  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 11 .init         0000001d  0000000000429538  0000000000429538  00029538  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 12 .plt          000059d0  0000000000429558  0000000000429558  00029558  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 13 .text         008b9288  000000000042ef40  000000000042ef40  0002ef40  2**6
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 14 .fini         0000000e  0000000000ce81c8  0000000000ce81c8  008e81c8  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
 15 .rodata       001a45e0  0000000000ce81e0  0000000000ce81e0  008e81e0  2**5
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 16 .eh_frame_hdr 0003478c  0000000000e8c7c0  0000000000e8c7c0  00a8c7c0  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 17 .eh_frame     000d5654  0000000000ec0f50  0000000000ec0f50  00ac0f50  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 18 .gcc_except_table 0006f298  0000000000f965a4  0000000000f965a4  00b965a4  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
 19 .ctors        00000fc0  0000000001206488  0000000001206488  00c06488  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 20 .dtors        00000010  0000000001207448  0000000001207448  00c07448  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 21 .jcr          00000008  0000000001207458  0000000001207458  00c07458  2**3
                  CONTENTS, ALLOC, LOAD, DATA
 22 .data.rel.ro  00003690  0000000001207460  0000000001207460  00c07460  2**5
                  CONTENTS, ALLOC, LOAD, DATA
```

```
23 .dynamic     00000340  000000000120aaf0  000000000120aaf0  00c0aaf0  2**3
                CONTENTS, ALLOC, LOAD, DATA
24 .got         000001b0  000000000120ae30  000000000120ae30  00c0ae30  2**3
                CONTENTS, ALLOC, LOAD, DATA
25 .got.plt     00002cf8  000000000120afe8  000000000120afe8  00c0afe8  2**3
                CONTENTS, ALLOC, LOAD, DATA
26 .data        003a38c8  000000000120dce0  000000000120dce0  00c0dce0  2**5
                CONTENTS, ALLOC, LOAD, DATA
27 .bss         000832d8  00000000015b15c0  00000000015b15c0  00fb15a8  2**5
                ALLOC
28 .comment     00000023  0000000000000000  0000000000000000  00fb15a8  2**0
                CONTENTS, READONLY
29 .debug_aranges 00024c10  0000000000000000  0000000000000000  00fb15cb  2**0
                CONTENTS, READONLY, DEBUGGING
30 .debug_pubnames 000a55ab  0000000000000000  0000000000000000  00fd61db  2**0
                CONTENTS, READONLY, DEBUGGING
31 .debug_info   021a472b  0000000000000000  0000000000000000  0107b786  2**0
                CONTENTS, READONLY, DEBUGGING
32 .debug_abbrev 000b7f84  0000000000000000  0000000000000000  0321feb1  2**0
                CONTENTS, READONLY, DEBUGGING
33 .debug_line   00221f03  0000000000000000  0000000000000000  032d7e35  2**0
                CONTENTS, READONLY, DEBUGGING
34 .debug_str    001c4df9  0000000000000000  0000000000000000  034f9d38  2**0
                CONTENTS, READONLY, DEBUGGING
35 .debug_loc    00376727  0000000000000000  0000000000000000  036beb31  2**0
                CONTENTS, READONLY, DEBUGGING
36 .debug_ranges 0051f900  0000000000000000  0000000000000000  03a35258  2**0
```

Nos ha dicho que es un elf64-x86 y a parte vemos unas secciones (29-36) al final de debug um parece interesante, al igual nos facilitan el trabajo.

Para listar las funciones:

```
nm /usr/bin/uex
```

Para saber desde donde se llaman podríamos hacer un:

```
objdump -D /usr/bin/uex | grep plt | grep callq
objdump -D /usr/bin/uex | grep plt | grep jmp
```

Y por mera curiosidad, si quisiéramos buscar las llamadas a la función time del sistema:

```
objdump -D /usr/bin/uex | grep "<time@plt>"
```

```
**********************************************
000000000042c5c8 <time@plt>:
45ae00:     e8 c3 17 fd ff      callq  42c5c8 <time@plt>
45b4c0:     e8 03 11 fd ff      callq  42c5c8 <time@plt>
46172a:     e8 99 ae fc ff      callq  42c5c8 <time@plt>
4618d3:     e8 f0 ac fc ff      callq  42c5c8 <time@plt> ----------> Very interestant
461913:     e8 b0 ac fc ff      callq  42c5c8 <time@plt>
467262:     e8 61 53 fc ff      callq  42c5c8 <time@plt>
484731:     e8 92 7e fa ff      callq  42c5c8 <time@plt>
4a91bf:     e8 04 34 f8 ff      callq  42c5c8 <time@plt>
4c4237:     e8 8c 83 f6 ff      callq  42c5c8 <time@plt>
4c431e:     e8 a5 82 f6 ff      callq  42c5c8 <time@plt>
4d07ac:     e8 17 be f5 ff      callq  42c5c8 <time@plt>
```

```
4d0893:       e8 30 bd f5 ff          callq  42c5c8 <time@plt>
4d09b4:       e8 0f bc f5 ff          callq  42c5c8 <time@plt>
5721d7:       e8 ec a3 eb ff          callq  42c5c8 <time@plt>
************************************************
```

Ala ya hemos visto que las gnu tools son la repera pero donde esté el buen Ida....

Abro con Ida el binario uex a ver que me dice, utilizo la versión de windows que soporta elf y bastante bien por lo que veo, ida64 ya que estoy trabajando con ubuntu 64bits, por cierto uex ha sido instalado con:

```
apt-get install uex

Md5:  ec0794140b25e0685b4d9ca458cb24e8  /usr/bin/uex
```

Bien lo que veo realmente me sorprende, en ida se han cargado todos los nombres de las clases y métodos del binario en c++, me pongo a curiosear y veo esto:

```
ueMain::ProcessTrial(void)
```

Supongo que se ha compilado con symbolos de debug incluidos y para nada se ha pasado un strip de funciones o sea que podemos ir saltando a las funciones del binario solo con clickar en el listado del ida (a nuestra izquierda), casi como si estuviéramos programando con nuestro ide favorito, con los nombres reales incluidos jeje ;)

Bueno por ahora nada de debugger pero es que no ha sido necesario no?

## 4-Debug

Veamos con gdb que podemos hacer...

```
gdb uex
```

Ponemos un bp en la función:

```
(gdb) b ueMain::ProcessTrial
Breakpoint 1 at 0x4618c0: file ../main.cpp, line 6705. -------> ummmm debug symbols
                                                                included for fun and profit ;)
```

Arrancamos el binario:

```
(gdb) r
Starting program: /usr/bin/uex
[Thread debugging using libthread_db enabled]
[New Thread 0x7fffebb99700 (LWP 4002)]

Breakpoint 1, ueMain::ProcessTrial (this=0x198e4d0) at ../main.cpp:6705
6705    ../main.cpp: No such file or directory.     ---------> solo faltaba que incluyeran el
                                                               main.c en el debian pakage jajaja.
        in ../main.cpp
```

Listamos el código:

```
(gdb) disas
Dump of assembler code for function _ZN6ueMain12ProcessTrialEv:
=> 0x00000000004618c0 <+0>: push   %rbp
   0x00000000004618c1 <+1>:    push   %rbx
   0x00000000004618c2 <+2>:    mov    %rdi,%rbx
   0x00000000004618c5 <+5>:    sub    $0x8,%rsp
   0x00000000004618c9 <+9>:    callq  0x461610 <_ZN6ueMain14GetInstallTimeEv> ;  fecha de la instalación.
   0X00000000004618ce <+14>:  xor    %edi,%edi
   0x00000000004618d0 <+16>:  mov    %rax,%rbp                  ; copia a rbp el time de la instalación
   0x00000000004618d3 <+19>:  callq  0x42c5c8 <time@plt>        ; Llama al time del sistema.
   0X00000000004618d8 <+24>: mov    %rax,%rcx                  ; copia el time del sistema a rcx
   0x00000000004618db <+27>:  sub    %rbp,%rcx                  ; Resta el time de la instalación con el del sistema.
   0X00000000004618de <+30>: test   %rcx,%rcx                  ; test rcx ;)
   0x00000000004618e1 <+33>:  jle    0x461958 <_ZN6ueMain12ProcessTrialEv+152>
   0x00000000004618e3 <+35>:  mov    %rcx,%rax
   0x00000000004618e6 <+38>:  movabs $0x1845c8a0ce512957,%rdx
   0x00000000004618f0 <+48>:  sar    $0x3f,%rcx
   0x00000000004618f4 <+52>:  imul   %rdx
   0x00000000004618f7 <+55>:  mov    $0x0,%eax
   0x00000000004618fc <+60>:  sar    $0xd,%rdx
  --Type <return> to continue, or q <return> to quit---
```

El listado nos sale con formato at y esto tiene una particularidad entre otras, que cambia el sentido en que operan las instrucciones.

Si en "*Intel*" con Ida tenemos:

```
.text:00000000004618D0              mov     rbp, rax
```

Copia el contenido del registro rax a rbp.

En sintaxis *"at"* los registros están invertidos:

```
0x00000000004618d0 <+16>:    mov    %rax,%rbp
```

Viendo el listado está claro que modificando la linea "*test %ecx,%ecx*", por un "*xor %ecx,%ecx*" por ejemplo ya nos saltaríamos el trial.

Buscamos los opcodes en gdb con el argumento /r :

```
(gdb) disas /r
Dump of assembler code for function _ZN6ueMain12ProcessTrialEv:
=> 0x00000000004618c0 <+0>:     55          push   %rbp
   0x00000000004618c1 <+1>:     53          push   %rbx
   0x00000000004618c2 <+2>:     48 89 fb    mov    %rdi,%rbx
   0x00000000004618c5 <+5>:     48 83 ec 08          sub    $0x8,%rsp
   0x00000000004618c9 <+9>:     e8 42 fd ff ff       callq  0x461610 <_ZN6ueMain14GetInstallTimeEv>
   0x00000000004618ce <+14>:    31 ff       xor    %edi,%edi
   0x00000000004618d0 <+16>:    48 89 c5    mov    %rax,%rbp
   0x00000000004618d3 <+19>:    e8 f0 ac fc ff       callq  0x42c5c8 <time@plt>
   0x00000000004618d8 <+24>:    48 89 c1    mov    %rax,%rcx
   0x00000000004618db <+27>:    48 29 e9    sub    %rbp,%rcx
   0x00000000004618de <+30>:    48 85 c9    test   %rcx,%rcx
   0x00000000004618e1 <+33>:    7e 75       jle    0x461958 <_ZN6ueMain12ProcessTrialEv+152>
   0x00000000004618e3 <+35>:    48 89 c8    mov    %rcx,%rax
   0x00000000004618e6 <+38>:    48 ba 57 29 51 ce a0 c8 45 18        movabs $0x1845c8a0ce512957,%rdx
   0x00000000004618f0 <+48>:    48 c1 f9 3f          sar    $0x3f,%rcx
   0x00000000004618f4 <+52>:    48 f7 ea    imul   %rdx
   0x00000000004618f7 <+55>:    b8 00 00 00 00       mov    $0x0,%eax
   0x00000000004618fc <+60>:    48 c1 fa 0d          sar    $0xd,%rdx
   0x0000000000461900 <+64>:    48 29 d1    sub    %rdx,%rcx
   0x0000000000461903 <+67>:    48 83 c1 1e          add    $0x1e,%rcx
   0x0000000000461907 <+71>:    48 0f 48 c8          cmovs  %rax,%rcx
   0x000000000046190b <+75>:    89 8b 8c 03 00 00    mov    %ecx,0x38c(%rbx)
   0x0000000000461911 <+81>:    31 ff       xor    %edi,%edi
---Type <return> to continue, or q <return> to quit---
```

Ya tenemos el opcode a buscar:

*48 85 c9 ----> test rcx, rcx*

Pero bueno vamos a buscar un patrón un poco mas grande para que no hallan coincidencias por ejemplo:

*48 85 c9 7e 75 48 89 c8*

Y sustituimos la parte que nos interesa, "*48 85 c9*" por:

*31 c9 90 ----> xor rcx, rcx*

Con ultraedit es muy fácil hacer búsquedas de todo tipo en hex, y por supuesto reemplazar también, parcheandose a si mismo jeje.

Con esto ya nos saltamos la trial, ahora nos queda el molesto nag.

Según vemos por la funciones que llama el binario en la pestaña exports y en la parte izquierda en el navegador de funciones utiliza las librerías gráficas wxwidgets que son multiplataforma.

Y como intuyo que es una ventana modal, claro candidato...

*wxDialogShowModal*

Abrimos el Ida y continuamos desde donde lo habíamos dejado... creo que por

*ueMain::ProcessTrial*

```
.text:00000000004618C0 _ZN6ueMain12ProcessTrialEv proc near    ; CODE XREF: UexApp::OnInit(void)+7B0
.text:00000000004618C0                 push    rbp
.text:00000000004618C1                 push    rbx
.text:00000000004618C2                 mov     rbx, rdi
.text:00000000004618C5                 sub     rsp, 8
.text:00000000004618C9                 call    _ZN6ueMain14GetInstallTimeEv ; ueMain::GetInstallTime(void)
.text:00000000004618CE                 xor     edi, edi        ; timer
.text:00000000004618D0                 mov     rbp, rax
.text:00000000004618D3                 call    _time
.text:00000000004618D8                 mov     rcx, rax
.text:00000000004618DB                 sub     rcx, rbp
.text:00000000004618DE                 test    rcx, rcx
.text:00000000004618E1                 jle     short loc_461958
.text:00000000004618E3                 mov     rax, rcx
.text:00000000004618E6                 mov     rdx, 1845C8A0CE512957h
```

Tenemos una xref "*CODE XREF: UexApp::OnInit(void)+7B0*", en Ida haciendo un click o situando el cursor y apretando la tecla intro, nos iremos al xref (click o intro en el comentario XREF de ida "*UexApp::OnInit(void)+4A9*") nos llevará allá:

```
.text:0000000000575C0D loc_575C0D:                      ; CODE XREF: UexApp::OnInit(void)+4A9
.text:0000000000575C0D                 mov     rdi, rax
.text:0000000000575C10                 call    _ZN6ueMain12ProcessTrialEv ; ueMain::ProcessTrial(void)
.text:0000000000575C15                 mov     edi, offset unk_D370A0
.text:0000000000575C1A                 call    T_3842
.text:0000000000575C1F                 test    rax, rax
.text:0000000000575C22                 mov     edx, offset dword_E1EA44
.text:0000000000575C27                 lea     r12, [rsp+4D8h+var_C8]
.text:0000000000575C2F                 mov     rcx, cs:_ZN12wxStringBase4nposE ; wxStringBase::npos
.text:0000000000575C36                 cmovz   rax, rdx
.text:0000000000575C3A                 xor     edx, edx
.text:0000000000575C3C                 mov     rsi, rax
.text:0000000000575C3F                 mov     rdi, r12
.text:0000000000575C42                 call    _ZN12wxStringBase8InitWithEPKwmm ; wxStringBase::InitWith(wchar_t
const*,ulong,ulong)
.text:0000000000575C47                 mov     rsi, [rbx+0E0h]
.text:0000000000575C4E                 lea     rbp, [rsp+4D8h+var_4B8]
.text:0000000000575C53                 mov     r9d, offset wxDefaultSize
.text:0000000000575C59                 mov     r8d, offset wxDefaultPosition
.text:0000000000575C5F                 mov     rcx, r12
.text:0000000000575C62                 mov     edx, 2780h
.text:0000000000575C67                 mov     rdi, rbp
.text:0000000000575C6A                 mov     [rsp+4D8h+var_4D8], 20081800h
.text:0000000000575C72                 call
_ZN18InProductMessagingC1EP8wxWindowiRK8wxStringRK7wxPointRK6wxSizel ;
InProductMessaging::InProductMessaging(wxWindow *,int,wxString  const&,wxPoint  const&,wxSize  const&,long)
.text:0000000000575C77                 mov     rdi, r12
.text:0000000000575C7A                 call    _ZN8wxStringD1Ev ; wxString::~wxString()
.text:0000000000575C7F                 mov     rdi, rbp
.text:0000000000575C82                 call    _ZN8wxDialog9ShowModalEv ; wxDialog::ShowModal(void) --->ShowModal
.text:0000000000575C87                 cmp     byte ptr [rbx+0A8h], 0
.text:0000000000575C8E                 jnz     short loc_575CA7
.text:0000000000575C90                 mov     rax, [rbx+0E0h]
```

```
.text:0000000000575C97          mov     r8d, [rax+38Ch]
.text:0000000000575C9E          test    r8d, r8d
.text:0000000000575CA1          jle     loc_575E65
```

Aquí tenemos el showModal;

```
.text:0000000000575C82          call    _ZN8wxDialog9ShowModalEv ; wxDialog::ShowModal(void)
```

Podríamos parchearlo sustituyendo el call con nop's, pero y si miramos de ir al xref a ver de donde venimos ..

```
.text:0000000000575C0D loc_575C0D:                      ; CODE XREF: UexApp::OnInit(void)+4A9
```

Interesante lo que nos dice...

```
.text:00000000005758F6          call    _ZN6UexApp16CreateMainWindowEP12ueDocManagerb ;
UexApp::CreateMainWindow(ueDocManager *,bool)
.text:00000000005758FB          cmp     byte ptr [rbx+0A8h], 0
.text:0000000000575902          mov     [rbx+0E0h], rax
.text:0000000000575909          jz      loc_575C0D      ----------->De aquí venimos
.text:000000000057590F
.text:000000000057590F loc_57590F:                      ; CODE XREF: UexApp::OnInit(void)+84F#j
.text:000000000057590F          mov     rdx, [rbx+368h]
.text:0000000000575916          mov     rax, [rbx+370h]
.text:000000000057591D          mov     r14, 0CCCCCCCCCCCCCCCDh
.text:0000000000575927          sub     rax, rdx
.text:000000000057592A          sar     rax, 3
.text:000000000057592E          imul    rax, r14
.text:0000000000575932          test    rax, rax
.text:0000000000575935          jz      loc_5759D8
.text:000000000057593B          test    eax, eax
.text:000000000057593D          jle     loc_5759D8
.text:0000000000575943          xor     ebp, ebp
.text:0000000000575945          xor     r12d, r12d
.text:0000000000575948          jmp     short loc_57599E
```

Bueno por lo visto el patch anterior nos lo podemos ahorrar(el de la llamada al time) ya que modificando la instrucción del jz está todo hecho:

```
.text:0000000000575909          jz      loc_575C0D
```

Aquí podemos nopear, utilizar un jnz o cualquier otra instrucción sin ser animales...como no hay comprobación de nops, crc, ni na de na.

Los opcodes de la instrucción a parchear, sacados de la pestaña hex del Ida:

```
0F 84 FE 02 00 00
```

La decisión del tipo de parcheo que la tome el lector a modo de ejercicio no?

## 5-Conclusión

No compilar con flags de debug aplicaciones comerciales por favor, que si no es muy fácil, jejeje.

Fuera bromas la conclusión con este binario es que solo con Ida se puede quitar la protección muy fácilmente, y me sorprende que un programa tan utilizado y maduro no implemente ningún tipo de seguridad.

Espero haber transmitido algo de lo poco que sé a los demás, si hay algún error o he dicho alguna barbaridad háganmelo saber por favor así rectifico o lo que sea companys ;)

Saludos a tod@s los CrackSlatinoS