TUTORIAL about Laserlock SPEEnc (Tennis Masters Series 2003):

Commercial protection : Laserlock SPEEnc v 2 (http://www.laserlock.com/)
Editor : Microïds
Release date : FR November 01, 2002
http://www.jeuxvideo.com/articles/0000/00002557_test.htm
Cracking level : [ ] easy [x] intermediate [x] confirmed [ ] expert
It is preferable to have practised or cracked game protections such as Safedisc/SecuROM before reading this tutorial (it also requires knowledge about the PE file format, manual unpacking and imports rebuilding) and perhaps have read my previous tutorial on Laserlock :p

Tools nedeed :
Softice 4.01
Icedump
Procdump/LordPE
Hiew/Hex Workshop
W32Dasm v8.93
Adump
Masm v8

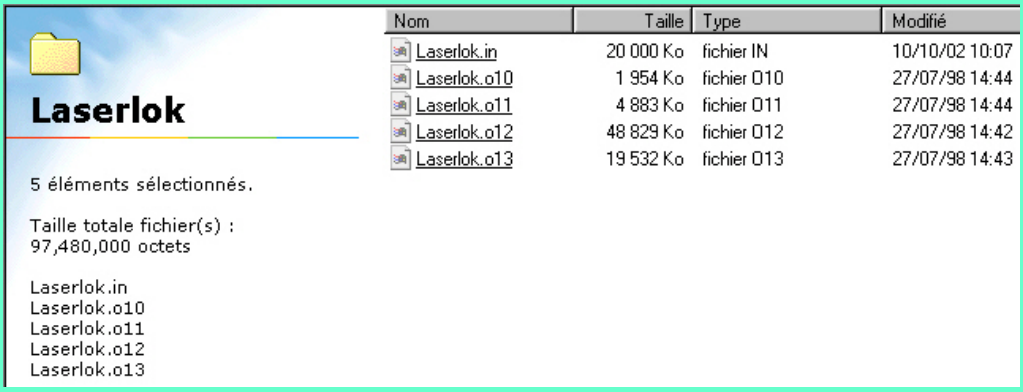
You must own the original disc of this game or a functional clone (there is a Laserlock profile in Alcohol 120%), to be able to apply this tutorial…
Moreover, it is the French version of Tennis Masters Series 2003…
The protected executable file is "Tennis Masters Series 2003.exe", which will now be called "tms2003.exe"…

OS : Win98 SE.
This tutorial can easily be applied to others debuggers (OllyDbg, for example) and other OS.

This tutorial is detailed in the following way:

A) Introduction and general information :

Laserlock is a commercial protection, created by MLS LaserLock Inc.
It is characterized by a Laserlok folder (hidden - > they take us for idiots) on CD's root.
[CD is characterized by a ring quite visible on center…]
If they believe that in putting an hidden attribute on important files, we could not see their little game … - > "a pure lamerz technique" :p
This Laserlok folder contains 5 files : laserlok.in, laserlok.o10, laserlok.o11, laserlok.o12 and laserlok.o13, all hidden (~90 Mo!!!) …



Protection is based on these "copy protected" files (they can't be copied the usual way).
So, protection has just to check the presence of these files, to determine whether it is disc copy or not.

On the executable file, protection can easily be detected by editing it with a hexadecimal editor:

```
00000000 4D5A 9000 0300 0000 0400 0000 FFFF 0000 MZ..............
00000010 B800 0000 0000 0000 4000 0000 0000 0000 ........@.......
00000020 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000030 0000 0000 0000 0000 0000 0000 3001 0000 ............0...
00000040 0E1F BA0E 00B4 09CD 21B8 014C CD21 5468 ........!..L.!Th
00000050 6973 2070 726F 6772 616D 2063 616E 6E6F is program canno
00000060 7420 6265 2072 756E 2069 6E20 444F 5320 t be run in DOS
00000070 6D6F 6465 2E0D 0D0A 2400 0000 0000 0000 mode....$.......
00000080 4F36 F801 0B57 9652 0B57 9652 0B57 9652 O6...W.R.W.R.W.R
00000090 F174 8F52 0F57 9652 F173 8F52 0957 9652 .t.R.W.R.s.R.W.R
000000A0 E348 9C52 0A57 9652 E348 9D52 0257 9652 .H.R.W.R.H.R.W.R
000000B0 704B 9A52 0E57 9652 884B 9852 2D57 9652 pK.R.W.R.K.R-W.R
000000C0 6448 9D52 1C57 9652 6448 9C52 A657 9652 dH.R.W.RdH.R.W.R
000000D0 D174 8B52 0757 9652 0B57 9652 0357 9652 .t.R.W.R.W.R.W.R
000000E0 F777 8452 0A57 9652 0B57 9752 E857 9652 .w.R.W.R.W.R.W.R
000000F0 6948 8552 1A57 9652 5F74 A652 0257 9652 iH.R.W.R_t.R.W.R
00000100 5F74 A752 9755 9650 6163 6B65 6420 6279 _t.R.U.Packed by
00000110 2053 5045 456E 6320 5632 2041 7374 6572  SPEEnc V2 Aster
00000120 696F 7320 5061 726C 616D 656E 7461 732E ios Parlamentas.
00000130 5045 0000 4C01 0700 1ED2 A13D 0000 0000 PE..L......=....
```

Laserlock's guys are cool ; they give us the version of SPEEnc (here, it is version 2).

On their site (copy from September 2003), Laserlock informed us about its protection (Mommy, I 'm really afraid :p) :

LaserLock consists of the following successful combination:

- Sophisticated Codes Encryption software
- Physical Signature on CD, made during has single and special glass-mastering process
- State-of-the-art Debug Prevention software engineering embedded in the s/w code

Actually, it is this powerful combination that makes LaserLock protection system so secure, single and applicable, especially when compared to the other protection systems available today!

LaserLock MARATHON is:

- Uncrackable, with No Generic Cracks
- Most resistant against the latest, advanced copying total software and the devices of the market
- The most compatible Copy protection system worldwide (CD-Rom and DVD-Rom drives)
- Transparent Totally to replicators and end-users, No passwords, No extra devices.
- Of low cost and high efficiency.

I don't know what to say :o

Different types of protections :

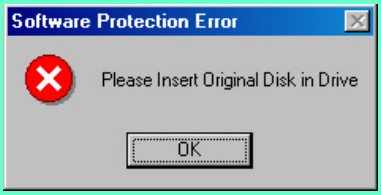
Executable file of this game is packed :
Packed by SPEEnc V2 Asterios Parlamentas
\=-SPeEnc loader rutin d02-07-01r by Asterios Parlamentas \ = -
Redirection of APIs of tms2003.exe (call/jmp API replaced by call Laserlock).
Redirection of GetProcAddress API on the IAT.
Previous protections are ensured first by execution of a routine, the SPEEnc (contained in the game executable file)…
This one allows **variability of addresses**, introduction of **polymorphic code** on its code, but also various techniques of **anti-debugging (anti-bpx, etc…)**, **decoding layers**, **integrity checks**, etc…

The different steps of this protection :

1. Setup of "variable" addresses
2. Setup of SPEEnc (SPEEnc has various anti-cracking techniques such as polymorphic code, anti-debugging code, etc…)
3. Executable code blocks of SPEEnc are progressively decoded

4. Then, there is the protection itself (CD's physical structure, made by the protection, is read).

If CD authentification failed, the following box appears :



5. SPEEnc makes some integrity checks (in particular on itself)…
6. SPEEnc then deciphers tms2003.exe and jumps to its OEP
7. APIs are redirected to a routine of SPEEnc, which calculates and returns the good address of redirected APIs

B) "Cracking" Approach :

1. Locating OEP to unpack the executable file :

After two small jump on OEP, we arrive at 0040117B and with a look at the code, which follows, we can already suspect the executable file to be packed… (see the pushad in 0040119F).

```
01A7:004010D1  EB79                    JMP       0040114C                (JUMP !)

...

01A7:0040114C  EB2D                    JMP       0040117B                (JUMP !)

...

01A7:0040117B  50                      PUSH      EAX
01A7:0040117C  55                      PUSH      EBP
01A7:0040117D  E800000000              CALL      00401182
01A7:00401182  5D                      POP       EBP
01A7:00401183  50                      PUSH      EAX
01A7:00401184  8BC5                    MOV       EAX,EBP
01A7:00401186  81ED82518B04            SUB       EBP,048B5182
01A7:0040118C  2D82010000              SUB       EAX,00000182
01A7:00401191  3E2B853E508B04          SUB       EAX,DS:[EBP+048B503E]
01A7:00401198  3E89853E508B04          MOV       DS:[EBP+048B503E],EAX
01A7:0040119F  60                      PUSHAD
01A7:004011A0  8D85C1518B04            LEA       EAX,[EBP+048B51C1]
01A7:004011A6  50                      PUSH      EAX
01A7:004011A7  8D9DEF588B04            LEA       EBX,[EBP+048B58EF]
01A7:004011AD  2BD8                    SUB       EBX,EAX
01A7:004011AF  53                      PUSH      EBX
01A7:004011B0  3EFFB542508B04          PUSH      DWORD PTR DS:[EBP+048B5042]
01A7:004011B7  6822127293              PUSH      93721222
01A7:004011BC  E82E070000              CALL      004018EF
01A7:004011C1  6C                      INSB
01A7:004011C2  5A                      POP       EDX
```

To arrive at the "real" OEP, lets first test the traditionals bpx APIs (for example, bpx GetVersion)…
Baooom, the PC explodes… lol… No, just a crash…:p
It is thus useless to put any bpx, because of anti-bpx…
How can we arrive at OEP of our unpacked executable file ?
By putting a bpr 402000 650000 R (memory breakpoint).
Why 402000? Because this way, we avoid the protection's instructions (from 4010D1 to 40192A) before the ret in 0040157C (which enables us to arrive on "variable addresses" code).
Prefer a large range, because a lot of games do not have their OEP in the neighbourhoods of 401000 (contrary to majority of sharewares executables)…
The 1st break brings us on a LODSB instruction at 00BA73FE :

```
01A7:00BA73F3  33D2              XOR         EDX,EDX
01A7:00BA73F5  8B450C            MOV         EAX,[EBP+0C]
01A7:00BA73F8  F7E3              MUL         EBX
01A7:00BA73FA  33D0              XOR         EDX,EAX
01A7:00BA73FC  33D3              XOR         EDX,EBX
01A7:00BA73FE  AC                LODSB
01A7:00BA73FF  C1C210            ROL         EDX,10
01A7:00BA7402  02C6              ADD         AL,DH
01A7:00BA7404  32C2              XOR         AL,DL
01A7:00BA7406  C1EA10            SHR         EDX,10
01A7:00BA7409  2AC6              SUB         AL,DH
01A7:00BA740B  32C2              XOR         AL,DL
01A7:00BA740D  AA                STOSB
01A7:00BA740E  43                INC         EBX
01A7:00BA740F  83F901            CMP         ECX,01
01A7:00BA7412  7403              JZ          00BA7417
01A7:00BA7414  49                DEC         ECX
01A7:00BA7415  EBDC              JMP         00BA73F3
01A7:00BA7417  61                POPAD
01A7:00BA7418  C9                LEAVE
01A7:00BA7419  C21000            RET         0010
```

We disable this bpr by a bd 0 and we put a bpx 00BA7417 to leave this loop…
Shucks! We must not use BPX !!! In this case, we crash at 00BA21FA…
So, prefer a bpm 00BA7417 X (hardware breakpoint).
We reactivate then the bpr by a be 0 and we break this time on REPZ MOVSB instruction in 00BA5078 :

```
01A7:00BA5077  FC                CLD
01A7:00BA5078  F3A4              REPZ MOVSB
01A7:00BA507A  8B18              MOV         EBX,[EAX]
01A7:00BA507C  83C004            ADD         EAX,04
01A7:00BA507F  3E8985A9D08B04    MOV         DS:[EBP+048BD0A9],EAX
01A7:00BA5086  3E899DB5D08B04    MOV         DS:[EBP+048BD0B5],EBX
01A7:00BA508D  3E8B9D68D08B04    MOV         EBX,DS:[EBP+048BD068]
01A7:00BA5094  3E899DADD08B04    MOV         DS:[EBP+048BD0AD],EBX
01A7:00BA509B  3E8B9D6CD08B04    MOV         EBX,DS:[EBP+048BD06C]
01A7:00BA50A2  3E899DB1D08B04    MOV         DS:[EBP+048BD0B1],EBX
01A7:00BA50A9  E8001F0000        CALL        00BA6FAE
01A7:00BA50AE  50                PUSH        EAX
01A7:00BA50AF  3EFF9580D08B04    CALL        DS:[EBP+048BD080]
01A7:00BA50B6  61                POPAD
01A7:00BA50B7  EB01              JMP         00BA50BA
```

We disable the bpr (bd 0) again and execute the REPZ MOVSB instruction at 00BA5078 by F10 (Step Over)…
Then reactivate the bpr (be 0) a last time and finally after 1 minute, we get OEP of the unpacked tms2003.exe ;)

Note: An lame method "to find" the OEP of an unpacked executable file consists in dumping on runtime (so it is decrypted/decompressed), disassembling this dump and looking in the code for these instructions :
push ebp
mov ebp, esp
It can constitute a check, but nothing more (because of several occurrences)…

2. Fixing the call Laserlock

If we attentively look at the code on OEP, we have this:

```
01A7:00646957  55                PUSH        EBP
01A7:00646958  8BEC              MOV         EBP,ESP
01A7:0064695A  6AFF              PUSH        FF
01A7:0064695C  68889C8100        PUSH        00819C88
01A7:00646961  68BCB86400        PUSH        0064B8BC
01A7:00646966  64A100000000      MOV         EAX,FS:[00000000]
01A7:0064696C  50                PUSH        EAX
01A7:0064696D  64892500000000    MOV         FS:[00000000],ESP
01A7:00646974  83EC58            SUB         ESP,58
01A7:00646977  53                PUSH        EBX
01A7:00646978  56                PUSH        ESI
01A7:00646979  57                PUSH        EDI
01A7:0064697A  8965E8            MOV         [EBP-18],ESP
01A7:0064697D  FF156077BA00      CALL        [00BA7760]
01A7:00646983  33D2              XOR         EDX,EDX
01A7:00646985  8AD4              MOV         DL,AH
01A7:00646987  891560C78F00      MOV         [008FC760],EDX
01A7:0064698D  8BC8              MOV         ECX,EAX
01A7:0064698F  81E1FF000000      AND         ECX,000000FF
01A7:00646995  890D5CC78F00      MOV         [008FC75C],ECX
01A7:0064699B  C1E108            SHL         ECX,08
```

We see that all the call API (but also JMP API) are replaced by call dword ptr [00BA7760].
It is the redirection of APIs.
(We will call them : "call Laserlock").

Warning : Keep in mind that all addresses like 00BAxxxx, are variable from one execution to another one ("variable addresses"). It is the case of these "call Laserlock". Here, I fixed them to make explanations easier (see the 2nd part of this tutorial, for more details…).

Let us see what there is, inside…

```
01A7:00BA6F30  50                PUSH    EAX
01A7:00BA6F31  50                PUSH    EAX
01A7:00BA6F32  55                PUSH    EBP
01A7:00BA6F33  53                PUSH    EBX
01A7:00BA6F34  56                PUSH    ESI
01A7:00BA6F35  52                PUSH    EDX
01A7:00BA6F36  51                PUSH    ECX
01A7:00BA6F37  8B5C241C          MOV     EBX,[ESP+1C]
01A7:00BA6F3B  E800000000        CALL    00BA6F40
01A7:00BA6F40  5D                POP     EBP
01A7:00BA6F41  81ED70C88B04      SUB     EBP,048BC870
01A7:00BA6F47  3E8BB558D08B04    MOV     ESI,DS:[EBP+048BD058]
01A7:00BA6F4E  8B56FC            MOV     EDX,[ESI-04]
01A7:00BA6F51  B900000000        MOV     ECX,00000000
01A7:00BA6F56  3E2B9D72598B04    SUB     EBX,DS:[EBP+048B5972]
01A7:00BA6F5D  8BC2              MOV     EAX,EDX
01A7:00BA6F5F  2BC1              SUB     EAX,ECX
01A7:00BA6F61  D1E8              SHR     EAX,1
01A7:00BA6F63  03C1              ADD     EAX,ECX
01A7:00BA6F65  391CC6            CMP     [EAX*8+ESI],EBX
01A7:00BA6F68  7707              JA      00BA6F71
01A7:00BA6F6A  740A              JZ      00BA6F76
01A7:00BA6F6C  8BC8              MOV     ECX,EAX
01A7:00BA6F6E  41                INC     ECX
01A7:00BA6F6F  EBEC              JMP     00BA6F5D
01A7:00BA6F71  8BD0              MOV     EDX,EAX
01A7:00BA6F73  4A                DEC     EDX
01A7:00BA6F74  EBE7              JMP     00BA6F5D
01A7:00BA6F76  8B44C604          MOV     EAX,[EAX*8+ESI+04]
01A7:00BA6F7A  8BD8              MOV     EBX,EAX
01A7:00BA6F7C  25FFFFFF7F        AND     EAX,7FFFFFFF
01A7:00BA6F81  81E300000080      AND     EBX,80000000
01A7:00BA6F87  3E038572598B04    ADD     EAX,DS:[EBP+048B5972]
01A7:00BA6F8E  8B00              MOV     EAX,[EAX]
01A7:00BA6F90  83FB00            CMP     EBX,00
01A7:00BA6F93  740E              JZ      00BA6FA3
01A7:00BA6F95  8944241C          MOV     [ESP+1C],EAX
01A7:00BA6F99  59                POP     ECX
01A7:00BA6F9A  5A                POP     EDX
01A7:00BA6F9B  5E                POP     ESI
01A7:00BA6F9C  5B                POP     EBX
01A7:00BA6F9D  5D                POP     EBP
01A7:00BA6F9E  58                POP     EAX
01A7:00BA6F9F  83C404            ADD     ESP,04
01A7:00BA6FA2  C3                RET
01A7:00BA6FA3  89442418          MOV     [ESP+18],EAX
01A7:00BA6FA7  59                POP     ECX
01A7:00BA6FA8  5A                POP     EDX
01A7:00BA6FA9  5E                POP     ESI
01A7:00BA6FAA  5B                POP     EBX
01A7:00BA6FAB  5D                POP     EBP
01A7:00BA6FAC  58                POP     EAX
01A7:00BA6FAD  C3                RET
```

This routine calculates the API address to return, with address of the "call Laserlock", put on stack at time of its call…
Always with the example of the call dword ptr [00BA7760] at 0064697D, let us trace this routine in step over (F10) until 00BA6FAD. After executing the ret, we arrive to API GetVersion (Kernel) located at address BFF92F1B… (addresses, which vary, depending on the OS version).

```
KERNEL32!GetVersion
01A7:BFF92F1B  E810F4FDFF        CALL    BFF72330
01A7:BFF92F20  A900002000        TEST    EAX,00200000
01A7:BFF92F25  7407              JZ      BFF92F2E
01A7:BFF92F27  B8030A0000        MOV     EAX,00000A03
01A7:BFF92F2C  EB14              JMP     BFF92F42
01A7:BFF92F2E  2500000080        AND     EAX,80000000
01A7:BFF92F33  83F801            CMP     EAX,01
01A7:BFF92F36  1BC0              SBB     EAX,EAX
01A7:BFF92F38  25000A0000        AND     EAX,00000A00
01A7:BFF92F3D  2DFCFFFF3F        SUB     EAX,3FFFFFFC
01A7:BFF92F42  C3                RET
```

A little F12 brings us back just after our call Laserlock in 00646983.
To find address of this API in the IAT, we have just to look in the eax register, when we are at 00BA6F8E, on the MOV EAX, [EAX] instruction.
Occurrence is found at 00903938 (idata section).

We must thus replace call Laserlock by call API…
Here, we have just to replace FF156077BA00 by FF1538399000.
e 0064697F 38,39,90,00.
One call fixed ;).

We have just to automate what we have previously done manually, by a small routine (a "call-fixer").
This constitutes an anti-dump protection. So, if these calls are not fixed and that we dump, then executable file "will attempt to go" in memory zones, which do not exist anymore (they were initialized by protection) and it will irreversibly crash…

You will perhaps remember the "Call-fixer" of the previous version of Laserlock (Laserlock version 5, for example, Desperados: Wanted Dead or Alive)?
Well, we can keep the same one to fix all our Call Laserlock of this version (there are only very few things to modify).
If you want more explanations about this routine, I send you back to my previous tutorial :p

Rem: - Compared to the previous version (version 5), there are no more checksums, applied on this routine, which calculates redirection address of the "call Laserlock", nor on the executable file itself , at time of its call. Thus, we can modify the code if it is necessary (and it is what I did, by patching address 00BA6F8E with a jmp edi !!!).
- The exploit, present in the version 5, which consisted in making the program patch itself and replacing the call Laserlock by call/jmp APIs, from the 50th API and which made possible to avoid CD authentification, has been corrected…

Call-fixer (Laserlock.asm):

```
title call_fixer
.386
.model small, stdcall
option casemap :none

.code
   _TextOffset          equ 00401000h
   _TextSize            equ 00411000h
   _RoutinePatch        equ 00BA6F8Eh

start:
   pushad
   mov esi,_RoutinePatch
   mov word ptr [esi],0E7FFh
   call   @1
@1:
   pop edi
   add edi, offset here1-offset @1
   mov edx, _TextOffset
   mov ecx, _TextSize

search_loop:
   cmp word ptr [edx], 15FFh
   jne try_again
   cmp dword ptr [edx+2], 00BA7760h
   jne try_again
   lea eax, [edx+6]
   pushad
   push eax
   ;jmp dword ptr [BA7760h]
   db 0FFh,25h,60h,77h,0BAh,0
here1:
   mov edx, dword ptr [ebp+1Ch]
   mov dword ptr [edx-4], eax
   mov eax, edi
   cmp ebx, 00
   jne @2
   mov byte ptr [edx-5], 25h
   inc eax
@2: add eax, offset here2-offset here1
   mov esi, _RoutinePatch+02
   jmp esi
here2:
   pop eax
   popad
try_again:
   inc edx
   dec ecx
   jne search_loop
   popad
   int 03
end start
```

Type r in adump to obtain an address where we can load our routine…
We assemble and load this routine in memory by the l command of adump.

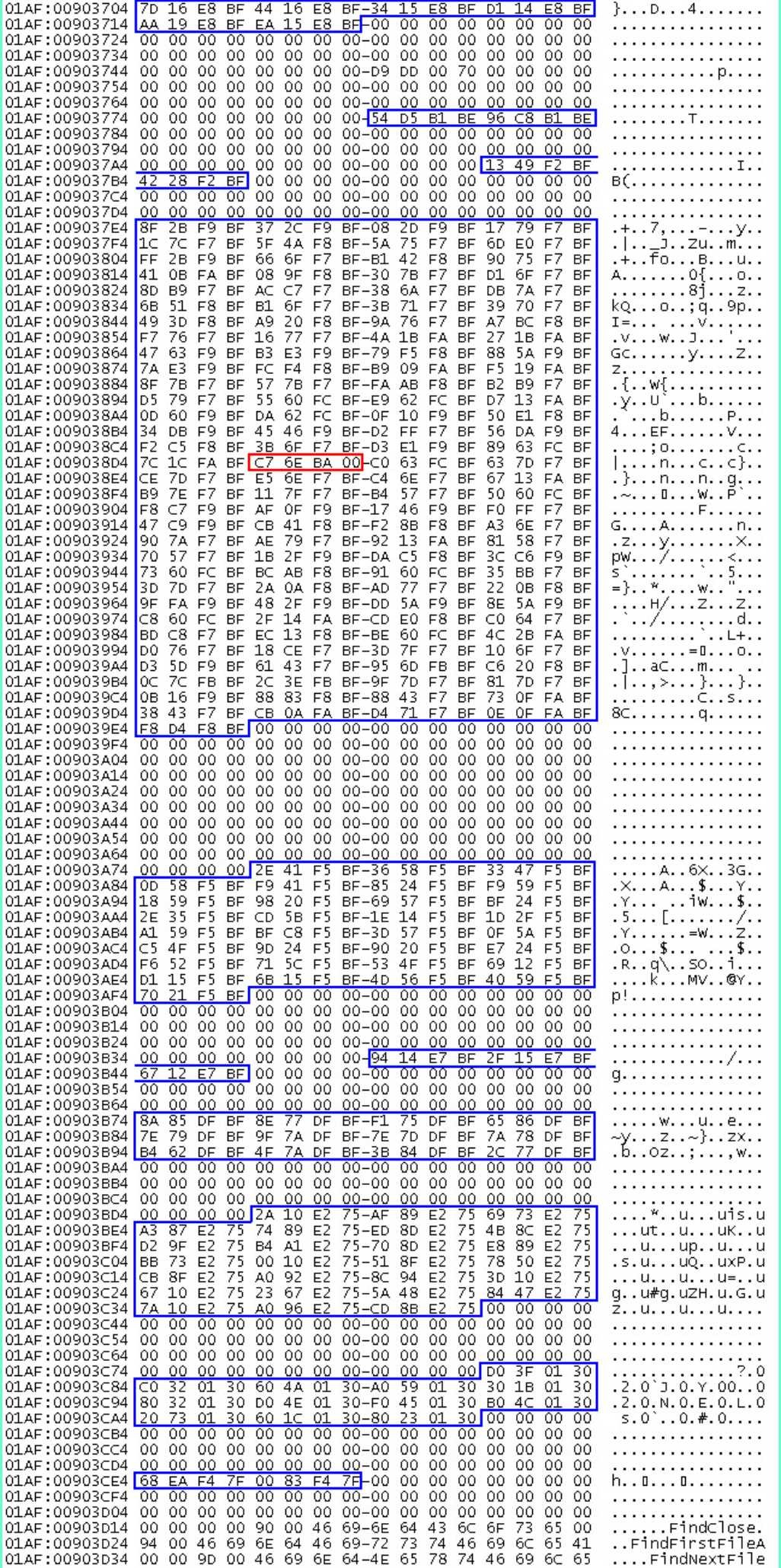
Note : Adump was a binary, which allocated some memory in order to load some routines (or anything else), on Softice.
With ollydbg, you can copy this routine just after PE. It is also possible to allocate some memory to do this on OllyDbg (Alt+M to see the Memory map, then a right click > Allocate Memory).

We break thus on OEP of tms2003.exe and we modifie eip to execute our routine (loaded in Adump).

Note: You have just to type *i3here one* to break on the int 03 of the routine, which indicates the end of its work…
Once the routine has finished its job, we go back to OEP, by a r eip OEP.

All calls are correctly fixed :)

Now, let us examine the IAT (it starts at 00903704 and finishes at 00903CEC).
You will certainly have noticed in this IAT, an "abnormal address", concerning Kernel imports…



Indeed, address at 009038D8 does not point to a BFxxxxxx address, as this should be the case, but to 00BA6EC7…
It is a redirection of the GetProcAddress API. This redirection is done by a modification of the IAT, contrary to the call Laserlock, where the redirection is done by a call to a specific routine, in charge of calculation of address to return and thus requires to transform in the code, all the call/jmp API into call Laserlock…

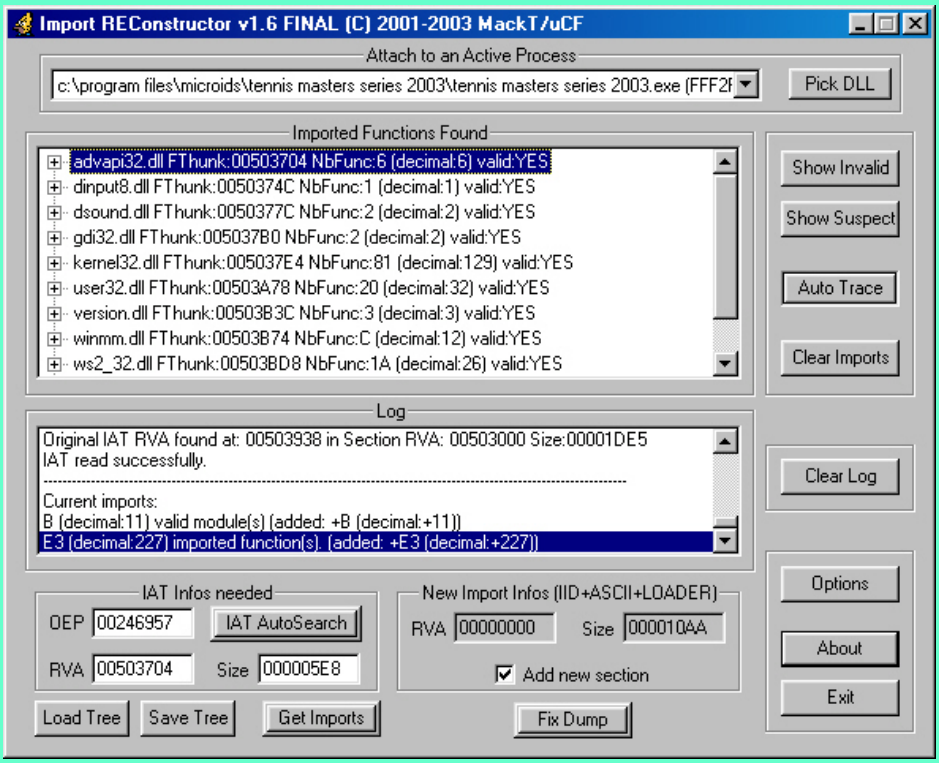
Here's this routine :

```
01A7:00BA6EC7  C8000000          ENTER   0000,00
01A7:00BA6ECB  55                PUSH    EBP
01A7:00BA6ECC  53                PUSH    EBX
01A7:00BA6ECD  56                PUSH    ESI
01A7:00BA6ECE  52                PUSH    EDX
01A7:00BA6ECF  51                PUSH    ECX
01A7:00BA6ED0  8B4508            MOV     EAX,[EBP+08]
01A7:00BA6ED3  8B5D0C            MOV     EBX,[EBP+0C]
01A7:00BA6ED6  E800000000        CALL    00BA6EDB
01A7:00BA6EDB  5D                POP     EBP
01A7:00BA6EDC  81ED0BC88B04      SUB     EBP,048BC80B
01A7:00BA6EE2  83F8FF            CMP     EAX,-01
01A7:00BA6EE5  7537              JNZ     00BA6F1E
01A7:00BA6EE7  53                PUSH    EBX
01A7:00BA6EE8  81E3FFFFFF0F      AND     EBX,0FFFFFFF
01A7:00BA6EEE  81FB6833770F      CMP     EBX,0F773368
01A7:00BA6EF4  5B                POP     EBX
01A7:00BA6EF5  7527              JNZ     00BA6F1E
01A7:00BA6EF7  81E3000000F0      AND     EBX,F0000000
01A7:00BA6EFD  C1C304            ROL     EBX,04
01A7:00BA6F00  83FB00            CMP     EBX,00
01A7:00BA6F03  7409              JZ      00BA6F0E
01A7:00BA6F05  83FB01            CMP     EBX,01
01A7:00BA6F08  740C              JZ      00BA6F16
01A7:00BA6F0A  33C0              XOR     EAX,EAX
01A7:00BA6F0C  EB19              JMP     00BA6F27
01A7:00BA6F0E  8D8530598B04      LEA     EAX,[EBP+048B5930]
01A7:00BA6F14  EB11              JMP     00BA6F27
01A7:00BA6F16  8D8500D08B04      LEA     EAX,[EBP+048BD000]
01A7:00BA6F1C  EB09              JMP     00BA6F27
01A7:00BA6F1E  53                PUSH    EBX
01A7:00BA6F1F  50                PUSH    EAX
01A7:00BA6F20  3EFF9592598B04    CALL    DS:[EBP+048B5992]
01A7:00BA6F27  59                POP     ECX
01A7:00BA6F28  5A                POP     EDX
01A7:00BA6F29  5E                POP     ESI
01A7:00BA6F2A  5B                POP     EBX
01A7:00BA6F2B  5D                POP     EBP
01A7:00BA6F2C  C9                LEAVE
01A7:00BA6F2D  C20800            RET     0008
```

It is the instruction call ds:[ebp+048B5992] at 00BA6F20, which corresponds to a call GetProcAddress.
You just have to replace in IAT (in 009038D8), the value 00BA6F20, by the address of the GetProcAddress API (for me, that will be BFF76DA8).
Here we are : we can dump the executable file :)
Now, we must create a new Imports Table or repair the one we have (and that Laserlock destroyed partially)…

3. Rebuilding the imports

The easiest way: creating a new Import Table :
Run an infinite loop on OEP of tms2003.exe, then run ImpRec, enter good values of the IAT and ImpRec "will attach" to the end of executable file (the previous obtained dump), the new Import Table (it does it by adding a section).



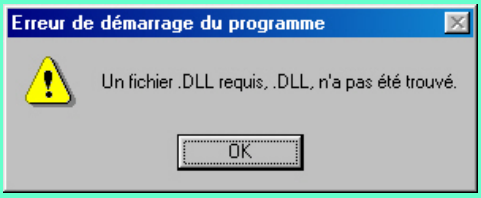
A functional dump is thus obtained, which also run on XP…
Note: You must give the OEP of the unpacked executable. Thus ImpRec also corrects the OEP on PE, which avoids us doing it manually with LordPE.

The hardest way: repairing the original Import Table :
I have dumped so that VO correspond to RO, for more convenience…
Import Table is easily found (generally, it is above the IAT):

```
01AF:00903000 D0 31 50 00 00 00 F7 BF-00 00 00 00 50 40 50 00  .1P.........P@P.
01AF:00903010 E4 37 50 00 64 34 50 00-00 00 F5 BF 00 00 00 00  .7P.d4P.........
01AF:00903020 54 42 50 00 78 3A 50 00-9C 31 50 00 00 00 F2 BF  TBP.x:P..1P.....
01AF:00903030 00 00 00 00 82 42 50 00-B0 37 50 00 F0 30 50 00  .....BP..7P..0P.
01AF:00903040 00 00 E8 BF 00 00 00 00-D0 42 50 00 04 37 50 00  .........BP..7P.
01AF:00903050 60 35 50 00 00 00 DF BF-00 00 00 00 EC 42 50 00  `5P..........BP.
01AF:00903060 74 3B 50 00 6C 36 50 00-00 00 00 30 00 00 00 00  t;P.l6P....0....
01AF:00903070 EA 43 50 00 80 3C 50 00-38 31 50 00 00 00 00 70  .CP..<P.81P....p
01AF:00903080 00 00 00 00 0C 44 50 00-4C 37 50 00 28 35 50 00  .....DP.L7P.(5P.
01AF:00903090 00 00 E7 BF 00 00 00 00-5A 44 50 00 3C 3B 50 00  ........ZDP.<;P.
01AF:009030A0 C4 35 50 00 00 00 E2 75-00 00 00 00 66 44 50 00  .5P....u....fDP.
01AF:009030B0 D8 3B 50 00 D0 36 50 00-00 00 F2 7F 00 00 00 00  .;P..6P....▯....
01AF:009030C0 CC 4A 50 00 E4 3C 50 00-68 31 50 00 00 00 AF BE  .JP..<P.h1P.....
01AF:009030D0 00 00 00 00 6A 4B 50 00-7C 37 50 00 00 00 00 00  ....jKP.|7P.....
01AF:009030E0 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
01AF:009030F0 8C 42 50 00 9E 42 50 00-AC 42 50 00 C0 42 50 00  .BP..BP..BP..BP.
01AF:00903100 8A 4A 50 00 98 4A 50 00-00 00 00 00 00 00 00 00  .JP..JP.........
01AF:00903110 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
01AF:00903120 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
01AF:00903130 00 00 00 00 00 00 00 00-F6 43 50 00 00 00 00 00  .........CP.....
```

To locate it, you have to look for a pointer to the name of a dll (imports), for example KERNEL32.dll, USER32.dll, etc…
You must work of course with VO (remove image base of memory addresses) and take care of possible alignments of the sections (not here) …
Here, the IT starts at 503000 and finishes at 5030F0, which includes 10 null words, indicating as it should be, the end of Import Table (size = 0F0).
Now, put these values for the IT in PE, by editing it with LordPE (EP Editor - > Directories - > Import Table).
For IAT, put RVA and Size to 00000000, and you will not have any problem…
At each launch of program, IAT is always overwritten (by physical addresses of your OS imports…).

Now, we launch our modified executable and we have the following error (A .DLL file, .DLL, is missing) :



lol… I like this error ;-)
We have to check values in Import Table, but we will do so after a remainder (I know, I already do this with my previous tutorial, but it is so concise and clear :p)

Image Import Descriptor Format or Import Table (read "peering inside the PE" of Matt Pietrek!) :

An entry (a DLL and all its functions) in the Image Import Descriptor consists of 5 Dword.

Dword 1 - Characteristics (hint name array)
This dword is a pointer to the first element of a pointers table.
Each pointer of this table points to the hint name, followed with a function name.

Dword 2 - TimeDateStamp

Dword 3 - ForwarderChain

Dword 4 - DLL's name
This dword is a pointer to the name of the DLL (null terminated ASCII string)

Dword 5 - Import Address Counts
This dword is a pointer towards the first element of an addresses table.
This addresses table functions in parallel with the one of pointers to the hint names (names of functions).

Extracted from Import Function tutorial.doc / Import-Function in section RDATA.doc by El.CaRaCoL. (In French)
Thank you El.CaRaCoL. ;-)

Import Table seems normal, apart from all the Dwords 4 (underlined in red)…
Indeed, it points to a null string instead of pointing to the name of a .dll (imported).
However, they point to the good place (in the middle of functions names).
Thus, Laserlock has partially destroyed IT , by erasing the names of .dll…
How can we find the missing names of these .dll?

For an entry, search Dword 5 (underlined in green) corresponding to Dword 4 of the same librairy (underlined in red). We find a pointer to the physical addresses of this dll functions and the 1st address (1st function) of this addresses table enables us to find the name of the library (for example, under Softice, by typing u, followed by this address). Of course, we can do the same with Dword 1 (underlined in blue) and find the name of a library, using one of its functions name…
Now, we have just to patch at the good places, with the library names …

```
00504000 4372 6561 7465 4669 6C65 4D61 7070 696E CreateFileMappin
00504010 6741 0000 AF00 466F 726D 6174 4D65 7373 gA....FormatMess
00504020 6167 6541 0000 2401 4765 744D 6F64 756C ageA..$.GetModul
00504030 6546 696C 654E 616D 6541 0000 2601 4765 eFileNameA..&.Ge
00504040 744D 6F64 756C 6548 616E 646C 6541 0000 tModuleHandleA..
00504050 0000 0000 0000 0000 0000 0000 0000 BE01 ................
00504060 4D65 7373 6167 6542 6F78 4100 2F02 5365 MessageBoxA./.Se
```

```
00504000 4372 6561 7465 4669 6C65 4D61 7070 696E CreateFileMappin
00504010 6741 0000 AF00 466F 726D 6174 4D65 7373 gA....FormatMess
00504020 6167 6541 0000 2401 4765 744D 6F64 756C ageA..$.GetModul
00504030 6546 696C 654E 616D 6541 0000 2601 4765 eFileNameA..&.Ge
00504040 744D 6F64 756C 6548 616E 646C 6541 0000 tModuleHandleA..
00504050 4B45 524E 454C 3332 2E64 6C6C 0000 BE01 KERNEL32.dll....
00504060 4D65 7373 6167 6542 6F78 4100 2F02 5365 MessageBoxA./.Se
```

And here's a summary table :

librairie n°	Dword 5 pointe en	Adresse de la 1ère fonction	Nom de la 1ère fonction	Dword 4 pointe en	Librairie (par déduction)
1	5037E4h	BFF92B8Fh	SetEvent	504050h	KERNEL32.dll
2	503A78h	BFF5412Eh	MessageBoxA	504254h	USER32.dll
3	5037B0h	BFF24913h	GetStockObject	504282h	GDI32.dll
4	503704h	BFEA167Dh	RegCreateKeyExA	5042D0h	ADVAPI32.dll
5	503B74h	BFDF858Ah		5042ECh	WINMM.dll
6	503C80h	30013FD0h		5043EAh	BINKW32.dll
7	50374Ch	7000DDD9h		50440Ch	DINPUT8.dll
8	503B3Ch	BFE91494h	GetFileVersionInfoSizeA	50445Ah	VERSION.dll
9	503BD8h	78E2102Ah		504466h	WS2_32.dll
10	503CE4h	7FF4EA68h		504ACCh	OLE32.dll
11	50377Ch	BEB1D554h		504B6Ah	DSOUND.dll

Once all these modifications are made, start again and a double error message will appear… (it's still better :p)

The 1st thing that I did was to check if the addresses table (Dword 5 - Import Address Table) corresponds to the one (Dword 1) with pointers to the hint names (names of the functions), concerning the KERNEL32.dll library.
Addresses table (Dword 5) for this library starts at 005037E4.
The table (Dword 1) of pointers to the functions names, concerning the same library, starts at 005031D0.

In **005037E4**, we find the BFF92B8F address, which corresponds to SetEvent API. Its function string is pointed in **005031D0**.
In **005037E4**+F0, we find the BFFA1C7C address, which corresponds to CompareStringA API. Its function string is pointed in **005031D0**+F0.
In **005037E4**+F4=005038D8, we find the BFF76DA8 address, which corresponds to **GetProcAddress API** , whereas in **005031D0**+F4, it is the **CreateSemafore** function (its string), which is pointed to !!!

Hum… this must be the problem, here. Moreover, what is this function and why is it there ?

While looking in Win32 Programmer' s Reference, we realize that there is an API of this style, but written as **CreateSemaphore** and not **as CreateSemafore** !!!
Then, impossible not to notice that the strings "CreateSemafore" and "GetProcAddress" have the same size (14 characters) !!!
Laserlock thus replaced the name of GetProcAddress API by a string with same size…
We have just to patch like this (replace by the good API, i.e. GetProcAddress):

```
00504690 0000 C001 4C43 4D61 7053 7472 696E 6757 ....LCMapStringW
005046A0 0000 BF00 4765 7443 5049 6E66 6F00 2100 ....GetCPInfo.!.
005046B0 436F 6D70 6172 6553 7472 696E 6741 0000 CompareStringA..
005046C0 2200 436F 6D70 6172 6553 7472 696E 6757 ".CompareStringW
005046D0 0000 0000 4372 6561 7465 5365 6D61 666F ....CreateSemafo
005046E0 7265 0000 A301 4865 6170 5369 7A65 0000 re....HeapSize..
005046F0 A202 546C 7341 6C6C 6F63 0000 A302 546C ..TlsAlloc....Tl
```

```
00504690 0000 C001 4C43 4D61 7053 7472 696E 6757 ....LCMapStringW
005046A0 0000 BF00 4765 7443 5049 6E66 6F00 2100 ....GetCPInfo.!.
005046B0 436F 6D70 6172 6553 7472 696E 6741 0000 CompareStringA..
005046C0 2200 436F 6D70 6172 6553 7472 696E 6757 ".CompareStringW
005046D0 0000 0000 4765 7450 726F 6341 6464 7265 ....GetProcAddre
005046E0 7373 0000 A301 4865 6170 5369 7A65 0000 ss....HeapSize..
005046F0 A202 546C 7341 6C6C 6F63 0000 A302 546C ..TlsAlloc....Tl
```

You can also, even if it is less clean, modify the pointer to the string "CreateSemafore", so that it points to "GetProcAddress" (for example the one, which is not far from the PE and that Laserlock has "redirected" for its own use…).

```
00000480 0000 A210 0000 B410 0000 0000 0000 0000 ................
00000490 4765 744D 6F64 756C 6548 616E 646C 6541 GetModuleHandleA
000004A0 0000 0000 4765 7450 726F 6341 6464 7265 ....GetProcAddre
000004B0 7373 0000 0000 4C6F 6164 4C69 6272 6172 ss....LoadLibrar
000004C0 7941 0000 4B45 524E 454C 3332 2E64 6C6C yA..KERNEL32.dll
000004D0 00EB 7901 4D53 4D49 3232 3633 352E 3030 ..y.MSMI22635.00
000004E0 4829 4B03 0300 0000 0000 0000 0000 0000 H)K.............
000004F0 0000 0055 6E6B 6F77 6E00 556E 6B6F 776E ...Unkown.Unkown
00000500 0031 302D 3130 2D30 3200 0000 0000 0000 .10-10-02.......
```

In this case, you have to put in 005032C4, a pointer to 000004A2 instead of 005046D2… (You must point in fact, 2 bytes before the name of the function, to point to the ordinal).

Now, the game launches without any problem and much more quickly too ;-)
Moreover, it is compatible XP.
Note : You can add the GetProcAddress ordinal (that can be easily found with ImpRec), but the cracked game launches without problems on other OS, if you didn't add it...

Lastly, it remains a CD-check, but this one is independent of the Laserlock commercial protection and is implemented by the editor (and is thus not very hard to crack).

4. Cracking the CD-check:

If you want to crack the game in order to be able to play without CD, ensure first that game is complete (full installation).
We launch the game, after having removed CD. A box with the following message : "Please insert the game CD" appears…

It is a traditional CD-check (like 90% of CD-checks), which determines initially the type of drives, you own, by **GetDriveTypeA** API, then when it finds a drive letter corresponding to a CD drive, it determines the CD volume (possibly inserted) by **GetVolumeInformationA** API and finally compares volume with the one, which corresponds to the game… *

Introduction of CD-checks corresponds to the beginnings of the Internet and their goals were only to prevent execution of games, illegally distributed by the Internet.
The democratization of CD writers made their use completely obsolete.
Who doesn't have a CD burner now ? Moreover, it is possible to burn a CD-RW with exactly the same label to play and avoid them, unless there are other checks (about presence of a specific file, CD size, etc.)…
These CD-checks are thus here only to annoy people and by no means prevents the intensive "hacking" of games…
(commercial protections have already great difficulty to fight, then even less them…).
And do you believe that game editors, when they develop or test their game, they insert a game CD, which does not exist yet ?
Stop believing and following instructions from editors & co, with their useless and imperative CD-checks like "you must absolutely insert CD"....

To locate the routine in question, you can disassemble with W32Dasm the previous dumped / rebuilt executable file…
You just have to search APIs like **GetDriveTypeA** and **GetVolumeInformationA** APIs .
Four GetDriveTypeA API occurences can be found, but the one, which corresponds to the CD-check is in 0045EC43, where there's the famous cmp eax, 00000005, just after its call…
Concerning the GetVolumeInformationA API, there are 2 occurrences and obviously the important one is in 0045EBE2…

* Here's the routine in question:

```
* Referenced by a CALL at Addresses:
|:0045DC77   , :0045DCE7   , :0045EA06   , :0045EADC   , :005EC431
|
:00401910 E92BD20500                jmp 0045EB40

.
.
.
.
.

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401910(U)
|
:0045EB40 6AFF                      push FFFFFFFF
:0045EB42 6861D27E00                push 007ED261
:0045EB47 64A100000000              mov eax, dword ptr fs:[00000000]
:0045EB4D 50                        push eax
:0045EB4E 64892500000000            mov dword ptr fs:[00000000], esp
:0045EB55 81EC3C010000              sub esp, 0000013C
:0045EB5B 53                        push ebx
:0045EB5C 55                        push ebp
:0045EB5D 56                        push esi
:0045EB5E 57                        push edi
:0045EB5F 6A01                      push 00000001
:0045EB61 8BD9                      mov ebx, ecx

* Reference To: KERNEL32.SetErrorMode, Ord:0264h
                                    |
:0045EB63 FF15F8379000              Call dword ptr [009037F8]
:0045EB69 8A442413                  mov al, byte ptr [esp+13]
:0045EB6D 6A00                      push 00000000
:0045EB6F 8D4C242C                  lea ecx, dword ptr [esp+2C]
:0045EB73 C6430800                  mov [ebx+08], 00
:0045EB77 8844242C                  mov byte ptr [esp+2C], al
:0045EB7B E82728FAFF                call 004013A7

* Possible StringData Ref from Data Obj ->":\"
                                    |
:0045EB80 BF20288800                mov edi, 00882820
:0045EB85 83C9FF                    or ecx, FFFFFFFF
:0045EB88 33C0                      xor eax, eax
:0045EB8A 6A01                      push 00000001
:0045EB8C F2                        repnz
:0045EB8D AE                        scasb
:0045EB8E F7D1                      not ecx
:0045EB90 49                        dec ecx
:0045EB91 8BE9                      mov ebp, ecx
:0045EB93 8D4C242C                  lea ecx, dword ptr [esp+2C]
:0045EB97 55                        push ebp
:0045EB98 E8662BFAFF                call 00401703
:0045EB9D 84C0                      test al, al
:0045EB9F 7423                      je 0045EBC4
:0045EBA1 8B7C242C                  mov edi, dword ptr [esp+2C]
:0045EBA5 8BCD                      mov ecx, ebp
:0045EBA7 8BD1                      mov edx, ecx

* Possible StringData Ref from Data Obj ->":\"
                                    |
:0045EBA9 BE20288800                mov esi, 00882820
:0045EBAE C1E902                    shr ecx, 02
:0045EBB1 F3                        repz
:0045EBB2 A5                        movsd
:0045EBB3 8BCA                      mov ecx, edx
:0045EBB5 55                        push ebp
:0045EBB6 83E103                    and ecx, 00000003
:0045EBB9 F3                        repz
:0045EBBA A4                        movsb
:0045EBBB 8D4C242C                  lea ecx, dword ptr [esp+2C]
:0045EBBF E8BA28FAFF                call 0040147E

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045EB9F(C)
|
:0045EBC4 8A442413                  mov al, byte ptr [esp+13]
:0045EBC8 6A00                      push 00000000
:0045EBCA 8D4C241C                  lea ecx, dword ptr [esp+1C]
:0045EBCE C784245801000000000000    mov dword ptr [esp+00000158], 00000000
:0045EBD9 8844241C                  mov byte ptr [esp+1C], al
:0045EBDD E8C527FAFF                call 004013A7

* Reference To: KERNEL32.GetVolumeInformationA, Ord:0177h
                                    |
:0045EBE2 8B2DF4379000              mov ebp, dword ptr [009037F4]
:0045EBE8 C684245401000001          mov byte ptr [esp+00000154], 01
:0045EBF0 C644241461                mov [esp+14], 61

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045EC92(C)
|
:0045EBF5 8B542414                  mov edx, dword ptr [esp+14]
:0045EBF9 8D4C2428                  lea ecx, dword ptr [esp+28]
:0045EBFD 51                        push ecx
:0045EBFE 8D44243C                  lea eax, dword ptr [esp+3C]
:0045EC02 52                        push edx
:0045EC03 50                        push eax
:0045EC04 E8B63CFAFF                call 004028BF
:0045EC09 83C40C                    add esp, 0000000C
:0045EC0C 6AFF                      push FFFFFFFF
:0045EC0E 6A00                      push 00000000
:0045EC10 50                        push eax
:0045EC11 8D4C2424                  lea ecx, dword ptr [esp+24]
:0045EC15 C684246001000002          mov byte ptr [esp+00000160], 02
```

```
:0045EC1D E82A32FAFF                call 00401E4C
:0045EC22 6A01                      push 00000001
:0045EC24 8D4C243C                  lea ecx, dword ptr [esp+3C]
:0045EC28 C684245801000001          mov byte ptr [esp+00000158], 01
:0045EC30 E87227FAFF                call 004013A7
:0045EC35 8B44241C                  mov eax, dword ptr [esp+1C]
:0045EC39 85C0                      test eax, eax
:0045EC3B 7505                      jne 0045EC42
:0045EC3D B86C208100                mov eax, 0081206C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045EC3B(C)
|
:0045EC42 50                        push eax

* Reference To: KERNEL32.GetDriveTypeA, Ord:0104h
                                    |
:0045EC43 FF15F0379000              Call dword ptr [009037F0]
:0045EC49 83F805                    cmp eax, 00000005
:0045EC4C 7538                      jne 0045EC86
:0045EC4E 8B44241C                  mov eax, dword ptr [esp+1C]
:0045EC52 85C0                      test eax, eax
:0045EC54 7505                      jne 0045EC5B
:0045EC56 B86C208100                mov eax, 0081206C

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045EC54(C)
|
:0045EC5B 6A00                      push 00000000
:0045EC5D 6A00                      push 00000000
:0045EC5F 6A00                      push 00000000
:0045EC61 6A00                      push 00000000
:0045EC63 6A00                      push 00000000
:0045EC65 8D4C245C                  lea ecx, dword ptr [esp+5C]
:0045EC69 6804010000                push 00000104
:0045EC6E 51                        push ecx
:0045EC6F 50                        push eax
:0045EC70 FFD5                      call ebp

* Possible StringData Ref from Data Obj ->"CDTMS2003"
                                    |
:0045EC72 BF14288800                mov edi, 00882814
:0045EC77 8D742448                  lea esi, dword ptr [esp+48]
:0045EC7B B90A000000                mov ecx, 0000000A
:0045EC80 33D2                      xor edx, edx
:0045EC82 F3                        repz
:0045EC83 A6                        cmpsb
:0045EC84 7414                      je 0045EC9A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045EC4C(C)
|
:0045EC86 8A442414                  mov al, byte ptr [esp+14]
:0045EC8A FEC0                      inc al
:0045EC8C 3C7A                      cmp al, 7A
:0045EC8E 88442414                  mov byte ptr [esp+14], al
:0045EC92 0F8E5DFFFFFF              jle 0045EBF5
:0045EC98 EB07                      jmp 0045ECA1

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045EC84(C)
|
:0045EC9A 8A442414                  mov al, byte ptr [esp+14]
:0045EC9E 884308                    mov byte ptr [ebx+08], al

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045EC98(U)
|
:0045ECA1 6A00                      push 00000000

* Reference To: KERNEL32.SetErrorMode, Ord:0264h
                                    |
:0045ECA3 FF15F8379000              Call dword ptr [009037F8]

.
.
.
.
.

:0045ED21 C3                        ret
```

This routine can be called from 5 different places in the code. First, it determines, if a drive letter corresponds to a CD drive, by a call to **GetDriveTypeA** api in 0045EC43.

• If it is not the case, we jump in 0045EC86, where al (letter designating a drive) is incremented, then we return to the previous loop (in 0045EBF5) until finding a CD drive or by default, terminating at letter z …

• If the routine finds a CD drive, we continue with a call to **GetVolumeInformationA** API (in 0045EC70). If a CD is inserted in drive, this API returns the volume of this CD, volume, which is immediately compared with the string "CDTMS2003" at 0045EC82.

If the good CD is inserted, the drive letter, where CD is inserted, is finally stored in memory in order to be tested further…

If CD is not present or it is not the good one, we start again the loop, which accomplishes these tests, by incrementing al (letter drive to test) to letter z …

As we saw, the previous routine can be called 5 times.

The call to this routine, in 0045DC77 is the one, which determines or not the MessageBoxA display (which is done by the call ebx in 0045DCDA) :

```
:0045DC77 E8943CFAFF                call 00401910
:0045DC7C 84C0                      test al, al
:0045DC7E 7570                      jne 0045DCF0

* Reference To: USER32.MessageBoxA, Ord:01BEh
                                    |
:0045DC80 8B1D783A9000              mov ebx, dword ptr [00903A78]

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045DCEE(C)
|
:0045DC86 E8693FFAFF                call 00401BF4
:0045DC8B 8BF0                      mov esi, eax
:0045DC8D 8D7E08                    lea edi, dword ptr [esi+08]
:0045DC90 8BCF                      mov ecx, edi
:0045DC92 E8474BFAFF                call 004027DE
:0045DC97 3D85030000                cmp eax, 00000385
:0045DC9C 761D                      jbe 0045DCBB
:0045DC9E 8B761C                    mov esi, dword ptr [esi+1C]
:0045DCA1 8A8685030000              mov al, byte ptr [esi+00000385]
:0045DCA7 84C0                      test al, al
:0045DCA9 7410                      je 0045DCBB
:0045DCAB 6885030000                push 00000385
:0045DCB0 8BCF                      mov ecx, edi
:0045DCB2 E8B93BFAFF                call 00401870
:0045DCB7 8B00                      mov eax, dword ptr [eax]
:0045DCB9 EB05                      jmp 0045DCC0

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0045DC9C(C), :0045DCA9(C)
|
:0045DCBB B858228100                mov eax, 00812258

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0045DCB9(U)
|
:0045DCC0 6835200500                push 00052035
:0045DCC5 50                        push eax
:0045DCC6 6886030000                push 00000386
:0045DCCB E8243FFAFF                call 00401BF4
:0045DCD0 8BC8                      mov ecx, eax
:0045DCD2 E8D73EFAFF                call 00401BAE
:0045DCD7 50                        push eax
:0045DCD8 6A00                      push 00000000
:0045DCDA FFD3                      call ebx
:0045DCDC 83F802                    cmp eax, 00000002
:0045DCDF 0F841D010000              je 0045DE02
:0045DCE5 8BCD                      mov ecx, ebp
:0045DCE7 E8243CFAFF                call 00401910
:0045DCEC 84C0                      test al, al
:0045DCEE 7496                      je 0045DC86
```

The loop, starting at 0045DC86, is executed each time we click on "Try again" and each time there is not or a bad CD, inserted (test in 0045DCE7).
If you have just avoid this 1st check, while launching a game, you will have this (Insert CD) :

It is the check at 005EC431. You will not leave this message as long as you will have not inserted the game CD …
Thus, to crack definitively this game, it is proper to force the jump in **0045EC84** (the comparison between the volume and the string "CDTMS2003"), rather than to force the 5 jumps following the calls to the check routine (in 0045DC77, 0045DCE7, 0045EA06, 0045EADC and 005EC431).

C) "Reversing" Approach :

1. Fixing "variable" addresses :
The first problem is these "variable" addresses : it is never the same values on each launching…
What we know, it is that addresses are something like 00BAxxxx, where xxxx vary each time…
We launch tms2003.exe with Symbol Loader and come here (on OEP of this packed program) :

```
01C7:004010D1  EB79  JMP 0040114C

puis en regardant un peu plus bas, on a du code très intéressant :

01C7:0040134A  6A40                PUSH      40
01C7:0040134C  6800100000          PUSH      00001000
01C7:00401351  6800020000          PUSH      00000200
01C7:00401356  6A00                PUSH      00
01C7:00401358  3EFF95BB578B04      CALL      DS:[EBP+048B57BB]
...
01C7:00401410  6A40                PUSH      40
01C7:00401412  6800100000          PUSH      00001000
01C7:00401417  50                  PUSH      EAX
01C7:00401418  6A00                PUSH      00
01C7:0040141A  3EFF95BB578B04      CALL      DS:[EBP+048B57BB]
...
```

After execution of call at 00401358, we have eax = 00B90000.
After execution of call at 0040141A, we have eax = 00BA0000. Hum…

And, while tracing a little, here's the code, which fixes the "random" aspect of addresses in protection :

```
01C7:00401446  50                  PUSH      EAX
01C7:00401447  E88A010000          CALL      004015D6
01C7:0040144C  59                  POP       ECX
01C7:0040144D  25FFFF0000          AND       EAX,0000FFFF
01C7:00401452  03C8                ADD       ECX,EAX
01C7:00401454  3E898DBF578B04      MOV       DS:[EBP+048B57BF],ECX
```

The variability of addresses is determined by the variable / random value, contained in eax and computed by routine located in 004015D6, called by the call 004015D6 on 00401447 address.
To remove any variability, type r eax 0, after execution of call 004015D6.
You will thus have same instructions of protection, always at the same addresses. That gets us out of a mess …
How this "random" value is calculated ?
Quite simply from returned values of GetLocalTime and GetSystemTime APIs …

```
01C7:004015D6  53                  PUSH      EBX
01C7:004015D7  51                  PUSH      ECX
01C7:004015D8  52                  PUSH      EDX
01C7:004015D9  57                  PUSH      EDI
01C7:004015DA  56                  PUSH      ESI
01C7:004015DB  83EC18              SUB       ESP,18
01C7:004015DE  8D442418            LEA       EAX,[ESP+18]
01C7:004015E2  50                  PUSH      EAX
01C7:004015E3  3EFF95AA578B04      CALL      DS:[EBP+048B57AA] <- Call GetLocalTime
01C7:004015EA  8D7C2418            LEA       EDI,[ESP+18]
01C7:004015EE  B908000000          MOV       ECX,00000008
01C7:004015F3  660307              ADD       AX,[EDI]
01C7:004015F6  83C702              ADD       EDI,02
01C7:004015F9  E2F8                LOOP      004015F3
01C7:004015FB  89442412            MOV       [ESP+12],EAX
01C7:004015FF  8D442418            LEA       EAX,[ESP+18]
01C7:00401603  50                  PUSH      EAX
01C7:00401604  3EFF95AE578B04      CALL      DS:[EBP+048B57AE] <- Call GetSystemTime
01C7:0040160B  8D7C2418            LEA       EDI,[ESP+18]
01C7:0040160F  B910000000          MOV       ECX,00000010
01C7:00401614  660307              ADD       AX,[EDI]
01C7:00401617  83C702              ADD       EDI,02
01C7:0040161A  E2F8                LOOP      00401614
01C7:0040161C  6689442410          MOV       [ESP+10],AX
01C7:00401621  668B4C2412          MOV       CX,[ESP+12]
01C7:00401626  66D3C1              ROL       CX,CL
01C7:00401629  66894C2412          MOV       [ESP+12],CX
01C7:0040162E  668B4C2410          MOV       CX,[ESP+10]
01C7:00401633  66D3C9              ROR       CX,CL
01C7:00401636  668B5C2412          MOV       BX,[ESP+12]
01C7:0040163B  6633CB              XOR       CX,BX
01C7:0040163E  668BC1              MOV       AX,CX
01C7:00401641  83C418              ADD       ESP,18
01C7:00401644  5E                  POP       ESI
01C7:00401645  5F                  POP       EDI
01C7:00401646  5A                  POP       EDX
01C7:00401647  59                  POP       ECX
01C7:00401648  5B                  POP       EBX
01C7:00401649  C3                  RET
```

Let us leave this last routine by executing the ret in 00401649 and let's continue to trace the program until 004014D5 :

```
01A7:0040148D  6A10              PUSH      10
01A7:0040148F  8D854B568B04      LEA       EAX,[EBP+048B564B]
01A7:00401495  50                PUSH      EAX
01A7:00401496  8D85C3568B04      LEA       EAX,[EBP+048B56C3]
01A7:0040149C  50                PUSH      EAX
01A7:0040149D  6A00              PUSH      00
01A7:0040149F  3EFF95B2578B04    CALL      DS:[EBP+048B57B2]
01A7:004014A6  3EFF95B6578B04    CALL      DS:[EBP+048B57B6]
 ==> 004014AD  53                PUSH      EBX
01A7:004014AE  3EFF9596578B04    CALL      DS:[EBP+048B5796]
01A7:004014B5  3E8B85BF578B04    MOV       EAX,DS:[EBP+048B57BF]
01A7:004014BC  50                PUSH      EAX
01A7:004014BD  BBCDD48B04        MOV       EBX,048BD4CD
01A7:004014C2  81EB30598B04      SUB       EBX,048B5930
01A7:004014C8  53                PUSH      EBX
01A7:004014C9  3EFFB542508B04    PUSH      DWORD PTR DS:[EBP+048B5042]
01A7:004014D0  68C2BAEDFE        PUSH      FEEDBAC2
01A7:004014D5  E815040000        CALL      004018EF
01A7:004014DA  3E8B9DBF578B04    MOV       EBX,DS:[EBP+048B57BF]
```

What does the routine called in 004014D5 make ?
It precisely deciphers beginning of code, which contains variable addresses …
Here, it is a block starting in 00BA0000 (esi) and with a size of 7B9D (ecx)…

```
01A7:004018EF  C8000000          ENTER     0000,00
01A7:004018F3  60                PUSHAD
01A7:004018F4  8B7514            MOV       ESI,[EBP+14]
01A7:004018F7  8BFE              MOV       EDI,ESI
01A7:004018F9  8B4D10            MOV       ECX,[EBP+10]
01A7:004018FC  83F900            CMP       ECX,00
01A7:004018FF  7427              JZ        00401928
01A7:00401901  8B5D08            MOV       EBX,[EBP+08]
01A7:00401904  33D2              XOR       EDX,EDX
01A7:00401906  8B450C            MOV       EAX,[EBP+0C]
01A7:00401909  F7E3              MUL       EBX
01A7:0040190B  33D0              XOR       EDX,EAX
01A7:0040190D  33D3              XOR       EDX,EBX
01A7:0040190F  AC                LODSB
01A7:00401910  C1C210            ROL       EDX,10
01A7:00401913  02C6              ADD       AL,DH
01A7:00401915  32C2              XOR       AL,DL
01A7:00401917  C1EA10            SHR       EDX,10
01A7:0040191A  2AC6              SUB       AL,DH
01A7:0040191C  32C2              XOR       AL,DL
01A7:0040191E  AA                STOSB
01A7:0040191F  43                INC       EBX
01A7:00401920  83F901            CMP       ECX,01
01A7:00401923  7403              JZ        00401928
01A7:00401925  49                DEC       ECX
01A7:00401926  EBDC              JMP       00401904
01A7:00401928  61                POPAD
01A7:00401929  C9                LEAVE
01A7:0040192A  C21000            RET       0010
```

Once this block of instructions is deciphered, it has just to jump on this last…

```
01C7:0040155A  BEC05A8B04        MOV       ESI,048B5AC0
01C7:0040155F  81EE30598B04      SUB       ESI,048B5930
01C7:00401565  03DE              ADD       EBX,ESI
01C7:00401567  3E899D7E578B04    MOV       DS:[EBP+048B577E],EBX
01C7:0040156E  61                POPAD
01C7:0040156F  3E8B857E578B04    MOV       EAX,DS:[EBP+048B577E]
01C7:00401576  89442408          MOV       [ESP+08],EAX
01C7:0040157A  58                POP       EAX
01C7:0040157B  5D                POP       EBP
01C7:0040157C  C3                RET
```

Notice that the jump is not done to 00BA0000 (ebx), but to 00BA0000 + 048B5AC0 - 048B5930 = 00BA0190 (ebx + esi), computed by instructions in 0040155A, 0040155F and 00401565…
Finally, we arrive then after execution of previous ret, in 00BA0190 (beginning of code with variable addresses):

```
01C7:00BA0190  EB04              JMP       00BA0196                (JUMP !)
01C7:00BA0192  E8EB04E9EB        CALL      ECA30682
01C7:00BA0197  FB                STI
01C7:00BA0198  E950EB04E8        JMP       E8BEECED
01C7:00BA019D  EB04              JMP       00BA01A3
01C7:00BA019F  E9EBFBE9E8        JMP       E9A3FD8F
01C7:00BA01A4  0300              ADD       EAX,[EAX]
01C7:00BA01A6  0000              ADD       [EAX],AL
01C7:00BA01A8  EB01              JMP       00BA01AB
01C7:00BA01AA  E858EB04E8        CALL      E8BEED07
01C7:00BA01AF  EB04              JMP       00BA01B5
01C7:00BA01B1  E9EBFBE958        JMP       59A3FDA1
01C7:00BA01B6  EB04              JMP       00BA01BC
01C7:00BA01B8  E8EB04E9EB        CALL      ECA306A8
01C7:00BA01BD  FB                STI
01C7:00BA01BE  E9EB018753        JMP       544103AE
01C7:00BA01C3  BBEB0724D8        MOV       EBX,D82407EB
01C7:00BA01C8  5B                POP       EBX
01C7:00BA01C9  EBF9              JMP       00BA01C4
```

i.e. polymorphic code (interleaved jumps, obfuscation code, junk code), supposed to make hard crackers' life…
Here is another example of this type of code :

```
01C7:00BA1EA7  EB01              JMP       00BA1EAA                (JUMP !)
01C7:00BA1EA9  CC                INT       3
 ==> 00BA1EAA  57                PUSH      EDI
01C7:00BA1EAB  BFEB07EF68        MOV       EDI,68EF07EB
01C7:00BA1EB0  5F                POP       EDI
01C7:00BA1EB1  EBF9              JMP       00BA1EAC
```

Rem: If you want to trace this code, you have to use the Step Into (F8), because of these "call eip+8", which are everywhere in this code…

2. Polymorphic code :

Before an approach of SPEEnc polymorphic code, here is a reminder on this subject by Pulsar (Thank you Pulsar) :

Polymorphic code or overlapping :

It can happen that tracing in SoftIce is obstructed by Appearances Changing Codes (CCA):
Softice disassembles an instruction at offset EIP (let us say that the number of bytes that it makes is called NBI), followed by an instruction, positioned to EIP+NBI....
It is thus rather easy to make CCA.
Let us imagine the following sequence :

```
jmp @1
dB E8h
@1:
mov eax, [eax]
inc eax
inc edx
```

In memory, you have something like :

```
EB01
E8
8B004042
```

but E8 is the byte, where begins an opcode like "call offset", so this code, if it is disassembled by increasing addresses will give :

```
EB01 jmp @1
E88B004042 call addresses
```

When you will trace by jumping the jump, you fall in the middle of the call opcode, and softice will re-disassemble this instruction, what will give the impression, that the code changes appearance - > CCA
You surely understood that it is thus very easy to make code like this, by adding various types of bytes. Instead of E8h, we can add 0CDh, 020h which corresponds to a int 20h - > VMMJump, which uses the DWORD, following the opcode… and SoftIce will thus display something like this :

```
int 20 VXDJump XXXX, XXXX
```

with XXXX, the XXXX value contained in the DWORD, following the int 20....

If you have not have understood it, this type of coding is particularly long, painful, and tiring to trace (not to say....), and you have to trace it VERY carefully, since you practically never see, which will be the TRUE following instruction.
To have already met this type of coding, allmost ALL Call address must be traced Step Into (F8).
A solution to trace the CCA is to do it "by the old way" by looking at memories addresses on the left of the screen, and as soon as there is a rather consequent jump, it is that the packer has finished its work.

Another major disadvantage of the CCA is the difficulty in putting BreakPoints on eXecution, SoftIce doesn't accept to put its INT3 in the middle of a code. The BPM address X give good results, but are limited to four. *Pulsar (from a text about anti-debugging)*

Now, we can study polymorphic code of SPEEnc ;)
To be able to clean it as well as possible, it is advised to study in detail its basic structure :

```
01C7:00BA0190  EB04                 JMP       00BA0196
01C7:00BA0193  EB04                 JMP       00BA0199
01C7:00BA0196  EBFB                 JMP       00BA0193
01C7:00BA0199  50                   PUSH      EAX
01C7:00BA019A  EB04                 JMP       00BA01A0
01C7:00BA019D  EB04                 JMP       00BA01A3
01C7:00BA01A0  EBFB                 JMP       00BA019D
01C7:00BA01A3  E803000000           CALL      00BA01AB
01C7:00BA01AB  58                   POP       EAX
01C7:00BA01AC  EB04                 JMP       00BA01B2
01C7:00BA01AF  EB04                 JMP       00BA01B5
01C7:00BA01B2  EBFB                 JMP       00BA01AF
01C7:00BA01B5  58                   POP       EAX
01C7:00BA01B6  EB04                 JMP       00BA01BC
01C7:00BA01B9  EB04                 JMP       00BA01BF
01C7:00BA01BC  EBFB                 JMP       00BA01B9

01C7:00BA01BF  EB01                 JMP       00BA01C2
01C7:00BA01C2  53                   PUSH      EBX
01C7:00BA01C3  BBEB0724D8           MOV       EBX,D82407EB
01C7:00BA01C4  EB07                 (JMP      00BA01CD)
01C7:00BA01C8  5B                   POP       EBX
01C7:00BA01C9  EBF9                 JMP       00BA01C4

01C7:00BA01CD  50                   PUSH      EAX <---

01C7:00BA01CE  EB01                 JMP       00BA01D1
01C7:00BA01D1  51                   PUSH      ECX
01C7:00BA01D2  B9EB077E0A           MOV       ECX,0A7E07EB
01C7:00BA01D3  EB07                 (JMP      00BA01DC)
01C7:00BA01D7  59                   POP       ECX
01C7:00BA01D8  EBF9                 JMP       00BA01D3

01C7:00BA01DC  55                   PUSH      EBP <---

01C7:00BA01DD  EB01                 JMP       00BA01E0
01C7:00BA01E0  51                   PUSH      ECX
01C7:00BA01E1  B9EB0788C6           MOV       ECX,C68807EB
01C7:00BA01E2  EB07                 (JMP      00BA01EB)
01C7:00BA01E6  59                   POP       ECX
01C7:00BA01E7  EBF9                 JMP       00BA01E2

01C7:00BA01EB  E800000000           CALL      00BA01F0
01C7:00BA01F0  5D                   POP       EBP

01C7:00BA01F1  EB04                 JMP       00BA01F7
01C7:00BA01F4  EB04                 JMP       00BA01FA
01C7:00BA01F7  EBFB                 JMP       00BA01F4
01C7:00BA01FA  50                   PUSH      EAX
01C7:00BA01FB  EB04                 JMP       00BA0201
01C7:00BA01FE  EB04                 JMP       00BA0204
01C7:00BA0201  EBFB                 JMP       00BA01FE
01C7:00BA0204  E803000000           CALL      00BA020C
01C7:00BA020C  58                   POP       EAX
01C7:00BA020D  EB04                 JMP       00BA0213
01C7:00BA0210  EB04                 JMP       00BA0216
01C7:00BA0213  EBFB                 JMP       00BA0210
01C7:00BA0216  58                   POP       EAX
01C7:00BA0217  EB04                 JMP       00BA021D
01C7:00BA021D  EBFB                 JMP       00BA021A
01C7:00BA021A  EB04                 JMP       00BA0220

01C7:00BA0220  EB01                 JMP       00BA0223
01C7:00BA0223  53                   PUSH      EBX
01C7:00BA0224  BBEB07802F           MOV       EBX,2F8007EB
01C7:00BA0225  EB07                 (JMP      00BA022E)
01C7:00BA022A  EBF9                 JMP       00BA0225

01C7:00BA022E  50                   PUSH      EAX <---

01C7:00BA022F  EB04                 JMP       00BA0235
01C7:00BA0232  EB04                 JMP       00BA0238
01C7:00BA0235  EBFB                 JMP       00BA0232
01C7:00BA0238  50                   PUSH      EAX
01C7:00BA0239  EB04                 JMP       00BA023F
01C7:00BA023C  EB04                 JMP       00BA0242
01C7:00BA023F  EBFB                 JMP       00BA023C
01C7:00BA0242  E803000000           CALL      00BA024A
01C7:00BA024A  58                   POP       EAX
01C7:00BA024B  EB04                 JMP       00BA0251
01C7:00BA024E  EB04                 JMP       00BA0254
01C7:00BA0251  EBFB                 JMP       00BA024E
01C7:00BA0254  58                   POP       EAX
01C7:00BA0255  EB04                 JMP       00BA025B
01C7:00BA0258  EB04                 JMP       00BA025E
01C7:00BA025B  EBFB                 JMP       00BA0258

01C7:00BA025E  EB01                 JMP       00BA0261
01C7:00BA0261  55                   PUSH      EBP
01C7:00BA0262  BDEB07DDC6           MOV       EBP,C6DD07EB
01C7:00BA0263  EB07                 (JMP      00BA026C)
01C7:00BA0267  5D                   POP       EBP
01C7:00BA0268  EBF9                 JMP       00BA0263

01C7:00BA026C  8BC5                 MOV       EAX,EBP <-

01C7:00BA026E  EB01                 JMP       00BA0271
01C7:00BA0271  51                   PUSH      ECX
01C7:00BA0272  B9EB07FAFA           MOV       ECX,FAFA07EB
01C7:00BA0273  EB07                 (JMP      00BA027C)
```

```
01C7:00BA0267  5D                  POP       EBP
01C7:00BA0268  EBF9                JMP       00BA0263

01C7:00BA026C  8BC5                MOV       EAX,EBP <-

01C7:00BA026E  EB01                JMP       00BA0271
01C7:00BA0271  51                  PUSH      ECX
01C7:00BA0272  B9EB07FAFA          MOV       ECX,FAFA07EB
01C7:00BA0273  EB07                (JMP      00BA027C)
01C7:00BA0277  59                  POP       ECX
01C7:00BA0278  EBF9                JMP       00BA0273

01C7:00BA027C  81ED205B8B04        SUB       EBP,048B5B20                    <-
01C7:00BA0282  3E83BD50D08B0401    CMP       DWORD PTR DS:[EBP+048BD050],01  <-
01C7:00BA028A  7505                JNZ       00BA0291                        <-
01C7:00BA028C  E90F6A0000          JMP       00BA6CA0                        <-
01C7:00BA0291  3E83BD50D08B0402    CMP       DWORD PTR DS:[EBP+048BD050],02  <-
01C7:00BA0299  7505                JNZ       00BA02A0                        <-
01C7:00BA029B  E9FB6B0000          JMP       00BA6E9B                        <-

01C7:00BA02A0  EB04                JMP       00BA02A6
01C7:00BA02A3  EB04                JMP       00BA02A9
01C7:00BA02A6  EBFB                JMP       00BA02A3
01C7:00BA02A9  50                  PUSH      EAX
...
```

We see a basic sheme with 2 different types of blocks (the 1st and the 2nd) with variations about register nature.

Note: It is not really a listing, such as we could obtain after disassembling… I modified this one to thus make it dynamic "and more comprehensible"...
For example, on the 2nd block, after execution of the MOV EBX, D82407EB in 00BA01C3, we executes the POP EBX (in 00BA01C3+5=00BA01C8) and not the (JMP 00BA01CD) "contained" in instruction MOV EBX, D82407EB, jump at which we arrive by JMP 00BA01C4…
You have to be carefull with the call eip+8, which constitutes "anti-tracing" trick, because a Step Over (F10) in one of these calls starts launching of the program… Thus, you have to trace all code in Step Into (F8), which combined to the polymorphic code, quickly becomes very painful...

The instructions "really" executed are those indicated by an arrow < - (these are those that the author wants to dissimulate, and present before introduction of polymorphic code and finally are important protection instructions…).
The junk code is consequent : on 100 instructions, approximately 90 relate to this last… The "real" code hardly represents 10% of this routine :(
If we reason with place occupied by these junk code instructions, we have approximately 80 bytes occupied by this one, on a total of 120 bytes, that is to say approximately 2/3…
The relative importance (in weight) of this junk code, in the protection code is thus significant !!!
But the fault of this polymorphic code comes to the fact that it is generic; its basic structure is easy to find. It is thus easy to destroy it :)
Poor protectionists… Perhaps another time ?

The polymorphic code being dissected, we can now destroy it using this routine :

```
title cca
.386
.model small, stdcall
option casemap :none

.code

   _TextStart        equ 00BA0000h
   _TextEnd          equ 00BA7B9Dh

start:
   pushad
   mov edx, _TextStart

@1: cmp edx, _TextEnd
   je @2

check1:
   cmp word ptr [edx], 01EBh
   jne check2
   cmp word ptr [edx+05], 07EBh
   jne check2
   cmp word ptr [edx+0Ah], 0F9EBh
   jne check2
   mov ecx, 0Eh
   jmp paste

check2:
   cmp word ptr [edx], 04EBh
   jne try_again
   cmp word ptr [edx+03], 04EBh
   jne try_again
   cmp dword ptr [edx+14h], 03
   jne try_again
   cmp word ptr [edx+29h], 04EBh
   jne try_again
   cmp word ptr [edx+2Ch], 0FBEBh
   jne try_again
   mov ecx, 2Fh
```

```
    jmp paste

paste:
    mov al, 90h
    mov edi, edx
    rep stosb

try_again:
    inc edx
    jmp @1

@2: popad
    int 03
    nop

end start
```

You have only to put, in _TextStart, address of polymorphic code/routine beginning and in _TextEnd, its end…
Assemble it and load it in adump.
The routine then will nop us all the useless instructions and then, let us show what is useful and interesting ;).
Dump this "unpolymorphic" code… (/DUMP 00BA0000 7B9D C:\SPEEnc.dat under SI).

3. Code in general :

We can now study SPEEnc, quietly…
We can already say, just by using a hexadecimal editor, that this SPEEnc is mainly executable code mingled with datas, comments and even imports!!!

```
00000000 5C3D 2D53 5065 456E 6320 6465 636F 6465 \=-SPeEnc decode
00000010 2072 7574 696E 6520 6430 322D 3037 2D30  rutine d02-07-0
00000020 3172 2062 7920 4173 7465 7269 6F73 2050 1r by Asterios P
00000030 6172 6C61 6D65 6E74 6173 5C3D 2D00 5769 arlamentas\=-.Wi
00000040 2400 0000 4000 0090 5000 2070 0300 F000 $...@...P. p....
00000050 0000 0030 5000 DB10 6D9B 30AF A1EF 1677 ...0P...m.0....w
00000060 F7BF A86D F7BF D076 F7BF 0010 4000 F776 ...m...v....@..v
00000070 F7BF 3B71 F7BF D471 F7BF 0E0F FABF 2E41 ..;q...q.......A
00000080 F5BF F8D4 F8BF DB7A F7BF FF02 0090 0F80 .......z........
00000090 FF76 AAE2 5C3D 2D69 4E66 4563 5465 4420 .v..\=-iNfEcTeD
000000A0 4272 4069 4E20 4C61 6273 2E20 5072 6F6A Br@iN Labs. Proj
000000B0 6563 743A 2073 4372 406D 426C 4564 2042 ect: sCr@mBlEd B
000000C0 7240 694E 2E2D 3D2F 506C 6561 7365 2049 r@iN.-=/Please I
000000D0 6E73 6572 7420 4F72 6967 696E 616C 2044 nsert Original D
000000E0 6973 6B20 696E 2044 7269 7665 0000 0000 isk in Drive....
000000F0 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

```
00002AC0 9090 9090 9090 9090 9090 E9D2 E3FF FF57 ...............W
00002AD0 4152 4E49 4E47 3A20 5468 6973 2050 4520 ARNING: This PE
00002AE0 6861 7320 6265 656E 206C 6F63 6B65 6420 has been locked
00002AF0 7769 7468 2042 4554 412D 5445 5354 494E with BETA-TESTIN
00002B00 4720 7665 7273 696F 6E20 6F66 2053 5045 G version of SPE
00002B10 456E 6300 5741 524E 494E 4721 008D B548 Enc.WARNING!...H
00002B20 598B 0483 C609 803E 7274 5468 3000 0400 Y......>rtTh0...
00002B30 9090 9090 9090 9090 9090 9090 9090 8D85 ................
```

```
00004730 9090 9090 9090 9090 9090 E918 0100 0043 ...............C
00004740 7265 6174 6546 696C 6541 0057 7269 7465 reateFileA.Write
00004750 4669 6C65 0043 6C6F 7365 4861 6E64 6C65 File.CloseHandle
00004760 0047 6574 5465 6D70 5061 7468 4100 4C6F .GetTempPathA.Lo
00004770 6361 6C41 6C6C 6F63 004C 6F63 616C 4672 calAlloc.LocalFr
00004780 6565 0044 656C 6574 6546 696C 6541 0046 ee.DeleteFileA.F
00004790 7265 654C 6962 7261 7279 0047 6574 4C61 reeLibrary.GetLa
000047A0 7374 4572 726F 7200 4765 7444 6973 6B46 stError.GetDiskF
000047B0 7265 6553 7061 6365 4100 5350 4545 6E63 reeSpaceA.SPEEnc
000047C0 2E44 7570 0045 7272 6F72 2053 7461 7274 .Dup.Error Start
000047D0 696E 6720 5072 6F67 7261 6D00 5468 6572 ing Program.Ther
000047E0 6520 6973 206E 6F74 2065 6E6F 7567 6820 e is not enough
000047F0 6D65 6D6F 7279 2074 6F20 7374 6172 7420 memory to start
00004800 6170 706C 6963 6174 696F 6E0D 0A50 6C65 application..Ple
00004810 6173 6520 636C 6F73 6520 6F74 6865 7220 ase close other
00004820 6170 706C 6963 6174 696F 6E73 2C20 616E applications, an
00004830 6420 7472 7920 6167 6169 6E2E 0053 6F66 d try again..Sof
00004840 7477 6172 6520 5072 6F74 6563 7469 6F6E tware Protection
00004850 2045 7272 6F72 0090 9090 9090 9090 9090  Error..........
00004860 9090 9090 908D B530 598B 0490 9090 9090 .......0Y.......
```

```
00007420 2020 2020 2020 2020 2020 2020 2020 2020
00007430 2042 7920 4173 7465 7269 6F73 2050 6172  By Asterios Par
00007440 6C61 6D65 6E74 6173 2E20 694E 6645 6354 lamentas. iNfEcT
00007450 6544 2042 7240 694E 204C 6162 732C 2050 eD Br@iN Labs, P
00007460 726F 6A65 6374 3A20 7343 7240 6D42 6C45 roject: sCr@mBlE
00007470 6420 4272 4069 4E2E 2020 2020 2020 2020 d Br@iN.
00007480 2020 2020 2020 2020 0D0A 0D0A 5573 6572          ....User
00007490 3332 2E64 6C6C 004D 6573 7361 6765 426F 32.dll.MessageBo
000074A0 7841 0045 7869 7450 726F 6365 7373 0077 xA.ExitProcess.w
000074B0 7370 7269 6E74 6641 004C 6F63 616C 416C sprintfA.LocalAl
000074C0 6C6F 6300 4C6F 6361 6C46 7265 6500 4765 loc.LocalFree.Ge
000074D0 744D 6F64 756C 6546 696C 654E 616D 6541 tModuleFileNameA
000074E0 0043 7265 6174 6553 656D 6166 6F72 6500 .CreateSemafore.
000074F0 4572 726F 7220 5374 6172 7469 6E67 2050 Error Starting P
00007500 726F 6772 616D 0041 2072 6571 7569 7265 rogram.A require
00007510 6420 2E44 4C4C 2066 696C 6520 2573 2077 d .DLL file %s w
00007520 6173 206E 6F74 2066 6F75 6E64 2E00 4361 as not found..Ca
00007530 6E6E 6F74 2066 696E 6420 696D 706F 7274 nnot find import
00007540 2025 733A 2573 2E00 4361 6E6E 6F74 2066  %s:%s..Cannot f
00007550 696E 6420 696D 706F 7274 2025 733A 5B4F ind import %s:[O
00007560 7264 696E 616C 2049 6D70 6F72 745D 2C20 rdinal Import],
00007570 3078 2558 2E00 5468 6520 2573 2066 696C 0x%X..The %s fil
00007580 650D 0A63 616E 2774 206C 6F61 6420 6174 e..can't load at
00007590 2074 6865 2064 6573 6972 6564 2061 6464  the desired add
000075A0 7265 7373 2C20 616E 6420 6973 206E 6F74 ress, and is not
000075B0 2072 656C 6F63 6174 6162 6C65 2E0D 0A43  relocatable...C
000075C0 6F6E 7461 6374 2079 6F75 7220 7665 6E74 ontact your vent
000075D0 6F72 2074 6F20 6765 7420 6120 7665 7273 or to get a vers
000075E0 696F 6E20 7468 6174 2069 7320 636F 6D70 ion that is comp
000075F0 6174 6962 6C65 2077 6974 6820 7468 6973 atible with this
00007600 2076 6572 7369 6F6E 206F 6620 5769 6E64  version of Wind
00007610 6F77 732E 0054 6865 7265 2069 7320 6E6F ows..There is no
00007620 7420 656E 6F75 7068 206D 656D 6F72 7920 t enouph memory
00007630 746F 2073 7461 7274 2061 7070 6C69 6361 to start applica
00007640 7469 6F6E 0D0A 506C 6561 7365 2063 6C6F tion..Please clo
00007650 7365 206F 7468 6572 2061 7070 6C69 6361 se other applica
00007660 7469 6F6E 732C 2061 6E64 2074 7279 2061 tions, and try a
00007670 6761 696E 2E00 5350 4545 6E63 2042 7920 gain..SPEEnc By
00007680 4173 7465 7269 6F73 2050 6172 6C61 6D65 Asterios Parlame
00007690 6E74 6173 2069 4E66 4563 5465 4420 4272 ntas iNfEcTeD Br
000076A0 4069 4E20 4C61 6273 2E20 5072 6F6A 6563 @iN Labs. Projec
000076B0 743A 2073 4372 406D 426C 4564 2042 7240 t: sCr@mBlEd Br@
000076C0 694E 2E20 6430 322D 3037 2D30 3172 2E00 iN. d02-07-01r..
000076D0 1C41 9500 1C43 9500 D16F F7BF DB7A F7BF .A...C...o...z..
```

Indeed, having an "infected brain" does not help to make good protections ^^ (just joking :p).

In order to study this code, I use W32Dasm :p…
As I am always in a hurry, I don't have time to wait for IDA's analysis, nor to comment its listing, making it more comprehensible. Moreover, polymorphic code is removed, W32Dasm can disassemble it without any problem …
To make correspondance between addresses in W32Dasm and those in memory (those fixed like previously), you have just to add address base (here, it is 00BA0000h).

4. Locating OEP easily:

When we are on OEP of tms2003.exe, if we take a look at the stack (esp=B7FE3C), we obtain something like that :

```
01AF:00B7FE1C 30 FE B7 00 00 00 00 00-64 A1 99 81 D4 6C BA 00  0.......d....l..
01AF:00B7FE2C 24 6D BA 00 62 6D BA 00-24 A1 99 81 57 69 64 00  $m..bm..$...wid.
01AF:00B7FE3C 60 B5 F8 BF 00 00 00 00-04 A1 99 81 00 00 00 00  `...............
01AF:00B7FE4C 54 65 6E 6E 69 73 20 6D-61 73 74 65 72 73 20 73  Tennis masters s
01AF:00B7FE5C 65 72 69 65 73 20 32 30-30 33 00 45 58 45 00 00  eries 2003.EXE..
01AF:00B7FE6C 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  ................
```

Thus, not only, polymorphic code is useless (it is generic and can be easily removed), but it also constitutes one fault, owing to repeated sequences of call eip+8... (they constitute "markers" on the stack).
This enables us to go up on instructions, which "jumps" towards the OEP…

This (going up on these instructions) is interesting for two reasons:
- Now, it is much easier to come at OEP (by a simple bpm 00BA6D87 X).
- the 2 last instructions (FINIT / RET) and even better with the following instructions, it enables to have a signature of this packer. This facilitates thus our life and tracers/unpackers' task, you could code…

```
:00005FFB 3E8B856E598B04          mov eax, dword ptr ds:[ebp+048B596E] <- on affecte à eax, l'OEP "codé"
:0000601E 3E83BDC0598B0400        cmp dword ptr ds:[ebp+048B59C0], 00000000
:00006034 7471                    je 000060A7
:00006073 CC                      int 03
:00006082 3E2B8574D08B04          sub eax, dword ptr ds:[ebp+048BD074] <- on "décode" l'OEP par une soustraction
:00006097 EB52                    jmp 000060EB
:000060A7 3E2B8586598B04          sub eax, dword ptr ds:[ebp+048B5986]
:00006128 3E89856E598B04          mov dword ptr ds:[ebp+048B596E], eax <- on remet l'OEP à sa place (00BA003E)

...

:00006C9F 61                      popad
:00006CA0 3E8B856E598B04          mov eax, dword ptr ds:[ebp+048B596E] <- l'OEP (VO) est affecté à eax
:00006CB5 3E038572598B04          add eax, dword ptr ds:[ebp+048B5972] <- ajout de l'Image Base (400000h)
:00006CF9 89442408                mov dword ptr [esp+08], eax         <- je mets l'OEP sur la pile
:00006D0B 58                      pop eax
:00006D49 5D                      pop ebp
:00006D87 DBE3                    finit
:00006D89 C3                      ret                                 <- je "saute" vers l'OEP
```

The routine of the top (in 00BA5FFB) makes it possible to calculate the OEP where SPEEnc must jump…

ebp+048BD074 is equal to 00BA7744, which contains the value 9B6D10DB.
ebp+048B596E is equal to 00BA003E, which contains the "encoded" OEP: 9B917A32. (in red, here)
ebp+048B5972 is equal to 00BA0042, which contains the Image Bases (400000h). (in green, here)

The Image Base is in SPEEnc since the beginning…
Image Base, which is initialized (at the beginning of protection), by this code:

```
01A7:004013EC    3E8B853E000B10    MOV EAX, DS:[EBP+100B003E]
01A7:004014F3    89041E            MOV [EBX+ESI], EAX
```

So, OEP "is just coded" and a simple subtraction makes it possible to find easily!!!
OEP (VO) = 9B917A32h - 9B6D10DBh = 00246957h

5. Anti-debugging tricks:

At first time, we could say that there were no anti-debugging tricks.
Indeed, game launches without any problem with SoftIce, loaded. We have no warning message, no crashes, exit or anything else...
So, there is no direct debugging detection.
But, when we put bp / bpm (hardware breakpoint), we discover that anti-debugging tricks are present...

First of all, a small log of FrogsIce (Thank you +Frog's Print):

```
=> Tennis m
**  IDT DETECTION     **  code 0F, at 01A7:00BA1F7E
Interrupt:  03h

=> Tennis m
**  IDT DETECTION     **  code 0F, at 01A7:00BA1FCF
Interrupt:  03h

=> Tennis m
** SOFTICE DETECTION **  code 00, at  01A7:00BA205C
Interrupt:03h  eax=00BA07A6h  ebx=800F9018h  ecx=00BA3119h
               edx=E2B9BDC3h  esi=000000BAh  edi=00B7FD10h  ebp=FC2EA6D0h

SEH proc address at cs:????????
```

Hook of interruption 3:

```
01C7:00BA1D46  8D85D6608B04      LEA      EAX, DWORD PTR DS:[EBP+048B60D6]
01C7:00BA1D5A  50                PUSH     EAX                 SE Handler (00BA07A6)
01C7:00BA1D98  6467FF360000      PUSH     DWORD PTR FS:[0000] Pointer to next SEH record
01C7:00BA1DDB  646789260000      MOV      FS:[0000],ESP
01C7:00BA1E5B  3E0F018DBA598B04  SIDT     FWORD PTR DS:[EBP+048B59BA]
01C7:00BA1EA0  3E8B9DBC598B04    MOV      EBX,DS:[EBP+048B59BC]
01C7:00BA1EB5  83C318            ADD      EBX,18
01C7:00BA1EC6  FA                CLI
01C7:00BA1ED5  8B5304            MOV      EDX,[EBX+04]
01C7:00BA1F15  668B13            MOV      DX,[EBX]
01C7:00BA1F26  3E8995C0598B04    MOV      DS:[EBP+048B59C0],EDX    -> on sauve l'ancienne routine (en 00BA0090)
01C7:00BA1F3B  8DB5EC7A8B04      LEA      ESI,[EBP+048B7AEC]
01C7:00BA1F7E  668933            MOV      [EBX],SI                 -> l'int 03 est détournée vers une routine
01C7:00BA1F8F  C1EE10            SHR      ESI,10
01C7:00BA1FCF  66897306          MOV      [EBX+06],SI              -> de Laserlok, située en 00BA21BC...
01C7:00BA1FE1  FB                STI
01C7:00BA205C  CC                INT      3                        -> on va en 00BA21BC (on efface les Drx au passage :p)
01C7:00BA218E  64678F060000      POP      DWORD PTR FS:[0000]
01C7:000021A2  58                POP      EAX
01C7:000021A3  8D95538A8B04      LEA      EDX, DWORD PTR [ebp+048B8A53]
```

Laserlock thus hooks the int 03 to its own routine (located in 00BA21BC), which take care of erasing Debug Registers (dr0, dr1 and dr2) and by the same time clearing the Hardware Breakpoints !!!
Moreover, this routine in 00BA21BC is called (in ring 0) much more than once, by the int 03, disseminated a little everywhere in the code…
By putting a bpx adresse/API, you go directly on this routine (but not when it was planed :p) and you got a pretty crash in 00BA21FA !!! (on Ring 0)
Quite simply because the address ds:[ebp+048B5986] in 00BA21FA (1) is not "valid"…
It is the same for the address ds:[ebp+048BD074] in 00BA231E (2), where you arrive if you avoided the 1st crash by modifying the eip.

And if you manage to return to windows after these crashes, you will got a crash of the explorer, each time you want to go anywhere that the Desktop (My documents, any folder, etc…) and even when you want to restart the computer …
Indeed, a crash (or an early exit) of SPEEnc lets the int 03, hooked (it is not very clean), which is not the case for Starforce. The explorer seems to use int

03, when folders are browsed. If you minimize the game between the hook and the unhook by the SPEEnc, after a breakpoint and an infinite loop, you will obtain explorer's crashes in case of browsing folders.

Now, why these crashes in (1) and (2) ?
Because of the "ebp excentric" value. When SPEEnc is normally executed, ebp has a certain value.
When we are in (1), ebp = FC2EA6D0, so ebp+048B5986 take 00BA0056 as value and eax = 9B6D10DB.
When we are in (2), ebp = ECCD9F74, so ebp+048BD074 take 00BA7744 as value (eax has its previous value).
So, when a int 03 is called at any other moment (that the ones specified by the SPEEnc), like a bp, there will be big chance to have an invalid memory access (on read / write), because of the different value of ebp register.

The restoration of IDT, on the int 03, is done only just before the redirection towards the OEP of the unpacked executable …

Note: The int 03 is hooked by the instructions in 00BA1F7E and 00BA1FCF… (see my Tutorial about Starforce for more detail on the IDT).
If we avoid the hook of int 03 (by avoiding the instructions in 00BA1F7E and 00BA1FCF), the program will crash… (this is due to the int 03, disseminated a little everywhere in the code).

Clearing Drx:

```
:000021BC 90                    nop
:000021CA 60                    pushad
:000021FA 3E8B8586598B04        mov eax, dword ptr ds:[ebp+048B5986]   <- crash si bpx !!!
:0000223E 50                    push eax
:0000224D 33C0                  xor eax, eax
:0000228C 0F23C0                mov dr0, eax
:0000229D 0F23C8                mov dr1, eax
:000022DD 0F23D0                mov dr2, eax
:0000230F 58                    pop eax
:0000231E 3E898574D08B04        mov dword ptr ds:[ebp+048BD074], eax   <- crash si bpx (si 1er crash détourné) !!!
:00002333 61                    popad
:00002342 CF                    iret
```

This routine is not very efficient, since I can break on Laserlock code without problems, using a bpm !!!
Indeed, when we put our 1st bpm, SoftIce uses the dr3 and as this one is not erased, we can thus break.
Thus, this routine limit only the use of hardware breakpoints to one!!!
And you are by no means prevented from putting several bpm and breaking without problems in the code separating two consecutive int 03…
Nopping the mov drx, eax makes crash the program, much further, because of the integrity checks of SPEEnc !!!
It is thus necessary to proceed differently to circumvent this…

Interruptions (int 03) disseminated a little everywhere in the code:

Presence of int 03 in 00BA1BD2, 00BA1C93, 00BA205C, 00BA23CA, 00BA2656, 00BA2847, 00BA29CE, 00BA3C48, 00BA3F82, 00BA43A1, 00BA445E, 00BA4A69, 00BA4EBB, 00BA515D, 00BA55FC, 00BA6073 and 00BA66B2. They are thus rather numerous and are as many possibilities of anti-bpx and anti-hardware breakpoint (by clearing the drx).

Some examples:

```
:00001B7D 3E83BDC0598B0400      cmp dword ptr ds:[ebp+048B59C0], 00000000
:00001BC2 741D                  je 00001BE1
:00001BD2 CC                    int 03
:00001BE1 60                    pushad
:00001BE2 3EFFB588D08B04        push dword ptr ds:[ebp+048BD088]
:00001BE9 3EFF9510D08B04        call dword ptr ds:[ebp+048BD010]
:00001BF0 61                    popad
```

```
:00001C6D 3E83BDC0598B0400      cmp dword ptr ds:[ebp+048B59C0], 00000000
:00001C83 744C                  je 00001CD1
:00001C93 CC                    int 03
:00001CDF 8B93B8000000          mov edx, dword ptr [ebx+000000B8]
:00001CF3 83C20E                add edx, 0000000E
:00001D04 E998F1FFFF            jmp 00000EA1
```

```
:00002351 53                    push ebx
:00002360 3EFF9524D08B04        call dword ptr ds:[ebp+048BD024]
:00002375 3E83BDC0598B0400      cmp dword ptr ds:[ebp+048B59C0], 00000000
:000023BA 741D                  je 000023D9
:000023CA CC                    int 03
:00002408 8BD8                  mov ebx, eax
:00002447 03C5                  add eax, ebp
:00002457 0500D08B04            add eax, 048BD000
:0000246A 3E338570D08B04        xor eax, dword ptr ds:[ebp+048BD070]
:000024AE 60                    pushad
:000024AF 6A00                  push 00000000
:000024BF 6A00                  push 00000000
:000024C1 6A02                  push 00000002
:000024D1 6A00                  push 00000000
:000024D3 6A00                  push 00000000
:000024E3 6800000040            push 40000000
:000024E8 3EFFB58CD08B04        push dword ptr ds:[ebp+048BD08C]
:0000252C 3EFF95B6598B04        call dword ptr ds:[ebp+048B59B6]
:00002533 3E898588D08B04        mov dword ptr ds:[ebp+048BD088], eax
:0000253A 61                    popad
```

```
:00002630 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:00002646 740F                      je 00002657
:00002656 CC                        int 03
:00002694 53                        push ebx
:000026A3 3EFF9524D08B04            call dword ptr ds:[ebp+048BD024]
:000026E7 3EC78584D08B0401000000    mov dword ptr ds:[ebp+048BD084], 00000001
:000026F2 FFD0                      call eax
:000026F4 3EC78584D08B0400000000    mov dword ptr ds:[ebp+048BD084], 00000000
:0000270D B303                      mov bl, 03
:0000274C F6F3                      div bl
:0000275C 69C0901E0000              imul eax, 00001E90
:0000279F 3E29858A598B04            sub dword ptr ds:[ebp+048B598A], eax
:000027E3 5B                        pop ebx
:000027F2 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:00002808 743E                      je 00002848
:00002847 CC                        int 03
:00002856 8B93B8000000              mov edx, dword ptr [ebx+000000B8]
:00002899 83C214                    add edx, 00000014
:000028AA E9F2E5FFFF                jmp 00000EA1
```

```
:00002979 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:000029BE 740F                      je 000029CF
:000029CE CC                        int 03
:00002A0C E8F7470000                call 00007208
```

```
:00003BF3 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:00003C09 743E                      je 00003C49
:00003C48 CC                        int 03
:00003C57 3E83BD78D08B0400          cmp dword ptr ds:[ebp+048BD078], 00000000
:00003C9C 0F8424040000              je 000040C6
```

```
:00003F5C 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:00003F72 740F                      je 00003F83
:00003F82 CC                        int 03
:00003F91 6A00                      push 00000000
```

```
:0000437B 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:00004391 740F                      je 000043A2
:000043A1 CC                        int 03
:000043DF 3E83BD78D08B0400          cmp dword ptr ds:[ebp+048BD078], 00000000
:00004424 0F8596020000              jne 000046C0
:00004438 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:0000444E 740F                      je 0000445F
:0000445E CC                        int 03
:0000446D 3E80BDF8598B0400          cmp byte ptr ds:[ebp+048B59F8], 00
:00004483 0F8480000000              je 00004509
```

etc…

You have certainly seen this recurring cmp dword ptr ds:[ebp+048B59C0], 00000000 and preceding all the int 03 (except the one in 00BA205C)…
And what does correspond ds:[ebp+048B59C0] to ?
It is there, that is stored the old routine ("the address") of the int 03 previously hooked …
However, as we saw, if we only try to avoid the hook of int 03, the program crashes … (because of all these int 03).
Then let's test this :
In 00BA1F26 (storage of the int 03 old routine), we put edx at 0 (r edx 0), so that it saves this value.
In 00BA1F7E, we avoid the instruction while making point eip on the following instruction (r eip eip+3).
In 00BA1FCF, we avoid also the instruction by a "r eip eip+4".
Lastly, we avoid the int 03 in 00BA205C (r eip eip+1).
And now you will be able to put all the bpx/bpx API you want :)
We solve in the same time the anti-bpx and the drx clearing…

But why this presence of cmp / je?
Because contrary to win 98, XP does not make possible any read/write attempt in IDT in ring 3 (normal execution mode of programs)…
This code ensures thus compatibility with this last system.

You will have noticed that this int 03 hook routine is included in a "SEH structure". Any attempt to read / write IDT values on XP send us directly to (39)07A6 (SE Handler), 390000 being Image base of the SPEEnc...

Restoration of IDT (on the int 03 vector):

Just before the "jump" towards the OEP of the unpacked executable, the IDT is restored:

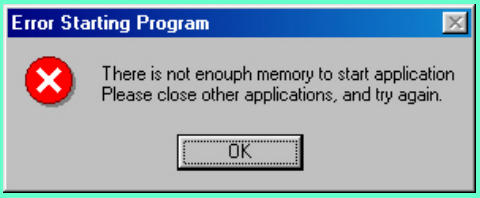
```
:00006B2C 3E83BDC0598B0400          cmp dword ptr ds:[ebp+048B59C0], 00000000
:00006B42 0F8457010000              je 00006C9F
:00006B85 3E8B9DBC598B04            mov ebx, dword ptr ds:[ebp+048B59BC] -> adresse de l'IDT
:00006B9A 83C318                    add ebx, 00000018
:00006BAB 3E8BB5C0598B04            mov esi, dword ptr ds:[ebp+048B59C0] -> on restaure l'ancienne routine (int 03)
:00006BEF FA                        cli
:00006BFE 668933                    mov word ptr [ebx], si
:00006C0F C1EE10                    shr esi, 10
:00006C4F 66897306                  mov word ptr [ebx+06], si
:00006C61 FB                        sti
:00006C9E 90                        nop
:00006C9F 61                        popad
```

Many exceptions (SEH):

SPEEnc contains many exceptions throughout its code:

```
:00002FCE 6800000040              push 40000000
:00002FD3 3EFFB58CD08B04          push dword ptr ds:[ebp+048BD08C]   -> pointe vers C:\WINDOWS\TEMP\SPEEnc.Dup
:00003017 3EFF95B6598B04          call dword ptr ds:[ebp+048B59B6]   -> CreateFileA
:0000301E 83F8FF                  cmp eax, FFFFFFFF
:00003021 7507                    jne 0000302A
:00003023 3EFF95B2598B04          call dword ptr ds:[ebp+048B59B2]
:0000302A 3E898588D08B04          mov dword ptr ds:[ebp+048BD088], eax
:0000306E 8D85D6608B04            lea eax, dword ptr [ebp+048B60D6]
:00003082 50                      push eax                           SE Handler (00BA07A6)
:000030C0 6467FF360000            push word ptr fs:[0000]            Pointer to next SEH record
:00003103 646789260000            mov fs:[0000], esp
:00003109 33C0                    xor eax, eax
:00003119 FF00                    inc dword ptr [eax]                -> une exception
:0000311B AD                      lodsd
:0000311C 5A                      pop edx
:0000311D A9C6BA3A6C              test eax, 6C3ABAC6
:00003122 E990909090              jmp 9090C1B7
:00003160 B801000000              mov eax, 00000001
:000031A2 FF00                    inc dword ptr [eax]
:000031A4 C700000000000           mov dword ptr [eax], 00000000
:000031E7 33DB                    xor ebx, ebx
:00003226 F6F3                    div bl                             -> une autre exception
:00003265 6800040000              push 00000400
:00003278 6A00                    push 00000000
:000032B7 3EFF9548D08B04          call dword ptr ds:[ebp+048BD048]   -> GlobalAlloc
:000032CC 83F800                  cmp eax, 00000000
:000032DD 7574                    jne 00003353                       -> saute si assez de mémoire...
```

Here it is the routine, which determines whether there is enough available memory for SPEEnc.
If it is not the case, the jump in 00BA32DD is not carried out and this following error message appears :



The use of all these exceptions, present a little everywhere in the code, is probably used as anti-tracing technique, but it's useless …

6. Integrity checks :

Here's the integrity checks routine of SPEEnc code, which makes you avoiding code modifications (if not -> risk of a crash) :

```
:00007248 53                      push ebx
:00007249 52                      push edx
:0000724A 51                      push ecx
:0000724B 83EC04                  sub esp, 00000004
:0000724E F7D0                    not eax
:0000725E 890424                  mov dword ptr [esp], eax
:0000726F 83F900                  cmp ecx, 00000000
:00007280 7502                    jne 00007284
:00007282 EB77                    jmp 000072FB
:00007284 33DB                    xor ebx, ebx
:00007286 8A1F                    mov bl, byte ptr [edi]
:00007288 668B442401              mov ax, word ptr [esp+01]
:0000729B 6633D2                  xor dx, dx
:0000729E 8A542403                mov dl, byte ptr [esp+03]
:000072B0 321C24                  xor bl, byte ptr [esp]
:000072B3 32FF                    xor bh, bh
:000072B5 66D1E3                  shl bx, 1
:000072B8 66D1E3                  shl bx, 1
:000072C9 663E33841DCDD08B04      xor ax, word ptr ds:[ebp+ebx+048BD0CD]
:000072D2 663E33941DCFD08B04      xor dx, word ptr ds:[ebp+ebx+048BD0CF]
:000072E9 66890424                mov word ptr [esp], ax
:000072ED 6689542402              mov word ptr [esp+02], dx
:000072F2 47                      inc edi
:000072F3 83F901                  cmp ecx, 00000001
:000072F6 7403                    je 000072FB
:000072F8 49                      dec ecx
:000072F9 EB89                    jmp 00007284
:000072FB 90                      nop
:00007309 8B0424                  mov eax, dword ptr [esp]
:0000731A F7D0                    not eax
:0000732A 83C404                  add esp, 00000004
:0000733B 59                      pop ecx
:0000733C 5A                      pop edx
:0000733D 5B                      pop ebx
:0000733E C3                      ret
```

This routine is called several times by the call 00BA7248, located in 00BA722C.
On the 3rd call, the check is done on the SPEEnc, about the block starting in 00BA0190 (esi) and with a size of 7540 (ecx).
The result of the operation is finally stored in eax.

```
:00007208 90                      nop
:00007216 50                      push eax
:00007217 3E8B85C9D08B04          mov eax, dword ptr ds:[ebp+048BD0C9]
:0000722C E817000000              call 00007248
:0000723F 3E8985C9D08B04          mov dword ptr ds:[ebp+048BD0C9], eax
:00007246 58                      pop eax
:00007247 C3                      ret
```

7. Decoding layers :

1st decoding layer:
We saw that it was carried out at 004014D5, in order to decipher the block starting at 00BA0000 (esi), with a size of 7B9D (ecx) (see the chapter fixing

the variable addresses). But in fact, instructions are valid, only for the block starting in 00BA0190 and finishing in 00BA04A5, which can be explained by the application of a second decoding layer…

2nd decoding layer:
Then, once our 1st block is deciphered, the following routine, at 00BA733F (identical to the previous one, but with polymorphic code, this time), called at 00BA0455, is charged to decipher the block starting at 00BA04A5, with a size of 44F2.

```
:0000733F 90                    nop
:0000734D C8000000              enter 0000, 00
:00007351 60                    pushad
:00007360 8B7514                mov esi, dword ptr [ebp+14]
:00007371 8BFE                  mov edi, esi
:00007381 8B4D10                mov ecx, dword ptr [ebp+10]
:00007392 83F900                cmp ecx, 00000000
:000073A3 7472                  je 00007417
:000073B3 8B5D08                mov ebx, dword ptr [ebp+08]
:000073F3 33D2                  xor edx, edx
:000073F5 8B450C                mov eax, dword ptr [ebp+0C]
:000073F8 F7E3                  mul ebx
:000073FA 33D0                  xor edx, eax
:000073FC 33D3                  xor edx, ebx
:000073FE AC                    lodsb
:000073FF C1C210                rol edx, 10
:00007402 02C6                  add al, dh
:00007404 32C2                  xor al, dl
:00007406 C1EA10                shr edx, 10
:00007409 2AC6                  sub al, dh
:0000740B 32C2                  xor al, dl
:0000740D AA                    stosb
:0000740E 43                    inc ebx
:0000740F 83F901                cmp ecx, 00000001
:00007412 7403                  je 00007417
:00007414 49                    dec ecx
:00007415 EBDC                  jmp 000073F3
:00007417 61                    popad
:00007418 C9                    leave
:00007419 C21000                ret 0010
```

This routine is then called at 00BA1B32, to decipher a block starting at 00BA9A46 for a size of 3421B (from datas…).
The checking of the disc is then carried out…
The call at 00BA1B32, calls this routine again to decipher the code at 00BA4997, with a size of 2617 (all the executable code of SPEEnc is thus deciphered…), then to decipher a block of datas on SPEEnc, starting at 00BA748C and with a size of 244.

Unpacking of the tms2003.exe is finally executed, using this routine also (00BA733F). It is called several times by the call at 00BA5ABC, to proceed by blocks, starting from the end of executable (relocations) and going up gradually…
It remains a last important question : is it possible to crack this version of Laserlock without having an original CD ?
The answer is yes :).

8. Faultsin original disc authentification :

To determine where disc checking is done (authentification), we have to put a breakpoint on GetDriveTypeA API (don't forget to avoid anti-debugging tricks). Then, we land in NIL32.dll module.
When re-running the executable file and putting a breakpoint on CreateFileA, we see that Laserlock creates some temporary files in C:\Windows\TEMP :
- nomouse
- nomouse.com
- nomouse.sp
- NIL32.dll
- SPEEnc.Dup

These files are erased, when the SPEEnc has finished its task (when the game starts).
So, it is possible to retrieve NIL32.dll, by stopping in the code before the time, it is created and after the time, it is deleted (for example, by breaking on GetDriveTypeA). We have just to go in the temporay folder to copy it. The hook (interception) of the int 03 by the SPEEnc must be avoided, because the explorer interferes with this hook (use of int 03 by the explorer in folders navigation) and leads to a crash...
We edit the dll PE with LordPE.
OEP is 0A150 (RVA), Image Base is 0x10000000.
We have just to put a bpm 1000A150 X to break at the OEP.

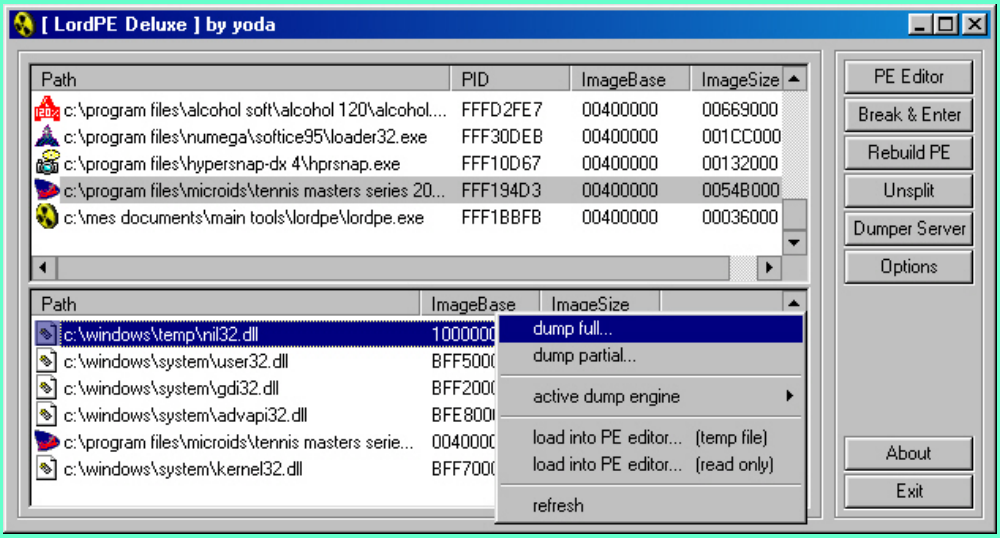
```
01A7:1000A150  E97C6FFFFF          JMP       100010D1
01A7:1000A155  B3C4                MOV       BL,C4
01A7:1000A157  323DA18B1BBE        XOR       BH,[BE1B8BA1]
01A7:1000A15D  E287                LOOP      1000A0E6
01A7:1000A15F  C5F8                LDS       EDI,EAX
01A7:1000A161  AA                  STOSB
01A7:1000A162  EBF9                JMP       1000A15D
01A7:1000A164  64F8                CLC
01A7:1000A166  4E                  DEC       ESI
01A7:1000A167  51                  PUSH      ECX
01A7:1000A168  E2D1                LOOP      1000A13B
01A7:1000A16A  7674                JBE       1000A1E0
01A7:1000A16C  B350                MOV       BL,50
01A7:1000A16E  40                  INC       EAX
01A7:1000A16F  A0ACBF5BB2          MOV       AL,[B25BBFAC]
01A7:1000A174  DDF1                ESC
01A7:1000A176  28C1                SUB       CL,AL
01A7:1000A178  D133                INVALID
01A7:1000A17A  714B                JNO       1000A1C7
01A7:1000A17C  0395AC5D00DD        ADD       EDX,[EBP+DD005DAC]
01A7:1000A182  94                  XCHG      EAX,ESP
```

We break on it. We run again (F5) to break a second time.
The NIL32.dll module is then unpacked (deciphered) and 1000A150 seems to be the OEP of the unpacked dll.

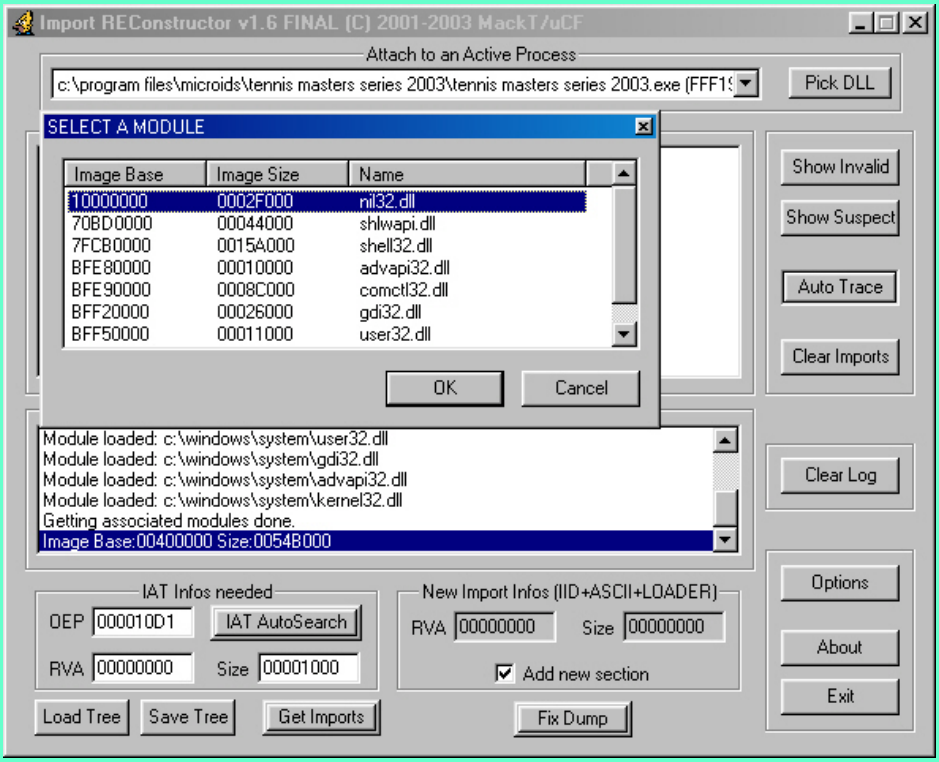
```
01A7:1000A150  55              PUSH    EBP
01A7:1000A151  8BEC            MOV     EBP,ESP
01A7:1000A153  53              PUSH    EBX
01A7:1000A154  8B5D08          MOV     EBX,[EBP+08]
01A7:1000A157  56              PUSH    ESI
01A7:1000A158  8B750C          MOV     ESI,[EBP+0C]
01A7:1000A15B  57              PUSH    EDI
01A7:1000A15C  8B7D10          MOV     EDI,[EBP+10]
01A7:1000A15F  85F6            TEST    ESI,ESI
01A7:1000A161  7509            JNZ     1000A16C
01A7:1000A163  833D54A0021000  CMP     DWORD PTR [1002A054],00
01A7:1000A16A  EB26            JMP     1000A192
01A7:1000A16C  83FE01          CMP     ESI,01
01A7:1000A16F  7405            JZ      1000A176
01A7:1000A171  83FE02          CMP     ESI,02
01A7:1000A174  7522            JNZ     1000A198
01A7:1000A176  A1B8BB0210      MOV     EAX,[1002BBB8]
01A7:1000A17B  85C0            TEST    EAX,EAX
01A7:1000A17D  7409            JZ      1000A188
01A7:1000A17F  57              PUSH    EDI
01A7:1000A180  56              PUSH    ESI
```

And if it is not the case, it does not matter, because we want to dump this dll, only to study it in dead listing.
In the code, we can see some call [00C0xxxx], corresponding to the Laserlock redirected call [API].
We have again variable addresses at this stage.

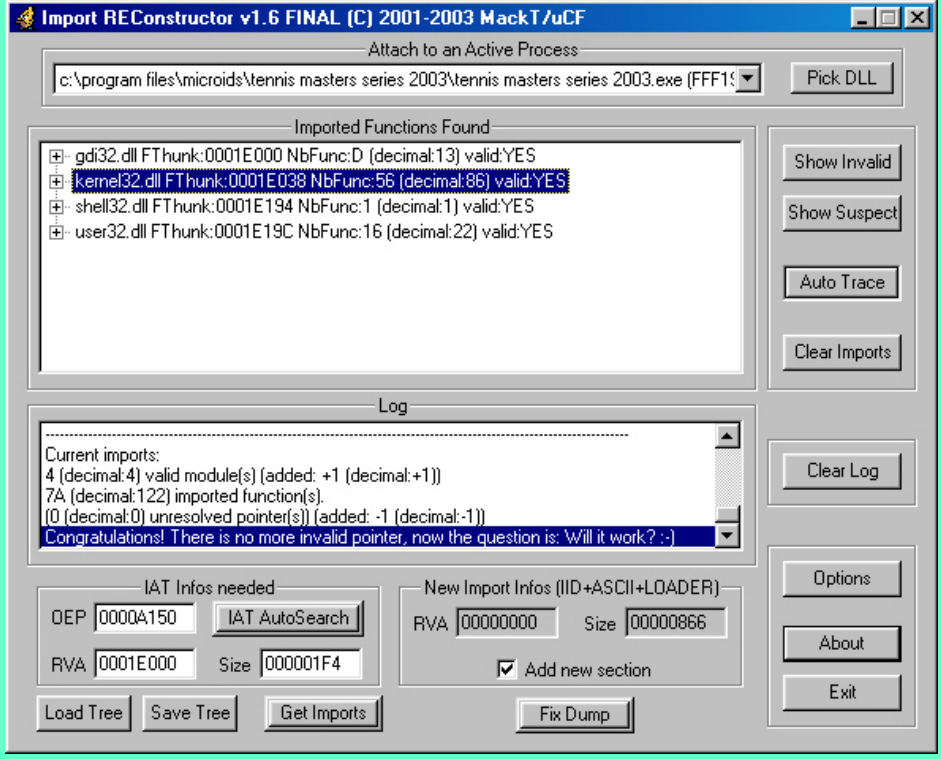
We launch a call-fixer to solve this problem.
Then, we dump at the OEP, with LordPE :



We create a new Import Table with ImpRec :



To do it, we choose the tms2003.exe process. We click on Pick DLL and we select the nil32.dll module.
Then, we have just to enter the beginning of the IAT : 1E000 (RVA), its size : 1F4 and to click on Get Imports.

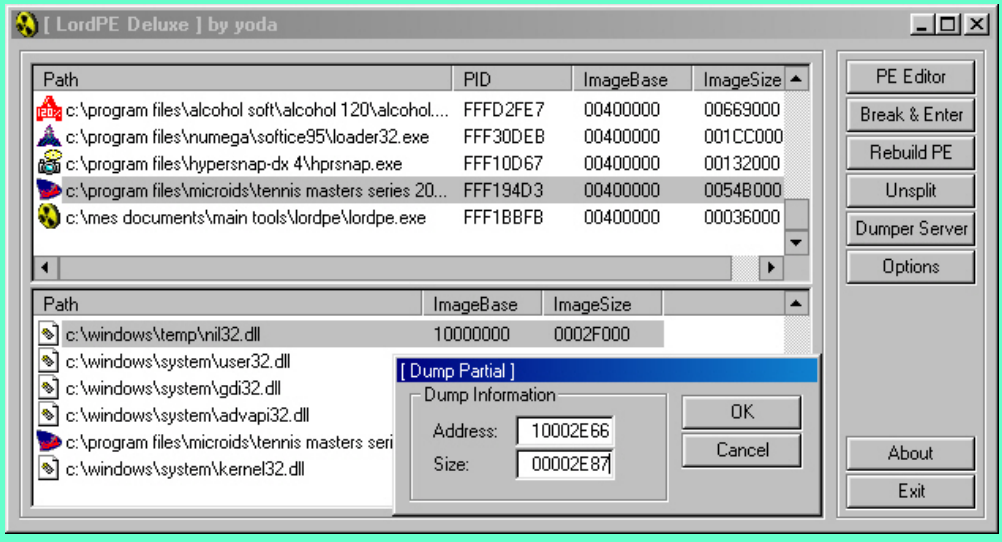


Don't forget to solve the GetProcAddress import to obtain the new Import Table (see previously).

By disassembling the dll, we can see a part of code, which is still ciphered (offsets from 0x2E66 to 0x5CED, that is to say, a size of 0x2E87) :

```
:10002E30 55                    push ebp
:10002E31 8BEC                  mov ebp, esp
:10002E33 83EC34                sub esp, 00000034
:10002E36 53                    push ebx
:10002E37 56                    push esi
:10002E38 57                    push edi
:10002E39 E892480000            call 100076D0
:10002E3E 6A00                  push 00000000
:10002E40 6A05                  push 00000005
:10002E42 E813490000            call 1000775A
:10002E47 83C408                add esp, 00000008
:10002E4A 90                    nop
...
:10002E65 90                    nop
:10002E66 9B                    wait
:10002E67 6F                    outsd
:10002E68 343A                  xor al, 3A
:10002E6A EB13                  jmp 10002E7F
:10002E6C AD                    lodsd
:10002E6D A0070210CD            mov al, byte ptr [CD100207]
:10002E72 3147B6                xor dword ptr [edi-4A], eax
:10002E75 9F                    lahf
:10002E76 D6                    BYTE 0d6h
:10002E77 58                    pop eax
:10002E78 5E                    pop esi
:10002E79 18A40702106C8F        sbb byte ptr [edi+eax-7093EFFE], ah
:10002E80 44                    inc esp
:10002E81 91                    xchg eax,ecx
:10002E82 C3                    ret
```

We can put a bpm 10002E66 X to break and dump the deciphered code with LordPE :



Then, we replace the deciphered code in the previous dumped .dll .

We obtain a fully rebuilt dll. By examining the disassembled code, what can we see ?

The same String Data References as the Laserlock v5 dll : "*LASERLOK* Copyright (c) 1992-1996", "Petros Skalkos **", "v5.00.607 Compile:21/3/2002".

Finally, protectionnists re-release always the same protection but in a different way.
They take their old protection dll (previous version) and have just implemented the SPEEnc, as a layer, enabling to hide the dll protection...

How can the CD authentification be bypassed ?
By putting a bp on the GetDriveTypeA API, we land at this code :

```
* Referenced by a CALL at Address:
|:10005619
|
:10005F99 55                      push ebp
:10005F9A 8BEC                    mov ebp, esp
:10005F9C 81ECD0000000            sub esp, 000000D0
:10005FA2 53                      push ebx
:10005FA3 56                      push esi
:10005FA4 57                      push edi
:10005FA5 E837BFFFFF              call 10001EE1
:10005FAA A334010210              mov dword ptr [10020134], eax
:10005FAF 833D3401021000          cmp dword ptr [10020134], 00000000
:10005FB6 0F8413000000            je 10005FCF
:10005FBC 6870100210              push 10021070
:10005FC1 E851B6FFFF              call 10001617
:10005FC6 83C404                  add esp, 00000004
:10005FC9 50                      push eax
:10005FCA E87BB8FFFF              call 1000184A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10005FB6(C)
|
:10005FCF E825BCFFFF              call 10001BF9
:10005FD4 85C0                    test eax, eax
:10005FD6 0F840A000000            je 10005FE6
:10005FDC B801000000              mov eax, 00000001
:10005FE1 E975010000              jmp 1000615B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10005FD6(C)
|
:10005FE6 833D3001021000          cmp dword ptr [10020130], 00000000
:10005FED 0F841A000000            je 1000600D
:10005FF3 6880100210              push 10021080
:10005FF8 E81AB6FFFF              call 10001617
:10005FFD 83C404                  add esp, 00000004
:10006000 50                      push eax
:10006001 E844B8FFFF              call 1000184A
:10006006 33C0                    xor eax, eax
:10006008 E94E010000              jmp 1000615B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10005FED(C)
|
:1000600D 8D8534FFFFFF            lea eax, dword ptr [ebp+FFFFFF34]
:10006013 50                      push eax
:10006014 6A64                    push 00000064

* Reference To: kernel32.GetLogicalDriveStringsA, Ord:0185h
                                  |
:10006016 FF15C8E00110            Call dword ptr [1001E0C8]
```

```
:1000601C 85C0                  test eax, eax
:1000601E 0F850A000000          jne 1000602E
:10006024 B801000000            mov eax, 00000001
:10006029 E92D010000            jmp 1000615B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000601E(C)
|
:1000602E C7459800000000        mov [ebp-68], 00000000

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10006139(U)
|
:10006035 8B4598                mov eax, dword ptr [ebp-68]
:10006038 0FBE840534FFFFFF      movsx eax, byte ptr [ebp+eax-000000CC]
:10006040 85C0                  test eax, eax
:10006042 0F84F6000000          je 1000613E
:10006048 8B4598                mov eax, dword ptr [ebp-68]
:1000604B 8D840534FFFFFF        lea eax, dword ptr [ebp+eax-000000CC]
:10006052 50                    push eax

* Reference To: kernel32.GetDriveTypeA, Ord:0168h
                                 |
:10006053 FF15C4E00110          Call dword ptr [1001E0C4]
:10006059 83F805                cmp eax, 00000005
:1000605C 0F85B0000000          jne 10006112
:10006062 8B4598                mov eax, dword ptr [ebp-68]
:10006065 8D840534FFFFFF        lea eax, dword ptr [ebp+eax-000000CC]
:1000606C 50                    push eax

* Possible StringData Ref from Data Obj ->"_Chk:%s"
                                 |
:1000606D 6890100210            push 10021090
:10006072 8D459C                lea eax, dword ptr [ebp-64]
:10006075 50                    push eax
:10006076 E8D3480000            call 1000A94E
:1000607B 83C40C                add esp, 0000000C
:1000607E 8D459C                lea eax, dword ptr [ebp-64]
:10006081 50                    push eax
:10006082 E8C3B7FFFF            call 1000184A
:10006087 8B4598                mov eax, dword ptr [ebp-68]
:1000608A 8A840534FFFFFF        mov al, byte ptr [ebp+eax-000000CC]

* Possible StringData Ref from Data Obj ->"x:\LASERLOK\LASERLOK.IN"
                                 |
:10006091 8B0D88000210          mov ecx, dword ptr [10020088]
:10006097 8801                  mov byte ptr [ecx], al

* Possible StringData Ref from Data Obj ->"x:\LASERLOK\LASERLOK.IN"
                                 |
:10006099 A188000210            mov eax, dword ptr [10020088]
:1000609E 50                    push eax

* Reference To: kernel32.GetFileAttributesA, Ord:0172h
                                 |
:1000609F FF15C0E00110          Call dword ptr [1001E0C0]
:100060A5 50                    push eax

* Possible StringData Ref from Data Obj ->"_Attr(%lx)"
                                 |
:100060A6 6898100210            push 10021098
:100060AB 8D459C                lea eax, dword ptr [ebp-64]
:100060AE 50                    push eax
:100060AF E89A480000            call 1000A94E
:100060B4 83C40C                add esp, 0000000C
:100060B7 8D459C                lea eax, dword ptr [ebp-64]
:100060BA 50                    push eax
:100060BB E88AB7FFFF            call 1000184A

* Possible StringData Ref from Data Obj ->"x:\LASERLOK\LASERLOK.IN"
                                 |
:100060C0 A188000210            mov eax, dword ptr [10020088]
:100060C5 50                    push eax
:100060C6 E8B4FDFFFF            call 10005E7F
:100060CB 83C404                add esp, 00000004
:100060CE 85C0                  test eax, eax
:100060D0 0F8529000000          jne 100060FF

* Possible StringData Ref from Data Obj ->"x:\LASERLOK\LASERLOK.IN"
                                 |
:100060D6 A188000210            mov eax, dword ptr [10020088]
:100060DB 8A00                  mov al, byte ptr [eax]

* Possible StringData Ref from Data Obj ->"x:\"
                                 |
:100060DD 8B0D8C000210          mov ecx, dword ptr [1002008C]
:100060E3 8801                  mov byte ptr [ecx], al
:100060E5 68A4100210            push 100210A4
:100060EA E828B5FFFF            call 10001617
:100060EF 83C404                add esp, 00000004
:100060F2 50                    push eax
:100060F3 E852B7FFFF            call 1000184A
:100060F8 33C0                  xor eax, eax
:100060FA E95C000000            jmp 1000615B

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:100060D0(C)
|
:100060FF 68B0100210            push 100210B0
:10006104 E80EB5FFFF            call 10001617
:10006109 83C404                add esp, 00000004
:1000610C 50                    push eax
:1000610D E838B7FFFF            call 1000184A
```

```
* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:1000605C(C), :10006134(U)
|
:10006112 8B4598                mov eax, dword ptr [ebp-68]
:10006115 898530FFFFFF          mov dword ptr [ebp+FFFFFF30], eax
:1000611B FF4598                inc [ebp-68]
:1000611E 8B8530FFFFFF          mov eax, dword ptr [ebp+FFFFFF30]
:10006124 0FBE840534FFFFFF      movsx eax, byte ptr [ebp+eax-000000CC]
:1000612C 85C0                  test eax, eax
:1000612E 0F8405000000          je 10006139
:10006134 E9D9FFFFFF            jmp 10006112

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:1000612E(C)
|
:10006139 E9F7FEFFFF            jmp 10006035

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10006042(C)
|
:1000613E 68BC100210            push 100210BC
:10006143 E8CFB4FFFF            call 10001617
:10006148 83C404                add esp, 00000004
:1000614B 50                    push eax
:1000614C E8F9B6FFFF            call 1000184A
:10006151 B801000000            mov eax, 00000001
:10006156 E900000000            jmp 1000615B

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:10005FE1(U), :10006008(U), :10006029(U), :100060FA(U), :10006156(U)
|
:1000615B 5F                    pop edi
:1000615C 5E                    pop esi
:1000615D 5B                    pop ebx
:1000615E C9                    leave
:1000615F C3                    ret
```

We have the classical APIs couple : GetLogicalDriveStingsA (returns the existing drives on the system) / GetDriveTypeA (returns the type of a specified drive).
The GetFileAttributesA API returns the LASERLOK.IN file attribute (hidden attribute). So, the flag can be reversed (or some nops can be placed).

We can also make inconditional, the jump on 10005620 (it has the same effect).

```
* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:100055A2(C)
|
:1000560F C705180102100000000    mov dword ptr [10020118], 00000000
:10005619 E87B090000             call 10005F99
:1000561E 85C0                   test eax, eax
:10005620 0F840F000000           je 10005635
:10005626 C705180102100A000000   mov dword ptr [10020118], 0000000A
:10005630 E905000000             jmp 1000563A

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:10005620(C)
|
:10005635 E80F0E0000             call 10006449

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:1000554A(U), :100055F6(U), :10005605(U), :1000560A(U), :10005630(U)
|
:1000563A A118010210             mov eax, dword ptr [10020118]
```

So, it is possible to crack Laserlock SPEEnc without the original disc.
Bypassing the check on LASERLOK.IN file attribute, seems to be enough to avoid the other physical structure checks,
as some SDRs let it assumed : "_*NTCD_Last5 ChkSum error*", "_*NTCD_Prev ChkSum error*", "_*NTCD_SIGN not found*", "_*NTCDFound*", etc...
However, Laserlock does not reach the security level of some other protections, at this stage .
Indeed, the CD physical structure (defective sectors or something else) is usually used to prevent copy, but is also used by some protections to extract a key, enabling the game executable to be deciphered...
Then, an original CD is needed to crack it, unless the deciphering algorithm implementation is bad (faults, bruteforce, etc...).

D) Conclusions :

Generalization:

What changes from one game to another?
I looked at another game, protected by the same version of SPEEnc. It is Codename : Outbreak…
When they say that Laserlock is not generic, it makes me laugh.

It is exactly the same principle…
The "differences" are :
- the "Image Base" of SPEEnc changes… It is 007A0000 instead of 00BA0000.
- the OEP is always "decoded" by subtraction. This time, 9F9B992D is subtracted instead of 9B6D10DB
- the instructions of "fixed" SPEEnc have slightly different addresses: the RET, which makes jump towards OEP is in 6C06 instead of 6D89 from tms2003…
This must be explained by a "pseudo-random" insertion of polymorphic code blocks into SPEEnc…
- API GetProcAddress is redirected twice in IAT (in 50C244 and 50C3A4) instead of one…
- the IT is more destroyed than in tms2003, since the pointers towards the name of dlls are also cleared (Dword 1).
In other words, nothing fabulous to deserve the name of "non generic" protection !!!

Infos (for this game) :
OEP = 8E3AC (VO)

IT begins at 50C000 (size: 1D0)
IAT begins at 50C1F4 (size: 9F0)

Now, you have enough information to make your own unpacker ;-)
Creating a unpacker in this case is more educational than another thing ; very few games have been protected with SPEEnc version of Laserlock…
(Codename : Outbreak, Tennis Masters Series 2003, Post Mortem and Warrior Kings, in France…).

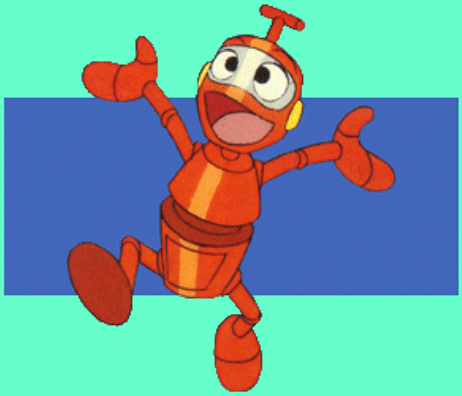
This protection is finally easier than it appeared at first ;) …
It still contains many errors of conception and is very far from being incrackable, as its authors ensure it…

You can always thank the author for this amusing and interesting protection ; he kindly left us his email…

```
00000400 5C3D 2D53 5065 456E 6320 6C6F 6164 6572 \=-SPeEnc loader
00000410 2072 7574 696E 6520 6430 322D 3037 2D30  rutine d02-07-0
00000420 3172 2062 7920 4173 7465 7269 6F73 2050 1r by Asterios P
00000430 6172 6C61 6D65 6E74 6173 5C3D 2D00 0010 arlamentas\=-...
00000440 0000 1AFA 1719 8E10 0000 A210 0000 B410 ................
00000450 0000 0000 0000 7E10 0000 0000 0000 0000 ......~.........
00000460 0000 C410 0000 4610 0000 0000 0000 0000 ......F.........
00000470 0000 0000 0000 0000 0000 0000 0000 8E10 ................
00000480 0000 A210 0000 B410 0000 0000 0000 0000 ................
00000490 4765 744D 6F64 756C 6548 616E 646C 6541 GetModuleHandleA
000004A0 0000 0000 4765 7450 726F 6341 6464 7265 ....GetProcAddre
000004B0 7373 0000 0000 4C6F 6164 4C69 6272 6172 ss....LoadLibrar
000004C0 7941 0000 4B45 524E 454C 3332 2E64 6C6C yA..KERNEL32.dll
000004D0 00EB 7901 4D53 4D49 3232 3633 352E 3030 ..y.MSMI22635.00
000004E0 4829 4B03 0300 0000 0000 0000 0000 0000 H)K.............
000004F0 0000 0055 6E6B 6F77 6E00 556E 6B6F 776E ...Unkown.Unkown
00000500 0031 302D 3130 2D30 3200 0000 0000 0000 .10-10-02.......
00000510 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000520 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000530 0000 0000 0000 0000 0000 0000 0000 0000 ................
00000540 0000 0000 0000 0000 0000 0000 EB2D 2D3D .............--=
00000550 5C65 2D6D 6169 6C3A 7374 6576 6540 6C61 \e-mail:steve@la
00000560 7365 726C 6F63 6B2E 636F 6D7C 4943 513A serlock.com|ICQ:
00000570 3231 3933 3936 3038 2F3D 2D50 55E8 0000 21939608/=-PU...
```

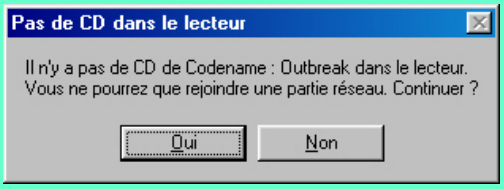
As for Microïds, I am disappointed, that they closed shop and I do not understand why…
This french editor:studio made good games like Syberia 1 & 2, Tennis Masters Series, War And Peace: 1796-1815, compared to the garbage that french studios like Davilex or Cyanide make, studios which are still alive!!!
They have already tested all the protections available on the market: **Laserlock** with Tennis Masters Series 2003, **Safedisc** with Syberia 2, **Starforce** with War And Peace: 1796-1815, **VOB ProtectCD** with Tennis Masters Series (the first)… In vain…

Here it is finished with Laserlock tutorials :p



Removing the CD-check of Codename : Outbreak :

Once Laserlock removed from Outbreak.exe, it remains a CD-check to remove :



The game launches (without movies) and then we reach the main menu…
But the two options "Play solo" and "Intro" are grayed and nonaccessible :

Moreover, a click on "Play solo" displays the following error message (it is not the case for "Intro"):

Lol… Even if you have the original, you got a limited version if you do not insert the CD!!!
We will destroy this. Disassemble Outbreak.exe, for example with W32Dasm. We look for GetDriveTypeA/GetVolumeInformationA APIs and we fall on this code :

```
* Referenced by a CALL at Addresses:
|:0046369B   , :0048E0D8
|
:00463334 55                      push ebp
:00463335 8BEC                    mov ebp, esp
:00463337 81C4A4FBFFFF            add esp, FFFFFBA4
:0046333D 53                      push ebx
:0046333E 56                      push esi
:0046333F 57                      push edi
:00463340 33D2                    xor edx, edx
:00463342 8995A4FBFFFF            mov dword ptr [ebp+FFFFFBA4], edx
:00463348 8995A8FBFFFF            mov dword ptr [ebp+FFFFFBA8], edx
:0046334E 8BF8                    mov edi, eax
:00463350 33C0                    xor eax, eax
:00463352 55                      push ebp
:00463353 68DD344600              push 004634DD
:00463358 64FF30                  push dword ptr fs:[eax]
:0046335B 648920                  mov dword ptr fs:[eax], esp
:0046335E C645FF00                mov [ebp-01], 00
:00463362 8D85F8FEFFFF            lea eax, dword ptr [ebp+FFFFFEF8]
:00463368 50                      push eax
:00463369 68FF000000              push 000000FF

* Reference To: kernel32.GetLogicalDriveStringsA, Ord:0185h
                                  |
:0046336E E8313EFAFF              Call 004071A4
:00463373 8BF0                    mov esi, eax
:00463375 33DB                    xor ebx, ebx
:00463377 E91E010000              jmp 0046349A

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:004634AF(C), :004634B9(C)
|
:0046337C 8D841DF8FEFFFF          lea eax, dword ptr [ebp+ebx-00000108]
:00463383 50                      push eax

* Reference To: kernel32.GetDriveTypeA, Ord:0168h
                                  |
:00463384 E8F33DFAFF              Call 0040717C
:00463389 83F805                  cmp eax, 00000005
:0046338C 0F8505010000            jne 00463497
:00463392 8A841DF8FEFFFF          mov al, byte ptr [ebp+ebx-00000108]
:00463399 E826FAF9FF              call 00402DC4
:0046339E 8885F8FDFFFF            mov byte ptr [ebp+FFFFFDF8], al
:004633A4 C685F9FDFFFF3A          mov byte ptr [ebp+FFFFFDF9], 3A
:004633AB C685FAFDFFFF5C          mov byte ptr [ebp+FFFFFDFA], 5C
:004633B2 C685FBFDFFFF00          mov byte ptr [ebp+FFFFFDFB], 00
:004633B9 8D85F8FDFFFF            lea eax, dword ptr [ebp+FFFFFDF8]
:004633BF 8BD7                    mov edx, edi
:004633C1 E8D260FAFF              call 00409498
:004633C6 6A00                    push 00000000
:004633C8 6A00                    push 00000000
:004633CA 8D45F8                  lea eax, dword ptr [ebp-08]
:004633CD 50                      push eax
:004633CE 8D45F8                  lea eax, dword ptr [ebp-08]
:004633D1 50                      push eax
:004633D2 8D45F8                  lea eax, dword ptr [ebp-08]
:004633D5 50                      push eax
:004633D6 6800010000              push 00000100
:004633DB 8D85ACFBFFFF            lea eax, dword ptr [ebp+FFFFFBAC]
:004633E1 50                      push eax
:004633E2 8D85A8FBFFFF            lea eax, dword ptr [ebp+FFFFFBA8]
:004633E8 8A95F8FDFFFF            mov dl, byte ptr [ebp+FFFFFDF8]
:004633EE E89119FAFF              call 00404D84
:004633F3 8D85A8FBFFFF            lea eax, dword ptr [ebp+FFFFFBA8]
:004633F9 BAF8344600              mov edx, 004634F8
:004633FE E8611AFAFF              call 00404E64
:00463403 8B85A8FBFFFF            mov eax, dword ptr [ebp+FFFFFBA8]
:00463409 E8461CFAFF              call 00405054
:0046340E 50                      push eax

* Reference To: kernel32.GetVolumeInformationA, Ord:01DFh
                                  |
:0046340F E8F83DFAFF              Call 0040720C
:00463414 85C0                    test eax, eax
:00463416 747F                    je 00463497
:00463418 E86FF7F9FF              call 00402B8C
:0046341D A1682E4900              mov eax, dword ptr [00492E68]
:00463422 C60000                  mov byte ptr [eax], 00
:00463425 8D85A4FBFFFF            lea eax, dword ptr [ebp+FFFFFBA4]
:0046342B 8D95F8FDFFFF            lea edx, dword ptr [ebp+FFFFFDF8]
:00463431 B900010000              mov ecx, 00000100
:00463436 E8D119FAFF              call 00404E0C
:0046343B 8B95A4FBFFFF            mov edx, dword ptr [ebp+FFFFFBA4]
:00463441 8D85ACFCFFFF            lea eax, dword ptr [ebp+FFFFFCAC]
:00463447 E8BCFCF9FF              call 00403108
:0046344C BA01000000              mov edx, 00000001
:00463451 8D85ACFCFFFF            lea eax, dword ptr [ebp+FFFFFCAC]
:00463457 E8BC01FAFF              call 00403618
:0046345C A1682E4900              mov eax, dword ptr [00492E68]
:00463461 C60002                  mov byte ptr [eax], 02
:00463464 E823F7F9FF              call 00402B8C
:00463469 85C0                    test eax, eax
:0046346B 752A                    jne 00463497
:0046346D 8D85ACFCFFFF            lea eax, dword ptr [ebp+FFFFFCAC]
:00463473 E80CFEF9FF              call 00403284

* Possible StringData Ref from Code Obj ->"OUTBREAK"
                                  |
:00463478 BAFC344600              mov edx, 004634FC
:0046347D 8D85ACFBFFFF            lea eax, dword ptr [ebp+FFFFFBAC]
:00463483 E85060FAFF              call 004094D8
:00463488 85C0                    test eax, eax
:0046348A 750B                    jne 00463497
```

```
:0046348C 8A85F8FDFFFF              mov al, byte ptr [ebp+FFFFFDF8]
:00463492 8845FF                    mov byte ptr [ebp-01], al
:00463495 EB28                      jmp 004634BF

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:0046338C(C), :00463416(C), :0046346B(C), :0046348A(C)
|
:00463497 83C304                    add ebx, 00000004

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00463377(U)
|
:0046349A 8BC3                      mov eax, ebx
:0046349C 99                        cdq
:0046349D 52                        push edx
:0046349E 50                        push eax
:0046349F 8BC6                      mov eax, esi
:004634A1 48                        dec eax
:004634A2 33D2                      xor edx, edx
:004634A4 3B542404                  cmp edx, dword ptr [esp+04]
:004634A8 750D                      jne 004634B7
:004634AA 3B0424                    cmp eax, dword ptr [esp]
:004634AD 5A                        pop edx
:004634AE 58                        pop eax
:004634AF 0F87C7FEFFFF              ja 0046337C
:004634B5 EB08                      jmp 004634BF

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004634A8(C)
|
:004634B7 5A                        pop edx
:004634B8 58                        pop eax
:004634B9 0F8FBDFEFFFF              jg 0046337C

* Referenced by a (U)nconditional or (C)onditional Jump at Addresses:
|:00463495(U), :004634B5(U)
|
:004634BF 33C0                      xor eax, eax
:004634C1 5A                        pop edx
:004634C2 59                        pop ecx
:004634C3 59                        pop ecx
:004634C4 648910                    mov dword ptr fs:[eax], edx
:004634C7 68E4344600                push 004634E4

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:004634E2(U)
|
:004634CC 8D85A4FBFFFF              lea eax, dword ptr [ebp+FFFFFBA4]
:004634D2 BA02000000                mov edx, 00000002
:004634D7 E8EC16FAFF                call 00404BC8
:004634DC C3                        ret
```

This routine of CD checking is very close to that of Tennis Masters Series 2003, except for the use of GetLogicalDriveStringsA API…
This API determines the drives configuration of your computer, making it possible to restrict the tests by GetDriveTypeA API on drives to the existing ones only.
The GetDriveTypeA API determines for a given drive, if it is a CD drive.
In a such case, the GetVolumeInformationA API returns the volume of inserted CD (if there is one).
Lastly, returned volume is compared with "OUTBREAK" string …

This routine is called twice, at 0046369B and at 0048E0D8…
The call in 0046369B is not followed by a test, contrary to the one in 0048E0D8 :

```
* Referenced by a CALL at Address:
|:0048E3CF
|
:0048E0D0 53                        push ebx
:0048E0D1 33DB                      xor ebx, ebx

* Possible StringData Ref from Code Obj ->"outbreaksetup.exe"
                                    |
:0048E0D3 B8E8E04800                mov eax, 0048E0E8
:0048E0D8 E85752FDFF                call 00463334
:0048E0DD 84C0                      test al, al
:0048E0DF 7402                      je 0048E0E3
:0048E0E1 B301                      mov bl, 01

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:0048E0DF(C)
|
:0048E0E3 8BC3                      mov eax, ebx
:0048E0E5 5B                        pop ebx
:0048E0E6 C3                        ret
```

You will have understood that you just have to nop the conditional jump in 0048E0DF, to launch the game (with the movies) and to play without CD!
And for the fans of String Data References :

```
* Possible StringData Ref from Code Obj ->"No CD disk in drive"
                                    |
:0048E4E2 B9D0E94800                mov ecx, 0048E9D0

* Possible StringData Ref from Code Obj ->"There is no Codename:Outbreak "
                                        ->"CD disk in your drive."
                                    |
:0048E4E7 BAE4E94800                mov edx, 0048E9E4
:0048E4EC A1D82D4900                mov eax, dword ptr [00492DD8]
:0048E4F1 8B00                      mov eax, dword ptr [eax]
:0048E4F3 E8E053FCFF                call 004538D8
```

E) Thanks:

Greetz:

ACiD BuRN, ArthaXerXès, Black Check, cdkiller (ProtectionID), CyberBob Jr, ^DAEMON^, Dark-Angel, DecOde12, diablo2oo2, Duelist, El-Caracol, EliCZ, Elraizer, evlncrn8, Fravia+, +Frog's Print, KeopS, Gádix, GRim@, G-RoM, Iczelion, kilby, Laxity, Lorian, LutiN NoIR, MackT, MrOcean, NeuRaL NoiSE, Neutral AtomX, Ni2, Nody, [NtSC], +ORC, Pedro, Peex, PEiD (snaker, Qwerton, Jibz), Psyché, Pulsar, +Pumqara, Ricardo Narvaja, Skuater, +Spath, +splaj, Stone, TeeJi, The Owl, tHeRaiN, TaMaMBoLo, +Tsehp, Tola, woodmann, +Xoanon, [yAtes], yoda... and all those which I forgot and who contribute actively to the scene by their tutorials, tools and others…

all the icedump TEAM, ARTeam, CracksLatinos, DREAD, FRET, ShmeitCorp, UCF2000, UNPACKiNG GODS, TMG, all game groups…

All +HCU Students
All ppl of RCE Messageboard

these great sites:
http://207.218.156.34/krobar/index.html
http://arteam.accessroot.com/
http://tuts4you.com/
http://www.woodmann.net/forum/index.php
http://www.woodmann.net/yates/index.htm

Special Greetz:

Christal: Thank you for all that you did for the French scene and your so great implication. My best regards :)
Laserlock: Thank you to the developers of this protection, for the few recreation hours (too few …)
+Frog's Print & +Spath: Thank you for FrogsIce :-)
GRim@: Thank you for your "Beginner" tutorials, with which I started…
R!SC: Thanks, Master, for your tuts about CD-ROM commercial protections, like Safedisc, SecuROM, VOB ProtectCD, etc…: -)
All members of FFF ;-)

Message to **TomRipley** : Plz, come back !!!

Final words:

I hope that you enjoyed reading this tutorial.

If you want to study this protection, I can upload these tutorial targets. So, mail me at :
dWx5c3NlMjAwOV9mckB5YWhvby5mcg==
(base64 encoded)

I tried to make an original presentation "cracking"/"reversing" to show that it was possible to crack commercial protections with a minimum of knowledge and without be aware of all the various anti-cracking techniques of protection…
A vague idea of these various techniques is largely enough as you have seen…
It is nevertheless much more interesting to reverse these techniques and it highly facilitates the attack ;)
But, the boundary between these two concepts tends to be currently erased.
Reality is always much more complex than a simple dichotomy…

Concerning Tennis Masters Series 2003, I would say that it is one of the best (more complete) games of tennis, on PC, with the Top Spin series …
It is very far from the poor/terrible level of the Roland Garros game on PC!!!

"If you like a game, Buy it !"

uLysse, on 27 / 02 /07

"There is a crack, a crack in everything... That's how the light gets in."

For any critical comments, suggestions, informations about the latest Laserlock protections, feel free to contact me at :
dWx5c3NlMjAwOV9mckB5YWhvby5mcg==
(base64 encoded)
Any crack request will be ignored !!!