© The CodeBreakers-Journal, Vol. 1, No. 1 (2004)
http://www.CodeBreakers-Journal.com

Sharepad - Transforming the Windows Notepad in Shareware

K. Anubis

Abstract

I had read in the past a challenge which consisted in transforming the windows' notepad into a shareware. I have no idea if this has already been done, and as I have always been excited by Reverse Engineering, I have always wanted to write an essay about it. I hope that you will also have fun in reading it as I had in writing it :o) This essay is written in 2 parts. Part 1 deals with a transformation which needs no GUI (interface), only pure coding under an hexeditor. Part 2 will deal with GUI and has the typical registration box with name and serial calculation.

Keywords: *Sharepad; Notepad; Code Injection; Adding Functionality*

I. Tools required

For the 1st part:

Hex-Workshop (for applying changes to the file)
HIEW (to not be bored by calculating the jumps :)
A Win-API reference (Win32.hlp for some APIs)
W32Dasm (for checking the imported functions, it goes easier!)
A resource editor (to locate some strings and IDs)

For the 2nd part:

Actually, the same as above, but used in a more deeper way Softice is also needed (when building the registration box) BRW - Borland Resource Workshop (for building the registration dialog and adding the new menu items).

We won't need it, but I allow me to attach to this essay the ones of LaZaRuS and NeuRaL_NoiSE I have mentioned above for information purposes

II. Target's URL/FTP

This is not a cracking essay (it is actually just the opposite). I used the Windows Notepad which was shipped with Win 98.

III. Program History

Nothing special to say here.

IV. Essay

Before starting: some general comments about the Notepad...
It is advised to have some PE files structure skills to approach this tutorial. Some notions (PEP, IAT, RVA,...) will not be explained when they will be used. Moreover, it is useful to know how to manipulate APIs (parameters pushing order,...) and to calculate jumps (jne,jmp,...). Finally, you should already have used a resources editor.

A short look to Procdump shows that the RawOffset and the VirtualOffset are the same. This will simplify a lot the calculations because the RVA is equal to the op code's offset under an hex editor (modulo the ImageBase which is 0x400000).

The PEP is in 0x10CC.

On a general way, when one cracks, the StringDataRefs button under Wdasm is often used. When one reverses, the Imported Functions button will be rather used to play with the APIs ;o).

Compatibility
The sharepad1 has been successfully tested under win 9x, win 2000 and win XP. It does not work under win 3.1 and win NT. The sharepad2 has been tested on the same matter and offers the same results except with win 2000 and win XP which crash the sharepad at its start when the API RegQueryValueExA is called. This API does not succeed to read the keys Name and Code for reasons I have not studied. To remedy the problem, it suffices to initialise these keys in the registry base. To do this, use the file init s2 win2k-xp.reg shipped with this tutorial. On the same way, the REGBOX is not fully displayed, but does work. As I am not using and programming under win 2000 and XP, I did not studied the point. The keygen provided for the sharepad2 is a DOS program.

The compatibility with win Me has not been tested.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

A. Part I : version without GUI

Aim:
The aim is to make a shareware of the notepad with the following restrictions:

```
- display of a msgbox at the beginning of the software
- display of a msgbox at the end of the software
- display of the word "SHAREWARE" in the title bar of the software
- menus "SAVE" and "SAVE AS..." are deactivated
```

This shareware must become a full version without the above restrictions as soon as one will have provided the activation (registration) key. This activation key is only constituted by the presence of the file ”sharepad.key” in the C:/ directory. Its presence or absence will make the notepad being a full version or a shareware.

1) Display of the word ”SHAREWARE” in the title bar of the software: We just look for the string ”Notepad” in an hexeditor, and we change one letter in this string until we get the right one (of course, the changes are turned back if the right string is not found!). The right string is found at 0xB5B2 :

```
0000B590 6700 6500 7300 3F00 0800 5500 6E00 7400 g.e.s.?...U.n.t.
0000B5A0 6900 7400 6C00 6500 6400 0A00 2000 2D00 i.t.l.e.d... .-.
0000B5B0 2000 4E00 6F00 7400 6500 7000 6100 6400  .N.o.t.e.p.a.d.
0000B5C0 0000 0000 0000 0000 1000 4300 6100 6E00 ..........C.a.n.
```

Then, ”Notepad” can be changed in ”SHAREWARE ”. But because ”SHAREWARE” is 9 letters long where ”Notepad” is 7 letters long, we have to adapt the change a little. If the last ”E” of ”SHAREWARE” is not at the same position as the ”d” of Notepad, all letters after this position will not be displayed! Instead of making hundreds of explanations, just compare the right change below with the original above :

```
0000B590 6700 6500 7300 3F00 0800 5500 6E00 7400 g.e.s.?...U.n.t.
0000B5A0 6900 7400 6C00 6500 6400 0A00 2D00 5300 i.t.l.e.d...-.S.
0000B5B0 4800 4100 5200 4500 5700 4100 5200 4500 H.A.R.E.W.A.R.E.
0000B5C0 0000 0000 0000 0000 1000 4300 6100 6E00 ..........C.a.n.
```

To turn back to ”Notepad”, we will see that later.

2) Deactivation of the menus ”SAVE” and ”SAVE AS...”: We can of course do that very easily with a resource editor. But as we have to turn back this change in the registered version of the sharepad, we have to know how to make this modification. Therefore, we make a copy of the notepad file (which I call 1.exe). We make a second copy (2.exe) which we will modify in the resource editor. Under this editor, we 1/ deactivate the menus ”SAVE” and ”SAVE AS...” and 2/ we grey them.
Then, in order to find out the difference between the two files, we enter the following DOS command...:

```
fc 1.exe 2.exe > 123.txt
```

...and the result is displayed in the automatically created 123.txt file:

```
Comparison of the files 1.exe and 2.exe
0000A076: 00 03
0000A086: 00 03
```

(Note: I am translating the French version, so I hope it is the same in the English one)

We notice that in order to grey and deactivate a menu, we have to change 00 in 03 at the appropriate place (check the result in the same time in the hexeditor).

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

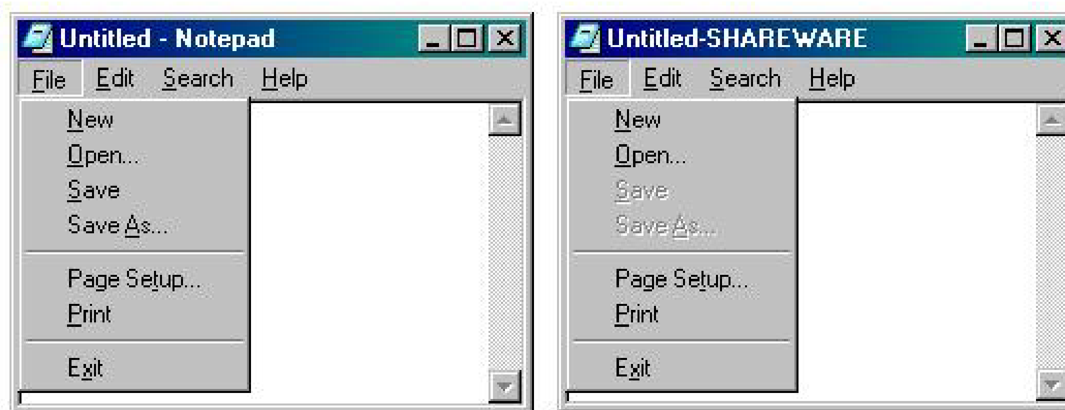

Fig. 1. Sharepad - Restrictions

Before approaching the coding part, here is the working logic of the sharepad:

3) Test of the presence of the "sharepad.key" key: TESTKEY@PEP: At the beginning, we first have to verify the presence of the registration key to determine the behaviour of the sharepad (i.e. shareware or full version). Therefore, we divert the program at the PEP with a jump which goes at the very end of the notepad in the padding after the .rsrc section. Why there? Because it is most of the time the place in a program where there is the biggest padding. And if this place would be too short, we would have to create a new section in which we could quietly work.

```
//******************** Program Entry Point ********
:004010CC 55                    push ebp
:004010CD 8BEC                  mov ebp, esp
:004010CF 83EC44                sub esp, 00000044
:004010D2 56                    push esi
```

modified in :

```
//******************** Program Entry Point ********
:004010CC E97FB90000            jmp 0040CA50
:004010D1 90                    nop
:004010D2 56                    push esi
```

At the end of the program, I choose the offset CA50 to start my code. The instructions that we have overwritten by writing the jump at the PEP are copied from this address. Then, we directly code the API to test the presence of the file "sharepad.key". How to select a suitable API for what we want to do? It's very easy, there are 2 conditions to fulfil. The first one is that the API can tell us if the file "sharepad.key" is really in `C:\` (I have chosen this default directory because everybody has it on its hard drive!). So, it will be to our interest to choose a kind of APIs like CreateFileA, _lopen, FindFirstFile, GetFileAttribute... this means something in relation with files. The second condition is that the chosen API is present in the notepad's IAT. Otherwise, its call has to be coded and this make the work harder (Note: this will be done in the second part of this tutorial, but not here, because the simple, fast and efficient coding is preferred). To know this, just have a look in the Imported Functions in Wdasm and choose the suitable APIs.

The chosen API is "_lopen". It is in the kernel32.dll DLL and has only two parameters to push. Although this API is now old fashioned, it is still very useful and pretty short to code. This simplifies a lot the coding task in comparison to "CreateFileA" for instance (have a look at this API in the win32hlp).

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

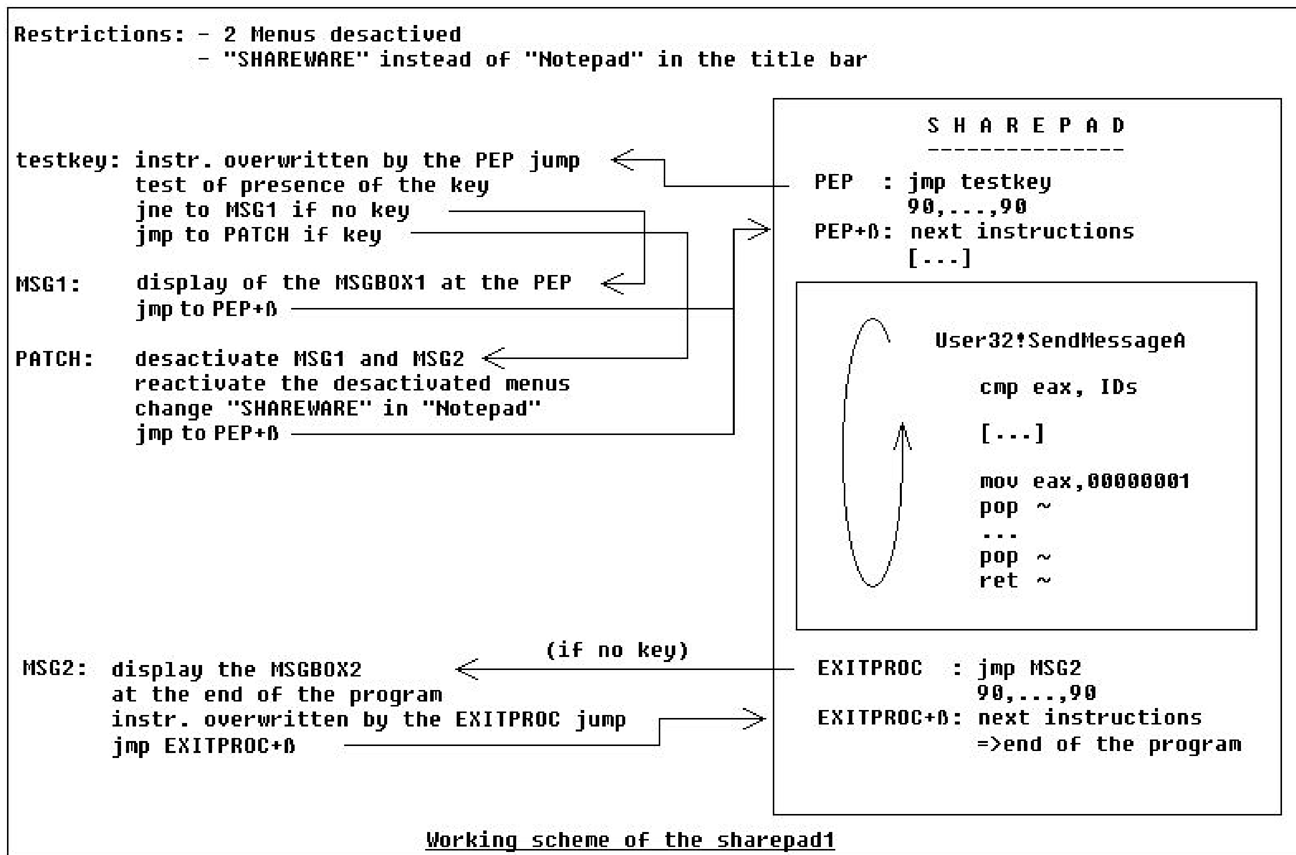

Fig. 2. Sharepad - Algorithm of Restrictions

Here is an overview of the _lopen API:

```
HFILE _lopen(
 LPCSTR lpPathName,// pointer to name of file to open
 int iReadWrite  // file access mode
);
```

the file access mode is :

```
Value          Meaning   Code
OF_READ          Opens the file for reading only   01
OF_READWRITE Opens the file for reading and writing   02
OF_WRITE Opens the file for writing only   03
```

Here, we will choose the pathname "`C:\sharepad.key`" because everybody has this directory on his hard drive, and we will select a file access mode of READWRITE, which has the value 2. Of course, we can put the sharepad.key file in the same directory as the executable. In this case, we'll just have to change the string "`C:\sharepad.key`" below in "`.\sharepad.key`". I have not put the key in the same directory as the .exe file, just to show that the key could be put anywhere and especially in a system directory.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

The string "C:\sharepad.key" is written without quotes in C9F0 directly in an hexeditor. Thus, the API looks like the following :

```
0000C9F0 433A 5C73 6861 7265 7061 642E 6B65 7900 C:\sharepad.key.
0000CA00 0000 0000 0000 0000 0000 0000 0000 0000 ................
.0040CA56: 6A02                     push        002 <- 1st parameter of the API
.0040CA58: 68F0C94000               push        00040C9F0 <- 2nd parameter of the API
.0040CA5D: FF1560634000             call        _lopen <- Call of the API
```

How to find the hexa code for an API? How to call an API? The method I am using is to search for the API under Wdasm in the Imported Functions. If you double click on the name of the API you need, you will always land on the same hex code (except for the last one, at the bottom part of the listing, but that is another story). So, if you double click on _lopen (and not lopen which is not there), you will always land on the same hex code (the FF1560634000). Well, actually there is only one issue of _lopen, but it does not change what I just said. This hex value contains a dword parameter which is bound to the API and corresponds to its address. Thus, in any part of the program, it will be possible to call this API by doing "call ¡ dword ¿". I.e. in using the sequence FF1560634000 for the _lopen API.

Once the API has been coded, we will use a second API to know the presence of the file "sharepad.key" (i.e. the answer of the API _lopen). This second API is GetLastError. It has no parameter to push, and return a specific value in eax according the presence or not of the file "sharepad.key" pushed in parameter in _lopen.

Here is the overview of the GetLastError API:

```
DWORD GetLastError(VOID)
```

Then, the eax value is tested (test eax, eax), and we jump to the MSGBOX1 (jne MSGBOX1) if the eax value is not equal to zero. Otherwise, we jump to the PATCH part (jmp PATCH) to set the sharepad back to the notepad version if eax is equal to zero. Finally, we get for the TESTKEY@PEP part, we get the following:

```
.0040CA50: 55                       push        ebp |Instructions overwritten
.0040CA51: 8BEC                     mov         ebp,esp |by the jump at the PEP
.0040CA53: 83EC44                   sub         esp,044  |
.0040CA56: 6A02                     push        002 <- 1st parameter of the API
.0040CA58: 68F0C94000               push        00040C9F0    <- 2nd parameter
                                                                of the API
.0040CA5D: FF1560634000             call        _lopen  <- Call of the API
.0040CA63: FF15C8634000             call        GetLastError  <- Treatment of
                                                             the sent back
                                                             message
.0040CA69: 85C0                     test        eax,eax <- "sharepad.key" here?
.0040CA6B: 7543                     jne        .00040CAB0    <- no, we go to
                                                             MSGBOX1@TEST
.0040CA6D: E9BE000000               jmp        .00040CB30 <- yes, we go to
                                                             PATCH@TEST
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Finally, we get under an hexeditor:

```
0000C9F0 433A 5C73 6861 7265 7061 642E 6B65 7900 C:\sharepad.key.
0000CA00 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA10 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA20 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA30 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA40 5445 5354 4B45 5940 5045 5000 0000 0000 TESTKEY@PEP.....  <-- Title of the
                                                                   part. Does not
                                                                   act in the code.
0000CA50 558B EC83 EC44 6A02 68F0 C940 00FF 1560 U....Dj.h..@...`
0000CA60 6340 00FF 15C8 6340 0085 C075 43E9 BE00 c@....c@...uC...
0000CA70 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA80 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Note : We will write many strings in the hexeditor (for the MSGBOXs). It is strongly advised to leave a blank line between each string for legibility reason and buffer management in the APIs.

4) Msgbox display at the start of the program: MSGBOX1@TEST: Actually, this messagebox comes after TESTKEY@PEP, but when the program is started, only the messagebox MSGBOX1 is displayed. This messagebox has "SHAREWARE!!!" for title, and "Please register your version." for message. It is coded as the following:

```
0000C990 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C9A0 5348 4152 4557 4152 4521 2121 0000 0000 SHAREWARE!!!....  <-- Title
0000C9B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C9C0 506C 6561 7365 2C20 7265 6769 7374 6572 Please, register  <-- Message
0000C9D0 2079 6F75 7220 7665 7273 696F 6E2E 0000  your version...
0000C9E0 0000 0000 0000 0000 0000 0000 0000 0000 ................
[...]
0000CA90 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CAA0 4D53 4742 4F58 3140 5445 5354 0000 0000 MSGBOX1@TEST....  <-- Title of the part.
                                                                   Does not act in the code.
0000CAB0 6A00 68A0 C940 0068 C0C9 4000 6A00 FF15 j.h..@.h..@.j...
0000CAC0 A864 4000 E909 46FF FF00 0000 0000 0000 .d@...F.........
0000CAD0 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Which gives in asm:

```
.0040CAB0: 6A00                        push       000
.0040CAB2: 68A0C94000                  push       00040C9A0     <-- Title
.0040CAB7: 68C0C94000                  push       00040C9C0     <-- Message
.0040CABC: 6A00                        push       000
.0040CABE: FF15A8644000                call       MessageBoxA
.0040CAC4: E90946FFFF                  jmp       .0004010D2     <-- Back to the PEP
                                                                after the 90(s).
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Fig. 3. Sharepad - Shareware-Messagebox

5) Transformation in registered version: PATCH@TEST (1st part): For the moment, we will only be content with displaying a messagebox which title and message are the same (for instance "SHAREWARE!!!" in C9A0). Then, we branch again this messagebox at the same place where MSGBOX1@TEST branches, i.e. just after the 90(s) at the PEP.

```
0000CB20 5041 5443 4840 5445 5354 0000 0000 0000 PATCH@TEST......
0000CB30 6A00 68A0 C940 0068 A0C9 4000 6A00 FF15 j.h..@.h..@.j...
0000CB40 A864 4000 E989 45FF FF00 0000 0000 0000 .d@...E.........
0000CB50 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Which is in asm:

```
.0040CB30: 6A00                   push     000
.0040CB32: 68A0C94000             push     00040C9A0     <-- Title
.0040CB37: 68A0C94000             push     00040C9A0     <-- Message
                                                          (=Title)
.0040CB3C: 6A00                   push     000
.0040CB3E: FF15A8644000           call     MessageBoxA
.0040CB44: E98945FFFF             jmp     .0004010D2     <-- Back to the
                                                          PEP after the
                                                          90(s).
```

What is the reason? Well, until now, we can test our shareware system!!!

This works fine. And of course, if we delete the file "sharepad.key", we automatically get back in the shareware version. The change is reversible at will. As for the key file "sharepad.key", there is nothing in it. Its contain is even not tested. It is its PRESENCE in the C: directory that makes that the user has registered or not:

```
- He knows that a key file is needed to be registered
- He knows the name of this file
- He knows WHERE to put this file
```

Furthermore, the aim of this tutorial is to introduce a shareware mechanism on a freeware, and not to set up a shareware security. On the security focus, this mechanism is null, and I remember that it is not the aim of this tutorial (I will also crack this security at the end of this part I to show it).

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Fig. 4. Sharepad - Keyfile missing

Fig. 5. Sharepad - Keyfile there!

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

6) Msgbox display at the end of the program: MSGBOX2: This messagebox is branched at the end of the program when we click on ”Exit” or on the X cross on the top right side of the window. These two commands call the API ExitProcess. We look under Wdasm in the ImportedFunctions, and we find (of course) only one occurrence in the listing...:

```
 * Reference To : KERNEL32.ExitProcess, Ord: 007Fh
 |
 :00401143 FF1598634000            Call dword ptr [00406398]
 :00401149 8BC6                    mov eax, esi
 :0040114B 5E                      pop esi
 :0040114C 8BE5                    mov esp, ebp
 :0040114E 5D                      pop ebp
 :0040114F C3                      ret

...which we transform in:

 :00401143 E9A8B90000              jmp 0040CAF0
 :00401148 90                      nop
 :00401149 8BC6                    mov eax, esi
 :0040114B 5E                      pop esi
 :0040114C 8BE5                    mov esp, ebp
 :0040114E 5D                      pop ebp
 :0040114F C3   ret
```

And in CAF0, we code the MSGBOX2:

```
0000C940 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C950 446F 6E27 7420 666F 7267 6574 2074 6F20 Don't forget to   <-- Message
0000C960 7265 6769 7374 6572 2E20 5265 6164 2072 register. Read r
0000C970 6567 2E74 7874 2066 6F72 2064 6574 6169 eg.txt for detai
0000C980 6C73 2E00 0000 0000 0000 0000 0000 0000 ls..............
0000C990 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C9A0 5348 4152 4557 4152 4521 2121 0000 0000 SHAREWARE!!!....  <-- Title
0000C9B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
[...]
0000CAD0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CAE0 4D53 4742 4F58 3240 4558 4954 0000 0000 MSGBOX2@EXIT....  <-- Title of the
                                                                  part. Does not
                                                                  act in the code.
0000CAF0 6A00 68A0 C940 0068 50C9 4000 6A00 FF15 j.h..@.hP.@.j...
0000CB00 A864 4000 FF15 9863 4000 8BC6 E938 46FF .d@....c@....8F.
0000CB10 FF00 0000 0000 0000 0000 0000 0000 0000 ................
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Which is in asm:

```
.0040CAF0: 6A00                   push      000 |Parameters of
                                                 the messagebox
.0040CAF2: 68A0C94000             push      00040D9A0 |
.0040CAF7: 6850C94000             push      00040D950   |
.0040CAFC: 6A00                   push      000 |
.0040CAFE: FF15A8644000           call      MessageBoxA  <-- API
.0040CB04: FF1598634000           call      ExitProcess  |Instructions
                                                          overwritten by
                                                          the jump at the
.0040CB0A: 8BC6                   mov       eax,esi |original ExitProcess
.0040CB0C: E93846FFFF             jmp      .000401149    <-- Back to the
                                                          EXITPROC after
                                                          the 90(s).
```

Fig. 6. Sharepad - Messagebox "‘Dont forget!"’

Summary until here

The following functions have been implemented:

```
- display of a msgbox at the start of the program
- display of a msgbox at the end of the program
- display of the word "SHAREWARE" in the title bar of the program
- menus "SAVE" and "SAVE AS..." deactivated
```

And the sharepad reacts on the presence of a deactivation key file in the C: directory. The added or modified code to the original notepad.exe file is the following:

At the PEP:

```
000010C0 2532 2E32 6400 0000 0D0A 0000 E97F B900 %2.2d...........
000010D0 0090 56FF 15E0 6340 008B F08A 003C 2275 ..V...c@.....<"u

[...]
```

At the EXITPROCESS:

```
00001140 508B F0E9 A8B9 0000 908B C65E 8BE5 5DC3 P.........^..].

[...]
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

After the end of the .reloc section:

```
0000C940 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C950 446F 6E27 7420 666F 7267 6574 2074 6F20 Don't forget to
0000C960 7265 6769 7374 6572 2E20 5265 6164 2072 register. Read r
0000C970 6567 2E74 7874 2066 6F72 2064 6574 6169 eg.txt for detai
0000C980 6C73 2E00 0000 0000 0000 0000 0000 0000 ls..............
0000C990 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C9A0 5348 4152 4557 4152 4521 2121 0000 0000 SHAREWARE!!!....
0000C9B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C9C0 506C 6561 7365 2C20 7265 6769 7374 6572 Please, register
0000C9D0 2079 6F75 7220 7665 7273 696F 6E2E 0000  your version...
0000C9E0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000C9F0 433A 5C73 6861 7265 7061 642E 6B65 7900 C:\sharepad.key.
0000CA00 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA10 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA20 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA30 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA40 5445 5354 4B45 5940 5045 5000 0000 0000 TESTKEY@PEP.....
0000CA50 558B EC83 EC44 6A02 68F0 C940 00FF 1560 U....Dj.h..@...`
0000CA60 6340 00FF 15C8 6340 0085 C075 43E9 BE00 c@....c@...uC...
0000CA70 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA80 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CA90 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CAA0 4D53 4742 4F58 3140 5445 5354 0000 0000 MSGBOX1@TEST....
0000CAB0 6A00 68A0 C940 0068 C0C9 4000 6A00 FF15 j.h..@.h..@.j...
0000CAC0 A864 4000 E909 46FF FF00 0000 0000 0000 .d@...F.........
0000CAD0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CAE0 4D53 4742 4F58 3240 5445 5354 0000 0000 MSGBOX2@TEST....
0000CAF0 6A00 68A0 C940 0068 50C9 4000 6A00 FF15 j.h..@.hP.@.j...
0000CB00 A864 4000 FF15 9863 4000 8BC6 E938 46FF .d@....c@....8F.
0000CB10 FF00 0000 0000 0000 0000 0000 0000 0000 ................
0000CB20 5041 5443 4840 5445 5354 0000 0000 0000 PATCH@TEST......
0000CB30 6A00 68A0 C940 0068 A0C9 4000 6A00 FF15 j.h..@.h..@.j...
0000CB40 A864 4000 E989 45FF FF00 0000 0000 0000 .d@...E.........
0000CB50 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000CB60 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Now, we will set up the neutralisation of the shareware elements when the file "sharepad.key" is in `C:\`.

7) Transformation in registered version: PATCH@TEST (2nd part): In this part, a mean has to be found in order that:

```
- the 2 msgboxes are no more displayed
- the 2 menus are activated
- "SHAREWARE" is replaced by the original word "Notepad"
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Let's have a look to these points in details...

**** Deactivation of the MSGBOX1 and MSGBOX2 msgboxes:***

Nothing else is easier. MSGBOX1 is absolutely not displayed because the way goes through PATCH@TEST and then goes directly back after the PEP. As for MSGBOX2, a simple patch of the "call messageboxA" instruction will neutralise it:

```
 FF15A8644000    call  MessageBoxA
```

...becomes...:

```
9015A8644000    call  MessageBoxA
```

...and no msgbox more! In asm, it will be written:

```
.0040CB30: B890000000                 mov        eax,000000090   <-- put 00000090 in eax
.0040CB35: A2FECA4000                 mov        [00040CAFE],al  <-- changes the byte at
                                                                     the address CAFE in 90
```

**** Activation of the 2 menus:***

We could use some APIs to reactivate the modifications of the beginning made "in hard" in the file. Actually, we will use the trick to patch the program in memory only. The file will still stay in a shareware form on the hard drive, but the patch is done in memory. For this, we need as for every patch process:

```
- the address of the byte to patch
- the value to patch
```

And for that, we use again the information of the fc command used before...:

```
Comparison of the files 1.exe and 2.exe
0000A076: 00 03
0000A086: 00 03
```

... but in doing the opposite, we put 00 instead of 03 (and we erase here the code of the msgbox which displayed the same title and message):

```
.0040CB3A: B800000000                 mov        eax,000000000   <-- put 00000000 in eax
.0040CB3F: A276A04000                 mov        [00040A076],al  <-- changes the byte at
                                                                     the address A076 in 00
.0040CB44: A286A04000                 mov        [00040A086],al  <-- changes the byte at
                                                                     the address A086 in 00
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

**** Replacement of the word ”SHAREWARE”:***

Same technique as the both previously cases. Actually, the change is to patch ”-SHAREWARE” in ” - Notepad”. In 32-bits, ” - Notepad” will be written : 20002D0020004E006F00740065007000061006400. We will replace ”-SHAREWARE” DWORD by DWORD. This gives...:

```
.0040CB49: B820002D00                    mov       eax,0002D0020 ;" -  "
.0040CB4E: A3ACB54000                    mov       [00040B5AC],eax
.0040CB53: B820004E00                    mov       eax,0004E0020 ;" N  "
.0040CB58: A3B0B54000                    mov       [00040B5AE],eax
.0040CB5D: B86F007400                    mov       eax,00074006F ;" t o"
.0040CB62: A3B4B54000                    mov       [00040B5B0],eax
.0040CB67: B865007000                    mov       eax,000700065 ;" p e"
.0040CB6C: A3B8B54000                    mov       [00040B5B2],eax
.0040CB71: B861006400                    mov       eax,000640061 ;" d a"
.0040CB76: A3BCB54000                    mov       [00040B5B4],eax
.0040CB7B: E95245FFFF                    jmp      .0004010D2
```

...followed by the last jump which branches back to the PEP after the 90(s). We finally get for PATCH@TEST (the previous version is overwritten!) :

```
0000CB20 5041 5443 4840 5445 5354 0000 0000 0000 PATCH@TEST......
0000CB30 B890 0000 00A2 FECA 4000 B800 0000 00A2 ........@.......
0000CB40 76A0 4000 A286 A040 00B8 2000 2D00 A3AC v.@....@.. .-...
0000CB50 B540 00B8 2000 4E00 A3B0 B540 00B8 6F00 .@.. .N....@..o.
0000CB60 7400 A3B4 B540 00B8 6500 7000 A3B8 B540 t....@..e.p....@
0000CB70 00B8 6100 6400 A3BC B540 00E9 5245 FFFF ..a.d....@..RE..
0000CB80 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

But that’s not all!!!

The sharepad will crash if we start it as it. Do not forget that we patch the MSGBOX2 which is in the section .reloc, as well as the 2 menus and the word ”-SHAREWARE” which are in the section .rsrc. Consequently, as we will write in memory in these sections, we have to verify their characteristics in order that the writing operation runs normally. A short look in Procdump shows us the following original data:

Name	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	00003E9C	00001000	00004000	00001000	60000020
.data	0000084C	00005000	00001000	00005000	C0000040
.idata	00000DE8	00006000	00001000	00006000	40000040
.rsrc	00004FB8	00007000	00005000	00007000	40000040
.reloc	00000A9C	0000C000	00001000	0000C000	42000040

We see that the sections .rsrc and .reloc are READ ONLY (0x40000000). We will change them in READ + WRITE (0xC0000000):

.rsrc	00004FB8	00007000	00005000	00007000	C0000040
.reloc	00000A9C	0000C000	00001000	0000C000	C2000040

Voil!!! The notepad is now a shareware and is called a sharepad! The activation key is the ”sharepad.key” file which is to be put in the `C:` directory. Of course, we will have obtained this key after consulting the file ”reg.txt” shipped with the sharepad and giving all (financial) details to get the activation key (i.e., send lot’s of specialchars).

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

And now, for the fun!

8) Cracking the sharepad: We have here of course the sharepad, but we have no idea how to turn it in a full version. And we do not have the key ”sharepad.key”. As I said it before, the solidity of the sharepad's security is NULL. By looking the program in an hexeditor, the code is very easy to detect. To deactivate the sharepad's mechanism, it simply suffices to invert the conditional jump of the TESTKEY@PEP part:

```
.0040CA58: 68F0C94000                    push        00040C9F0
.0040CA5D: FF1560634000                  call        _lopen
.0040CA63: FF15C8634000                  call        GetLastError
.0040CA69: 85C0                          test        eax,eax
.0040CA6B: 7543                          jne        .00040CAB0    <- inversion here in 7443
.0040CA6D: E9BE000000                    jmp        .00040CB30
```

A tiny hint: do not leave the key in `C:\`, otherwise you will have a cracked version which is ... shareware :o) Or then, nop directly the whole jump with 9090.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

B. Part II : GUI Version

Aim:

Build a registration box (regbox) in GUI with name + serial, and code the serial calculation routine. The restrictions of the shareware version will be those of the part I.

We'll first work out the drawing of the regbox, then we'll elaborate the mechanism's structure of the shareware, and we'll code it (still in the padding at the end of the .reloc section).

Some advises before starting this part:

I have had lots of problem which were actually none, due to silly reactions from software (SI, Hiew,...), or from code parts which were RIGHT but did not work. If you see that you become crazy on a point for a while without finding a solution, reboot the computer in order to flush the RAM and the software. Often, SI, Hiew or others are running illogically and bring a big mess...

For instance, when a breakpoint is put in SI, you have to know that the byte of the bpx address is replaced by the byte "CC" which corresponds to "int 03". That's the way SI does recognise breaks and can pop-up. Of course, in SI's code windows, you will see ordinary data/code. But in switching between SI, Wdasm and Hiew, I have often found this "CC" back instead of my instructions in the notepad... The solution is to patch the origin value under an hexeditor, and it's good again! Conclusion: MAKE ALWAYS A BACKUP COPY OF THE FILE YOU ARE WORKING ON.

As for HIEW, when a jump is written in asm mode and validated by F9, there are often some "40" (this comes from the ImageBase which value is 0x400000 and which is inopportunely added) which appear and transform a "je 00405656" in "je 00805656" when you are tracing with SI. That's really cool :o/

I start this tutorial having NO knowledge in using the system registry (writing/reading), and a weak knowledge in using APIs (the one of part one of this tutorial). We will learn in the meanwhile! ;o)

In order to choose the IDs, we can take in theory any number as long as it is not already used in the software. Practically, while the IDs comparison in the software, some jumps are "stupid" and make it better to take bigger IDs as the biggest of the software. Example here with the notepad:

We choose an ID of 0x250 for a new menu. Unfortunately, we will jump in 40128D (and 401294). If we branch in 40129A, the code "will work" (i.e. will be bugless) but due to the "jl/jle" jumps, we will never reach our branching (which can although be a bug :o/ !!!).

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
* Possible Ref to Menu: MenuID_0001, Item : "Cut Ctrl+X"
|
:00401288 3D00030000              cmp eax, 00000300
:0040128D 7C21                    jl 004012B0 ; we jump here

* Possible Ref to Menu: MenuID_0001, Item : "Copy Ctrl+C"
|
:0040128F 3D01030000              cmp eax, 00000301
:00401294 0F8E3E040000            jle 004016D8 ; we jump here

* Possible Ref to Menu: MenuID_0001, Item : "Paste Ctrl+V"
|
:0040129A 3D02030000              cmp eax, 00000302 ; branching for our menu
:0040129F 0F8456040000            je 004016FB
```

So we will choose an ID above 310. We will take one for instance above 350 (848 in decimal). Generally, a quick check under a resources editor shows what is the last IDs.

Last advise: RESOURCES ARE ALWAYS FIRST MODIFIED AND LEFT, THE CODE IS DONE ONLY AFTER THIS STEP!!! If you had to modify again the resources (even shortly!) after you have put some code in the padding, you can generally code everything from the beginning (in particularly if you have put your code in the .rsrc section, because it will be overwritten in the new compilation... and the same for your code). To avoid this problem and to be able to modify the resources after you started coding, you have to create a new section and code in it.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

In this second part of the sharepad, I will take the following components/IDs :

```
Registration box : ID=1664 (any link with a drink is... only pure accident ;o)
                             I prefer the Mort-Subite :oD )
Edittext (name) : ID=900
Edittext (code) : ID=901
Text (name) : ID=902
Text (code) : ID=903
Button (validate) : ID=904
Button (cancel) : ID=905

Sub-menu "Register..." : ID=910 (Ctrl+T)
Sub-menu "About Sharepad" : ID=911

Shortcut Ctrl+T : ID=950

Script of the regbox under BRW:

1664 DIALOG 6, 15, 180, 75
STYLE DS_MODALFRAME | WS_POPUP | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "Registration"
FONT 8, "MS Sans Serif"
{
 EDITTEXT 900, 44,10,119,14, WS_BORDER
 EDITTEXT 901, 44,30,119,14, WS_BORDER
 LTEXT "Name:", 902, 18,12,20,14
 LTEXT "Code:", 903, 18,32,20,14
 DEFPUSHBUTTON "Validate", 904, 44, 56, 50, 14
 PUSHBUTTON "Cancel", 905, 113, 56, 50, 14
}

And of the menu:

 POPUP "Re&gistration"
 {
  MENUITEM "Reg&ister...\tCtrl+T", 910
  MENUITEM SEPARATOR
  MENUITEM "A&bout Sharepad", 911
 }

And of the shortcut (Ctrl+T):
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
1 ACCELERATORS
{
 VK_INSERT, 769, VIRTKEY, CONTROL
 VK_F1, 5, VIRTKEY
 VK_F3, 8, VIRTKEY
 VK_F5, 12, VIRTKEY
 VK_BACK, 25, VIRTKEY, ALT
 "^Z", 25, ASCII
 "^T", 950, ASCII
 "^X", 768, ASCII
 "^C", 769, ASCII
 "^V", 770, ASCII
}
--------------------------------

2 ACCELERATORS
{
 VK_INSERT, 769, VIRTKEY, CONTROL
 VK_F1, 5, VIRTKEY
 VK_F3, 8, VIRTKEY
 VK_F5, 12, VIRTKEY
 VK_BACK, 25, VIRTKEY, ALT
 "^Z", 25, ASCII
 "^T", 950, ASCII
 "^X", 768, ASCII
 "^C", 769, ASCII
 "^V", 770, ASCII
 VK_ESCAPE, 28, VIRTKEY
 "C", 28, VIRTKEY, CONTROL
 "D", 28, VIRTKEY, CONTROL
 "Z", 28, VIRTKEY, CONTROL
}
```

So, a regbox is added with 2 EDIT fields (”Name:” and ”Code:”, resp. IDs 900 and 901), as well as 2 buttons (”Validate” and ”Cancel”, resp. IDs 904 and 905). As for the menu, it is inserted between ”Search” and ”Help” a menu ”Registration” which contains two sub-menus (”Register... Ctrl+T” and ”About Sharepad”, resp. IDs 910 and 911). All of this is entirely done with Borland Resource Workshop. No other software is used to build up and include these resources.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

The size of notepad.exe grows from 52 to 56 Ko. Out of curiosity, we quickly check the difference with Procdump:

```
Before the resources' compilation:

  Name        Virtual Size    Virtual Offset  Raw Size    Raw Offset  Characteristics
 .text        00003E9C        00001000        00004000    00001000    60000020
 .data        0000084C        00005000        00001000    00005000    C0000040
 .idata       00000DE8        00006000        00001000    00006000    40000040
 .rsrc        00004FB8        00007000        00005000    00007000    40000040
 .reloc       00000A9C        0000C000        00001000    0000C000    42000040

After the resources' compilation:

  Name        Virtual Size    Virtual Offset  Raw Size    Raw Offset  Characteristics
 .text        00003E9C        00001000        00004000    00001000    60000020
 .data        0000084C        00005000        00001000    00005000    C0000040
 .idata       00000DE8        00006000        00001000    00006000    40000040
 .reloc       00000A9C        00007000        00001000    00007000    42000040
 .rsrc        00004FB8        00008000        00006000    00008000    40000040
```

The size of the sections has not been modified except for .rsrc, .rsrc and .reloc have been inverted while the recompilation. This is the result of the resources editor... This is no problem for us.

Well, now we have to branch the regbox (ID 1664) on the menu ”Register... Ctrl+T” (ID 910) and the MSGBOX5 on ”About Sharepad” (ID 911).

In order to know how to, it's better to have some skills about the handling of events in a program under windows. The handling of an event is : ”What does happen when a button, a menu, ... (a resource) is clicked, or that an action is done in the program?”. The program is like a piano. It remains silently as long as no action is done, but as soon as it is the case, the action is analysed and the program acts consequently (in using the ID of the executed action). For physicists, this corresponds to the Galileo inertia principle : ”Each body (aka the program) remains in the uniform movement in which it is, unless that any force (aka user's action) acts on it and make it changing its status”. So, when a resource is clicked, an ID is sent to windows. This last, through the API User32!SendMessageA, will handle the sending of this information and send the activated ID in eax (to the program). A loop in the program will then compare each ID to the one loaded in eax, and will execute the corresponding part of code after the good comparison.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

For instance:

```
* Possible Ref to Menu: MenuID_0001, Item : "Cut Ctrl+X"
|
:00401288 3D00030000              cmp eax, 00000300
:0040128D 7C21                    jl 004012B0

* Possible Ref to Menu: MenuID_0001, Item : "Copy Ctrl+C"
|
:0040128F 3D01030000              cmp eax, 00000301
:00401294 0F8E3E040000            jle 004016D8

* Possible Ref to Menu: MenuID_0001, Item : "Paste Ctrl+V"
|
:0040129A 3D02030000              cmp eax, 00000302
:0040129F 0F8456040000            je 004016FB
```

We will thus use here a long jump (0F8X...) to branch to our code will find place after the last section as for the part I of this tutorial. By the way, we will set up first the strings we need for this time:

```
Shareware restrictions:
- One MSGBOX1 at the program's PEP (as in part I)
title="SHAREWARE!!!" message="Please register."
- One MSGBOX2 at the program's EXITPROCESS (as in part I)
title="SHAREWARE!!!" message="Do not forget to register. Read the file reg.txt."

Menu "Registration":
- One MSGBOX3 (or Goodboy) in case of successful registration
title="Bravo!" message="Thank you for your support."
- One MSGBOX4 (or Badboy) in case of unsuccessful registration
title="Error!" message="Bad Code"
- One MSGBOX5 for the part "About Sharepad"
title="Sharepad" message="Reversed by Anubis (Shmeitcorp)!"
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

This makes a total of 9 strings to write (MSGBOX1 and MSGBOX2 have the same title). This gives:

```
0000D1A0 5348 4152 4557 4152 4521 2121 0000 0000 SHAREWARE!!!....
0000D1B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D1C0 506C 6561 7365 2072 6567 6973 7465 722E Please register.
0000D1D0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D1E0 446F 206E 6F74 2066 6F72 6765 7420 746F Do not forget to
0000D1F0 2072 6567 6973 7465 722E 2052 6561 6420  register. Read
0000D200 7468 6520 6669 6C65 2072 6567 2E74 7874 the file reg.txt
0000D210 2E00 0000 0000 0000 0000 0000 0000 0000 ................
0000D220 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D230 4272 6176 6F21 0000 0000 0000 0000 0000 Bravo!..........
0000D240 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D250 5468 616E 6B20 796F 7520 666F 7220 796F Thank you for yo
0000D260 7572 2073 7570 706F 7274 2E00 0000 0000 ur support......
0000D270 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D280 4572 726F 7221 0000 0000 0000 0000 0000 Error!..........
0000D290 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D2A0 4261 6420 436F 6465 0000 0000 0000 0000 Bad Code........
0000D2B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D2C0 5368 6172 6570 6164 0000 0000 0000 0000 Sharepad........
0000D2D0 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D2E0 5265 7665 7273 6564 2062 7920 416E 7562 Reversed by Anub
0000D2F0 6973 2028 5368 6D65 6974 636F 7270 2921 is (Shmeitcorp)!
0000D300 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

I left one line between each string for clarity reasons.

Afterwards, the handling of our regbox is coded, then the MSGBOX5 for ”About Sharepad”. We use here the ultra classical technique to overwrite an instruction (which is preferably not a jump, this can avoid some problems...) with our jump. This ”wild” branching jumps to our code which we will inject, and which is usually located at the end of a section in its padding, or in a new created section. Thus, we will have good chances to be located at the very end of the program, after the .reloc and .rsrc. Then, at the beginning of this new code, the overwritten instructions are recopied, and we jump back to the next instruction located after our ”wild” branching.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Here, we pretty have the choice. We will make the branching at the "Ctrl+C" for instance...:

```
 * Possible Ref to Menu : MenuID_0001, Item: "Cut Ctrl+X"
 |
 :00401288 3D00030000              cmp eax, 00000300
 :0040128D 7C21                    jl 004012B0

 * Possible Ref to Menu : MenuID_0001, Item: "Copy Ctrl+C"
 |
 :0040128F 3D01030000              cmp eax, 00000301
 :00401294 0F8E3E040000            jle 004016D8

 * Possible Ref to Menu : MenuID_0001, Item: "Paste Ctrl+V"
 |
 :0040129A 3D02030000              cmp eax, 00000302
 :0040129F 0F8456040000            je 004016FB

...which becomes:

 * Possible Ref to Menu : MenuID_0001, Item: "Cut Ctrl+X"
 |
 :00401288 3D00030000              cmp eax, 00000300
 :0040128D 7C21                    jl 004012B0
 :0040128F E98CC00000              jmp 0040D320 <<== we branch here, jump to D320
 :00401294 0F8E3E040000            jle 004016D8

 * Possible Ref to Menu : MenuID_0001, Item: "Paste Ctrl+V"
 |
 :0040129A 3D02030000              cmp eax, 00000302
 :0040129F 0F8456040000            je 004016FB
```

The sentence " * Possible Ref to Menu : MenuID_0001, Item: "Copy Ctrl+C" " disappears, because there is no longer its ID (301) in the overwritten code.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

In D320, we add the ID comparison code of our menu (ID-COMPARAISON). Here is the result in asm for the MSGBOX5 display...:

```
(ID-COMPARISON)
.0040D320: 60                pushad                        <-- backup of all
                                                               registers
.0040D321: 3D8E030000        cmp        eax,00000038E      <-- regbox chosen?
.0040D326: 0F8484000000      je        .00040D3B0          <-- yes, so its code
                                                               is executed
.0040D32C: 3D8F030000        cmp        eax,00000038F      <-- "About Sharepad"
                                                                MSGBOX5 chosen?
.0040D331: 0F8439000000      je        .00040D370          <-- yes, so its code
                                                               is executed
.0040D337: 61                popad                         <-- backdown of all
                                                               registers
.0040D338: 3D01030000        cmp        eax,000000301      <-- instruction
                                                               overwritten
                                                               by our jump in 40128F
.0040D33D: E9523FFFFF        jmp       .000401294          <-- back to the code just
                                                               after our wild jump

(MSGBOX5)
.0040D370: 6A00              push       000
.0040D372: 68C0D24000        push       00040D2C0          <-- Title
.0040D377: 68E0D24000        push       00040D2E0          <-- Message
.0040D37C: 6A00              push       000
.0040D37E: FF15A8644000      call       MessageBoxA        <-- MSGBOX5 display
.0040D384: 61                popad                         <-- Back down off
                                                               all registers
.0040D385: E92345FFFF        jmp       .0004018AD          <-- Back to the API
                                                               SendMessage loop

...and under the hexeditor:

0000D310 4944 2D43 4F4D 5041 5249 534F 4E00 0000 ID-COMPARISON...    <-- Title of the
                                                                         part. Does
                                                                         not act
                                                                         in the code.
0000D320 603D 8E03 0000 0F84 8400 0000 3D8F 0300 `=..........=...    |Code
0000D330 000F 8439 0000 0061 3D01 0300 00E9 523F ...9...a=.....R?    |Code
0000D340 FFFF 0000 0000 0000 0000 0000 0000 0000 ................    |Code
0000D350 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D360 4D53 4742 4F58 3500 0000 0000 0000 0000 MSGBOX5.........    <-- Title of the
                                                                         part. Does
                                                                         not act
                                                                         in the code.

0000D370 6A00 68C0 D240 0068 E0D2 4000 6A00 FF15 j.h..@.h..@.j...    |Code
0000D380 A864 4000 61E9 2345 FFFF 0000 0000 0000 .d@.a.#E........    |Code
0000D390 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D3A0 5245 4742 4F58 0000 0000 0000 0000 0000 REGBOX..........    <-- Title of
                                                                         the part.
                                                                         Does not
                                                                         act in
                                                                         the code.
0000D3B0 0000 0000 0000 0000 0000 0000 0000 0000 ................
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

The titles of the parts do not act because the different parts of the code are connected to each others by jumps or calls which step above these titles. On the same matter, any other op-code will call/use these strings through their address (offset).

(Addendum to the original French version of this article - valid only for the English version!)

We want now to run a short test for the Msgbox5 and to test the above code. If we try, the software will crash due to a virtual address problem. The solution is to add 1000 bytes to the virtual address of the .rsrc section in which we are coding, rising it from 4FB8 to 5FB8. Otherwise our code is outside this size, and can not be interpreted by the computer. The change is illustrated below, compare it with the last PE header above:

```
  Name        Virtual Size    Virtual Offset  Raw Size        Raw Offset  Characteristics
 .rsrc        00005FB8        00008000        00006000        00008000    40000040
```

(End of the addendum)

We can now do a short test in live for the display of the Msgbox5 which works well. To resume until here, in case you have a problem:

```
- resources are modified (menus, IDs... added)
- branching on Ctrl+C with a "jump D320" which overwrites the "cmp eax, 301"
- in D320:
       backup of all registers (otherwise there's a crash!)
       addition of the IDs' comparisons added by reverse process
       backuped registers' backdown (otherwise there's also a crash!)
       addition of the instruction overwritten by "jump D320"
       jump back just after the "jump D320"
- the MSGBOX5 code is written and ended on the messages' loop of the software in 4018AD.
```

Some comments have to be provided about the above code construction. You may ask yourself ”How do we know that we have to handle the registers?” or ”Do we write the instruction overwritten by our jump BEFORE or AFTER the code we add?”...

When, beginning in the RE, we want to display a MSGBOX, we never care about the registers. Indeed, the API MessageBoxA doesn't modify the registers, so it is not worth to handle them. On the other hand, in the code added here, some tests are called out (as ”cmp” op-code). The response of these tests is recorded in the registers, and if this response is overwritten without saving the registers, then it will be lost and the processor will not appreciate (=will crash) at a certain moment. So the logic is:

```
- registers' backup
- [...]
- MY ADDED CODE
- [...]
- registers' backdown
```

... and there, the addition is ”clean” AND works. This logic is to apply for any code addition, which is somehow ”elaborated”.

For the place of the overwritten instruction (sometimes there are more than one!), this actually depends on the code. We have to handle each case at a time. To illustrate that, let's take for instance the 2 msgboxess of part I of this tutorial. The one is located at the PEP, the other at the end of the program (exitprocess). For the first one, we want it to be displayed before the program's code is executed, so the overwritten instructions will be put AFTER it. For the second one, we want it to be displayed after some code. So it will be the opposite case. All right? ;)

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

About the MSGBOX5, the popad/61 (in D337) is actually not at all indispensable. I have put it by analogy with what I just have written, but it isn't very useful actually... If it disturbs you, you can always replace it by a nop/90. Whereas the 4018AD, how do we know that it is this value? It suffices to trace a while under Wdasm in the jumps of the code which corresponds to the commands Ctrl+C,V,X. We always land on this value at the end of the procedures. And when one is used to reverse, this kind of value at the end of the SendMessage! loop is quickly noticed.

Now, we will code the "call" (the use) of the regbox in D326. But before, we have to think 2 minutes about the "how to...?".

By clicking on the menu "Register...", windows will send back the ID 911 (38Fh) to the notepad which will jump to the above code. Nothing special until here. Afterwards, our REGBOX has to be displayed. For that purpose, we have approximately 2 solutions!

To display a dialog box, we have the choice between two well-known ways (i.e. 2 APIs): CreateDialogParam and DialogBox-Param. Each of these 2 methods its inconvenient and advantages.

CreateDialogParam is a modeless dialog box. This kind of dialog box allow to do some modifications in other opened windows of the program or of an another one. An example is the "Find" dialog box of a text editor. When this box is displayed, it is still possible to use other commands in the menu of the text editor, or to use other(s) program(s).

DialogBoxParam is a modal dialog box. While using this dialog box, the focus is blocked. For instance, with the dialog box "Print...". This focus can be blocked by two different matters: only the proprietary application of the running dialog box is blocked (it is called "application modal") or the focus is blocked for everything as long as the dialog box is not closed (it is called "system modal").

Some other important differences also occurs:

- Display:

- CreateDialogParam CREATES the dialog box in memory without obligatory displaying it. In this case, the API ShowWindow is used.
- DialogBoxParam displays automatically the dialog box.

- Destruction/closing:

- CreateDialogParam: the dialog box is closed by using the API DestroyWindow. If instead of using it, the API EndDialog would be used, we would not seen the dialog box displayed anymore, but it would always be in memory.
- DialogBoxParam: the dialog box is closed by using the API EndDialog.

- Messages handling (WM_COMMAND,...):

- CreateDialogParam must be followed by its own messages handling structure that has to be coded. At the ends, the code is longer as for DialogBoxParam.
- DialogBoxParam generates its own messages handling loop. We do not have to code it. The coding of this API is simple and short.

- Passing the API parameters:

- Passing the parameters for the two APIs is strictly the same.

All these data already show us how we could choose the suitable API in our case. It would be more judicious to choose DialogBoxParam which is easier to code, even if a (small) API has to be additionally used to handle the display. But the major factor in the choice is the availability of these two APIs in the import table. Indeed, we are here reversing and not programming, so we don't have all the wished elements at our disposal. Here, our choice is dictated by the APIs which are in the Imported Functions (i.e. APIs already used by the program). And if we analyse a program like the notepad, there is ONLY the API CreateDialogParam available. So we have no choice!

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Actually, we do have the choice... :o)

Indeed, it is possible to call an API which is not in the import table. If we do not want to debuild the import table and then to rebuild it, there is than almost only one method for, which is a famous one. But to be able to use this method, we need two other APIs which are GetModuleHandle and GetProcAddress. The first one retrieves the system DLLs handle (kernel32, user32,...) which are ALREADY loaded in memory. The second one retrieves the API handle we wish and which is located in the DLL of which we just retrieve the handle. With these handles, we can then call all the APIs we wish!

In order to illustrate this, I allow me to quote an appropriate short extract from the marvellous tutorial of LaZaRuS:

The code for ”Start Notepad”:

```
:00000204 3D9C020000            cmp eax, 0000029C ;; is "Start Notepad" chosen?
:00000209 7525                  jne 00000230 ;; if not, then jump
:0000020B 68ACE64000            push 0040E6AC ;; push "KERNEL32.DLL"
:00000210 FF1590E24000          call dword ptr [0040E290] ;; "GetModuleHandle"
:00000216 68071B4100            push 00411B07 ;; "WinExec"
:0000021B 50                    push eax ;; handle of Kernel32.dll
:0000021C FF15DCE24000          call dword ptr [0040E2DC] ;; GetProcAddress
:00000222 6A01                  push 00000001 ;; SW_SHOW
:00000224 68EA174100            push 004117EA ;; "Notepad.exe"
:00000229 FFD0                  call eax ;; call WinExec
:0000022B E9B616FFFF            jmp FFFF18E6 ;; back to MessageLoop
```

If you do not understand the code, I send you back to his tutorial. I will not offence him and re-explain it. And if you want more details on the method, go to the source and see the tutorial of NeuRaL_NoiSE (in its Phase 5). It is excellently explained!

So we have the choice then?! Actually, not so much...

As I just have written it, we obligatorily have to have GetModuleHandle and GetProcAddress in the import table, otherwise it is really lost. And unfortunately, GetProcAddress is not to be found in the notepad’s import table. So we are now bound to definitely use CreateDialogParam. We will have to code the messages loop handling and to close the dialog box by using DestroyWindow.

This will be longer and harder as for DialogBoxParam :(But on the other hand, we’ll learn more! ;o)

Let’s see now the structure of the API CreateDialogParam:

```
HWND CreateDialogParam(
    HINSTANCE hInstance,// handle to application instance
    LPCTSTR lpTemplateName,// identifies dialog box template
    HWND hWndParent,      // handle to owner window
    DLGPROC lpDialogFunc,// pointer to dialog box procedure
    LPARAM dwInitParam         // initialisation value
   );
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

And we close this API with DestroyWindow. Its structure has only one parameter to push:

```
BOOL DestroyWindow(
    HWND hWnd  // handle to window to destroy
   );
```

As we do not have a big idea how to begin, we will have a look to the API CreateDialogParam which is located in the notepad under Wdasm/SI. Only one occurrence of the API in the program is to be found (display of the small DialogBox of the printer "now printing"), which gives/shows the five following parameters to be pushed:

```
:00404085 56                  push esi                      ; init. value = 0
:00404086 A100504000          mov eax, dword ptr [00405000]
:0040408B 68E13B4000          push 00403BE1                 ; pointer to procedure
                                                              = 8B 44 24 08
:00404090 50                  push eax                      ; handle to owner window
                                                              = CAC

* Possible Ref to Menu : MenuID_0001, Item: "Time/Date F5"  |
|                                                           |
* Possible Reference to Dialog: DialogID_000C               | Wdasm is being confused ;)
|                                                           |
* Possible Reference to StringResource ID=00012: " - Notepad"  |
|
:00404091 6A0C                push 0000000C                 ; dialog box template
                                                              = 0x0C (i.e. 12 under BRW)
:00404093 FF3540554000        push dword ptr [00405540]     ; handle to appli. inst.
                                                              = 00 00 40 00 (400000)

* Reference To : USER32.CreateDialogParamA, Ord: 0050h
|
:00404099 FF155C644000        Call dword ptr [0040645C]     ; API CreateDialogParamA
```

Explanations with the values I have chosen for my code :

```
initialisation value   = 0 ; we don't care. We put zero.
pointer to procedure   = 40D980 ; procedure of the code to execute in
                                  the displayed window(REGBOX).
handle to owner window = 8C ; how to find this value?? see below... ;)
dialog box template    = 680 ; 0x680 = 1664d. Got it?
handle to appli. inst. = 400000 ; it is generally the ImageBase (here = 400000).
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

For the "pointer to procedure", it is like a jump which will execute the code of the REGBOX window (messages handling, serial calculation,...). I have fixed this value after I have coded the call of the REGBOX. For the "handle to owner window", there is a very easy way to get this tricky value: run the software from which you will the handle. Under SI, enter "task" and you will see a lot of data, included the names of the running tasks. Choose in the list the name of the software you are looking the handle for (this name is not always the same as the software's one you are looking for and have run, that's the reason why we enter "task" first!). Then, enter "hwnd name_of_the_soft", and look the first column which contains the handles of the soft. The first value is the one you are looking for (0x8C in our case). It is tabbed regarding the rest of the column.

The others values shouldn't be a problem to you.

Let's go now to the practical part, that means coding the dialog box. We will start to code the API CreateDialogParam with its 5 parameters, and define a junk instruction (for the moment) for its procedure.

We continue the coding at the following place:

```
(ID-COMPARISON)
.0040D320: 60               pushad                          <-- registers' back up
.0040D321: 3D8E030000       cmp         eax,00000038E  <-- regbox chosen?
.0040D326: 0F8484000000     je         .00040D3B0      <-- yes, so its code
                                                           is executed
.0040D32C: 3D8F030000       cmp         eax,00000038F  <-- MSGBOX5 "About
                                                           Sharepad" has
                                                           been chosen?
.0040D331: 0F8439000000     je         .00040D370      <-- yes, so its
                                                           code is executed
.0040D337: 61               popad                      <-- back down of
                                                           all registers
.0040D338: 3D01030000       cmp         eax,000000301  <-- overwritten instruction
                                                           by our jump in 40128F
.0040D33D: E9523FFFFF       jmp        .000401294      <-- back to the
                                                           code just after
                                                           our wild jump
```

As we call now the REGBOX, we'll have the ID 38E and jump in D3B0. The API CreateDialogParam is directly placed at this address, followed by an instruction which saves the handle of the created dialog box (this handle is returned in eax after the creation of the dialog box). The offset [405390], used to save eax, has been arbitrariness chosen in the padding of the .data section, it was the first large padding I have met while descending the exe code under an hexeditor. Moreover, this section is C0000040 (the C meaning "read and write" in the section), so we have everything we need!

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Here is the code for the REGBOX:

```
.0040D3B0: 6A00                push        0000 |
.0040D3B2: 6880D94000          push        00040D3E0         |
.0040D3B7: 688C000000          push        00000008C         | Our 5 parameters of the API
                                                                 CreateDialogParam
.0040D3BC: 6880060000          push        000000680         |
.0040D3C1: 6800004000          push        000400000         |
.0040D3C6: FF155C644000        call        CreateDialogParamA   <-- API CreateDialogParam
                                                                    calling the REGBOX
.0040D3CC: A390534000          mov         [000405390],eax      <-- back up of the
                                                                    REGBOX's handle
.0040D3D1: E9373FFFFF          jmp        .0004018AD            <-- we exit/jump to our
                                                                    well known loop!
[...]
.0040D3E0: 33C0                xor         eax,eax
.0040D3E2: C21000              retn        00010
```

In the API CreateDialogParam, I have said that we had the procedure of the code to execute in the displayed REGBOX window. It has to be in 40D3E0, as defined in 40D3B2 by the push 40D3E0. This procedure corresponds to the 2 above instructions in 40D3E0 which are for the moment junk instructions, so that the program can run. We'll put here later the serial calculation routine code.

Under an hexeditor, we get:

```
0000D3A0 5245 4742 4F58 0000 0000 0000 0000 0000 REGBOX..........  <-- Title of the part.
                                                                        Does not act in the code.
0000D3B0 6A00 68E0 D340 0068 8C00 0000 6880 0600 j.h..@.h....h...  |Code1
0000D3C0 0068 0000 4000 FF15 5C64 4000 A390 5340 .h..@...\d@...S@  |Code1
0000D3D0 00E9 D744 FFFF 0000 0000 0000 0000 0000 ...D............  |Code1 + padding
0000D3E0 33C0 C210 0000 0000 0000 0000 0000 0000 3...............  |Code2
```

We can now run the program to test it and click on the registration menu. joie! Here is the REGBOX displayed :o)

Et voil!

Well, that's not so happily, because we can not close the REGBOX. But the most beautiful thing can only give what it has, don't you think?

Now, we have to manage to close the REGBOX by clicking on the "Cancel" button (ID=0x389 or 905d) or on the X cross of the window. Moreover, the "Validate" button has also to be activated. But for the moment, in order to close the program, just use any close command of the main notepad's window.

If we think on the same way as we did for the menus insertions we did at the beginning, we face here the same case. We have two buttons (Validate and Cancel) to which we have to bound some code (an action). We have a proprietary window of these two buttons, which is the REGBOX (ID=1664, Handle=? The Handle is always changing, we can find it somewhere in the stack). So we just have to make a "cmp eax, ID" loop to know which action has been done. This loop starts directly in D3E0 at the place of the XOR (the junk instruction) that we had written. Then, we will redirect the REGBOX to an exit (Cancel) or to the serial calculation (Validate).

Let's work!

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

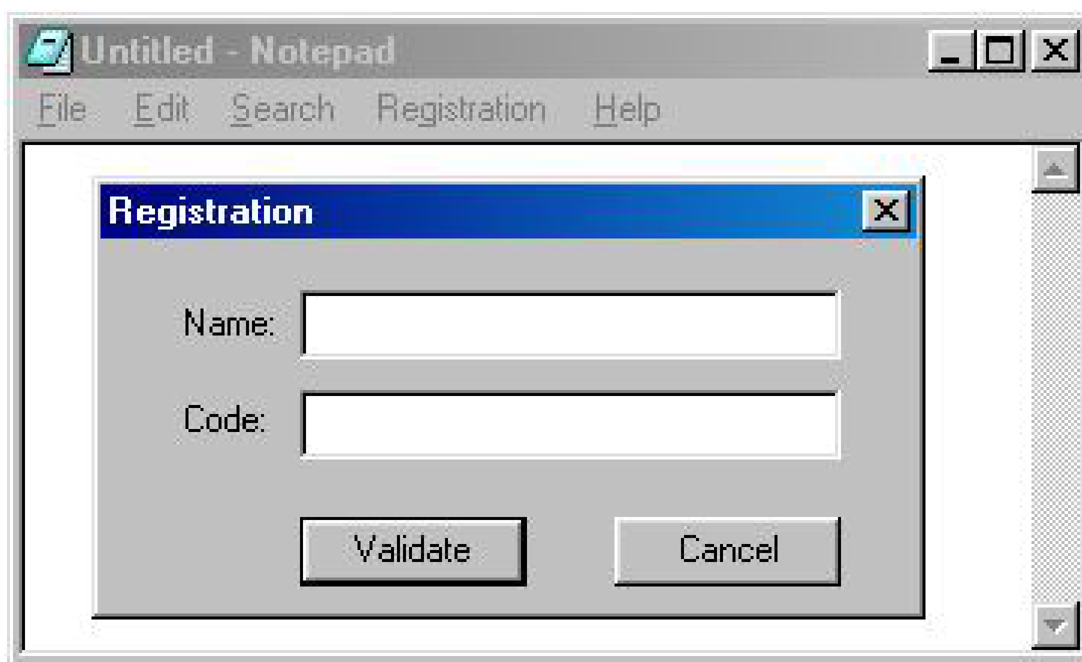

Fig. 7. Sharepad - Registration Box

As I said above in the analysis of the two APIs to display a dialog box (CreateDialogParam and DialogBoxParam), before we can manage the buttons' IDs, we have to manage the messages that the REGBOX sends to the windows OS (which sends them back to the notepad, i.e. our code).

Here is the code I propose (in D3E0 then):

```
* Under an hexeditor

0000D3E0 558B EC81 7D0C 1000 0000 750E FF75 08FF U...}.....u..u..
0000D3F0 15A0 6440 00E9 2D00 0000 817D 0C11 0100 ..d@..-....}....
0000D400 0075 248B 4510 3D89 0300 0075 0EFF 7508 .u$.E.=....u..u.
0000D410 FF15 A064 4000 E90C 0000 003D 8803 0000 ...d@......=....
0000D420 7505 E829 0000 00C9 C300 0000 0000 0000 u..)...........
0000D430 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D440 5345 5249 414C 2D43 414C 4390 0000 0000 SERIAL-CALC.....
0000D450 6A00 6830 D240 0068 30D2 4000 6A00 FF15 j.h0.@.h0.@.j...
0000D460 A864 4000 C9C3 0000 0000 0000 0000 0000 .d@.............
0000D470 0000 0000 0000 0000 0000 0000 0000 0000 ................
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
* And as asm listing (continuation of REGBOX)

.0040D3E0: 55                       push       ebp
.0040D3E1: 8BEC                     mov        ebp,esp
.0040D3E3: 817D0C10000000           cmp        [ebp+0C],000000010
.0040D3EA: 750E                     jne       .00040D3FA
.0040D3EC: FF7508                   push       [ebp+08]
.0040D3EF: FF15A0644000             call       DestroyWindow
.0040D3F5: E92D000000               jmp       .00040D427
.0040D3FA: 817D0C11010000           cmp        [ebp+0C],000000111
.0040D401: 7524                     jne       .00040D427
.0040D403: 8B4510                   mov        eax,[ebp+10]
.0040D406: 3D89030000               cmp        eax,000000389
.0040D40B: 750E                     jne       .00040D41B
.0040D40D: FF7508                   push       [ebp+08]
.0040D410: FF15A0644000             call       DestroyWindow
.0040D416: E90C000000               jmp       .00040D427
.0040D41B: 3D88030000               cmp        eax,000000388
.0040D420: 7505                     jne       .00040D427
.0040D422: E829000000               call      .00040D450
.0040D427: C9                       leave
.0040D428: C3                       retn

[...]

.0040D450: 6A00                     push       000
.0040D452: 6830D24000               push       00040D230
.0040D457: 6830D24000               push       00040D230
.0040D45C: 6A00                     push       000
.0040D45E: FF15A8644000             call       MessageBoxA
.0040D464: C9                       leave
.0040D465: C3                       retn
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Analysis of the code:

We will use the ebp register to work. So we back up it, then we assign it (copy to it) the esp value (from the stack, to handle windows OS messages). This is done in D3E0.

We have two kind of messages to handle. The first one is to check if the ebp+C value does correspond to 10h (windows events' handle: mouse cursor on the REGBOX?, use of the X cross to close the window...). The second is to check if it does correspond to 111h ("Validate" and "Cancel" buttons' handle).

In D3EA, if the message does not correspond to 10h, we jump in D3FA and we check if it does correspond to 111h. If it is not the case, we jump to the end of the routine and we loop. There is a huge quantity of messages which are sent and received by windows. Even when we do nothing with the computer. So in this bulk information, we set up a filter (the "cmp ebp+C, value") to catch what we are interested in.

In D3EC, the REGBOX will be closed if we have pressed the X cross. We go through the API DestroyWindow and we jump to the end of the code to the messages' loop.

Until now, we manage the messages' handle which was not provided with the API CreateDialogParam, as written above in the analysis/comparison of the 2 APIs to display the dialog boxes.

Now, we will manage the messages of the REGBOX buttons.

In D403, the ebp+10 word in eax is isolated. This operation equals to choose the lParam of the dword sent by windows. Thus, we directly have in eax the ID of the pressed button. Easy, clean and powerful! For the explanations on lParam and all the related things, go on the Iczelion homepage (win32asm.cjb.net - tutorials 10 and 11), here again, I will not re-explain what has already masterly done.

We compare now the sent ID with the actions to do.

In D406, we check if the ID sent by windows corresponds to our "Cancel" button (ID=389h). If it is not the case, we jump to the next ID. If it is the case, we close the REGBOX with the same code used for the X cross, then we jump to the end of the code.

En D41B, we compare the ID sent by our button "Validate" (ID=388h). If it is not the case, we jump to the end of the code, otherwise, the call in D422 is executed. This call goes in D450 and displays for the moment a (junk) msgbox which shows us that everything is working properly. This msgbox will be then replaced by the serial calculation. A short comment about the jne in D420: although that it is not useful because there are only two controls in the REGBOX, I have put it to be rigorous. Thus, only the "Validate" button will have access to the call of the serial calculation!

Until now, the REGBOX is finished... at least in the handling of its events. The "Validate" button sends back a msgbox with the same string as title and prompt. This part of the code (in D450) will be replaced next by the serial calculation routine. The "Cancel" button close the REGBOX with the help of the API DestroyWindow, as well as when the X cross is used to close the REGBOX.

We now have to code the serial calculation, then to manage the behaviour of the notepad according to registered/not registered.

I call now each of you to use his own experience in tracing a serial calculation under Softice. We will "copy" the schemes' structure of the easiest serial calculation routines when one begins in learning cracking. I will make here the same comment as I did for the part I of this tutorial. What is important here is HOW TO build a dialog box which will calculate a serial. It is not to have an acute SECURITY for the protection of the serial calculation!

Back to our subject. And let's start with the beginning:

Question: what are the 2 APIs on which a breakpoint is put we enter a fake name+serial under Softice??

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Answer: GetDlgItemText and GetWindowText! Hmemcpy is here not the subject as it leads in the system DLLs.

So we choose any one, and we check (under wdasm) if it is available in the notepad's import table. As the two APIs are present, we choose the one we want. Personally, I have a preference for GetDlgItemText. Here is their declaration for information purpose:

```
int GetWindowText(
    HWND hWnd,// handle of window or control with text
    LPTSTR lpString,// address of buffer for text
    int nMaxCount  // maximum number of characters to copy
   );

UINT GetDlgItemText(
    HWND hDlg,// handle of dialog box
    int nIDDlgItem,// identifier of control
    LPTSTR lpString,// address of buffer for text
    int nMaxCount  // maximum size of string
   );
```

Note that these 2 APIs send back the length of the input string in eax.

Well, the parameter's list is obvious. We will code GetDlgItemText in the place of the msgbox which displays us the same title and prompt. Actually, we proceed on the same way as for coding CreateDialogParam.

A few words on what's coming next... We will input the two edit fields of our REGBOX with the help of the API GetDlgItemText, then we will create the serial which will be compared with the user's serial. Then, we check if it corresponds with the help of a comparison test. If it does not match, a "Bad code" msgbox is displayed (we have already put the string which corresponds to this case at the beginning of this part II). Otherwise, we will create an entry in the system registry and write the name + serial, and deactivate the shareware restrictions (which have not been coded for the moment, this comes later!).

Once this part coded, we will build the shareware restrictions (the same as the ones of part I), and we will test at the launch of the notepad if the entries in the system registry are 1/present or not and 2/right. If it is the case, we'll jump to the code which kicks the shareware restrictions, otherwise we allow the notepad to start without changing something, leaving as it the shareware restrictions.

For the buffers' management, we will put them in the same section as the one we have used to save the REGBOX's handle (see above), either 53A0 or 53B0 or... Moreover, we will use the following areas: The variables/offsets used to save temporary data:

```
4053A0 : [name input by the user/the system registry] (on 0x20 bytes)
4053C0 : length of the [name] (on 1 byte)
4053D0 : [code/serial input by the user/the system registry] (on 0x10 bytes)
4053E0 : boolean/flag (on 1 byte)
4053F0 : [serial calculated by the sharepad]
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

I remember that the 2 EDIT fields have as ID 900d/0x384 (for the name) and 901d/0x385 (for the serial). Thus we start in D450 to code the input of the 2 EDIT fields with GetDlgItemText, and we add a small artfulness:

```
(SERIAL-CALC)
.0040D450: 6A20          push   020              <-- buffer max. length (32d)
.0040D452: 68A0534000    push   0004053A0        <-- memory offset of the input name
.0040D457: 6884030000    push   000000384        <-- ID of the field EDIT_name
.0040D45C: FF7508        push   [ebp+08]         <-- handle of the REGBOX
                                                     (in the stack)
.0040D45F: FF157C644000  call   GetDlgItemTextA  <-- we get the input name...
.0040D465: A3C0534000    mov    [0004053C0],eax  <-- ... its length is saved here
.0040D46A: 6A10          push   010              <-- buffer max. length (16d)
.0040D46C: 68D0534000    push   0004053D0        <-- memory offset of the
                                                     input serial
.0040D471: 6885030000    push   000000385        <-- ID of the field EDIT_serial
.0040D476: FF7508        push   [ebp+08]         <-- handle of the REGBOX
                                                     (in the stack)
.0040D479: FF157C644000  call   GetDlgItemTextA  <-- we get the input serial...
.0040D47F: C3            retn
```

...and we get the input name in 4053A0, and the input serial in 4053D0. Nothing really hard until now! The C3 is just there in order that the soft does not crash and to verify the offsets under Softice (by ”d 4053A0” and ”d 4053D0” in breaking with a bpx in 40D450). The ”mov [offset],eax” in D465 actually save the length of the input name in the [offset].

When we code this part, we really have to take care to two particular things. The first stacked instruction (in D450) is the buffer's length. When you put the buffer's offset in the second instruction (in D452), you have to take account of its length (in D450). Otherwise, there is a risk to overwrite some instructions/values which are below. Same thing when the serial's input is coded. We can not start anywhere after the name's buffer. We have to take account of the name's buffer size, otherwise it will result in a big mess in the handling of the data...

Afterward comes the most funny part of this tutorial: the creation of the serial. Well, here it's fully up to you to imagine everything! I have chosen to sum the double of the ascii values of the name's letters, to multiply this sum by a constant and to xor this value by another constant. Nothing less! Well, actually it is not really important how we do calculate the real serial (ours ;)), it is how we'll handle it which is important.

Well! We'll first code our official notepad serial creation (yeah! :D), then we'll convert the hexadecimal value of the serial to its decimal value with the help of the API wsprintf (which is in the notepad's IAT). Finally, we'll compare this value to the one input by the user in the REGBOX with the API lstrcmp (which is also in the notepad's IAT). A short test to know the result of this comparison will lead us to a bad boy or a good boy and displays the corresponding msgbox which strings are located in the sentences we have written at the beginning of this part II. We'll end the procedures as usually with the instructions leave/retn (0xC9/0xC3).

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

We start in D47F in the place of the C3 we have put (and that we trash now), and we code SERIAL-CALC:

```
.0040D47F: C605E053400000     mov    [0004053E0],000    <-- explanation comes later...
.0040D486: 33C0               xor    eax,eax            |the registers we need
.0040D488: 33D2               xor    edx,edx            |are reset
.0040D48A: 33DB               xor    ebx,ebx            |
.0040D48C: 8B0DC0534000       mov    ecx,[0004053C0]    <-- length of the name in ecx
.0040D492: 8A82A0534000       mov    al,[edx+0004053A0] <-- the name's letters
                                                            are put in eax
.0040D498: 8D1C43             lea    ebx,[ebx+eax*2]    <-- formula to calculate
                                                            the serial
.0040D49B: 42                 inc    edx                <-- next letter's turn
.0040D49C: 3BCA               cmp    ecx,edx            <-- we check if all
                                                            the letters
                                                            have been done
.0040D49E: 75F2               jne    .00040D492         <-- if not, the calculation
                                                            continues
.0040D4A0: 81C321430000       add    ebx,000004321      <-- otherwise we add
                                                            our constant...
.0040D4A6: 81F334120000       xor    ebx,000001234      <-- ...and xor the result
                                                            with another one
.0040D4AC: 53                 push   ebx                <-- the result is pushed
                                                            on the stack
.0040D4AD: 689C104000         push   00040109C          <-- see explanation below
.0040D4B2: 68F0534000         push   0004053F0          <-- result's offset
                                                            in decimal
.0040D4B7: FF150C644000       call   wsprintfA          <-- hexa/decimal conversion
.0040D4BD: 68D0534000         push   0004053D0          <-- input serial
.0040D4C2: 68F0534000         push   0004053F0          <-- calculated serial
.0040D4C7: FF15B8634000       call   lstrcmpA           <-- comparison
(we'll add something here later!!!)
.0040D4CD: 85C0               test   eax,eax            <-- are they the same??
.0040D4CF: 7416               je     .00040D4E7         <-- yes, then we jump to
                                                            good boy
.0040D4D1: 6A00               push   000                |
.0040D4D3: 6810D84000         push   00040D280          |Bad boy!
.0040D4D8: 6830D84000         push   00040D2A0          |msgbox display
.0040D4DD: 6A00               push   000                |"Bad Code"
.0040D4DF: FF15A8644000       call   MessageBoxA        |
.0040D4E5: C9                 leave
.0040D4E6: C3                 retn
.0040D4E7: 6A00               push   000                |
.0040D4E9: 68C0D74000         push   00040D230          |Good boy!
.0040D4EE: 68E0D74000         push   00040D250          |msgbox display
.0040D4F3: 6A00               push   000                |"Thank you for your support."
.0040D4F5: FF15A8644000       call   MessageBoxA        |
.0040D4FB: C9                 leave
.0040D4FC: C3                 retn
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Under a Hexeditor, we get:

```
0000D440 5345 5249 414C 2D43 414C 4390 0000 0000 SERIAL-CALC.....
0000D450 6A20 68A0 5340 0068 8403 0000 FF75 08FF j h.S@.h.....u..
0000D460 157C 6440 00A3 C053 4000 6A10 68D0 5340 .|d@...S@.j.h.S@
0000D470 0068 8503 0000 FF75 08FF 157C 6440 00C6 .h.....u...|d@..
0000D480 05E0 5340 0000 33C0 33D2 33DB 8B0D C053 ..S@..3.3.3....S
0000D490 4000 8A82 A053 4000 8D1C 4342 3BCA 75F2 @....S@...CB;.u.
0000D4A0 81C3 2143 0000 81F3 3412 0000 5368 9C10 ..!C....4...Sh..
0000D4B0 4000 68F0 5340 00FF 150C 6440 0068 D053 @.h.S@....d@.h.S
0000D4C0 4000 68F0 5340 00FF 15B8 6340 0085 C074 @.h.S@....c@...t
0000D4D0 166A 0068 80D2 4000 68A0 D240 006A 00FF .j.h..@.h..@.j..
0000D4E0 15A8 6440 00C9 C36A 0068 30D2 4000 6850 ..d@...j.h0.@.hP
0000D4F0 D240 006A 00FF 15A8 6440 00C9 C300 0000 .@.j....d@......
```

I will still enlighten 2-3 little things.

In 40D47F, there is a flag (a boolean) which will be us useful later. I do not explain it here.

In 40D4AD, there is a ”strange” push. Where does this offset come from? Well, in C/C++, when the hexa/decimal conversion with the API wsprintf is used, we write:

wsprintf(buffer_decimal, %d, buffer_hexa);

The %d being the parameter which means to wsprintf that we wish to convert in decimal. So we have to push this ”%d” in the API. And to do that, we must have it as a null terminated string in the ascii part under an hexeditor. I could have add it by hand, but I have first checked if it was not already in the code. Under an hexeditor, I have run a search for the %d and I have found only one occurrence in 40109C. You have to take care that the %d has to be a null terminated string (i.e. followed by 00 in the hexa part of the editor), otherwise by pushing %d on the stack, we would also push what's coming next, leading to a crash. It's only left to push the offset 40109C on the stack to push the parameter %d, what I did! :) Wsprintf sends then in 4053F0 the decimal value of the calculated serial back.

In 40D4C7, lstrcmp sends 0 in eax back if the 2 strings are identical. Otherwise eax is different from 0. The test at the next line checks eax, and the jump in 40D4CF acts consequently.

From now on, the REGBOX is right finished. We can input a name and a serial to register, and the REGBOX handles this serial to check if it does correspond or not!

In order that you can have fun, I have shipped a keygenerator with this tutorial. In my case, the name + serial are ”Anubis” and ”21969” :o) I have quickly coded this keygen in Pascal and without graphic user interface. It is a DOS program. If you have some troubles to run it (win2k,...), use the command prompt.

The bad boy part can be left as it. We will now develop the good boy part.

This part will consist on writing the correct name + serial in the system registry in a key that we'll have to create. Once this task done, we'll have to deactivate the shareware restrictions (in also deleting the registration menu!) and give the prompt to the notepad.

For the task on the system registry, we'll use all the notepad's ADVAPI32 APIs (see under Wdasm in the imports functions). As we will work on and with the system registry (sysreg), it is an evidence that we make a back up before doing anything a precisely blasting the sysreg out!

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Here are the structures of the APIs related to the sysreg:

```
LONG RegCloseKey(
    HKEY hKey  // handle of key to close
   );

LONG RegCreateKey(
    HKEY hKey,// handle of an open key
    LPCTSTR lpSubKey,// address of name of subkey to open
    PHKEY phkResult  // address of buffer for opened handle
   );

LONG RegSetValueEx(
    HKEY hKey,// handle of key to set value for
    LPCTSTR lpValueName,// address of value to set
    DWORD Reserved,// reserved
    DWORD dwType,// flag for value type
    CONST BYTE *lpData,// address of value data
    DWORD cbData  // size of value data
   );
```

We'll begin to put a jump in the place of the good boy, and to do a new GOODBOY part in which we'll code the writing process to the sysreg and then the good boy msgbox (which is taken off from the SERIAL-CALC part). This writing will be done in the path `HKEY\CURRENT_USER\Software\Microsoft\Notepad` with the following order:

```
RegCreateKey
RegSetValueEx (for the name)
RegSetValueEx (for the code)
RegCloseKey
```

Before starting to code, a short analysis is required!

By looking at these 3 APIs in the notepad's listing under wdasm, we notice that:

```
    - RegCreateKey is present only once
    - RegSetValueEx is present twice (through two different calls)
    - RegCloseKey is present only once
```

If we have a look to the listing between RegCreateKey and RegCloseKey, we have for instance this:

```
 * Possible StringData Ref from Data Obj ->"iPointSize"
 |
 :00402601 6820524000              push 00405220
 :00402606 FF75FC                  push [ebp-04]
 :00402609 E808FEFFFF              call 00402416
 :0040260E FF351C504000            push dword ptr [0040501C]

 [...]

 * Possible StringData Ref from Data Obj ->"fSavePageSettings"
 |
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
:00402627 6838524000              push 00405238
:0040262C FF75FC                  push [ebp-04]
:0040262F E8E2FDFFFF              call 00402416
:00402634 682C584000              push 0040582C

* Possible StringData Ref from Data Obj ->"lfFaceName"
|
:00402639 6850524000              push 00405250
:0040263E FF75FC                  push [ebp-04]
:00402641 E8ECFDFFFF              call 00402432
```

According to the kind of information which is to be input in the sysreg (string or binary data), we'll use the "call 00402416" (binary data) or "call 00402432" (string). In order to better compare, have a look to the directory `HKEY\CURRENT_USER\Software\Microsoft\Notepad` and check the difference between lfFaceName and fSavePage-Settings.

As we wish to input some text on the same way as lfFaceName does, we'll use the same code structure. We also notice that all the values are input to the sysreg on the same way: a function is called (the call) in pushing it 3 parameters. For instance, for lfFaceName is pushed:

```
- 0040582C = Value of the string (for me it is "Anubis")
- 00405250 = Name of the string (for me it is "Name")
- [ebp-04] = handle of the opened key
```

I am doing a short parenthesis. Until here, I have never used the sysreg in programming in any language. In order to be able to write the following code, I have traced under SI the above 3 APIs in the notepad by using the menu Edit¿Set font... which memorises the parameters in the sysreg. The "difficulty" was to well follow the different status of the stack in order to know to what does the [ebp+XX] correspond and to understand how are managed the calls. Thus, I do not get my information out of a black hat like a white rabbit, but through many tracing of these APIs. If my explanations seems insufficient to you, do the same. It's the best way to learn.! Put a bpx regcreatekeya (do not forget the "a" otherwise you land in the kernel instead of landing in advapi - and discard the ";" in front of advapi32 in the winice.dat file if it is not already done!), and each time when you trace with F10, look and write down the esp value to understand.

So after analysing the 3 APIs of the notepad under SI, I get the following code:

Caution, we have been get rid of the goodboy msgbox of the SERIAL-CALC part. Here is the new end of this part to compare with the previous version...:

```
(End of SERIAL-CALC)
.0040D4CF: 743F           je        .00040D510     ;we jump to GOODBOY
                                                    if the serial is ok, otherwise...
.0040D4D1: 6A00           push       000           |...we go to the badboy msgbox
.0040D4D3: 6810D84000     push       00040D280     |
.0040D4D8: 6830D84000     push       00040D2A0     |
.0040D4DD: 6A00           push       000           |
.0040D4DF: FF15A8644000   call       MessageBoxA   |
.0040D4E5: C9             leave
.0040D4E6: C3             retn
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

...in order to jump to this code snippet which is called GOODBOY:

```
(GOODBOY)
.0040D510: 55            push    ebp            ;backs ebp up
.0040D511: 8BEC          mov     ebp,esp        ;changes the variable
                                                for the stack
.0040D513: 83EC04        sub     esp,004        ;shift the stack for one position
.0040D516: 8D45FC        lea     eax,[ebp-04]   |RegCreateKeyA
.0040D519: 50            push    eax            |
.0040D51A: 6848514000    push    000405148      |
.0040D51F: 6801000080    push    080000001      |
.0040D524: FF15F0624000  call    RegCreateKeyA  |
.0040D52A: 85C0          test    eax,eax        ;if the key's creation fails, we jump...
.0040D52C: 7541          jne    .00040D56F      ;...here
.0040D52E: 68A0534000    push    0004053A0      |RegSetValueEx (for the name)
.0040D533: 6856524000    push    000405256      |
.0040D538: FF75FC        push    d,[ebp-04]     |
.0040D53B: E8F24EFFFF    call   .000402432      |
.0040D540: 68D0534000    push    0004053D0      |RegSetValueEx (for the code)
.0040D545: 68A4D24000    push    00040D2A4      |
.0040D54A: FF75FC        push    d,[ebp-04]     |
.0040D54D: E8E04EFFFF    call   .000402432      |
.0040D552: FF75FC        push    d,[ebp-04]     |RegCloseKey
.0040D555: FF15E8624000  call    RegCloseKey    |
.0040D55B: 6A00          push    000            |Msgbox goodboy
.0040D55D: 6830D24000    push    00040D230      |
.0040D562: 6850D24000    push    00040D250      |
.0040D567: 6A00          push    000            |
.0040D569: FF15A8644000  call    MessageBoxA    |
.0040D56F: 8BE5          mov     esp,ebp        ;change the original variable back
.0040D571: 5D            pop     ebp            ;pop ebp on the stack
.0040D572: C9            leave
.0040D573: C3            retn
```

We start with a very common variable change: we use ebp instead of esp for the stack variable. For this aim, we have to back up the ebp value at the beginning, and to back it down at the end of the procedure. In order that the stack data are coherent with the written [ebp+XX], we have to re-set the stack. This is the aim of the "sub esp,04" in D513.

Then, we can open the sysreg to write with RegCreateKeyA. Three parameters are pushed to the API. I could not identify the meaning of eax, in my code it is the serial, but as it work it does not matter. I have just copied this line code from the dead listing . The "push 405148" corresponds to the name of the key to open (`HKCU\Software\Mircosoft\Notepad`), and the "push 80000001" is the handle of the opened key. That's for the opening!

We verify if the opening has been properly done by testing eax which should be null, otherwise we jump at the end of the code. This avoids the risk to misuse the sysreg...

We go forth with the writing of the name. Here are also 3 parameters pushed in the following order: "Anubis" (in 4053A0, which comes from our GetDlgItemTextA), "Name" (in 405256) and of the key's handle. As I have no "Name" string at my disposal, and rather to add it, I have looked for one available. By looking for the word "Name", we find only one occurrence in 405256 which is actually:

```
00005250 6C66 4661 6365 4E61 6D65 0000 0000 0000 lfFaceName......
```

But if we point on the "N" of Name (405256), no problem to recover the suitable part! Reverse rulez ;o)

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

We then have the call in 402432 after we have pushed the handle of the opened key. This call will first calculate the length of the parameter push as value (here ”Anubis”) and set the API RegSetValueEx.

Same trick for the serial. We push ”21969” (stored in memory in 4053D0), then we push the word Code that I have taken from my string ”Bad Code” here:

```
0000D2A0 4261 6420 436F 6465 0000 0000 0000 0000 Bad Code........
```

Here we point to the ”C” of Code (40D2A4), and it’s done!

In 40D54D, we have the second call which writes the serial in the sysreg.

Then, the handle of the key is pushed, and the sysreg is closed. There’s only left to display the good boy msgbox, to reset the original variables and voila!

Our sharepad allows until here to register and writes the name + right serial in the sysreg. Here is an extract of my sysreg after I registered:

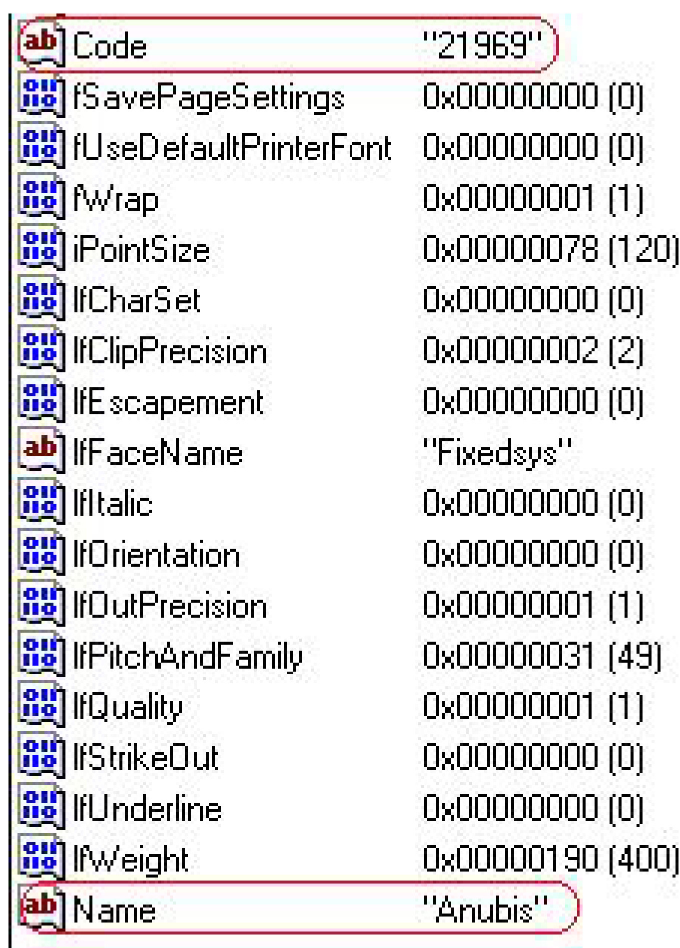

Fig. 8. Sharepad - Sysreg-Extraction

Well, this begins to be a little presentable :o)

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

We have finished the biggest part. Before going to the shareware restrictions, we will code an analogue part to the one we just coded. When we start the sharepad, this one must verify if it is already (properly) registered or not. For that purpose, we have to deviate the software at the PEP, go and read the information in the sysreg (if there is no information, we jump to the sharepad version), check this information in sending the name to the serial calculation routine and in comparing the result to the serial of the sysreg (if the result is not correct, we jump to the sharepad version). And if all of this is correct, then we can jump to the DISACTIV part which unlocks the sharepad in notepad. Let's work!

This part will be called TEST-REG and will be followed by DISACTIV after going through SERIAL-CALC. We still have to code TEST-REG and DISACTIV, but also do some modifications in SERIAL-CALC (see below).

To read in the sysreg, we'll use the following API suite:

```
RegOpenKey
RegQueryValueEx (for the name)
RegQueryValueEx (for the serial)
RegCloseKey
```

The APIs structure is:

```
LONG RegOpenKey(
    HKEY hKey,// handle of open key
    LPCTSTR lpSubKey,// address of name of subkey to open
    PHKEY phkResult  // address of handle of open key
   );

LONG RegQueryValueEx(
    HKEY hKey,// handle of key to query
    LPTSTR lpValueName,// address of name of value to query
    LPDWORD lpReserved,// reserved
    LPDWORD lpType,// address of buffer for value type
    LPBYTE lpData,// address of data buffer
    LPDWORD lpcbData  // address of data buffer size
   );
```

As usually, we check under Wdasm how these APIs are used, and we draw ourselves inspiration from it like poets :o) Here is the one for RegOpenKeyA (with the change of registers at the beginning)...:

```
:00402693 55                    push ebp
:00402694 8BEC                  mov ebp, esp
:00402696 83EC40                sub esp, 00000040
[...]
:004026AF 8D4DFC                lea ecx, dword ptr [ebp-04]
:004026B2 51                    push ecx

* Possible StringData Ref from Data Obj ->"Software\Microsoft\Notepad"
|
:004026B3 6848514000            push 00405148
:004026B8 6801000080            push 80000001

* Reference To: ADVAPI32.RegOpenKeyA, Ord:0094h
|
:004026BD FF15EC624000          Call dword ptr [004062EC]
:004026C3 85C0                  test eax, eax
:004026C5 7407                  je 004026CE
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

And so on for RegQueryValueEx...:

```
:004027C8 6A20                    push 00000020
:004027CA 8D4DDC                  lea ecx, dword ptr [ebp-24]
:004027CD 682C584000              push 0040582C
:004027D2 A22B584000              mov byte ptr [0040582B], al
:004027D7 51                      push ecx

* Possible StringData Ref from Data Obj ->"lfFaceName"
|
:004027D8 6850524000              push 00405250
:004027DD FF75FC                  push [ebp-04]
:004027E0 E8C5FCFFFF              call 004024AA
:004027E5 6A78                    push 00000078
```

...which call 004024AA sends back to the procedure. We are here in the same logical pathway for the sysreg reading as for the writing. Our code will thus be pretty closed to the one we did to write in the sysreg (do not forget to change the registers back at the end):

```
(TEST-REG)
.0040D5A0: 55                push        ebp           ;we directly come from the PEP and we...
.0040D5A1: 8BEC              mov         ebp,esp       ;...change the stack variable...
.0040D5A3: 83EC40            sub         esp,040       ;...which is recalibrated
.0040D5A6: 8D4DFC            lea         ecx,[ebp-04]  |RegOpenKeyA
.0040D5A9: 51                push        ecx           |
.0040D5AA: 6848514000        push        000405148     |
.0040D5AF: 6801000080        push        080000001     |
.0040D5B4: FF15EC624000      call        RegOpenKeyA   |
.0040D5BA: 85C0              test        eax,eax       ;if the opening fails...
.0040D5BC: 7539              jne        .00040D5F7     ;...we jump to the end
.0040D5BE: 6A20              push        020           |Reading of the name ("Anubis") in the sysreg
.0040D5C0: 68A0534000        push        0004053A0     |
.0040D5C5: 8D4DDC            lea         ecx,[ebp-24]  |
.0040D5C8: 51                push        ecx           |
.0040D5C9: 6856524000        push        000405256     |
.0040D5CE: FF75FC            push        d,[ebp-04]    |
.0040D5D1: E8D44EFFFF        call       .0004024AA     |
.0040D5D6: 6A10              push        010           |Reading of the serial ("21969") in the sysreg
.0040D5D8: 68D0534000        push        0004053D0     |
.0040D5DD: 8D4DDC            lea         ecx,[ebp-24]  |
.0040D5E0: 51                push        ecx           |
.0040D5E1: 68A4D24000        push        00040D2A4     |
.0040D5E6: FF75FC            push        d,[ebp-04]    |
.0040D5E9: E8BC4EFFFF        call       .0004024AA     |
.0040D5EE: FF75FC            push        d,[ebp-04]    |RegCloseKey
.0040D5F1: FF15E8624000      call        RegCloseKey   |
.0040D5F7: 8BE5              mov         esp,ebp       ;registers are set back...
.0040D5F9: 5D                pop         ebp           ;...the ebp is popped
.0040D5FA: 55                push        ebp           |Here, we put the instructions
.0040D5FB: 8BEC              mov         ebp,esp       |overwritten by the jump
.0040D5FD: 83EC44            sub         esp,044       |at the PEP...
.0040D600: E9CD3AFFFF        jmp        .0004010D2     ;...and we jump just after our wild jump of the PEP
```

At the PEP, the deviation looks like this:

```
.004010CC: E96FCA0000        jmp        .00040DB40       ;our jump
.004010D1: 90                nop                         ;we nop the remaining uncompleted instructions
.004010D2: 56                push        esi             ;the above procedure returns here
.004010D3: FF15E0634000      call        GetCommandLineA
```

Small analysis of the code to read the sysreg:

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

From D5A0 to D5A3, I have simply copied the Wdasm listing to keep ”the behaviour” of the software. Same thing for RegOpenKeyA, as I want to read at the same place (in the same key) like the notepad.

Then comes the ”test eax,eax” which checks the proper opening of the key (if eax=0, I remember that the API returns 0 in eax if the things happened properly. Otherwise, there is an error number different from 0). I have on the other hand transformed the jump in D5BC. In the Wdasm listing, there is an ununderstandable ”je” which I turn into a ”jne”, as I did in for the writing of the sysreg.

Then we read the value of the name (Name). We push 0x20 as maximum length buffer, as we did for the writing. We push the offset of the buffer which receives ”Anubis” in 4053A0. After that, I have not understand the [ebp-24], so I have recopied it! We push then the name of the key to read (405256, i.e. Name), followed by the handle of the opened key and the call to the RegQueryValueEx Api in 4024AA.

Same thing for the reading of the serial where we push the buffer’s size (0x10), its offset (4053D0), the name of the key (Code i.e. 40D2A4).

The buffers’ offsets receiving the name and the serial are the same used for the writing in the sysreg and for the reading of the 2 fields of the REGBOX.

The end of the procedure is the same as for the reading of the sysreg: we change again the registers, then we add the code overwritten by our wild jump from the PEP and we jump to the instruction which follows this jump at the PEP. This jump at the PEP is temporary, and is used only to verify that our code is working. In deed, we’ll have to verify that the serial read in the sysreg is the right one. We’ll have to return to the serial calculation for that purpose.

A short tracing under SI of this code part (with a bpx RegOpenKeyA) will show us that the RegSomethings return all 0 in eax, confirmed by a d 4053A0 (to display ”Anubis”) and a d 4053D0 (to display ”21969”). It works!

Once the information retrieved form the sysreg, it is sent back to the serial calculation routine and we act consequently. But, as we go through this routine when we come from the REGBOX, we do not have to land in a big mess and switch correctly on the software according where we come from. To clarify the things, here is a scheme:

I must admit that I had not think at the beginning that it would be a so huge work, otherwise I would not have probably begun ;o) But with this road map, we will quietly continue to code, and you’ll see that it is not so hard as it seems!

We’ll use a boolean or a flag (up to you :)). When it will be equals to 1, we’ll come from the sysreg, and when it will be equal to 0 we’ll come from the REGBOX. Thus, with the help of some tiny tests and jumps well coded, we’ll make the program doing what we want from him. Now you can understand the utility of...:

```
.0040D47F: C605E053400000     mov         byte ptr [0004053E0],00 <-- explaination later...
```

...which is in the serial calculation routine (far away above). Our flag is in 4053E0, in the state 01 (sysreg) or 00 (REGBOX).

Let’s continue! After exiting the reading process of the sysreg, we have ”Anubis” in 4053A0 which we returns at the beginning of the serial calculation routine in 40D486 (after having changed our flag 4053E0 to 1). We continue the end of the sysreg reading code:

```
(End of TEST-REG)
.0040D5EE: FF75FC          push       [ebp-04]         |RegCloseKey
.0040D5F1: FF15E8624000    call       RegCloseKey      |
.0040D5F7: 8BE5            mov        esp,ebp          ;change of the registers...
.0040D5F9: 5D              pop        ebp              ;...ebp is restored
.0040D5FA: C605E053400001  mov        [0004053E0],001  ;here is our sysreg flag!
.0040D601: E980FEFFFF      jmp       .00040D486        ;we jump to the serial calculation
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

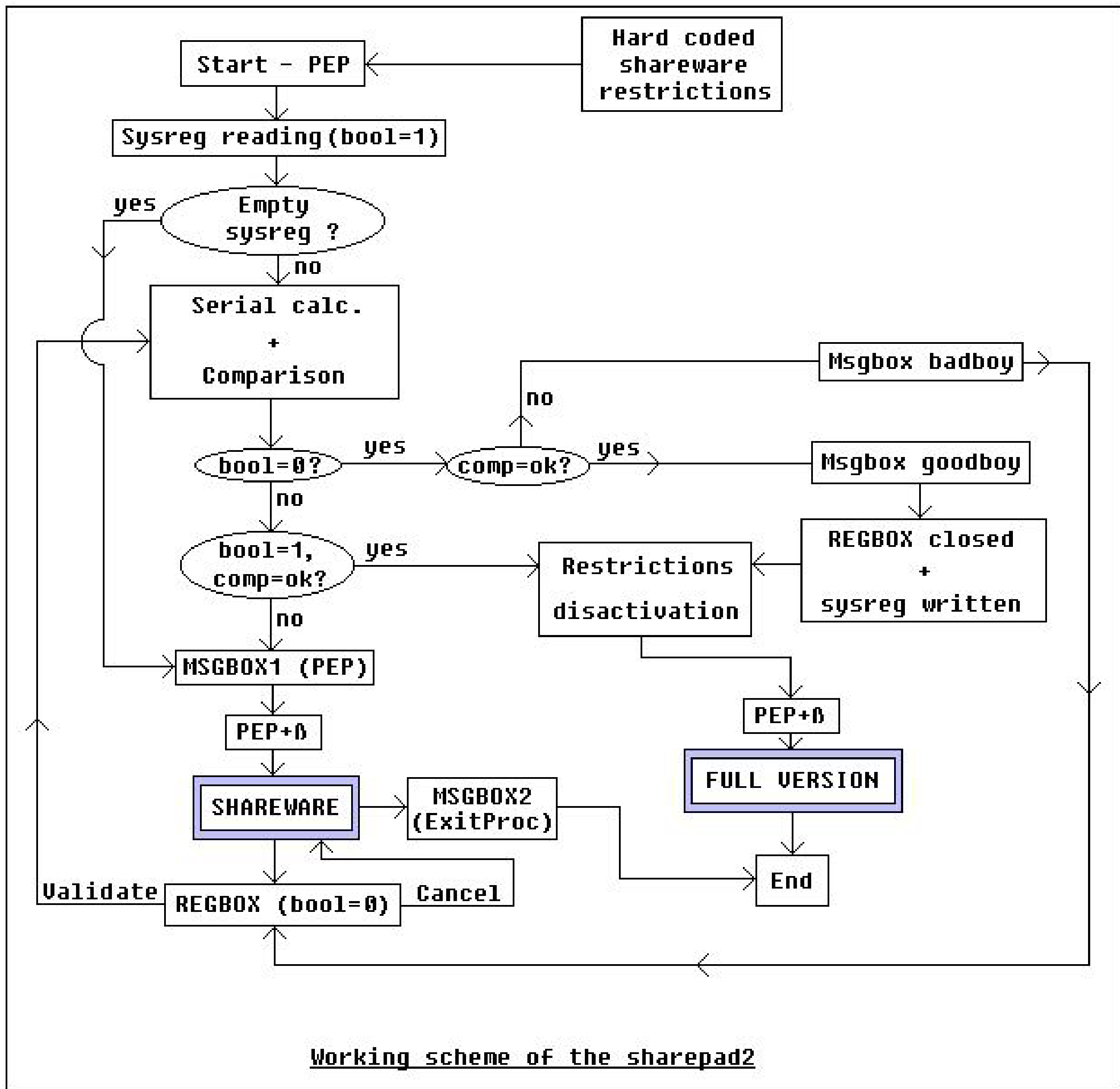

Fig. 9. Sharepad - Working Scheme of Sharepad2

There's here an important point to not forget! By deviating this code to the serial calculation, we arrive with the name and the serial in 2 different buffers. AND THAT'S ALL! You'll say me "yes, and so what??". Well, if we do not specify the length of the name buffer, the serial calculation will be done with an infinite loop, and the programme will crash at the beginning... So we have to use an API lstrlen just after having read the name in the sysreg. We'll then push the length in the destined length buffer which is [4053C0].

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

The modified part of TEST-REG looks like this now:

```
(End of TEST-REG)
.0040D5D1: E8D44EFFFF          call       .0004024AA        ;returns the name "Anubis" from the sysreg
.0040D5D6: 68A0534000          push        0004053A0        |returns the length of this name
.0040D5DB: FF15B0634000        call        lstrlenA         |in eax
.0040D5E1: A3C0534000          mov         [0004053C0],eax  ;we put eax in the buffer for the serial routine
.0040D5E6: 6A10                push        010              |begin of the API which returns the serial of the sysreg
.0040D5E8: 68D0534000          push        0004053D0
.0040D5ED: 8B4DDC              mov         ecx,[ebp-24]
.0040D5F0: 51                  push        ecx
.0040D5F1: 6838D84000          push        00040D838
.0040D5F6: FF75FC              push        [ebp-04]
.0040D5F9: E8AC4EFFFF          call       .0004024AA
.0040D5FE: FF75FC              push        [ebp-04]
.0040D601: FF15E8624000        call        RegCloseKey
.0040D607: 8BE5                mov         esp,ebp
.0040D609: 5D                  pop         ebp
.0040D60A: C605E053400001      mov         [0004053E0],001
.0040D611: E970FEFFFF          jmp        .00040D486
```

Here is the result under an hexeditor:

```
0000D590 5445 5354 2D52 4547 0000 0000 0000 0000 TEST-REG........
0000D5A0 558B EC83 EC40 8D4D FC51 6848 5140 0068 U....@.M.QhHQ@.h
0000D5B0 0100 0080 FF15 EC62 4000 85C0 7539 6A20 .......b@...u9j
0000D5C0 68A0 5340 008D 4DDC 5168 5652 4000 FF75 h.S@..M.QhVR@..u
0000D5D0 FCE8 D44E FFFF 68A0 5340 00FF 15B0 6340 ...N..h.S@....c@
0000D5E0 00A3 C053 4000 6A10 68D0 5340 008B 4DDC ...S@.j.h.S@..M.
0000D5F0 5168 38D8 4000 FF75 FCE8 AC4E FFFF FF75 Qh8.@..u...N...u
0000D600 FCFF 15E8 6240 008B E55D C605 E053 4000 ....b@...]...S@.
0000D610 01E9 70FE FFFF 0000 0000 0000 0000 0000 ..p.............
0000D620 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Now, we have to set up the switches :)

There is a modification to do in the SERIAL-CALC part we have already coded, and a switch to put at the beginning of the DISACTIV part.

In order to get a better overview, I put again the road map of the sharepad with the suited glasses to display you the different parts ;o)

So in SERIAL-CALC, we have to check the value of the boolean/flag in [4053E0] and act (jump) consequently. We'll add 2 instructions just after the "call lstrcmpa":

```
.0040D4C7: FF15B8634000              call        lstrcmpA
.0040D4CD: 803DE053400000            cmp         [0004053E0],000  ;is our flag at 0?
.0040D4D4: 0F85E6010000              jne        .00040D6C0        ;no, so we come from
                                                                  ;the sysreg and go to DISACTIV
.0040D4DA: 85C0                      test        eax,eax          ;otherwise we continue as usually to...
.0040D4DC: 7432                      je         .00040D510        ;...GOODBOY or...
.0040D4DE: 6A00                      push        000
.0040D4E0: 6810D84000                push        00040D810
.0040D4E5: 6830D84000                push        00040D830
.0040D4EA: 6A00                      push        000
.0040D4EC: FF15A8644000              call        MessageBoxA      ;...badboy msgbox
.0040D4F2: C9                        leave
.0040D4F3: C3                        retn
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

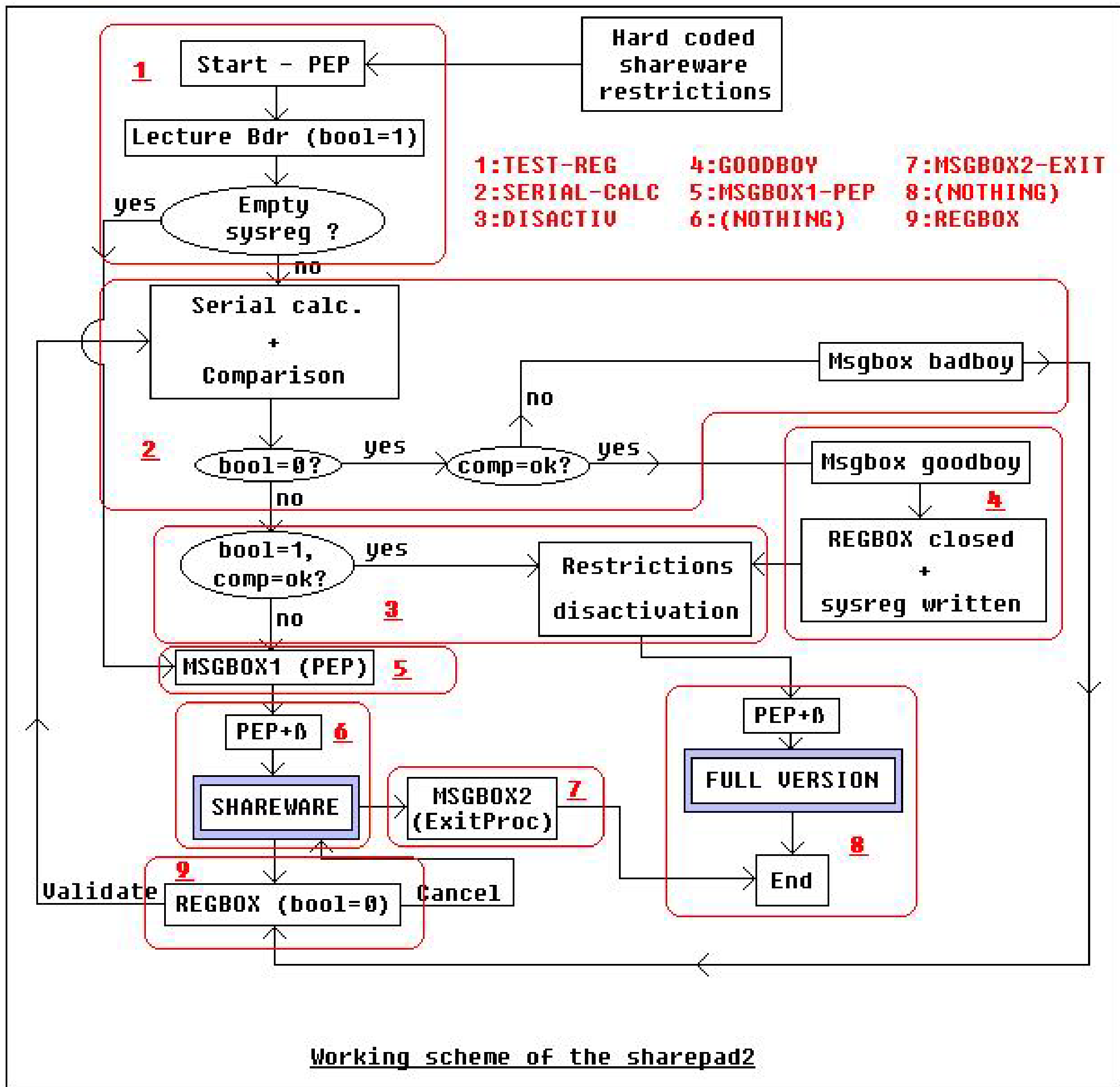

Fig. 10. Sharepad - Working Scheme of Sharepad2 Part II

Then, always in order that the program works when we arrive in DISACTIV where there is still nothing, we put a jump in 40DC60 to the PEP.

```
.0040D6C0: E90D3AFFFF                    jmp        .0004010D2
```

Voil! You can test the program, it won't completely work according our map because the DISACTIV part remains to be done, but it is always a pleasure to see the result of all this work until here.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

Before writing the DISACTIV part, I have chosen to write MSGBOX1-PEP and MSGBOX2-EXIT. As I don't know how much place will take DISACTIV, I have preferred to get rid of these 2 msgboxes which are very small. On the same hand, to keep the "esthetical" aspect of the different parts, the 2 MSGBOX-PEP and -EXITPROC parts are shifted of one line to the bottom after TEST-REG (in the hexeditor) and their jump is recalibrated by substrating 0x10 to the 2nd byte of the instruction (I hope that everybody follows me :)).

```
* MSGBOX1-PEP
.0040D640: 6A00                     push       000         |Msgbox1 at the PEP
.0040D642: 68A0D14000               push       00040D1A0   |
.0040D647: 68C0D14000               push       00040D1C0   |
.0040D64C: 6A00                     push       000         |
.0040D64E: FF15A8644000             call       MessageBoxA |
.0040D654: 55                       push       ebp         ;overwritten instructions
.0040D655: 8BEC                     mov        ebp,esp     ;by the jump at the PEP
.0040D657: 83EC44                   sub        esp,044     ;
.0040D65A: E9733AFFFF               jmp       .0004010D2   ;we land just after our wild jump of the PEP
```

MSGBOX1-PEP directly begins to display the msgbox, and finishes in returning to the PEP once the overwritten instructions have been executed. We'll land here from the DESACTIV part (which is below in this tutorial).

```
* MSGBOX2-EXIT
.0040D680: 6A00                     push       000          |Msgbox2 at the EXITPROCESS
.0040D682: 68A0D14000               push       00040D1A0    |
.0040D687: 68E0D14000               push       00040D1E0    |
.0040D68C: 6A00                     push       000          |
.0040D68E: FF15A8644000             call       MessageBoxA  ;
.0040D694: FF1598634000             call       ExitProcess  ;
.0040D69A: E9AA3AFFFF               jmp       .000401149    ;we land just after
                                                             our wild jump of the EXITPROCESS
```

For MSGBOX2-EXIT, it's exactly the same thing as for the sharepad1.

Here is the result under an hexeditor:

```
0000D630 4D53 4742 4F58 312D 5045 5090 0000 0000 MSGBOX1-PEP.....
0000D640 6A00 68A0 D140 0068 C0D1 4000 6A00 FF15 j.h..@.h..@.j...
0000D650 A864 4000 558B EC83 EC44 E973 3AFF FF00 .d@.U....D.s:...
0000D660 0000 0000 0000 0000 0000 0000 0000 0000 ................
0000D670 4D53 4742 4F58 322D 4558 4954 0000 0000 MSGBOX2-EXIT....
0000D680 6A00 68A0 D140 0068 E0D1 4000 6A00 FF15 j.h..@.h..@.j...
0000D690 A864 4000 FF15 9863 4000 E9AA 3AFF FF00 .d@....c@...:...
0000D6A0 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

Don't forget to put a wild jump at the EXITPROCESS of the notepad in 401143 which leads to MSGBOX2-EXIT (see part I of this tutorial if you do not remember how it works...).

We also can make the others restrictions here. So here, it's not very complicated, we'll apply the same shareware restrictions as those of the part I of this tutorial. I write them here again:
- Menus "Save" & "Save as..." are greyed
- The word "Shareware" is displayed in the title bar
- A msgbox warns us that the notepad is starting in shareware modus
- Same thing at the exit of the notepad
- Menu "Register" calls our REGBOX (already done)

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

*** The menus are greyed:**

We make a back up of the notepad for safety reasons (the one you are working on now!), and we open the software in a resources editor. We grey AND deactivate the 2 menus to save and we save/close the notepad.

We compare then the 2 files before/after the modifications with the DOS "fc" command:

```
Comparison of the files N.exe and N2.exe
0000C556: 00 03
0000C566: 00 03
```

Missa dicta est! We patch these 2 offsets to get the modification in hard in the exe file.

*** Shareware in the title bar:**

We have to find the right occurrence of " - Notepad" in the hexeditor, and to replace it with "-SHAREWARE" (character for character, including the spaces!). We find it at the line 86D0. This line might be different for you depending which kind of resources editor you have used at the beginning. But that's not so important, with the find option you'll find the right one. Here is the result before and after modifications:

```
000086C0 7300 3F00 0800 5500 6E00 7400 6900 7400 s.?...U.n.t.i.t.
000086D0 6C00 6500 6400 0A00 2000 2D00 2000 4E00 l.e.d... .-. .N.
000086E0 6F00 7400 6500 7000 6100 6400 0000 0000 o.t.e.p.a.d.....

000086C0 7300 3F00 0800 5500 6E00 7400 6900 7400 s.?...U.n.t.i.t.
000086D0 6C00 6500 6400 0A00 2D00 5300 4800 4100 l.e.d...-.S.H.A.
000086E0 5200 4500 5700 4100 5200 4500 0000 0000 R.E.W.A.R.E.....
```

Now we'll deactivate the shareware protections which we have just coded. Because, we are notepad's officially REGISTRED users!!! I have the feeling that some persons will be happy in redmond... ;o) All the below modifications have already been explained in the part I of this tutorial.

*** The MSGBOX-PEP and -EXITPROC:**

We'll patch the 1st byte of their call with 90 and they will be deactivated. I write them here again for memory.

```
.0040D64E: FF15A8644000                call        MessageBoxA
.0040D68E: FF15A8644000                call        MessageBoxA
```

*** SHAREWARE in the title bar:**

We'll replace it by the word "Sharepad".

*** Activation of the 2 menus greyed:**

Inverse patch of the hard coded one, so 03-¿00 at the offsets C556 and C566.

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

*** Deleting of the REGBOX menu:**

Same technique as for the 2 msgboxes. We'll patch by 00 the 1st letter of the menu "Register" so "R" which is at the offset C744.

```
0000C730 2600 4E00 6500 7800 7400 0900 4600 3300 &.N.e.x.t...F.3.
0000C740 0000 1000 5200 6500 2600 6700 6900 7300 ....R.e.&.g.i.s.
0000C750 7400 7200 6100 7400 6900 6F00 6E00 0000 t.r.a.t.i.o.n...
0000C760 0000 8E03 5200 6500 6700 2600 6900 7300 ....R.e.g.&.i.s.
```

Finally, we get for DISACTIV:

```
.0040D6C0: 85C0              test    eax,eax              ;the input serial is correct??
.0040D6C2: 0F8578FFFFFF      jne    .00040D640            ;no, so we jump to the MSGBOX1-PEP, otherwise
.0040D6C8: C6054ED6400090    mov     [00040D64E],090      ;we patch MSGBOX1-PEP
.0040D6CF: C6058ED6400090    mov     [00040D68E],090      ;we patch MSGBOX2-EXITPROC
.0040D6D6: C60556C5400000    mov     [00040C556],000      ;the 1st greyed menu is reactivated
.0040D6DD: C60566C5400000    mov     [00040C566],000      ;the 2nd greyed menu is reactivated
.0040D6E4: C605DC86400068    mov     [0004086DC],068 ;"h" |we patch (S)"HAREWARE" by (S)"harepad "
.0040D6EB: C605DE86400061    mov     [0004086DE],061 ;"a" |
.0040D6F2: C605E086400072    mov     [0004086E0],072 ;"r" |
.0040D6F9: C605E286400065    mov     [0004086E2],065 ;"e" |
.0040D700: C605E486400070    mov     [0004086E4],070 ;"p" |
.0040D707: C605E686400061    mov     [0004086E6],061 ;"a" |
.0040D70E: C605E886400064    mov     [0004086E8],064 ;"d" |
.0040D715: C605EA86400020    mov     [0004086EA],020 ;" " |
.0040D71C: C60544C7400000    mov     [00040C744],000      ;we patch the letter "R" of "R&egister"
.0040D723: 55                push    ebp                  |
.0040D724: 8BEC              mov     ebp,esp              |overwritten instructions by the
                                                           jump at the PEP
.0040D726: 83EC44            sub     esp,044              |
.0040D729: E9A439FFFF        jmp    .0004010D2            ;we land after our wild jump at the PEP
```

Of course, as we write in memory in the sections (.rsrc actually), we do not have to forget to change their characteristics in 0xC00000040 (read + write) otherwise the computer crashes! This means to change the byte 0x23F (in the PE header) which is at 40 in C0.

Voil! Voil! Three times voil! We have finished the alpha phase. The sharepad is operational. Now we'll look for the small bugs, which corresponds to the beta phase... I make my tests with the sysreg having a serial entry and a name entry. It doesn't care if the serial does match or not, but there is a case I do not know for the moment: the one when the sysreg is empty when the software starts (reading of a key which doesn't exist). As I expect a crash at this step, I will run this test at the end.

Well, by tracing under SI all possible ways of the general scheme, we notice some jumps which are shifted of 0x10 bytes in their code line (especially the one from GOODBOY to DESACTIV). This comes from different moving of the sections which we have done by adding the switches. More over, when we register in the REGBOX with the right serial, this one is not destroyed by the API DestroyWindow (hehe, we should send this API to redmond!). On the other hand, the writing of the good information in the sysreg is done properly, and this even if wrong/fake information is already there.

Here is the end of GOODBOY:

```
.0040D562: 6850D24000        push    00040D250
.0040D567: 6A00              push    000
.0040D569: FF15A8644000      call    MessageBoxA
.0040D56F: FF7508            push    [ebp+08]
.0040D572: FF15A0644000      call    DestroyWindow
.0040D578: 8BE5              mov     esp,ebp
.0040D57A: 5D                pop     ebp
.0040D57B: E9B0000000        jmp    .00040D630
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

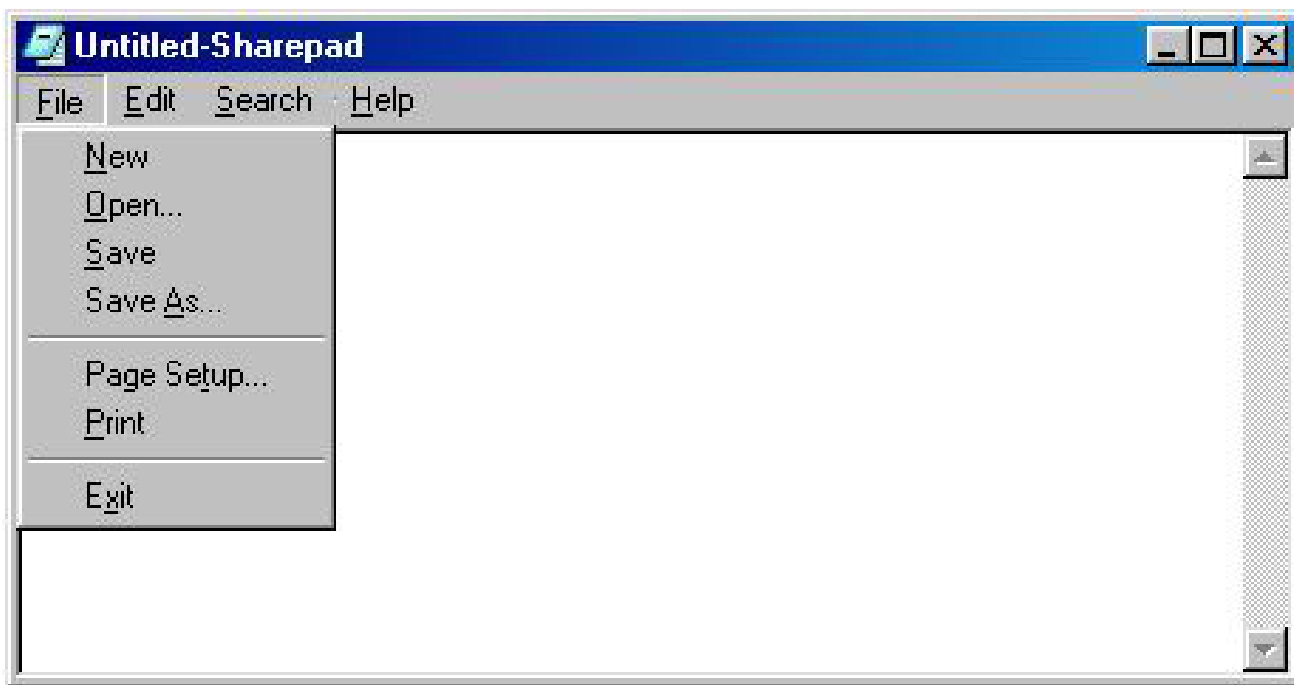

Fig. 11. Sharepad - Full version of Sharepad2

D630 is the (foreseen) previous place for the beginning of DISACTIV. Now, there is MSGBOX1 and 2. DISACTIV starts in D6C0 with the ”test eax,eax” which we have done just before. So we can directly jump after, that means in D6C8.

Then we notice that once the deactivation are done, the software returns to the PEP and exits alone. If we start it again, it is in full version. To palliate to this attitude (my idea was that the software would patch itself in real time in memory...), we'll add a short sentence in the goodboy messagebox which becomes ”Thank you for you support. Restart the software.”

```
0000D250 5468 616E 6B20 796F 7520 666F 7220 796F Thank you for yo
0000D260 7572 2073 7570 706F 7274 2E20 5265 2D72 ur support. Re-r
0000D270 756E 2074 6865 2070 726F 6772 616D 2E00 un the program..
0000D280 4572 726F 7221 0000 0000 0000 0000 0000 Error!..........
0000D290 0000 0000 0000 0000 0000 0000 0000 0000 ................
```

And there, the user understands what he has to do when the software disappears.

For the handle of the REGBOX, we could compare the [ebp+08] of the messages treatment routine (in REGBOX in D3E0) with the one where I want to close the REGBOX. But as the software exits when we've just registered, we no longer need to manage this event! It's sometimes good to know to adapt and to seize good opportunities :o)

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

So we change all this in (end of GOODBOY):

```
.0040D562: 6850D24000                push       00040D250
.0040D567: 6A00                      push       000
.0040D569: FF15A8644000              call       MessageBoxA
.0040D56F: FF7508                    push       [ebp+08]       ;we finally leave as it!
.0040D572: FF15A0644000              call       DestroyWindow
.0040D578: 8BE5                      mov        esp,ebp
.0040D57A: 5D                        pop        ebp
.0040D57B: E948010000                jmp       .00040D6C8      ;we only change here.
```

Hehe, our sharepad has an air now! We'll still check one thing, the one when we start the software whereas the sysreg is empty. I have not taken into account this thing in my coding. We delete the 2 keys "Name" and "Code" from the sysreg, then a short look under SI will tell us what we have to do.

[some instructions under SI later...(use the loader to trace from the PEP on)]

Hehehehe!!! It works fine! We don't need to change something!

Here is what happens:
We start by opening the sysreg (we have eax=0, so it's all right), then we read the key "Name" which does not exist (we are now with eax=a stack offset) and we get a length of 0xB (for me). Then we read the key "Code" which does not exist (eax=junk value), and we close the sysreg (eax=0, so ok). We then go to the serial calculation routine which tells that the returned values in eax for "Name" and "Code" do not match, and we jump after the switches in MSGBOX1-PEP. All what must be done !!!

A last thing still disturbs. When we input nothing in the REGBOX and we validate, the software crashes. By tracing with a "bpx getdlgitemtexta", we'll have a closer look to all of this.

[some instructions under SI later...]

Well! The length of the (empty) input "name" is 0. There will be an infinite loop while the instruction which compares ecx and edx in SERIAL-CALC. To remedy it, this case has to be managed in 2 places (reading of the name in the sysreg and reading of the name in the REGBOX). I proceed as following...:

For the sysreg:

```
(End of TEST-REG)
.0040D5D1: E83449FFFF          call       .0004024AA        ;returns the name "Anubis" in the sysreg
.0040D5D6: 68A0534000          push        0004053A0        |returns the length of this name
.0040D5DB: FF15B0634000        call        lstrlenA         |in eax
.0040D5E1: A3C0534000          mov         [0004053C0],eax  ;we put eax in its buffer for the serial routine
.0040D5E6: 6A10                push        010              |begin of the API which returns the serial
                                                             from the sysreg
.0040D5E8: 68D0534000          push        0004053D0
[...]
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

...which we transform into (comment only on the added lines):

```
.0040D5D1: E8D44EFFFF                call       .0004024AA
.0040D5D6: 68A0534000                push        0004053A0
.0040D5DB: FF15B0634000              call       lstrlenA
.0040D5E1: 85C0                      test       eax,eax          ;eax is null?
.0040D5E3: 7501                      jne       .00040D5E6        ;no, so we jump
.0040D5E5: 40                        inc        eax              ;yes, so we set it to 1
.0040D5E6: A3C0534000                mov        [0004053C0],eax
.0040D5EB: 6A10                      push       010
.0040D5ED: 68D0534000                push       0004053D0
.0040D5F2: 8B4DDC                    mov        ecx,[ebp-24]
.0040D5F5: 51                        push       ecx
.0040D5F6: 68A4D24000                push       00040D2A4
.0040D5FB: FF75FC                    push       [ebp-04]
.0040D5FE: E8A74EFFFF                call      .0004024AA
.0040D603: FF75FC                    push       [ebp-04]
.0040D606: FF15E8624000              call       RegCloseKey
.0040D60C: 8BE5                      mov        esp,ebp
.0040D60E: 5D                        pop        ebp
.0040D60F: C605E053400001            mov        [0004053E0],001
.0040D616: E96BFEFFFF                jmp       .00040D486
```

For the REGBOX (which is now in SERIAL-CALC):

```
.0040D9F7: 6884030000                push       000000384
.0040D9FC: FF7508                    push       [ebp+08]
.0040D9FF: FF157C644000              call       GetDlgItemTextA
.0040DA05: A3C0534000                mov        [0004053C0],eax
.0040DA0A: 6A10                      push       010
.0040DA0C: 68D0534000                push       0004053D0
.0040DA11: 6885030000                push       000000385
[...]
```

...which we transform into (comment only on the added lines):

```
.0040D45F: FF157C644000              call       GetDlgItemTextA
.0040D465: 85C0                      test       eax,eax          ;eax is null?
.0040D467: 7501                      jne       .00040D46A        ;no, so we jump
.0040D469: 40                        inc        eax              ;yes, so we set it to 1
.0040D46A: A3C0534000                mov        [0004053C0],eax
.0040D46F: 6A10                      push       010
.0040D471: 68D0534000                push       0004053D0
.0040D476: 6885030000                push       000000385
.0040D47B: FF7508                    push       [ebp+08]
.0040D47E: FF157C644000              call       GetDlgItemTextA
.0040D484: C605E053400000            mov        [0004053E0],000
.0040D48B: 33C0                      xor        eax,eax          ;caution! we come
                                                                 ;from the sysreg here (new offset)
.0040D48D: 33D2                      xor        edx,edx
.0040D48F: 33DB                      xor        ebx,ebx
.0040D491: 8B0DC0534000              mov        ecx,[0004053C0]
.0040D497: 8A82A0534000              mov        al,[edx+004053A0]
.0040D49D: 8D1C43                    lea        ebx,[ebx+eax*2]
.0040D4A0: 42                        inc        edx
.0040D4A1: 3BCA                      cmp        ecx,edx
.0040D4A3: 75F2                      jne       .00040D497
.0040D4A5: 81C321430000              add        ebx,000004321
.0040D4AB: 81F334120000              xor        ebx,000001234
.0040D4B1: 53                        push       ebx
.0040D4B2: 689C104000                push       00040109C
.0040D4B7: 68F0534000                push       0004053F0
.0040D4BC: FF150C644000              call       wsprintfA
.0040D4C2: 68D0534000                push       0004053D0
.0040D4C7: 68F0534000                push       0004053F0
.0040D4CC: FF15B8634000              call       lstrcmpA
.0040D4D2: 803DE053400000            cmp        [0004053E0],000
.0040D4D9: 0F85E1010000              jne       .00040D6C0
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
.0040D4DF: 85C0                       test        eax,eax
.0040D4E1: 742D                       je         .00040D510
.0040D4E3: 6A00                       push        000
.0040D4E5: 6880D24000                 push        00040D280
.0040D4EA: 68A0D24000                 push        00040D2A0
.0040D4EF: 6A00                       push        000
.0040D4F1: FF15A8644000               call        MessageBoxA
.0040D4F7: C9                         leave
.0040D4F8: C3                         retn
```

...following that, do not forget to recalibrate the jump of TEST-REG which points on the new position of "xor eax,eax":

```
.0040D616: E970FEFFFF                 jmp         .00040D48B   ;D486 becomes D48B
```

And for a valid empty "Name" field, the serial will be: 20757

Voil! The sharepad is definitely finished. At the end, the modifications will have taken a little less than 600 Bytes (0.6 Ko) for the code to add in the padding of the section .rsrc. The adding/modification of the resources strangely does not seem to have modified the size of the executable file.

Reverse rulez!

If you have hold out until here, either you have used your mouse to come directly to the end here or you are completely crazy! ;o) (and I am crazier than you to have written something so huge...). Ah! I realise that I have not written the code for the shortcut "Ctrl+T" which is in the menu. Well, it is not really important, this is not a big improvement for the sharepad, and I am too lazy to do it now ;o), so we trash it!

Far away from me the idea to add some text for "nothing", but this tutorial of the part II being so huge, I put here the WHOLE source code (except the strings) with the offsets and the final structure. If you have lost your way above, you will find here the complete working solution! ;o)

```
* Diversion at the PEP

.004010CC: E9CFC40000                 jmp         .00040D5A0
.004010D1: 90                         nop
.004010D2: 56                         push         esi
.004010D3: FF15E0634000               call         GetCommandLineA


* Diversion in the handling of the IDs

 Possible Ref to Menu: MenuID_0001, Item: "Cut Ctrl+X"
 |
 :00401288 3D00030000              cmp eax, 00000300
 :0040128D 7C21                    jl 004012B0
 :0040128F E98CC00000              jmp 0040D320
 :00401294 0F8E3E040000            jle 004016D8

 Possible Ref to Menu: MenuID_0001, Item: "Paste Ctrl+V"
 |
 :0040129A 3D02030000              cmp eax, 00000302
 :0040129F 0F8456040000            je 004016FB


* Diversion at the EXITPROCESS
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
.00401143: E938C50000                   jmp       .00040D680
.00401148: 90                           nop
.00401149: 8BC6                         mov        eax,esi
.0040114B: 5E                           pop        esi
.0040114C: 8BE5                         mov        esp,ebp
.0040114E: 5D                           pop        ebp
.0040114F: C3                           retn
```

```
* My different parts

ID-COMPARISON
.0040D320: 60                           pushad
.0040D321: 3D8E030000                   cmp        eax,0000038E
.0040D326: 0F8484000000                 je        .00040D3B0
.0040D32C: 3D8F030000                   cmp        eax,0000038F
.0040D331: 0F8439000000                 je        .00040D370
.0040D337: 61                           popad
.0040D338: 3D01030000                   cmp        eax,000000301
.0040D33D: E9523FFFFF                   jmp       .000401294


MSGBOX5
.0040D370: 6A00                         push       000
.0040D372: 68C0D24000                   push       00040D2C0
.0040D377: 68E0D24000                   push       00040D2E0
.0040D37C: 6A00                         push       000
.0040D37E: FF15A8644000                 call       MessageBoxA
.0040D384: 61                           popad
.0040D385: E92345FFFF                   jmp       .0004018AD


REGBOX
.0040D3B0: 6A00                         push       000
.0040D3B2: 68E0D34000                   push       00040D3E0
.0040D3B7: 688C000000                   push       00000008C
.0040D3BC: 6880060000                   push       000000680
.0040D3C1: 6800004000                   push       000400000
.0040D3C6: FF155C644000                 call       CreateDialogParamA
.0040D3CC: A390534000                   mov        [000405390],eax
.0040D3D1: E9D744FFFF                   jmp       .0004018AD

and

.0040D3E0: 55                           push       ebp
.0040D3E1: 8BEC                         mov        ebp,esp
.0040D3E3: 817D0C10000000               cmp        [ebp+0C],0010
.0040D3EA: 750E                         jne       .00040D3FA
.0040D3EC: FF7508                       push       [ebp+08]
.0040D3EF: FF15A0644000                 call       DestroyWindow
.0040D3F5: E92D000000                   jmp       .00040D427
.0040D3FA: 817D0C11010000               cmp        [ebp+0C],000111
.0040D401: 7524                         jne       .00040D427
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
.0040D403: 8B4510                    mov       eax,[ebp+10]
.0040D406: 3D89030000                cmp       eax,000000389
.0040D40B: 750E                      jne      .00040D41B
.0040D40D: FF7508                    push      [ebp+08]
.0040D410: FF15A0644000              call      DestroyWindow
.0040D416: E90C000000                jmp      .00040D427
.0040D41B: 3D88030000                cmp       eax,000000388
.0040D420: 7505                      jne      .00040D427
.0040D422: E829000000                call     .00040D450
.0040D427: C9                        leave
.0040D428: C3                        retn


SERIAL-CALC
.0040D450: 6A20                      push      020
.0040D452: 68A0534000                push      0004053A0
.0040D457: 6884030000                push      000000384
.0040D45C: FF7508                    push      [ebp+08]
.0040D45F: FF157C644000              call      GetDlgItemTextA
.0040D465: 85C0                      test      eax,eax
.0040D467: 7501                      jne      .00040D46A
.0040D469: 40                        inc       eax
.0040D46A: A3C0534000                mov       [0004053C0],eax
.0040D46F: 6A10                      push      010
.0040D471: 68D0534000                push      0004053D0
.0040D476: 6885030000                push      000000385
.0040D47B: FF7508                    push      [ebp+08]
.0040D47E: FF157C644000              call      GetDlgItemTextA
.0040D484: C605E053400000            mov       [0004053E0],000
.0040D48B: 33C0                      xor       eax,eax
.0040D48D: 33D2                      xor       edx,edx
.0040D48F: 33DB                      xor       ebx,ebx
.0040D491: 8B0DC0534000              mov       ecx,[0004053C0]
.0040D497: 8A82A0534000              mov       al,[edx+004053A0]
.0040D49D: 8D1C43                    lea       ebx,[ebx+eax*2]
.0040D4A0: 42                        inc       edx
.0040D4A1: 3BCA                      cmp       ecx,edx
.0040D4A3: 75F2                      jne      .00040D497
.0040D4A5: 81C321430000              add       ebx,000004321
.0040D4AB: 81F334120000              xor       ebx,000001234
.0040D4B1: 53                        push      ebx
.0040D4B2: 689C104000                push      00040109C
.0040D4B7: 68F0534000                push      0004053F0
.0040D4BC: FF150C644000              call      wsprintfA
.0040D4C2: 68D0534000                push      0004053D0
.0040D4C7: 68F0534000                push      0004053F0
.0040D4CC: FF15B8634000              call      lstrcmpA
.0040D4D2: 803DE053400000            cmp       [0004053E0],000
.0040D4D9: 0F85E1010000              jne      .00040D6C0
.0040D4DF: 85C0                      test      eax,eax
.0040D4E1: 742D                      je       .00040D510
.0040D4E3: 6A00                      push      000
.0040D4E5: 6880D24000                push      00040D280
.0040D4EA: 68A0D24000                push      00040D2A0
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
.0040D4EF: 6A00                         push      000
.0040D4F1: FF15A8644000                 call      MessageBoxA
.0040D4F7: C9                           leave
.0040D4F8: C3                           retn


GOODBOY
.0040D511: 8BEC                         mov       ebp,esp
.0040D513: 83EC04                       sub       esp,004
.0040D516: 8D45FC                       lea       eax,[ebp-04]
.0040D519: 50                           push      eax
.0040D51A: 6848514000                   push      000405148
.0040D51F: 6801000080                   push      080000001
.0040D524: FF15F0624000                 call      RegCreateKeyA
.0040D52A: 85C0                         test      eax,eax
.0040D52C: 7541                         jne      .00040D56F
.0040D52E: 68A0534000                   push      0004053A0
.0040D533: 6856524000                   push      000405256
.0040D538: FF75FC                       push      [ebp-04]
.0040D53B: E8F24EFFFF                   call     .000402432
.0040D540: 68D0534000                   push      0004053D0
.0040D545: 68A4D24000                   push      00040D2A4
.0040D54A: FF75FC                       push      [ebp-04]
.0040D54D: E8E04EFFFF                   call     .000402432
.0040D552: FF75FC                       push      [ebp-04]
.0040D555: FF15E8624000                 call      RegCloseKey
.0040D55B: 6A00                         push      000
.0040D55D: 6830D24000                   push      00040D230
.0040D562: 6850D24000                   push      00040D250
.0040D562: 6850D24000                   push      00040D250
.0040D567: 6A00                         push      000
.0040D569: FF15A8644000                 call      MessageBoxA
.0040D56F: FF7508                       push      [ebp+08]
.0040D572: FF15A0644000                 call      DestroyWindow
.0040D578: 8BE5                         mov       esp,ebp
.0040D57A: 5D                           pop       ebp
.0040D57B: E948010000                   jmp      .00040D6C8


TEST-REG
.0040D5A0: 55                           push      ebp
.0040D5A1: 8BEC                         mov       ebp,esp
.0040D5A3: 83EC40                       sub       esp,040
.0040D5A6: 8D4DFC                       lea       ecx,[ebp-04]
.0040D5A9: 51                           push      ecx
.0040D5AA: 6848514000                   push      000405148
.0040D5AF: 6801000080                   push      080000001
.0040D5B4: FF15EC624000                 call      RegOpenKeyA
.0040D5BA: 85C0                         test      eax,eax
.0040D5BC: 7539                         jne      .00040D5F7
.0040D5BE: 6A20                         push      020
.0040D5C0: 68A0534000                   push      0004053A0
.0040D5C5: 8D4DDC                       lea       ecx,[ebp-24]
.0040D5C8: 51                           push      ecx
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
.0040D5C9: 6856524000              push      000405256
.0040D5CE: FF75FC                  push      [ebp-04]
.0040D5D1: E8D44EFFFF              call     .0004024AA
.0040D5D6: 68A0534000              push      0004053A0
.0040D5DB: FF15B0634000            call      lstrlenA
.0040D5E1: 85C0                    test      eax,eax
.0040D5E3: 7501                    jne      .00040D5E6
.0040D5E5: 40                      inc       eax
.0040D5E6: A3C0534000              mov       [0004053C0],eax
.0040D5EB: 6A10                    push      010
.0040D5ED: 68D0534000              push      0004053D0
.0040D5F2: 8B4DDC                  mov       ecx,[ebp-24]
.0040D5F5: 51                      push      ecx
.0040D5F6: 68A4D24000              push      00040D2A4
.0040D5FB: FF75FC                  push      [ebp-04]
.0040D5FE: E8A74EFFFF              call     .0004024AA
.0040D603: FF75FC                  push      [ebp-04]
.0040D606: FF15E8624000            call      RegCloseKey
.0040D60C: 8BE5                    mov       esp,ebp
.0040D60E: 5D                      pop       ebp
.0040D60F: C605E053400001          mov       [0004053E0],001
.0040D616: E970FEFFFF              jmp      .00040D48B


MSGBOX1-PEP
.0040D640: 6A00                    push      000
.0040D642: 68A0D14000              push      00040D1A0
.0040D647: 68C0D14000              push      00040D1C0
.0040D64C: 6A00                    push      000
.0040D64E: FF15A8644000            call      MessageBoxA
.0040D654: 55                      push      ebp
.0040D655: 8BEC                    mov       ebp,esp
.0040D657: 83EC44                  sub       esp,044
.0040D65A: E9733AFFFF              jmp      .0004010D2


MSGBOX2-EXITPROC
.0040D680: 6A00                    push      000
.0040D682: 68A0D14000              push      00040D1A0
.0040D687: 68E0D14000              push      00040D1E0
.0040D68C: 6A00                    push      000
.0040D68E: FF15A8644000            call      MessageBoxA
.0040D694: FF1598634000            call      ExitProcess
.0040D69A: E9AA3AFFFF              jmp      .000401149


DISACTIV
.0040D6C0: 85C0                    test      eax,eax
.0040D6C2: 0F8578FFFFFF            jne      .00040D640
.0040D6C8: C6054ED6400090          mov       [00040D64E],090
.0040D6CF: C6058ED6400090          mov       [00040D68E],090
.0040D6D6: C60556C5400000          mov       [00040C556],000
.0040D6DD: C60566C5400000          mov       [00040C566],000
.0040D6E4: C605DC86400068          mov       [0004086DC],068 ;"h"
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

```
.0040D6EB: C605DE86400061              mov        [0004086DE],061 ;"a"
.0040D6F2: C605E086400072              mov        [0004086E0],072 ;"r"
.0040D6F9: C605E286400065              mov        [0004086E2],065 ;"e"
.0040D700: C605E486400070              mov        [0004086E4],070 ;"p"
.0040D707: C605E686400061              mov        [0004086E6],061 ;"a"
.0040D70E: C605E886400064              mov        [0004086E8],064 ;"d"
.0040D715: C605EA86400020              mov        [0004086EA],020 ;" "
.0040D71C: C60544C7400000              mov        [00040C744],000
.0040D723: 55                          push       ebp
.0040D724: 8BEC                        mov        ebp,esp
.0040D726: 83EC44                      sub        esp,044
.0040D729: E9A439FFFF                  jmp       .0004010D2
```


Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.

V. Final Notes

After all these essays where protections were studied, it was worth to try rebuilding what we have ”de-built” in cracking, but on the same matter as we did until now, wasn’t it?? I am sure that this Art or Science of Reverse Engineering is just at its beginning, and that a lot of more marvellous things are possible and will come in the future by new generations of Reversers. For the first time in history, it is possible to create and transform as far as the imagination wants it. Can we still talk about ’limits’ ? I am not sure that the answer is yes. The future will say it.

I hope that this small essay I have written will also open gates in your mind as it did with me by reading the ones of LaZaRuS and NeuRaL_NoiSE. We are at the beginning of a new area, it’s your power to explore it and go forth. A little bit as in the Matrix, isn’t ? ;o)

Also, since the time where I have read/discovered some years ago the essays from LaZaRuS and NeuRaL_NoiSE until now, some very good RE essays have been written in the meanwhile. I can not mention them all, but my greetings are going to these people too ;o)

This essay has been written along I was coding the 2 sharepads, so I apologise if it is sometimes scrambled!

I can be contacted here: anubis@iname.com or on the IRC chan of my team that you will find on our homepage: http://www.Shmeitcorp.tk. If this last url is no more valid, just search in an engine, you will surely find us ;o)

This tutorial has been originally published in French in the issue nr.5 of our Mementos (cracking & reversing tutorials collection, available on our homepage) in November 2002. Thank you to all of you guys, I would never have become what I am today if I had not had the chance to be accepted in your (our) team!

A big and special thank to Christal who helped me to solved a tricky point on which I stuck in the part II of this tutorial. Also thanks to the Shmeitcorp members who have read this tutorial and helped to improve it ;o)

To LaZaRuS and NeuRaL_NoiSE: if you read me, please contact me!! I have a lot of things I’d like to discuss with you ;o)

Also, forgive my lame English!

Great thanks and/or greetz fly to (no order) :
Fravia+, LaZaRuS, NeuRaL_NoiSE, +Malattia and Ringzer0, +ORC, +Mammon, +Spath, +Razzia, +Frog’s Print, Iczelion, Masta, Tsehp, Carpathia, Crackz, Anarchriz, +Sandman, Zero, Santmat, The_Analyst & The Immortal Descendants, Mr.Philex, Christal, Teeji, Pass Partout, TaMaMBoLo, Lutin Noir, Silversandstorm, Lord Soth, Defiler, Detten from/and BIW, Chafe from/and TMG, tkc, all Shmeitcorp members but also Iron Maiden, Cacophony, Dimmu Borgir, Ozzy Osbourne, Immortal, Manowar, Naglfar, Graveworm, Lord Belial, Marduk, Dissection, Mystic Circle, Cradle Of Filth and much more!
If I have forgotten you, drop me a line and I will add your name!

Wisdom is the Mother of all Knowledge.

VI. Oh duh

Doesn’t apply, does it?

Copyright © 2004 and published by the CodeBreakers-Journal. Single print or electronic copies for personal use only are permitted. Reproduction and distribution without permission is prohibited.