Tutor4:

We need some more practice:-)
Let's remember the to do routine stuff again....

- watching a program in action
- disassembling the program and looking for familiar strings
- finding and analyzing the protection routine of the program
- Modifying the program so it does the thing you want.

Well... The essence of cracking brought back to 4 sentences ;-) Let's do another one...

The program we are going to take a closer look at this time will be: Sweet Little Piano www.ronimusic.com

Step 1: Run the program:
Run it.
Cool. Fast and small:)
Bah. Two text files ...
A very big Unregistered Shareware on the caption bar...
Check out the Help menu for registration options ... Hmm... a Password option... Select it and enter just something to see what happens ... Click on OK.. nothing....
Maybe it accepted it? hmm.. no ... the caption bar still says Unregistered....
Ok close it ... bah ... more text files ... and a notification that the settings are not saved in the unregistered version ... well ... kind of irritating those text files !
Let's fix it :-)

Step 2: Disassemble the program:
Disassemble the program. Good, small is fast :-) Always....
Now, we don't have any strings that pop up when we want to register something...
Let's browse for strings like registered, unregistered, the string about the unsaved settings.
hmm... evaluation time left ... password.txt.... passworddialog....
sweet little piano - Unregistered <<-- looks like our caption bar ;-) go on...
Thanks for registering ... cool! So it thanks you anyway :-)
Let's jump to the place ... Double click on it an we will land right on top of the registration registration routine...

Step 3: Analyzing the protection routine.... / Understanding the jumping Mechanism...
Let's analyze the protection routine.

```
////////////////////// Code snip ////////////////////////////
ADDRESS   MACHINE CODE        ASSEMBLER INSTRUCTIONS

* Possible Reference to Dialog: PASSWORDDIALOG, CONTROL_ID:0064, ""
                                |
:00401715 6A64                  push 00000064
:00401717 53                    push ebx

* Reference To: USER32.GetDlgItemTextA, Ord:0000h
                                |
:00401718 E8A5B50000            Call 0040CCC2
:0040171D E822FFFFFF            call 00401644
:00401722 85C0                  test eax, eax
:00401724 741E                  je 00401744
:00401726 6A30                  push 00000030

* Possible StringData Ref from Data Obj ->"SweetPiano"
                                |
:00401728 6866D24000            push 0040D266

* Possible StringData Ref from Data Obj ->"Thanks for registering!"
```

```
                                |
:0040172D 68FED14000            push 0040D1FE
:00401732 53                    push ebx

//////////////////// Code snip /////////////////////////
```

PasswordDialog ... a call to GetDlgItemTextA ... Another call.... a test...
and depending on the test a je.... The je jumps over the thank you ...
And just ends the dialog box ... without telling you that you entered something wrong...
So this is right ... we did indeed not see that we typed something wrong ... but apparently we are supposed to see if we type something right :-)

Again execute the je jump, and look where it goes to ... return from the jump....
Now lets try to rewrite what goes on here...

```
call ShowPasswordDialog
call GetEnteredText
call IsEnteredTextGood
test value in eax
je     QuietExit

ShowThanksForRegistering

QuietExit:
```

---

the source code must have looked like this :

```
GetDlgItemText(_ID_Serial);
if (EnteredTextGood) ShowThanksForRegistering

// else nothing....
```

This is another interesting piece of code.... test eax, eax ... This assembler instruction tests if the value of eax is equal to itself ... If it is it is equal ... so a je instruction jumps ... if it is not equal, it does not jump....
To crack this program we can change the je instruction into two nop instructions... and we are done...

We have seen here, that the call has put a value in eax.... something which is not equal to zero or a zero... In our previous example we saw that the called Is_Serial_Valid call set some value in memory ... Here we see that the called
Is_Serial_Valid call sets the eax register of our processor to some value....

Step 4: Changing the original program...
So modify it :-)
Open Hiew ... you know the rest by now...