Tutor5

In the previous 4 tutorials we have handled the basics of cracking ...
We have summarized the basics of cracking into the following few points:

- 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.

In this tutorial I will show you how to insert your **own** code into the original programs code, so that the program does exactly what you want it to do ;-)

To show you how to do it, I have again written my own program. First I will show you how to crack the program I made, afterwards, we will together crack a commercial program, which has almost the same protection as my little 20 kb exe ;-)

We will again do the cracking as we have learned ... systematically.

Step 1: Run the program:
Our cool dino again :-) but what do we see there ? A very ugly caps:
UNREGISTERED EVALUATION - Tutorial 3 ...
Man I hate caps ;-)

Let's explore the file menus…
Hmm.. The Help menu has a register option. Select it. Enter a fake name and a Serial. Press Ok.
Hmm.. An ugly error message :-)
Let's fix it …

Step 2: Disassemble the program:
When we disassemble and look at the string references, we see a few of the strings that we can recognize. The big ugly caps Unregistered string, the error while entering password string….
Let's select the: Error entering name / password, string…. We will land in the following piece of disassembly:

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

* Referenced by a (U)nconditional or (C)onditional Jump at Address:
|:00401521(C)
|

* Possible StringData Ref from Data Obj ->"**UNREGISTERED EVALUATION** - "
                                       ->"Tutorial 5"
                                |
:00401546 68E6404000            push 004040E6
:0040154B FF7508                push [ebp+08]

* Reference To: USER32.SetWindowTextA, Ord:020Ch
                                |
:0040154E E8D1020000            Call 00401824
:00401553 6A10                  push 00000010

* Possible StringData Ref from Data Obj ->"ERROR"
                                |
:00401555 6840404000            push 00404040
                                                        You will land here
* Possible StringData Ref from Data Obj ->"Error entering name / password, "
                                       ->"please retry "
                                |
:0040155A 6846404000            push 00404046
```

```
:0040155F FF7508                push [ebp+08]

* Reference To: USER32.MessageBoxA, Ord:018Ah
                                 |
:00401562 E899020000            Call 00401800
//////////////////// Code snip ////////////////////////////
```

Step 3: Analyzing the protection routine.... / Understanding the jumping Mechanism...

Let's try to understand this piece of code…..
The code is executed after a jump **from** address 401521. What it does is: setting the text UNREGISTERED EVALUATION. After it has done this, it pops up a Message box telling you that you have entered the wrong name / password combination.
Well … Let's go back to where the decision was made … Address 401521 ….
Go to code location 401521 (see Appendix 1 for how to).

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

:0040150E 89D7                  mov edi, edx
:00401510 09C0                  or eax, eax       ; length of entered name zero ?
:00401512 7459                  je 0040156D
:00401514 FF7508                push [ebp+08]
:00401517 E8CFFEFFFF            call 004013EB     ; check something
:0040151C 83C404                add esp, 00000004
:0040151F 09C0                  or eax, eax       ; is eax zero ?
:00401521 7423                  je 00401546       <<-- You will land here …
                                                  ; jump to unregistered dialog…

* Possible StringData Ref from Data Obj ->"Registered Version - Tutorial  5 "
                                 |
:00401523 680F414000            push 0040410F

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

What we see here is pretty familiar to us by now. This je jumps over the piece of code, which tells us that we are registered. After this, the other code is executed.
In pseudo-code the above piece of code would look something like the following :

---

if SerialIsValid ShowThankYouForRegistering
else ShowWrongSerial

We know now what to do :-)… Let's change the jump to NOP's. Then this program will always show us that we are registered :-) easy huh ?

Step 4: Changing the original program...
Open a copy of the program in hiew and change the je you see into two nops. Just as we did in our very first tutorial. Piece of cake…..

Step 5: Testing your cracked program...
Let's run it. Go to the registration dialog box. Enter a fake name and serial combination.
Yes !! we are done :-)
It registered us ….Look how nice the ugly caps disappeared from the title bar…. It now says registered version :-)
I hear you think… Was it this easy ?? Is Intern not going to learn us anything new this tutorial ??
Well … In fact you are learning now … That you always should test your crack before spreading it / giving it away to others.
Ok close the program. And restart it… :-(Look … The ugly caps is still there …..
How depressing …. Probably we overlooked something while analyzing our routine …

Let's look a little bit close at the disassembly …

Step 3, Second Try : Analyzing the protection routine.... / Understanding the jumping Mechanism...

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

:0040150E 89D7                    mov edi, edx
:00401510 09C0                    or eax, eax       ; length of entered name zero ?
:00401512 7459                    je 0040156D
:00401514 FF7508                  push [ebp+08]
:00401517 E8CFFEFFFF              call 004013EB     ; check something
:0040151C 83C404                  add esp, 00000004
:0040151F 09C0                    or eax, eax       ; is eax zero ?
:00401521 7423                    je 00401546       <<--You will land here …
                                                    ; jump to unregistered dialog…

* Possible StringData Ref from Data Obj ->"Registered Version - Tutorial  5 "
                                   |
:00401523 680F414000              push 0040410F

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

This code looks something like this in c :

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

if (strlen(name) != 0 ) {
        if ( CheckTheSerialAndName()  == TRUE ){
                MessageBox(  "Thank you for registering ");
        }
        else{
                MessageBox( "Entered a wrong serial");
        }
}

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

Well … you probably did notice that the call at address 401517, which I commented as : Check Something is the *CheckTheSerialAndName* function …

We made this check for name serial combination totally useless after we removed the jump at address 401521. But what happens here is, this function is also called at another time from another place. Programmers write a function like this once, and like to call it at times when they need to decide whether you are a registered or an unregistered user. These moments are usually at the moment you try to register with a name and a serial number. And the next moment is when the program starts, when it wants to display a nag, or wants to set title bars or wants to determine if your trial time is up ….
This type of functions you will encounter very often.

Let's trace into the call. At the beginning of the function you will see this:

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

* Referenced by a CALL at Addresses:
|:00401292   , :00401517
|
:004013EB 55                      push ebp
////////////////////// Code snip ////////////////////////////
```

This means this function is called from two adressess… from : 00401292 and from 00401517.

If you push the ret button, to go back to the place we traced into the call, you will see that the call that we see is at 401517. That means this function is also called from address 401292.
Let's go to this address.

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

:00401292 E854010000              call 004013EB     ; isnameserialvalid ?
:00401297 83C40C                  add esp, 0000000C
:0040129A 09C0                    or eax, eax       ; eax = 0 ?
:0040129C 740D                    je 004012AB       ; jump to not registered

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

hmm… see … This piece of code is almost the same :-)

If we nop out the je here, we will be done… let's do it.
Step 4, second try: Changing the original program...
Load the program in hiew, and nop the je instruction …

Step 5, second try: Testing your cracked program...
Run it … and yup :-) It works ….
We did it …

Afterthoughts :

In this case we had to nop two jumps. Sometimes some programmers call the IsSerialValid routine several times …Sometimes you will see a list of 20 references :-) (just seen it once)
We could try to locate all the separate jumps, and noping them … But a cleaner and much more elegant way of fixing this small problem would be the following :

Writing your own code :-)
How ?
We know that the functions of this type usually return a value back. A value like false or true…
In the disassemblys you will see that after a call, one of the registers is being checked. (see also tutorial 3). In this program the register eax is checked. And you will see that in almost **all** cases the eax register is checked. So a function call like : IsSerialValid returns a value in eax. A zero or a one.
What we can do to fix the problem is making this function return the value we want in eax :-)

To do this you have to assemble a few assembler instructions in Hiew. Let's practice on this program.
Delete your cracked version, and make a copy of tutorial5.exe
Open it in hiew. Go to the **beginning** of the call at address 4013EB.

Now enter assemble mode, and assemble the following instructions.

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

mov eax, 1                        ; make eax equal to 1
ret                               ; return to caller

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

Apply the changes, save the file and run it.

Cool huh ? we have a registered version now. The function always returns a 1 in eax now meaning the serial is valid !
And you only had to patch it at one address :-)

What happens now is this, the function is called, and it immediately returns a 1 in eax without doing its usual complicated stuff to check the entered serial.

In this tutorial we have seen that you always have to check if the patch you applied works.
We also saw that registration routines are written once, but used often by programmers. (they call it code-reuse, but we call it lazy programmers ;-)) Usually such registration routines will be called at program startup and while checking an entered name / serial combination.
Finally we saw that we can write our assembler code into the original exe, to finally let the program behave the way we want :-) Here we assigned a value to one of the registers, and made the procedure return to its caller, without performing any code that was meant to be performed :-)