Cracking PCGuard Protected Aplications

by

macilaci

Published
by
Tsehp
2002

Introduction

The cracking is old as a programming. I think there´s no programmer who wasn´t just for a while a cracker. Even when he was just curious what that program does, even when he wanted to know what techniques are used by that software, he turns for a while to a cracker. In general there´s no line or a border between programming and cracking as such. These things are both components of computer science and cannot be divided.

Last time I visited some anti cracking site I saw a clause about a sharing knowledge. Why are crackers sharing their knowledge and protectors not? Why there are many crackers´ sites – for free and for protectors not? The answer is simple: Because of motivations. The protectors want to protect their (or theirs customers) applications against unauthorised use, that means they have commercial interests. Their best weapon is a secrecy. Hidden files, registry entries, algorithms, encryption – all this is here to maintain their secrecy. They don´t share knowledge at that level as crackers because of commerce. And a commerce is a barrier what in our world can´t be so easily bridged. That´s the reason why are protector companies disappearing from the market.

And the main purpose of this document is to share knowledge.

Tools used

The protection scheme – PCGuard is designed in such way, that it makes most of known tools unusable or hardly usable.

Disassemblers: IDA
Debuggers: WinICE, Icedump
Other tools: Imprec – an import reconstructor
Monitoring tools: Filemon, Regmon
Assembler: Masm

These and many other tools can be found on various sites. You can use search engine to find them (www.google.com, www.altavista.com).

The essay

This application can be downloaded from www.eeye.com site on request. Version used in this essay is 3.8. The protection mechanism PCGuard is in version 4.05.

The protection consist of two independent layers. First is the PCGuard´s layer which is set to 40 days after first run. After that period it will display an error message shown in fig 1.

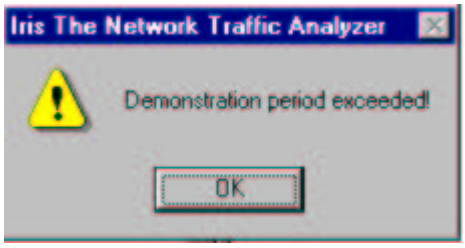

Fig. 1

The second protection layer consist of some licensing mechanism. Using the registry entries and some hidden files sets and maintains your license for 15 days. Even if you can obtain some way a valid license number this program will work only for 40 days according to layer one. After 15 day period the program will display a dialog box showing that your license expired. After 40 days it will display only the shown message box.

Regmon, Filemon & Co.

Using Regmon and Filemon requires some experience in finding suspicious entries such as hidden files and registry entries that are maintained by protection layers.
Layer one:
Let´s set the date beyond your 40 day trial period. Regmon and Filemon show us some suspicious registry entries and a hidden file:

\windows\inst32ba.dll

HKEY_CLASSES_ROOT\{22622989-0F06453D-0A87DE3B}

When you delete this registry entry and the hidden file in your windows directory, the trial period counts again from zero.
Layer two:
Experimentig with setting the time and running the program you may find following:

HKEY_LOCAL_MACHINE\Software\Acudata

\windows\1234-5678-9ABC-DEF1-2345\license.sls
\windows\1234-5678-9ABC-DEF1-2345\checkout.sls
\windows\1234-5678-9ABC-DEF1-2345\activeuser.sls
\windows\iris the network traffic analyzer.dat

and plus some hidden files:

\windows\system\winsusrx.dll
\windows\system\winsusrm.dll

Deleting above registry entry and above files restores your trial period.

WinICE and Icedump

The first layer contains some anti debugging tricks, so running without icedump is not recommended.

Strategy:
1. To find the Original Entry Point (OEP)
2. Dump the Application
3. Restore and fix the import table
4. Cracking the Application

The most difficult thing to do is the first point. Since the protection layer has some implemented protection mechanisms you have to be careful. Setting hardware breakpoints is not recommended – it will result in an error and setting software breakpoints in some cases causes error too. Rules to set these breakpoints are simple: Set only software breakpoints and never set a breakpoint on API entry. I3HERE should be off..

First time I used some bpx on *createwindowexa*+some_bytes to set the breakpoint inside the API. But it was too complicated. Second time I used msvcrt API function *time*. Setting a breakpoint on time and F12 will take you back to iris at 004673C1. Scrolling the winice´s code window upwards you can find the begin of this function at 00467040. Using the stack and a bit of assembly knowledge you can locate the entry point at 0048054C.

There is an easier method to do this. Sipmly do a breakpoint inside msvcrt API function *__set_app_type* and it will take you to the entry point routine at 00480579.

So the entry point was located at 0048054C. Doing a breakpoint on it and running the application again we can finally dump the program. When winice pops up, clear all breakpoints and with */pedump 400000 eip c:\iris.dmp* command the application will be dumped. Continue running the application.

When you try to run the good old dumping tools for win32 gui (e.g. procdump), these will simly fail because of modifying the process infos. You may get a two kB file with nothing inside. And with a raw dump you will get an error when disassembling IDA,so there´s need to reconstruct the PE format too. These are reasons why was the pedump command chosen.

Imprec

The next step is the import table reconstruction. Inside winice locate the start and the end of import table.

Next run Imprec and fill in the found values : OEP 0008054C, RVA 0008B000, Size 0000169C. Click on Get Imports and next on Auto Trace, since some entries are unresolved. We see now that there are some invalid file thunks. To fix them click on show invalid. Inside winice you will see at that address only garbage, so we can now cut these thunks. Go on the invalid import entry and click on Cut thunk(s) as shown below in fig 2.

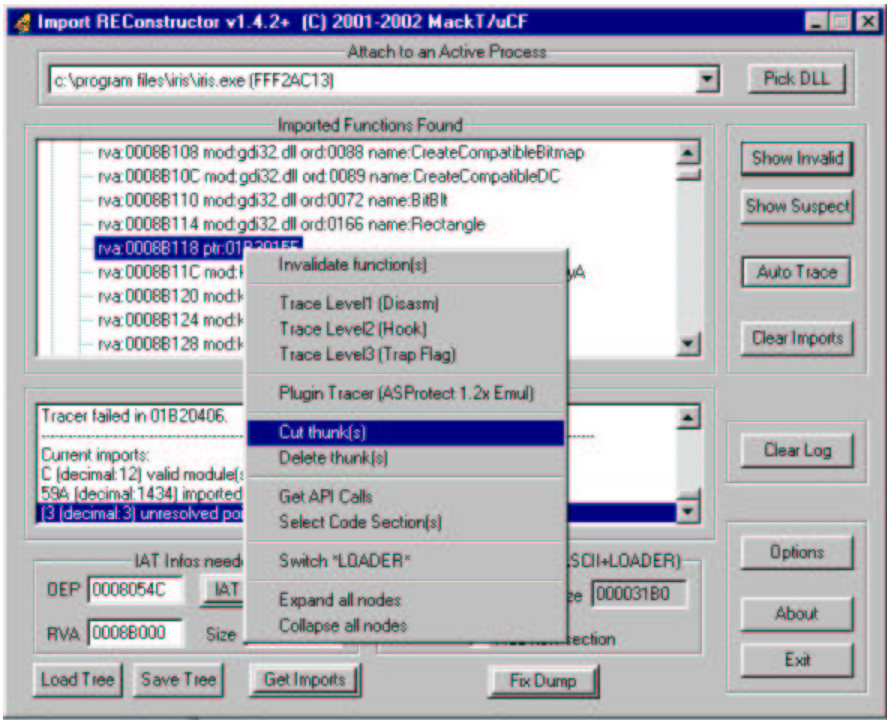

Fig.2

The valid thunks have reference only to one library. So locate next invalid thunk and find the invalid entry with doing cut thunk on this until you will get all entries valid. Check the add new section check box and click on Fix dump, locate the iris dump file and choose save. Done with step 3.

IDA

The first layer of protection was removed, the next step is the second layer. Doing a breakpoint on *time* function you will get to 004673C1 and a bit later to 00444A25 location. Looking at the assembly we see following:

```
004673C1        call    ds:time
004673C7        lea     edx, [ebp+var_28]
004673CA        push    edx
004673CB        call    ds:localtime
004673D1        push    eax
004673D2        call    ds:asctime
004673D8        mov     edi, eax
004673DA        or      ecx, 0FFFFFFFFh
004673DD        xor     eax, eax
.
.
00467453        lea     ecx, [ebp+var_44]
00467456        lea     edx, [ebp+var_528]
0046745C        push    ecx
0046745D        push    edx
0046745E        call    ds:StartLic
00467464        test    eax, eax
00467466        jz      loc_467AAF
```

and at 00444A25:

```
00444A25        call    ds:time
00444A2B        lea     edx, [esp+160h+var_14C]
00444A2F        push    edx
00444A30        call    ds:localtime
00444A36        push    eax
00444A37        call    ds:asctime
00444A3D        mov     edi, eax
00444A3F        or      ecx, 0FFFFFFFFh
00444A42        xor     eax, eax
.
.
00444AB8        lea     ecx, [esp+15Ch+var_144]
00444ABC        push    ecx
00444ABD        call    ds:HeartBeatUpdate
00444AC3        test    eax, eax
00444AC5        jz      loc_444B5C
```

Next question is: What are the Startlic and the HeartBeatUpdate functions? Both are tested with eax, so it is just a eax=TRUE matter? Yes and no. Yes because when the license is valid the function returns 1, no because it does not only that.

Startlic: Looking on the stack we will see that there is pushed fisrt some pointer to some bytes and second is the structure with license number. Some bytes: as you´ll trace the program you will see that these bytes are actually encrypted ascII date with simple xor:

```
00467437        mov     al, [ebp+edx+var_44] ;read the byte
0046743B        lea     edi, [ebp+var_44]
0046743E        xor     al, 0EFh                                ; xor with 0xef
00467440        or      ecx, 0FFFFFFFFh
00467443        mov     [ebp+edx+var_44], al ;write next  byte
00467447        xor     eax, eax
```

```
00467449            inc     edx
0046744A            repne scasb
0046744C            not     ecx
0046744E            dec     ecx
0046744F            cmp     edx, ecx
00467451            jb      short loc_467437                ;loop back
```

And after the Startlic it is decrypted and compared to the originally extracted date in ascII format:

```
0046747D            mov     al, [ebp+edx+var_44];read byte
00467481            lea     edi, [ebp+var_44]
00467484            xor     al, 0CDh                                ; xor with cd
00467486            or      ecx, 0FFFFFFFFh
00467489            mov     [ebp+edx+var_44], al ;write byte
0046748D            xor     eax, eax
0046748F            inc     edx
00467490            repne scasb
00467492            not     ecx
00467494            dec     ecx
00467495            cmp     edx, ecx
00467497            jb      short loc_46747D                ;loop back
```

At location 004674A2 is the date compared. So the program sends a challenge and should get a proper response.

Heartbeatupdate: The same with this function. The only value that is pushed into the stack is the pointer to challenge. This can be located at 000444ABD.

This gave me idea to emulate the whole library, so the new eeyelic.dll will be created.Of course can be these locations patched, but the dll could be used together with the protected application. The file is tamper proof, so this is andvantage (in case of virus attack).

Building the dll

The library should contain following functions: HeartBeatUpdate, StartWizard, GetLicUserName, StartLic, GetMeter and GetMeteringType – these are imported by the application and should contain a proper response function for the program. Below I will provide the asm source of the dll:

```
 .386
.model flat,stdcall
option casemap:none
include \masm32\include\windows.inc
include \masm32\include\user32.inc
include \masm32\include\kernel32.inc
includelib \masm32\lib\user32.lib
includelib \masm32\lib\kernel32.lib

.data

.code
DllEntry proc hInstance:HINSTANCE, reason:DWORD, reserved1:DWORD
        mov  eax,TRUE
        ret
DllEntry Endp

HeartBeatUpdate proc
        push    ebp
        mov     ebp, [esp+8]
        mov     edi, [esp+8]
        or      ecx, 0FFFFFFFFh
        xor     eax, eax
        xor     edx, edx
        repne scasb
        not     ecx
        dec     ecx
```

```
looop:
        mov     cl, [ebp+edx]
        mov     edi, ebp
        xor     cl, 022h
        xor     eax, eax
        mov     [ebp+edx], cl
        or      ecx, 0FFFFFFFFh
        inc     edx
        repne scasb
        not     ecx
        dec     ecx
        cmp     edx, ecx
        jb      looop
        mov         eax,1
        pop         ebp
        retn 4
HeartBeatUpdate endp


StartWizard proc

        ret
StartWizard endp


GetLicUserName proc
        retn 4
GetLicUserName endp


StartLic proc
        push    ebp
        mov     ebp, [esp+0ch]
        mov     edi, [esp+0ch]
        or      ecx, 0FFFFFFFFh
        xor     eax, eax
        xor     edx, edx
        repne scasb
        not     ecx
        dec     ecx
laoop:
        mov     cl, [ebp+edx]
        mov     edi, ebp
        xor     cl, 022h
        xor     eax, eax
        mov     [ebp+edx], cl
        or      ecx, 0FFFFFFFFh
        inc     edx
        repne scasb
        not     ecx
        dec     ecx
        cmp     edx, ecx
        jb      laoop
        mov         eax,1
        pop         ebp
        mov eax,1
        retn 8
StartLic endp


GetMeter proc
        xor eax,eax
        retn
GetMeter endp


GetMeteringType proc
        xor eax,eax
        ret
GetMeteringType endp
End DllEntry
```

The def file:

```
LIBRARY eeyelic
EXPORTS HeartBeatUpdate
EXPORTS StartWizard
EXPORTS GetLicUserName
EXPORTS StartLic
EXPORTS GetMeter
EXPORTS GetMeteringType
```

And the makefile:

```
\masm32\bin\ml /c /coff /Cp eeyelic.asm
\masm32\bin\Link /DLL /DEF:eeyelic.def /SUBSYSTEM:WINDOWS /LIBPATH: \masm32\lib eeyelic.obj
```

Conclusion

Although the application has two protection layers and the one is ´crack proof´, the protection can be still bypassed by a simple program which emulates the also PCGuard protected dll.

Exercise: Write a loader for the main program that deletes the \windows\inst32ba.dll file and HKEY_CLASSES_ROOT\{22622989-0F06453D-0A87DE3B} registry entry before starting the iris.exe.