REVERSING THE PROTECTION SCHEME OF "HELLRAISER SYSTEM UTIL V4" CRACKME IN ?? MOVES BY GYVER75

FOREWORDS

Crackmes.de is probably the most important site where a novel reverser can test his abilities: little programs written in different languages (C, Delphi, Assembler, Visual Basic etc...) just expect us to be solved. Clearly, every Crackme has a different difficult level; personally I choose a level # 8 crackme for three principal reasons:

1) **Curiosity**: I've never reversed this type of target: Because is it so hard? Will I be able to solve it? Probably both..
2) **Challenge**: isn't it a real challenge to try discovering some weakness behind a protection scheme?
3) **Increasing my knowledge:** my Basket's coach said: "even just playing with stronger men, you can learn something! " ☺

With this spirit, I solved the "***HellRaiser System Util V4***"; the scope is simply unpack the target and find the right serial to unlock the buffer slider of its form. Indeed, because this CrackMe has not yet a solution, I decided to write this little paper... so I hope you'll have a good leisure and as always, sorry for my bad English (thanks to Shub for his review)!

1.1 TABLE OF CONTENTS

1.2 TOOLS USED

- **A ring 3 debugger:** it's a real challenge to discover a working alternative to **OllyDbg** ☺, it's the most famous and probably powerful ring 3 debugger available on Microsoft Operative Systems; it's free and extensible with many plugins. Personally, in my Ollydbg copy, I installed these ones:
  - **Scherzo's LCB plugin**: Useful to import / export Labels, Comments and Breakpoints into / from a file; in this way you will never miss any data under Olly;
  - **OllyVbHelper plugin**: Find and label DllFunctionCall and MSVBVM imports;
  - **Stealth64 1.2 beta plugin**: don't misunderstand me, i don't like use plugins to hide the presence of OllyDbg, in particular way when we play with these targets, but i suggest you to use this plugin if you have a O.S 64 bit. In this case, do it: **Plugins** -> **Stealth64** -> **Options** -> **[Misc]** check **x64 compatibility mode**!
  - **ODbgScript plugin**: in reality we will never use this plugin with this target but it's really a MUST!
- **A Pe File Analyser**: in order to take some infos on sections, Dlls and functions used by the "Victim", this type of tools is mandatory; I prefer **CFF Explorer VII** with **Resource Tweaker plugin** installed;
- **Import REconstructor 1.7c**: useful to automate the reconstruction of Import Address Table partially or totally destroyed;
- **Process Explorer v12** and **Process Monitor v 2.8:** sysinternals tools useful to 'play' with our Crackme;
- **Comdlg32.ocx**: probably if you will try to reverse this target under **Windows 7 / Vista**, you will need this file. Because Microsoft developed only a 32 bit version of this component, to install this one under 64 bit O.S, you must do:
  - Copy the file in " **%Systemroot%\SysWOW64** " (**System32** for 32 bit O.S);
  - Type in Cmd: " **regsvr32 %Systemroot%\SysWOW64\ComDlg32.ocx** ";
- **Brain**: it's really needed for this kind of passions ☺!

1.3 SECOND STEP: STUDYING THE PE STRUCTURE

Thanks to **CFF Explorer**, we have other information (see Figure 1):

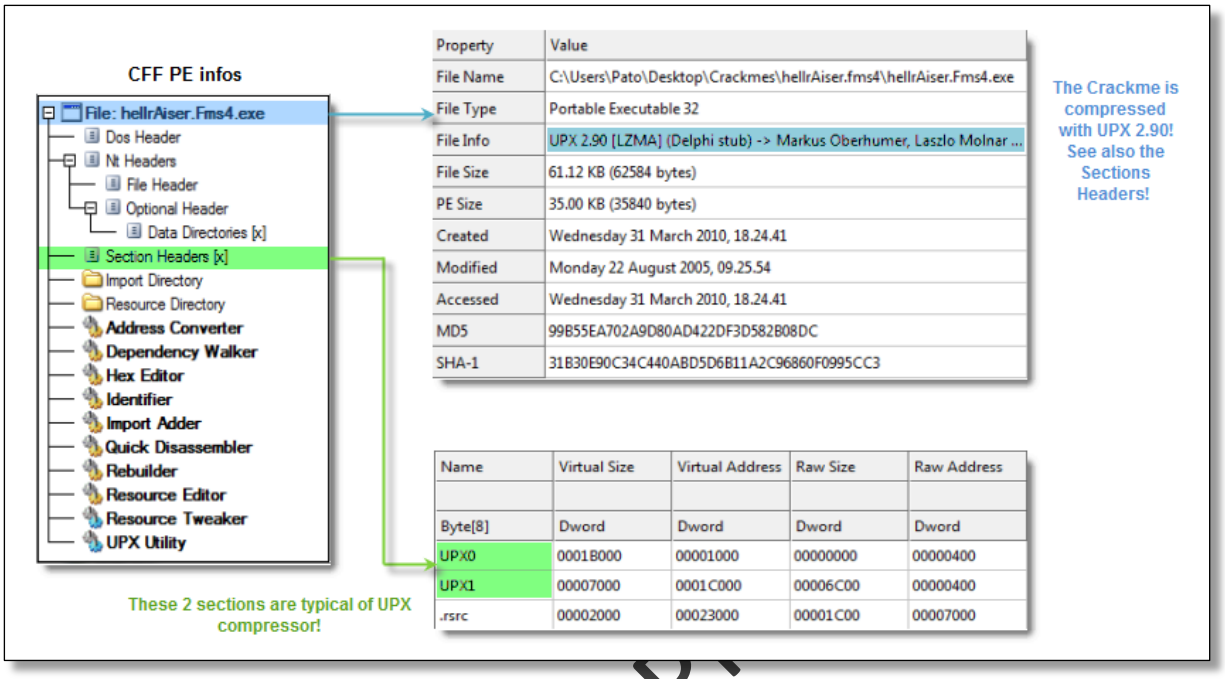

Figure 1. hellrAiser.Fms4.exe is compressed with the famous and the ... most simple UPX packer!

Well, if there's someone here who doesn't know how unpack an UPX packed target please raise your hand! Personally, I used **UPX Utility** inside the CFF Explorer and what I got is an unexpected result (see Figure 2):

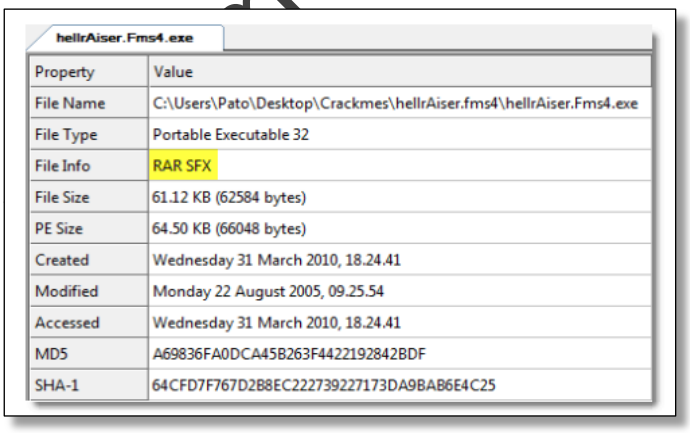

Figure 2. hellrAiser.Fms4.exe is a RAR archive!

Nice, our Crackme is indeed a self-extracting RAR archive packed with UPX! So the next step is really simple: use **WinRar** or **7Zip** to open the archive.

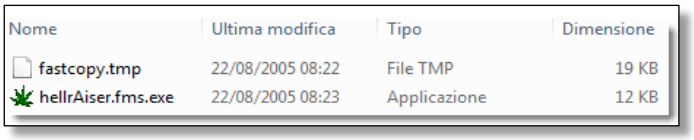

Figure 3. Result of 7Zip operation.

Bingo! We found **fastcopy.tmp** without any problem! Till there it was really a simple thing, but also a quite easy method to create a loader, without coding a thing.

Indeed we could get to this same conclusion also using the utility **Process Monitor** by Sysinternals. This tool, as its name clearly states, monitors the activity of every Process\Thread. What we do is to add a filter which instructs the application to only monitor our target and inspect its File system activity. We can see that two interesting entries soon appear:

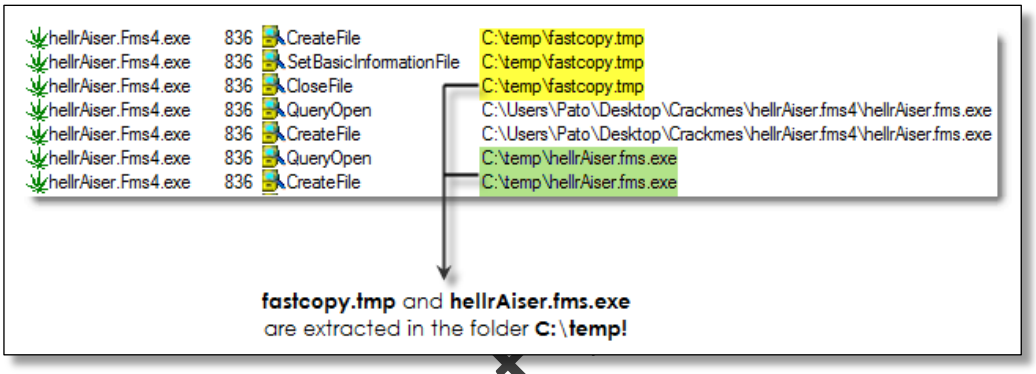

Figure 4. Snapshot of Process Monitor when it scans the activity of hellrAiser.Fms4.exe.

We therefore know that our target extracts its 2 files into **c:\temp**! On the other hand the bad news is that now we have a couple of files to investigate instead just one: double work, means twice the compensation? We will see ..

1.4 FIRST STEP: PLAYING WITH THE TARGET

If you have already read my previous tutorials (available on ARTeam web site), you should know how important is this stage; after loading the Crackme, push any buttons of the interface, enter as many serials as possible and try to understand how the target reacts! In other words you must **manually fuzz** the program in order to find few good attack points. Figure 5 clearly reports what I did to play with the proggie:

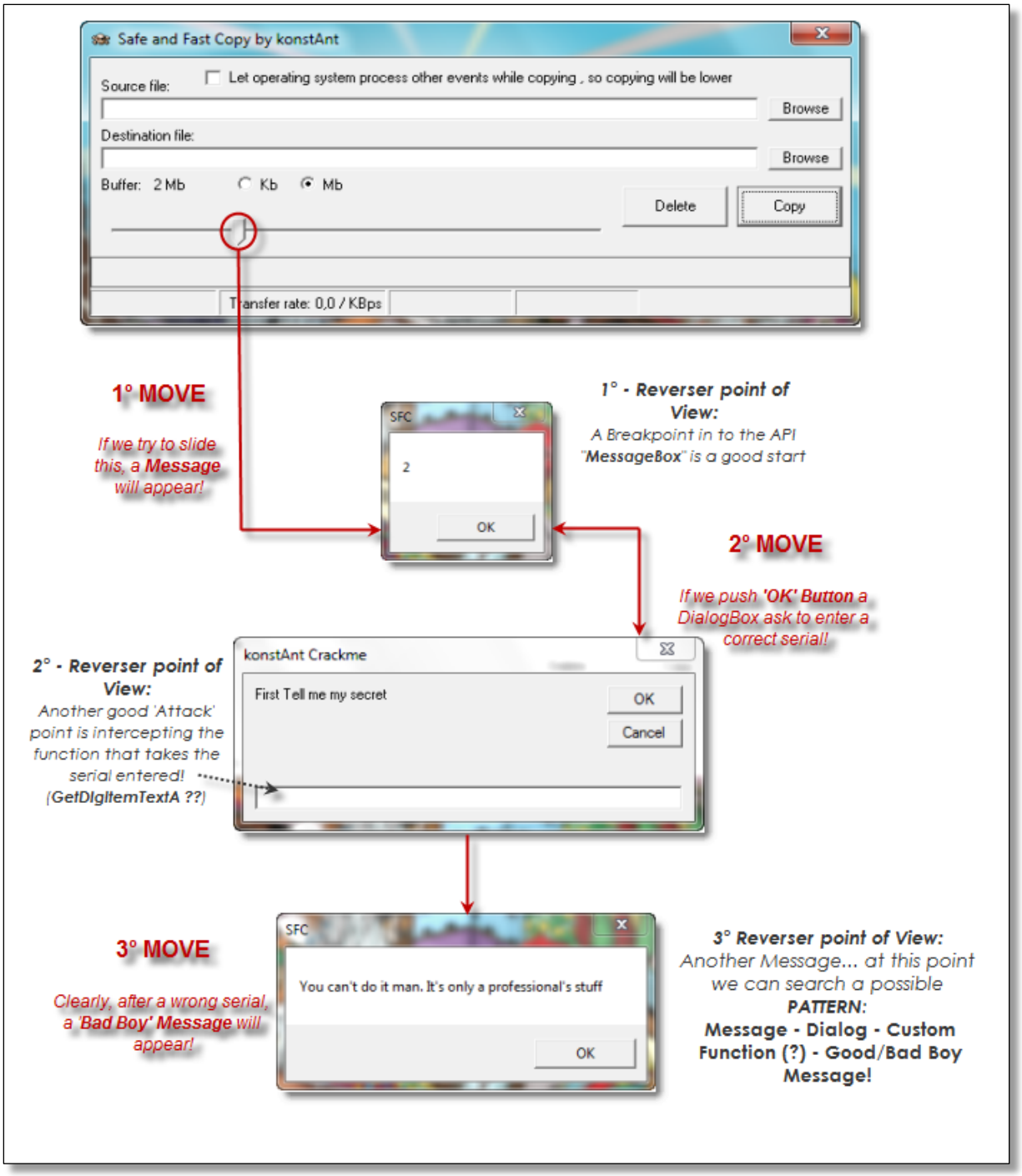

Figure 5: Searching some good point of Attack…

Note that all the assumptions you can do must be verified, anyway the most important thing is to understand the structure of the target in order to discover some weakness!

Another idea is to launch the Crackme and look how it works, for example using **Process Explorer v1.2**. Doing this we can understand if it's a multithreaded program or simply a loader (a program that creates a new child process which is executed in turn when the father dies) ☺. I was indeed quite surprised to realize that for this crackme this was case, a loader. Indeed, if we start the target and monitor it with Process Explorer (**File** -> **Run** -> ***'Name of Program'***) the following events occur:

1. In the Processes' view, as a son of the process **Procexp.exe**, you can find the process '**hellrAiser.Fms4.exe** ' (*Child process*);
2. If you continue looking at the processes list, after a little delay, a second Process appears in the root level of Process's Tree (*no Parent Process*), which is called '**fastcopy.tmp'**;
3. The ' **hellrAiser.Fms4.exe** ' process is therefore terminated;

The final result is better explained in the Figure 6:

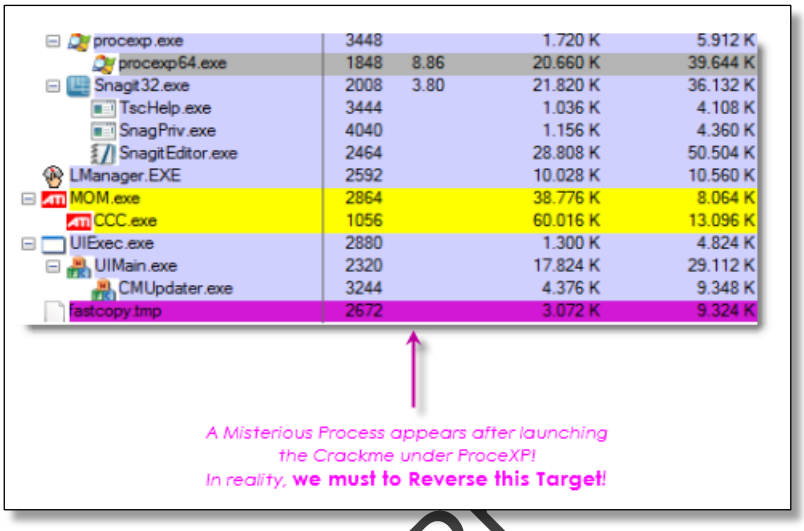

Figure 6. The 'hellrAiser.Fms4.exe' is simply a loader, the real target is 'fastcopy.tmp'.

At this stage it should be quite evident that, inside **hellrAiser.Fms4.exe** we will need to hook the calls to functions like **CreateProcess** and **ResumeThread** and only inside **fastcopy.tmp** we will search APIs like **MessageBox** and **GetDlgItemText.** Indeed, a good reverser should ask himself: "Where are stored the raw data that will belong to the new Process? Are they in a particular section inside the PE structure of the father process or simply generated by this last one, for example allocating a memory block and filling it with some Hex values?"

The typical lifecycle of loaders such these (which are also called **droppers**) is to execute a piece of code that creates and allocates a new buffer, then fill it with some blob data (which are blob by the loader's point of view), eventually decrypted them, and finally, just before terminating, turn the execution handle to that buffer.

In order to gather details to answer these questions, we will do a deeper analysis, beginning from the PE structure.