Removing Sentinel SuperPro dongle from Applications

and details on dongle way of cracking

Shub-Nigurrath of ARTeam

Version 1.0 – September 2006

Keywords

dongle, simulation, emulation, sentinel superpro

1. Abstract

Welcome back to another long tutorial of mine. This time I will focus on the Sentinel dongles on which I spent a little time recently. What I first understood are the different approaches one can follow to unprotect Sentinel protected applications (and generally speaking dongle protected apps) and that the tutorial on this subject are spread around into little, sometimes old, pieces, Most of the times difficult to understand or apply to modern applications.

Thanks to the work of people like GoatAss, CyberHeg, Crackz (just to name few) and TORO (which I here want to publicly thanks for all the long chats with him on this subject) what I will say here will not be completely new. I anyway updated their techniques, tested on a recent application and changed a little the routines you can find around, because don't work anymore.
An excellent source of material on dongles is the Woodmann's page on this subject [1].

Consider this tutorial as a whitepaper on Sentinel dongles (without having the pretence of telling everything about) where I tried to clearly explains concepts, with some new contributions as well. I focused on Sentinel because is one of the most requested and used in dongles.

Have phun,
Shub-Nigurrath

The techniques described here are general and not specific to any commercial applications. The whole document must be intended as a document on programming advanced techniques, how you will use these information will be totally up to your responsibility.
Where commercial applications are explicitly used, they are just for their protection, and no cracks details are given.

2. Possible approaches: emulations vs simulation

Generally speaking there are two possible approaches to an application protected with a dongle: emulation or simulation. I will use these two terms to distinguish the two approaches, other might not agree on the two meanings I will use, but it's better to call these two approaches just A and B ;-)
First of all I have to build a common understanding platform from where I can start the detailed discussion.

2.1. How a dongle works

I think that this subject should be already known, and here is not the place where to explain details on specific dongles, anyway few things are required to understand the general way to approach these dongles. Generally speaking an application protected by a dongle requires few components; I tried to summarize a general overview with Figure 1.

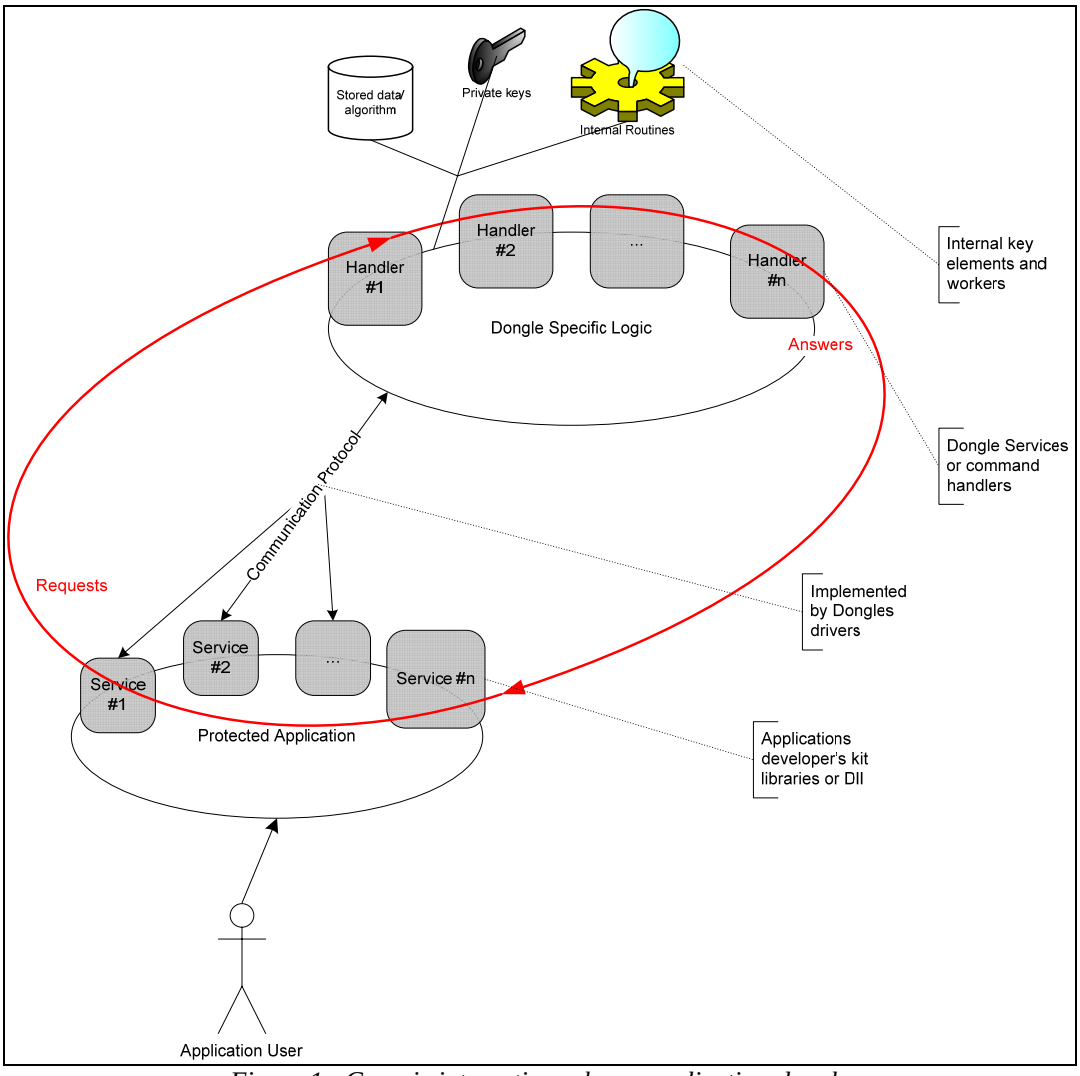

Figure 1 - Generic interaction scheme application-dongle

Going from bottom to the top of the figure, the important components are:

- The application to be protected. The application interacts locally with the dongle application developers' kit, calling its APIs. This by the application point of view is just like calling a proxy object, giving its requests and waiting for an answer or data. It's a classical design pattern, for those of you who know these software engineering objects.
- The Application Developers' library. This component is made of APIs we can assume working just like proxies: running locally into the application they are exposing a programming interface to the application and implement a communication protocol on the other side. They are just message dispatchers and requests collectors (like any other API in the world anyway). The libraries is linked to the program either through a dll or a classical library (dynamic vs static linking) and are offering to the application what I called Services (from 1 to n)
- The Communication protocol. This element requires the dongle drivers to be installed in the system, because this protocol is implemented by them.
- Dongle handlers. When the dongle receives a request it must invoke a specific request handler. The dongle's logic of course is more complex than this, but we can assume it to be just like in Figure 1. The dongle handlers are using internal dongle resources (keys, algorithm, data and CPU eventually) to answer the question posed by the application.
- Internal dongle resources. The dongle might hold keys, databases, and CPUs to execute some algorithms (this is also the case of Sentinel). The more hard to defeat is the dongle the more complex are the internal resources usually. Tracing execution of these components or reading out the data is often not possible or at least hard.

If you stop a moment thinking should become evident that there's a trusting mechanism on which the whole communication is based. Our attacks tries to subvert this trusting model (see something similar in [2]).

As you can understand, there are several possibilities to smash this trusting model between the application and the dongle. Precisely the two most important possibilities are called by me emulation and simulation.

2.2. Emulation of a dongle

The root consists in writing an external program running on the system which intercepts (hooks) requests going to the dongle drivers, build the same answers the dongle would have given and send them back exactly like the dongle would have done. There are different techniques to do this, but it's often useless to complicate life and a normal hooking of the system APIs responsible of sending requests to the dongle drivers, is enough.
TORO, but not only, released several really good working emulators, but there are other famous emulators around (e.g Glasha).
The problem with these programs is that their life is strictly connected to the specific dongle they're emulating. The other important limit is that they will require at least one time the original dongle to be inserted, to collect answers to be later emulated!

2.3. How an emulator works

Here I will use TORO's Sentinel emulator (released on several forums, like exetools). As all the emulators the first task to complete is to launch the emulation data collector which requires the original dongle to be inserted.

This collector is essentially a *dongle communication protocol recorder*: it monitors requests of the program and answers from the key the stores everything, see Figure 2.

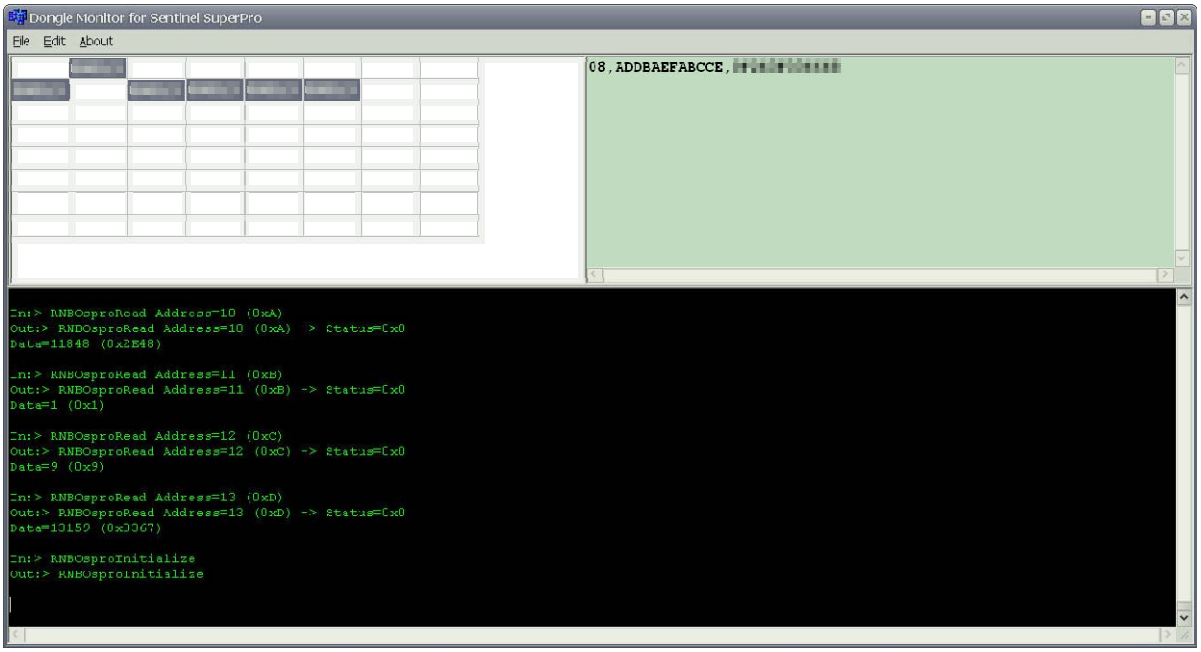

Figure 2 - Dongle Monitor for Sentinel SuperPro: collecting requests from the application and answers from the dongle

The mechanism is quite simple; we have two well known hooks to grip: the dongles API called by the application and the driver. This program to efficiently work must hook just before Windows passes the requests to the ring0 driver, and indeed this is what it does.
As you can see the application calls several functions, belonging to the Sentinel programming interface (see [3]), we will discuss about them later.

The second component is what I call a *dongle communication protocol player* which plays information collected at step before. This is done with the same hooking technique, in the opposite direction of course.

You can see the advantages of this technique, it doesn't modify the application just kip the problem. On the other hand there are some limitations: you need at least once the original dongle and the emulated data will contain your dongle data and eventually serials. It's good but sometimes it's not..

There are two other more advanced techniques you can use to write an emulator. These consist in writing a Ring0 driver emulator, emulating at all the driver's functionalities and again at Ring0 writing a filter driver which intercepts the calls to the real dongle driver. The logic behind is anyway always the same, intercept the calls to the dongle (at Ring3 or Ring0) and answers back, using stored data, what the real dongle would have answered.

There's another complication you might find for some applications: dongles' companies started to add packets encryption/decryption to the communication between application and dongle. The packet is encrypted by the application before sending it to the dongle, and the answer too, when comes back to the application.

Writing an emulator for these dongles means also emulating these algorithms. In these cases the only valid approaches are the reverse of the algorithm, correlating packets or the ripping of encrypt/decrypt algorithms from the application.

2.4. How a Simulator works

Emulating a dongle means including into the protected application the code required to emulate the answers the dongle would give to the application's requests, so as to free the application from the need of any external thing: dongle, driver, simulators. The patched application will be again a normal application. To do this we will require patching the piece of the protection schema inside the application (see Figure 1).
I summarized the approach in Figure 3.

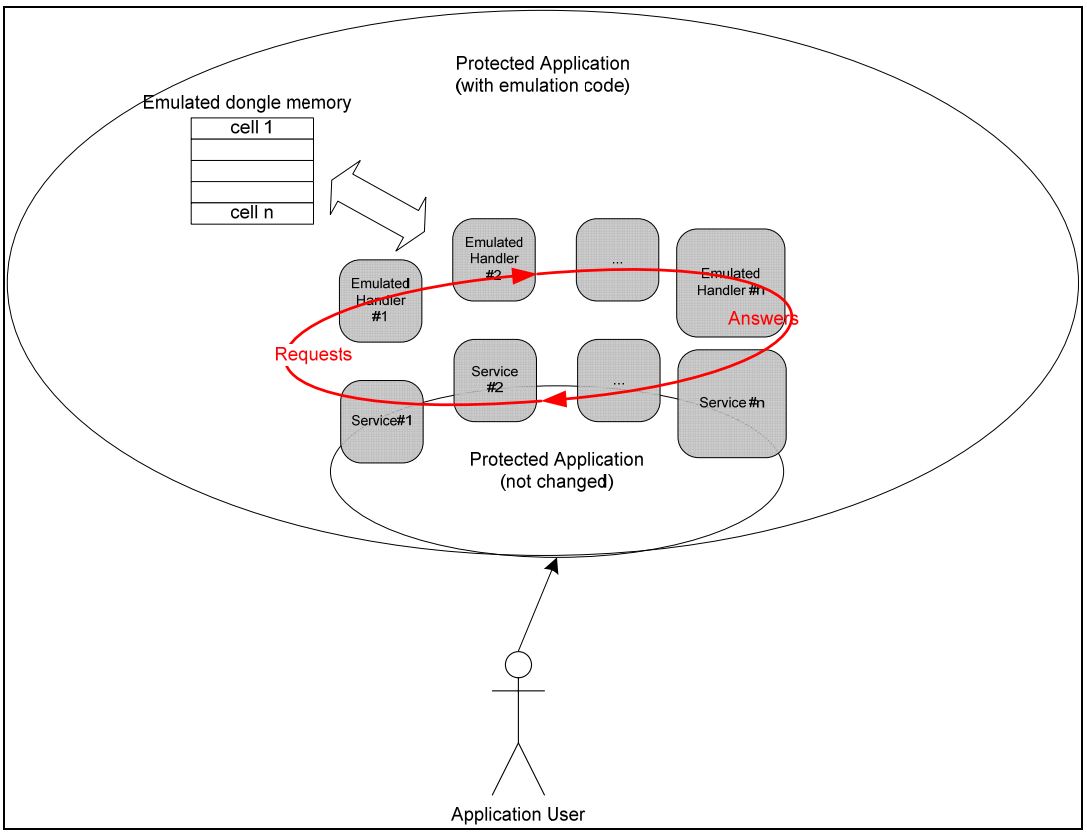

Figure 3 - Simulation of a dongle into the application

Figure 3 clearly shows that we need an Emulated dongle memory, inside the patched application, and some routines which emulate the original service handlers: the big difference is that the original answers will be read from the emulated memory instead of from the dongle. The application outside its dongle interactions parts will be untouched.

I will concentrate most on this latter approach for this tutorial.