Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking
Shub-Nigurrath of ARTeam
Version 1.0 – September 2006

1. Abstract.................................................................................................................... 2
2. Possible approaches: emulations vs simulation .................................................. 3
   2.1. How a dongle works .............................................................................................. 3
   2.2. Emulation of a dongle .......................................................................................... 4
   2.3. How an emulator works ...................................................................................... 4
   2.4. How a Simulator works ..................................................................................... 6
3. Disassembling a Sentinel Protected Program ...................................................... 7
   3.1. Disassembling with IDA ..................................................................................... 7
   3.2. Disassembling with OllyDbg ............................................................................. 10
4. Some details on the Sentinel Applications’ Programming Interface ..................... 12
   4.1. What is Sentinel SuperPro .................................................................................. 12
   4.2. Structure of the key memory ............................................................................ 12
      4.2.1 Restricted and Programmable Cells ............................................................ 13
      4.2.2 Access Codes ............................................................................................... 14
      4.2.3 Cell Types .................................................................................................... 14
   4.3. API Function reference ..................................................................................... 15
5. Re-writing Sentinel APIs ....................................................................................... 17
   5.1. sproFormatPacket .............................................................................................. 17
   5.2. sproFindFirstUnit ............................................................................................. 19
   5.3. sproOverwrite ................................................................................................... 20
   5.4. sproFindNextUnit ............................................................................................. 20
   5.5. sproRead ........................................................................................................... 20
      5.5.1 sproRead Approach #2 .................................................................................. 24
   5.6. sproQuery .......................................................................................................... 25
6. What’s more .......................................................................................................... 29
7. References ............................................................................................................. 30
8. Conclusions .......................................................................................................... 31
9. History ................................................................................................................... 31
10. Greetings .............................................................................................................. 31

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.

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.
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

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.

![Dongle Monitor for Sentinel SuperPro: collecting requests from the application and answers from the dongle](image)

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 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.
3. Disassembling a Sentinel Protected Program

As an example I used a part of a commercial application (any application from FreedomScientific is good also [4]) which is protected with a Sentinel SuperPro dongle. The whole application is not important, which is important here is that it also installs a standalone dongle license verification program, I included in this tutorial’s distribution. This program is for us just like a crackme..

The instruments most suited for a dongle program are IDA and OllyDbg¹ plus some signature dongle specific libraries (you can freely find them in forums, I also included those used here into this tutorial’s distribution) which reports where the program is calling the dongle services proxies.

3.1. Disassembling with IDA

For the moment, just to get used to IDA and the code structure we will concentrate on showing the concepts explained till here.

The original and patched programs look like in Figure 4.

![Figure 4 - Original and patched dongleviewer](image)

NOTE
Before starting if you have still not done it, copy the *.sig files distributed with this tutorial, into the sig folder under the IDA installation folder.

If you somehow got IDA version 5.0 (officially, for the lucky ones, or from warez boards) fire up your local copy: drag and drop the original dongleviewer.exe and press OK. IDA will start analysis of the program. Take a short break and wait till it finishes.

¹ Indeed thanks to some Ollydbg plugins, we will use later, which are able to read .sig files, IDA is not strictly needed. It’s superior code analyzing features are anyway handy.
Once IDA is ready go to the Libraries folder, like in following snapshot:

If the tag is not present you can use the flower button 🌸.

Select the tab and press right-click (or key Ins) and add the new signatures you copied before. I suggest adding them all to be sure, since you exactly do not still know which version of the SDK has been used to create the application. You should have something similar to Figure 5, given the highest number of matches it seems that the application has been compiled using version 6.x (not 6.2) of the Sentinel SDK. Moreover there are no external dlls so the SDK used has been statically linked to the application.

![Figure 5 – Applied IDA signatures](image)

Now you should see the result of the names applied into the Names tab: what IDA did is to assign library names to code functions.

According to IDA help this is the legend of the names:

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>library function</td>
</tr>
<tr>
<td>F</td>
<td>regular function</td>
</tr>
<tr>
<td>C</td>
<td>instruction</td>
</tr>
<tr>
<td>A</td>
<td>ASCII string</td>
</tr>
<tr>
<td>D</td>
<td>Data</td>
</tr>
<tr>
<td>I</td>
<td>imported name</td>
</tr>
</tbody>
</table>

Anticipating the following chapters we will focus on one of the Sentinel APIs: sproRead. What I would focus on now is the call graph.

In the Strings tab you should search for the driver which is effectively involved:

```
.text:00476800  00000011 C  \\\\\\SENTINEL.VXD
```

---

2 You should see that IDA already applied the compiler specific signatures.
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

Double click on it, press x to find references to that symbol and you will land into the function I386SPRO500MSOFTCIA(x) which the signatures files identified for us (see Figure 6)\(^3\).

![Figure 6 schema of I386SPRO500MSOFTCIA(x)](image)

Now just press to generate the graph of the functions references to I386SPRO500MSOFTCIA(x). The result is quite huge, but what I would point your attention to is the base of the graph: IDA colorizes for us the functions coming from signature files (light blue), thus because we only applied Sentinel signatures, these are the library functions of Sentinel dongle, used by the program or just linked to it (remember static linking of a *.lib file links it all, depending on the linker options).

You also have to be sure of this thing: if IDA reports that a function is not referenced this doesn’t always means that it will not be called at runtime, there’s the possibility of dynamic messages, changing code and errors in IDA parsing engine. Anyway it’s a result to consider or a point to explore after.

Figure 7 shows the base part of this graph: note that I386SPRO500MSOFTCIA is called by other internal functions and then entry library functions such as sproExtendedRead (which is not used by the program but it’s inside it, as wasted code) and sproQuery or sproRead (which is instead heavily used, and cannot be otherwise).

\(^3\) If you do see the classical code view (pres IDA 5.0), right click on any empty space of the code view and select “Graph View”
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

This type of analysis is of big help for us because tells on which library calls we will have to work. This same information could have been given by the monitor, as explained in Section 2.3. But, you must understand that a monitor only monitors calls effectively done by the program and not all the possible calls. The risk is that if something is not called during your simulation (a function not called, a menu not activated) you will do incomplete patches. Crossing the two approaches is safer and gives you higher probability to do a complete patch.

NOTE
You can understand how a Simulator might hook the system, intercepting calls to CreateFileA.

3.2. Disassembling with OllyDbg

It is also possible to use OllyDbg to disassemble the program, even without IDA. You will find extremely useful the GODUP plugin (included into this archive), see Figure 8.

This plugin is able to read the IDA .sig files\(^4\) and import function names into the disassembled program\(^5\), see Figure 9.

\(^4\) Indeed not all the possible types, but the Sentinel sig here used are supported.
\(^5\) You will have to insert the signatures path manually.
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

Apply all the signatures\(^6\) as before and press CTRL-N to see the names Olly has defined for the current program, search for names starting with “spro” (all the top level Sentinel functions start with these letters) and you will find, among others the following:

<table>
<thead>
<tr>
<th>Address</th>
<th>Type</th>
<th>User</th>
<th>Function</th>
<th>Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00476F50</td>
<td>text</td>
<td>User</td>
<td>sproFindNextUnit</td>
<td>1 argument</td>
</tr>
<tr>
<td>00476F6F</td>
<td>text</td>
<td>User</td>
<td>sproFormatPacket</td>
<td></td>
</tr>
<tr>
<td>00477639</td>
<td>text</td>
<td>User</td>
<td>sproGetVersion</td>
<td></td>
</tr>
<tr>
<td>00477519</td>
<td>text</td>
<td>User</td>
<td>sproOverwrite</td>
<td></td>
</tr>
<tr>
<td>004776F0</td>
<td>text</td>
<td>User</td>
<td>sproQuery</td>
<td></td>
</tr>
</tbody>
</table>

As you can see there’s no comparison between the few functions recognized with GODUP or with IDA: IDA wins with around 300 recognitions vs about 50 of OllyDbg.

The best alternative is then to export the MAP file from IDA and import to OllyDbg using this same plugin\(^7\), see Figure 10. This last way of work gives the best results at all!

---

\(^6\) Remember to check “Overwrite existing labels”

\(^7\) There’s another plugin doing the same importing work, but it’s much much more slower than GODUP.
4. Some details on the Sentinel Applications’ Programming Interface

We did enough for the moment, time to stop a little and briefly see how the Sentinel’s PDK works. I will take information from the document [3], you should read it just after this tutorial to complete your training path.

**NOTE**

Generally speaking if you want to patch a dongle protected application the really first thing to do is to read the developers’ manual to understand what the application might use and how to use the dongle services.

4.1. What is Sentinel SuperPro

In addition to providing you with a full-featured, easy-to-use software protection system, Sentinel SuperPro gives you the ability to increase demo limits, upgrade demos to fully-licensed versions and provide access to additional features all without having to ship a new hardware key or visit the customer’s site. Sentinel SuperPro 6.1 provides you with an added capability—the ability to allow your customers to use one key for multiple clients. Sentinel SuperPro 6.1 also allows you to program keys specifically for use by your distributors, so you can limit how many product keys they can activate and update. Sentinel SuperPro hardware keys come in two form-factors: parallel port or USB, see Figure 11.

![Figure 11 - Sentinel SuperPro keys](image)

4.2. Structure of the key memory

Every Sentinel SuperPro key contains 128 bytes of memory, organized as 64 cells (words) of 16 bits each. Cells are addressed as locations 0 through 3F hex, see Figure 12.

When you program a cell, you assign it various attributes. These attributes determine how the cell (and the word it contains) is used by your application. Cell attributes include the cell type, the access code and the cell value.
Generally, each cell contains one of the following types of words:

- **Data Words:** A data word can store data such as sublicenses, customer information, serial numbers, passwords, and check digits. You code your application to read the word and then evaluate and act upon the stored value. A data word cell may be programmed as read-only or read/write.

- **Counter Words:** A counter word contains an initial value you set that is then decremented by your application. A typical use of a counter word is to limit the number of times a demo application can be executed.

- **Algorithms:** An algorithm contains a bit pattern that defines how the hardware key should encrypt query data sent by your application. The key uses an algorithm—plus an internally stored proprietary algorithm—to transform the query data and then return a value to your application. You design your application to send queries to the key and then evaluate and act upon the responses.

Algorithms are **active** or **inactive**. Only active algorithms can return a valid response to a query. The **active/inactive bit** in the cell value controls whether or not the algorithm is active.

### 4.2.1 Restricted and Programmable Cells

Cells 00 through 07 in each key are restricted cells that contain fixed, preprogrammed system information, see Figure 13.

<table>
<thead>
<tr>
<th>Cell</th>
<th>Contents</th>
<th>Readable?</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Key serial number; sequentially assigned per key.(^a)</td>
<td>Yes</td>
</tr>
<tr>
<td>01</td>
<td>Developer ID; unique to your company/product.</td>
<td>Yes</td>
</tr>
<tr>
<td>02 – 07</td>
<td>Reserved for use by Rainbow Technologies.</td>
<td>No</td>
</tr>
</tbody>
</table>

Figure 13 – Sentinel SuperPro Key Restricted Cells Contents

\(^a\) serial number are on 16 bit and not guaranteed to be unique

Cells 08 through 3F are available for programs.
4.2.2 Access Codes

Every cell has an access code associated with it that controls how the cell can be used by your application—it defines the cell’s cell type attribute. For example, some cell types have an access code that permits cell values to be both read and overwritten, while others are read-only or not writable at all. Access codes are numbers from 0 to 3, see Figure 14.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0    | Read/write data word  
Your application can read the word in the cell and, if the write password is supplied, modify its contents. |
| 1    | Read-only (locked) data word  
Your application can read the word in the cell, but cannot change it without the overwrite passwords. |
| 2    | Counter word  
The cell contains a word (value) that your application can decrement using the write password. The cell’s value cannot be changed (other than by decrementing it) without the overwrite passwords. |
| 3    | Locked and hidden/algorithm word  
Your application cannot read the cell’s value. Modification requires the overwrite passwords. The cell value (contents) is hidden (unreadable). |

Figure 14 – Sentinel SuperPro Key Cell Access Codes

4.2.3 Cell Types

Each cell is assigned a code that defines how you want to use the selected cell. This code is called a cell type. The cell type classifies the type of data stored in the cell, which in turn affects how the cell can be used. Each cell type is identified by a two-letter abbreviation; for example, CW identifies a counter word. Some cell types are designed to be used in groups. For example, algorithms can have counters and passwords associated with them. Other cell types have address restrictions, meaning they can be assigned only to specific cells on the key, see Figure 15.

<table>
<thead>
<tr>
<th>Cell Type</th>
<th>Access Code</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>0</td>
<td>Undefined</td>
</tr>
<tr>
<td>AA</td>
<td>3</td>
<td>Active Algorithm</td>
</tr>
<tr>
<td>AH</td>
<td>3</td>
<td>Algorithm Half</td>
</tr>
<tr>
<td>AP</td>
<td>3</td>
<td>Activation Password</td>
</tr>
<tr>
<td>CA</td>
<td>2</td>
<td>Algorithm Counter Word</td>
</tr>
<tr>
<td>CW</td>
<td>2</td>
<td>Counter Word</td>
</tr>
<tr>
<td>DI</td>
<td>1</td>
<td>Developer ID</td>
</tr>
<tr>
<td>DL</td>
<td>1</td>
<td>Locked Data Word</td>
</tr>
<tr>
<td>DW</td>
<td>0</td>
<td>Data Word</td>
</tr>
<tr>
<td>IA</td>
<td>3</td>
<td>Inactive Algorithm</td>
</tr>
<tr>
<td>RW</td>
<td>3</td>
<td>Reserved Word</td>
</tr>
<tr>
<td>SN</td>
<td>1</td>
<td>Serial Number</td>
</tr>
</tbody>
</table>

Figure 15 - Sentinel SuperPro Key Cell Types
### 4.3. API Function reference

The main API used for the Sentinel SDK are reported in Figure 16.

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sproActivateO</td>
<td>Activates an inactive algorithm so it can be used by the sproQuery() function.</td>
</tr>
<tr>
<td>sproDecrementO</td>
<td>Decrements a counter word or read/write data word by one. If the counter is associated with an active algorithm, decrementing to zero deactivates the algorithm.</td>
</tr>
<tr>
<td>sproEnumServer()</td>
<td>Enumerates the number of servers running on the network, according to the specified developer ID.</td>
</tr>
<tr>
<td>sproExtendedRead()</td>
<td>Reads the value and access code of any unhidden memory cell in the key.</td>
</tr>
<tr>
<td>sproFindFirstUnit()</td>
<td>Searches all attached keys for a specified developer ID.</td>
</tr>
<tr>
<td>sproFindNextUnit()</td>
<td>Searches for the next key with the same developer ID.</td>
</tr>
<tr>
<td>sproFormatPacketO</td>
<td>Validates the size of the packet (RNBO_SPRO_APIPACKET) and initializes field defaults. <em>This function must be called once before any other API function is called.</em></td>
</tr>
<tr>
<td>sproGetContactServer()</td>
<td>Returns the contact server set for a particular API packet.</td>
</tr>
<tr>
<td>sproGetFullStatus()</td>
<td>Returns extended status information. It is provided for support purposes only.</td>
</tr>
<tr>
<td>sproGetHardLimit()</td>
<td>Retrieves the maximum number of licenses supported by the hardware key (the hard limit).</td>
</tr>
<tr>
<td>sproGetKeyInfo()</td>
<td>Gets information about the key from a particular server.</td>
</tr>
<tr>
<td>sproGetSubLicense()</td>
<td>Finds a sublicense in a particular cell.</td>
</tr>
<tr>
<td>sproGetVersion()</td>
<td>Returns the Sentinel SuperPro driver's version number.</td>
</tr>
<tr>
<td>sproInitialize()</td>
<td>Performs any required initialization of the driver.</td>
</tr>
<tr>
<td>sproOverwrite()</td>
<td>Changes the value and/or access code of any cell except the reserved cells 00-07.</td>
</tr>
<tr>
<td>sproQuery()</td>
<td>Sends a data string to the key, encrypts it using a specified algorithm, and returns the encrypted string to the application.</td>
</tr>
<tr>
<td>sproRead()</td>
<td>Reads the value of any unhidden cell in the key.</td>
</tr>
<tr>
<td>sproReleaseLicense()</td>
<td>Releases a license by specifying the cell address as zero, or releases a sublicense from a particular cell by specifying the cell address of the sublicensing cell as well as the number of sublicenses to be released.</td>
</tr>
</tbody>
</table>
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sproSetContactServer()</td>
<td>Sets the contact server for a particular API packet.</td>
</tr>
<tr>
<td>sproWrite()</td>
<td>Changes the value and/or access code of any cell with an access code of 0 (read/write data)</td>
</tr>
</tbody>
</table>

*Figure 16 - main Sentinel Superpro APIs*

Officially the SDK reports its names with the prefix “RNBO” but usually are mentioned without it. The most important by the cracking point of view are: sproFormatPacket, sproInitialize, sproFindFirstUnit, sproRead, sproQuery, sproOverwrite.

This is a typical calling sequence, you might find often in the programs:

1. sproFormatPacket() – Initializes the packet.
2. sproInitialize() – Performs required initialization.
3. sproFindFirstUnit() – Establishes communication with the key and gets a license.
4. sproRead() – Reads the cell and returns the value in it.
5. sproQuery() – Sends the query string and points to a location for the response value.

For example the sproRead looks like in Figure 17, we will compare this with the new one we are going to write.
What is important are the return values of the APIs, what the program expect from a valid call.

5. Re-writing Sentinel APIs
Now starts the real part of the tutorial for those already experts of the subject. Till now the tutorial covered already known things, required for completely new readers in order to get a satisfactory level of knowledge. The real patching starts here.

5.1. sproFormatPacket
Given that we will simulate all the other APIs as well, usually the only important thing for this API is that it always returns **SP_SUCCESS(0)**.
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

Figure 18 reports the original API from IDA, Figure 19 reports the same from OllyDbg.
The patch is quite simple as reported in Figure 20, it only zero out [ESP+C], the return value. As you can see I started to overwrite the original API at the first jump, at 0x476EF8, this is intentional and used to skip some elementary anti-tampering checks on Sentinel APIs entrypoints.

A Note, the PDK reports the following description about this API:

This function essentially acts as a “get license” call. If the Sentinel SuperPro key is found, the RB_SPRO_APIPACKET record will contain valid license data, otherwise, the packet will be marked invalid. If you try to call this function on an APIPACKET that already has a license, the SP_INVALID_OPERATION error is returned.

It returns a value different from SP_SUCCESS(0) only when the given parameter RB_SPRO_APIPACKET is invalid. This type of errors is typical at developing stage, thus already resolved for us. This means that for usually this API should not be patched, I anyway usually patch it just to avoid surprises.

5.2. sproFindFirstUnit

Also for this API, the important thing is to return 0 (SP_SUCCESS). One thing can be anyway useful to store away, the developerID which is developer’s specific, as reported by the Developers’ guide:

```c
unsigned short int sproFindFirstUnit(
    RB_SPRO_APIPACKET packet,
    unsigned short int developerID
);
```

developerID, is assigned to you by Rainbow Technologies or your distributor. It identifies the Sentinel SuperPro device to search for.
If you place a Breakpoint at the beginning of the sproFindFirstUnit, like any other API receiving a RB_SPRO_APIPACKET packet as first parameter, you will notice that this packet is just the ASCII string “Br”.

5.3. sproOverwrite

This function is used to change the value and access code of any word in the Sentinel SuperPro key. For us the only patch is to return a SP_SUCCESS, just to let the program go over. Figure 22 shows some apparently useless instructions; they are useless effectively, just for debugging purposes.

![Figure 22 - Patched sproOverwrite](image)

5.4. sproFindNextUnit

This API finds the next Sentinel SuperPro key based on the developer ID, found calling sproFindFirstUnit, it is often called by applications to correctly handle dongles and must then be patched as the sproFindFirstUnit, see Figure 23.

![Figure 23 - Patched sproFindNextUnit](image)

5.5. sproRead

This and the following sproQuery are the two most important functions we need to emulate. All the previous APIs have been patched in order to allow the program to freely and without errors come to these APIs. Inside these two APIs it is most of the work left.

The original prototype is:

```c
unsigned short int sproRead(
    RB_SPRO_APIPACKET packet,
    unsigned short int address,
    unsigned short int *data
);
```
Removing Sentinel SuperPro dongle from Applications and details on dongle way of cracking

where
- **packet**: a pointer to the RB_SPRO_APIPACKET record.
- **address**: the Sentinel SuperPro key memory cell address of the word to read.
- **data**: a pointer to the location that will contain the data read from the Sentinel SuperPro key.

Figure 17 reports the original sproRead. What we are going to do is to emulate the dongle memory locally, using the same space previously used by the original sproRead. The overall schema is reported in Figure 24: the new sproRead will read from a simulated memory region the values the original API would have returned to the calling program. The data input parameter is used as an index to get the correct value from the local memory.

Figure 24 - Schema of Patched sproRead

The function is reported below. The approach is quite similar to those already presented by Crackz, GoatAss and others, but their solutions were not working or erroneous or not complete (at least for my experience), due to misalignments in emulated memory sections: they were not correctly considering size of single cells. If you need more emulated memory reduce the offset added to EAX at address 0x00477308, as you can see there’s a lot of space still available.

```assembly
00477300  >$ 56  PUSH ESI  ; sproRead
00477301  | 57  PUSH EDI
00477302  | R8 00000000 CALL DongleVI.00477307 ; call itself body, used to ...
00477307  |$ 58  POP EAX  ; have EAX pointing to next instruction
00477308  | 83C0 59 ADD EAX,59  ; start of new sproRead, EAX points to itself offset base of simulated memory
0047730B  | 034424 10 ADD EAX,DWORD PTR SS:[ESP+10] ; adds the data offset, pay attention all is DWORD based
0047730F  | 90  NOP  ; read the memory cell
00477310  | 8B08  MOV ECX,DWORD PTR DS:[EAX] ; EAX contains the read cell
00477312  | 91  XCHG EAX,ECX  ; adjust return values and returns
00477313  > 8B4C24 14 MOV ECX,DWORD PTR SS:[ESP+14]
```

Version 1.0 2008  Page 21 of 31
Removing Sentinel SuperPro dongle from Applications and details on dongle way of cracking

```assembly
00477317  |  66:36:8901  MOV  WORD PTR SS:[ECX],AX   ; copy AX to parameter passed to function (address)
0047731B  |  33C0  XOR  EAX,EAX                        ; set EAX=SP_SUCCESS
0047731D  |  90  NOP
0047731E  |  90  NOP
0047731F  |  90  NOP
00477320  |  90  NOP
00477321  |  5F  POP  EDI
00477322  |  5E  POP  ESI
00477323  \.  C2 0C00  RETN 0C            ; returns to caller
00477326  90  NOP
00477327  90  NOP
00477328  90  NOP
00477329  90  NOP
0047732A  90  NOP
0047732B  90  NOP
0047732C  90  NOP
0047732D  90  NOP
0047732E  90  NOP
0047732F  90  NOP
00477330 >  90  NOP
00477331  90  NOP
00477332  90  NOP
00477333  90  NOP
00477334  90  NOP
00477335  90  NOP
00477336  90  NOP
00477337  90  NOP
00477338  90  NOP
00477339  90  NOP
0047733A  90  NOP
0047733B  90  NOP
0047733C  90  NOP
0047733D  90  NOP
0047733E  90  NOP
0047733F  90  NOP
00477340  90  NOP
00477341  90  NOP
00477342  90  NOP
00477343  90  NOP
00477344  90  NOP
00477345 >  90  NOP
00477346  90  NOP
00477347  90  NOP
00477348  90  NOP
00477349  90  NOP
0047734A  90  NOP
0047734B  90  NOP
0047734C  90  NOP
0047734D  90  NOP
0047734E  90  NOP
0047734F  90  NOP
00477350  90  NOP
00477351  90  NOP
00477352  90  NOP
00477353  90  NOP
00477354  90  NOP
00477355  90  NOP
00477356  90  NOP
00477357  90  NOP
00477358  90  NOP
00477359  90  NOP
0047735A  90  NOP
0047735B  90  NOP
0047735C  90  NOP
0047735D  90  NOP
0047735E  90  NOP
0047735F  90  NOP
00477360 > FF  DB  FF          ; first cell of simulated memory
00477361  FF  DB  FF
00477362  FF  DB  FF
00477363  FF  DB  FF
00477364  FF  DB  FF
00477365  FF  DB  FF
00477366  FF  DB  FF
00477367  FF  DB  FF
```
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

Now that we have an sproRead simulator, we still need to fill the local memory cells. This last step is totally target dependant, and must be obtained checking what the program does of the returned values. It’s one of the most boring things because you must trace the program and what it does with returned data.

I suggest the following approach:

1. fill all the memory cells with FF values, easy to follow
2. place a BP at the beginning of the patched sproRead and trace till it reaches the corresponding memory cell
3. write the value of data (value at [ESP+10]) in the cell: for example if accessed cell is 0A, write 0A 0A as dummy WORD value of the 0Ath cell.
4. trace the written memory into the caller with HW Breakpoints till you see what the program does with that value. Usually you will find a CMP or similar.

Figure 25 - Loader part of Patched sproRead
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

5. Write back to the memory cell the real value you found, instead of the dummy one. Remember to write the WORD values using the LSB logic.

Code above already contains what I found for the DongleViewer program. As you can see not all the cells have been used by the program because they’re still left to FF.

For example the program DongleViewer calls sproRead with these data values: 10h, 11h, 12h, 17h, 13h, 14h

Only the cell 13h is directly checked by this program. Here:

Hence the program joins AX with the ECX value read before, and obtains a value such as FFFF13FF. The value is then directly checked here.

It becomes evident that the correct value for cell 13 is 0001 (considering the WORD alignment), the value is then stored at 0x00477373 in reverse order (considering the LSB logic) as 0100.

NOTE
Try experimenting, at address 0x0042C590 you can see that the program checks this memory section with different values, try changing the simulated memory cell 13h and see what happens.

5.5.1 sproRead Approach #2

I also used this other approach for the sproRead, which is more stable in some cases and also correctly cleans memory, so for tracing is a better solution. The logic is almost the same so you can read and understand it on your own (note that the memory section below reported is not initialized).
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

005D4A6A | B4C24 14        MOV ECX,DWORD PTR SS:[ESP+14]
005D4A6E | 66:36:8901      MOV WORD PTR SS:[ECX],AX
005D4A72 | 33C0            XOR EAX,EAX
005D4A74 | 90              NOP
005D4A75 | 5F              POP EDI
005D4A76 | 5E              POP ESI
005D4A77 | C2 0C00         RETN 0C
005D4A79 | 90              NOP
005D4A7B | 90              NOP
005D4A7D | 90              NOP
005D4A7E | 90              NOP
005D4A7F | 90              NOP
005D4A80 | 90              NOP
005D4A81 | 90              NOP
005D4A82 | 90              NOP
005D4A83 | 90              NOP
005D4A84 | 90              NOP
005D4A85 | 90              NOP
005D4A86 | 90              NOP
005D4A87 | 90              NOP
005D4A88 | 90              NOP
005D4A89 | 90              NOP
005D4A8A | 90              NOP
005D4A8B | 90              NOP
005D4A8C | 90              NOP
005D4A8D | 90              NOP
005D4A8E | 90              NOP
005D4A8F | 90              NOP
005D4A90 | FFFF            ???
005D4A92 | FFFF            ???
005D4A94 | FFFF            ???
005D4A96 | FFFF            ???
005D4A98 | FFFF            ???
005D4A9A | FFFF            ???
005D4A9C | FFFF            ???
005D4A9E | FFFF            ???
005D4AA0 | FFFF            ???

<omissis>

005D4A96 | FFFF            ???
005D4A98 | FFFF            ???
005D4A9A | FFFF            ???
005D4A9C | FFFF            ???
005D4A9E | FFFF            ???
005D4AA0 | FFFF            ???
005D4ABF | 90              NOP

5.6. sproQuery
The sproQuery is a little more difficult to write, it’s always the most complex function to reverse. The API simply passes data to an algorithm or a Look-up Table stored on the key and returns a value, which is then checked by the application.

You can find the correct values if you have a valid key of the program, tracing the answers, or if you do not have any valid dongle, tracing the application and guessing what it does with returned data (follow the same approach of sproRead).

```
unsigned short int RNBOsproQuery(RB_SPRO_APIPACKET packet,
unsigned short int address,
VOID *queryData,
```
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

VOID *response,
unsigned long *response32,
unsigned short int length
);

Where:

- **Packet**: a pointer to the RB_SPRO_APIPACKET record.
- **Address**: the address of the word to query.
- **QueryData**: the pointer to the first byte of the query bytes.
- **Response**: the pointer to the first byte of the response bytes.
- **Response32**: the pointer to the location that will contain a copy of the last four bytes of the query response.
- **Length**: this is the number of query bytes to send to the active algorithm and also the length of the response buffer.

If successful, the function returns SP_SUCCESS (0).

The original sproQuery function is reported in Figure 26. The patched API is instead the following:

```
004776F0 >  53                  PUSH EBX                 ; sproQuery
004776F1    56                  PUSH ESI
004776F2    57                  PUSH EDI
004776F3    8B4424 10           MOV EAX,DWORD PTR SS:[ESP+10] ; point to packet
004776F7    0BC0                OR EAX,EAX ; prepare return 0
004776F9    8B7C24 1C           MOV EDI,DWORD PTR SS:[ESP+1C] ; points to response
004776FD    B8 0DC593CB         MOV EAX,CB93C50D
00477702    36:8907             MOV DWORD PTR SS:[EDI],EAX ; copy EAX to
; queryvalue address
; the first values
00477705 >  B8 6DFAADDB         MOV EAX,DBADFA6D
0047770A    36:8947 04          MOV DWORD PTR SS:[EDI+4],EAX ; copy to queryvalue
; the higher values
0047770E    90                  NOP
0047770F    90                  NOP
00477710    90                  NOP
00477711    90                  NOP
00477712    90                  NOP
00477713    90                  NOP
00477714    90                  NOP
00477715    90                  NOP
00477716    90                  NOP
00477717    90                  NOP
00477718    90                  NOP
00477719    33C0                XOR EAX,EAX
0047771B    5F                  POP EDI
0047771C    5E                  POP ESI
0047771D    5B                  POP EBX
0047771E    C2 1800             RETN 18
```

The logic is the same of Figure 24, but the implementation different.

When the program is stopped at 0x004776F0 the data stack is the following:

```
packet    ESP+4  0013F20C  003P5ACC  ASCII "Br"
address   ESP+8  0013F210  00000008
querydata ESP+C  0013F214  0051412C DongleVi.0051412C
```
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

<table>
<thead>
<tr>
<th></th>
<th>ESP+10</th>
<th></th>
<th>ESP+14</th>
<th></th>
<th>ESP+18</th>
<th></th>
<th>ESP+20</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>response</td>
<td>0013F218</td>
<td></td>
<td>0013F21C</td>
<td></td>
<td>0013F220</td>
<td></td>
<td>0013F228</td>
<td></td>
</tr>
<tr>
<td>response32</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>nothing</td>
<td>ESP+1C</td>
<td>0013F224</td>
<td></td>
<td>00000000</td>
<td></td>
<td>0013F228</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nothing</td>
<td>ESP+20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

but, when you are at 0x004776F3 the stack becomes:

ESP ==> 0013F1FC 0051412C DongleVi.0051412C
ESP+4 0013F200 003F5A40
Packet ESP+8 0013F204 003F5ACC ASCII "Br"
Packet ESP+C 0013F208 0042C3DF RETURN to DongleVi.sub_42C35E+81
Address ESP+14 0013F210 00000008
Querydata ESP+18 0013F214 0051412C DongleVi.0051412C
Response ESP+1C 0013F218 0013F230
Response32 ESP+20 0013F21C 00000000
Length ESP+24 0013F220 00000006

And these the register EDI points to querydata (0x0051412C). You can copy then to the pointed buffer the values the program wants to see.
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

Figure 26 - Original sproQuery
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

The real problem with sproQuery is that there are no other methods to find the queryvalue and response values than tracing the program. You can follow the same try-and-see procedure described for sproRead, if the program is poorly implemented you will easily find the correct values.

For the example we are following this is what happens

1. place an Hardware breakpoint on access at the addresses 0x0013F244 and 0013F248 when you are stopped at 0x004776F0 and press F9 to see where these values are accessed and against what compared.

2. the application stops at 0x00453BF1, see Figure 27.

   ![Figure 27 - Where DongleViewer stops after BP into sproQuery](image)

   The buffer, filled by previous call to sproQuery, is now stored in EDI and is compared byte by byte against 0x00514132 stored in ESI. If something is wrong the program jumps out. The alternative fixing it also to change the JNZ into a JZ but remember we told we were concentrating on the library part of the application and not on the application’s specific code.

3. If you press again F9 you will see that the buffer has been deleted.

This has been a lucky case (I chosen it not casually ;-).

6. What’s more

Well we reached the end of this tutorial. I have not completely covered the dongle argument, but only some specific issues. There’s still to cover the envelopes things and other complex interactions with the keys. Anyway the general approach here described remains valid for all the Sentinel protected programs and partially for all the dongle protected programs.

I would point out that the solution presented here is ideal when you have an original program with a dongle and you don’t know how to distribute it, because emulation would reveal your license. You can fish which responses sproQuery and sproRead report to the program, when a correct dongle is inserted, using the monitor described at Section 2.3. Then you can use those values to emulate the dongle, understanding what values are important for the
Removing Sentinel SuperPro dongle from Applications
and details on dongle way of cracking

program (checked) and what not. What you will finally obtain is a working distribution, derived from a real key, but with no more your personal data inside!
Another advantage of only changing the parts of the program derived from the Sentinel SDK is that these parts are invariant to the following releases of the program being the SDK library not dependant from the program’s producer. Thus a patcher with a Research & Replace engine has a big chance to survive to following releases of the same program.

**NOTE**
As an exercise to see if you have understood the concepts here described I included into the distribution of this tutorial also a Sentinel crackme (Sprocrackme.exe also available at [1]) for which you can apply the techniques here described.

7. References


8. Conclusions
Well, this is the end of this story, I hope all the things here said will be useful to better understand how process is handled by the OS and in which manners we can keep process control and avoid programs using dongles. I suggest as usual to use this tutorial for learning more in deep how the dongle and Sentinel system work and to use these examples to evolve your RCE techniques and not to crack programs.

All the code provided with this tutorial is free for public use, just make a greetz to the authors and the ARTeam if you find it useful to use. Don’t use these concepts for making illegal operation, all the info here reported are only meant for studying and to help having a better knowledge of application code security techniques.

9. History
- Version 1.0 – First public release!

10. Greetings
I wish to tank all the ARTeam members, TORO and who read the beta versions of this tutorial,.. and of course you, who are still alive at the end of this quite long and complex document!