Quick Tour for Linux

Differences under Linux

The Quick Tour chapter in the manual guides you through protecting a simple program using the Shell Method and the Windows GUI.

For Linux you cannot use the DinkeyAdd and DinkeyRemote GUI programs because these do not exist under Linux. (All the tools in the Linux SDK are designed to be run from the terminal). In this guide we will use DinkeyAddCmd and DinkeyRemoteCmd (you could instead call the DinkeyAdd and DinkeyRemote Modules using the C sample code but this is more complicated).

Before you continue please make sure you have read the README.txt in the root directory of the SDK and have run the "inst" script so that our dongles can be recognised by Linux.

Note – all the Linux programs in the SDK are provided in both 32-bit and 64-bit formats. For example DinkeyLook is provided as DinkeyLook32 (32-bit) and DinkeyLook64 (64-bit). For simplicity we will refer to this program as DinkeyLook, but please use the program that is most appropriate for your Linux kernel.

Protecting a Sample Program

The sample program we will protect is the *hello* program which is located in the QuickTour directory of the SDK. To protect this file use the terminal program DinkeyAddCmd and the parameter file quicktour.dapfj.

However, before we do this we should review the settings in the quicktour.dapfj. Please open this file in a text editor – it is in JSON format. To view the full specification of this file go to Structures | DAPF file – JSON Format in the manual.

You need to modify the *Models* key to match the model of dongle that you are using. If you are using a Lite dongle then you will also need to modify the *ProductCode* to match the Product Code in the dongle. If you are unsure of the dongle model or Product Code then attach the dongle and run DinkeyLook.

You also need to modify the input and output path of the file to be protected, so please change the *InputName* and *OutputName* keys in the *Files* object. The input path should be the location of hello in the QuickTour directory. The protected program will be written to the output location and filename specified and the original unprotected program will remain untouched (so long as *InputName* is different to *OutputName*). You can specify absolute paths or paths relative to DinkeyAddCmd.

You are now ready to protect the file. Attach the dongle to your computer and execute the following:

```
./DinkeyAddCmd64  ../QuickTour/quicktour.dapfj
```

(this assumes that you are in the DinkeyAdd directory of the SDK).

This will program the dongle (not necessary for Lite models) and protect hello so that it will only run if the dongle is present.

Try that now. With the dongle attached the *protected hello* program should run normally. If you remove the dongle from your computer and try running the *protected hello* it will display an error message and then terminate.

Note – if you are using a Net dongle then there is an extra step involved. You need to run *DinkeyServer* on the computer that has the dongle attached. For more information see the Network Dongles chapter in the manual.

You have now added protection to a simple program. To find out how to protect *your* program you need to read the Protection Method chapter of the manual. If you have a Plus or Net model then you may like to read the next section which shows you how to modify the protection parameters stored in the dongle.

Modifying Protection Parameters

If hello was protected to a Plus or Net dongle then it was initially limited to 5 executions. To find out how many executions you have left please attach the dongle to your machine again and run DinkeyLook.

The value under *Execs Left* tells you how many more times you can successfully run the protected hello. Please repeatedly run the *protected hello* program until it comes up with the following message: *Error! hello has used up all of its allowed executions*. You have now successfully run the protected hello your allotted five times and you cannot run it again. The value of Execs Left in the dongle is 0. However, you can change this value by running the DinkeyRemote and DinkeyChange programs.

To generate the update code we will use the program DinkeyRemoteCmd and the parameter file quicktour.drpfj.

However, before we do this we should review the settings in the quicktour.drpfj. Please open this file in a text editor - it is in JSON format. To view the full specification of this file go to Structures | DRPF file – JSON Format in the manual.

You need to modify the *ProductCode*, *DongleNumber* and *UpdateNumber* keys to match those of the dongle you want to update. Although you can find this information from DinkeyLook it is also displayed by DinkeyChange so please run this program now. You will probably only need to change the *DongleNumber* value.

The other settings in the DRPF tell you what parameter will be changed in the dongle. The relevant one is *"AddExecutions": 5*. This indicates that we want to add 5 more executions to the dongle.

You are now ready to produce the update code. Execute the following from the DinkeyRemote directory:

```
./DinkeyRemoteCmd64 ../QuickTour/quicktour.drpfj
```

This will output the update code to the file UpdateCode.ducf (to change the file name and location you can modify the *OutputFileName* key in the UpdateCode object of the DRPF. The default location is ~/.DinkeyRemote so you may want to change this). You would normally send this file to your customer so they can update the dongle with DinkeyChange.

Now execute the following command (from the SDK root directory) to update the dongle with this update code:

```
./DinkeyChange64 <path>/UpdateCode.ducf
```

(you should specify the relative or absolute path to the UpdateCode.ducf file).

The executions value has now been restored to 5. You can run DinkeyLook to verify this.

Now you have to learn how to use DinkeyAdd, DinkeyRemote and DinkeyChange to protect *your* software and modify the protection parameters in the dongle. To understand how to use these programs in more detail please read the manual before you protect your own software.