Chapter 4 - Key Code Protection

Example: Quick Format 7.0

The first type of protection scheme I would like to explain is the Key Code Scheme. This is used in Quick Format 7.0, which is a Public Domain Shareware program that is very good at formatting floppies and designing your own labeling scheme those floppies. It is definately worth the Shareware fee, so I suggest if you like the program to buy it, you can probably find it on many online services and from user groups.

The author decided to put in a registration algorithm which requires its users to type in a key code to access the advanced features of the program. When you first launch the Quick Format 7.0 application, a dialog box comes up and asks you to enter in the key code. If no key code is entered or you hit the return key, the program would continue to run, but with the advanced options turned off.

The author is doing a couple of different little things. First, he is going to check somewhere within his resource files to see if the current application being used is registered. Usually there is a register byte in a resource somewhere in the app. He will then do a compare to see if it really is registered. If it is, it continues like normal, if it isn't registered, it will jump to another routine which turns off the advanced options and then runs the app as normal.

There are a few options we have when deprotecting this app. We can use MacsBug to trace through for the routine, then disassemble it to see where it does the compare; or we can use ResEdit to find a resource that looks suspicious and delete it. The latter might be a little tedious and it is always much more interesting tracing through code.

Now we will deprotect Quick Format 7.0. For best results and for speed and memory purposes quit all other applications you are currently running. When you are back at the Finder, hit "Command-Reset" or reach in back of your machine and press the hardware interrupt switch, this will activate MacsBug. Your screen should have cleared and you should be looking at a white screen with numbers on the left hand quadrant of the screen running from TOP to BOTTOM. Those numbers are the various addresses and registers in memory.

Running along the bottom of the screen from LEFT to RIGHT are two separate boxes. The box on the big box with the numbers in it is a disassembly of the location in memory you just broke into with MacsBug. The smaller box under it is the MacsBug command line. On the far left you should notice a blinking cursor. From the command line you can execute different commands to help you trace through programs, especially useful in deprotecting software. At the command line type "?" (help). This will print up a list of different topics. If you keep hitting return it will give you information about each topic in

the order they are shown on the screen. So play around and hit return a few times to get an idea of what commands you can use.

Now that you are done playing let's get started. I almost always set a TRAP for an _InitGraf. You can do this by typing

'ATB INITGRAF'

A message should appear above the disassembly box saying 'A-Trap Break at A86E (_InitGraf)' everytime. What this means is that the program will be stopped and MacsBug will take over everytime the program tries to execute an _InitGraf. This works the same way for all of the other trpas that the Mac toolbox has as well.

Ok, now type 'G' on the MacsBug command line. This should bring you right back to the Finder where you started, and you will regain control of your Mac. Locate Quick Format 7.0 and launch it. Almost immediately your screen should change back into the MacsBug screen. There should be a message saying 'A-Trap break at XXXXXXX : A86E (_InitGraf)'. This means when you launched the program, MacsBug halted it because the program tried to pass an _InitGraf trap. Now that the program is halted, you can TRACE through the program to find the copy protection. You may not successfully pinpoint the protection to any one specific area until you have traced through a number of times.

Use the 'T' command to trace through. The object is to continue hitting 'T' and return until the protection scheme comes up. Eventually it will. When you do get it up look at the last few lines of code that was passed and you should see something like this:

Addr	Instruction	Hex Bytes
583834	JSR SETUPMEN	4EBA FE14
583838	JSR INITIALI	4EAD 02E2
58383C	JSR INITGLOB	4EBA FEBE
583840	JSR VIRALCHE	4EBA FF22
583844	JSR CHECKMOR	4EBA F82A

Now, it's pretty obvious from just looking at the labels they used that you can determine what is going on. In most cases people would not use LABELS like the ones above, but since it is shareware and not a $500 commercial package I can see why the author opted the easier route for programming ease. The first JSR would probably be him initializing his menus and stuff. The second JSR would be to initialize the screen and the fonts or whatever, the third JSR would be initializing the global variables he would need and the fourth would be to check for any virus, persay. The fifth however is the routine he uses to check if the program has been registered and brings up the dialog asking you to enter a key code. If it hasn't been registered with the correct keycode the program turns off some options. But, that is not necessary, as by omitting this JSR CHECKMOR you will remove the check and the program will run with all options available.

Write down the last 10 or so bytes on a piece of paper noting that 4EBA F82A is what you will have to change. Since you want to omit these bytes you are best off using two NOP (or No OPeration) commands. The hex value for a NOP is 4E71. Now run FEdit and open up the Quick Format 7.0 program and do a HEX SEARCH for the bytes you wrote down on the paper. Then change the proper values and you will be all set. Here is what you should be looking for and the change you should be making:

Byte Changes (You should find the SEARCH string only ONE TIME!)

Search : 4EBA FE14 4EAD 02E2 4EBA FEBE 4EBA FF22 4EBA F82A
Change : 4E71 4E71

The protection showed above is obviously an easy scheme to get around, and to tell you the truth, there really aren't that many hard schemes on the Mac, like there are or were on the Apple II. It is important to check the routine you are disabling. Sometimes variables (or globals) are passed in between different parts of protection schemes, if you skip the entire protection scheme there is a pretty good chance you will miss a variable (or global) getting passed and your program will crash on you in the future.

The best way to check is to use the program after you have initially deprotected it, if it works ok, then chances are no globals were passed. In the example above, all of the globals were passed in the prior two JSRs, which made things very easy.