Dr. Hax Column

If you have an application thats copy protected, or which has a registration or expiration scheme, its probably easier to fix than you think. All you need is some knowledge of Macintosh programming, a debugger such as MacsBug 6.1, and the willingness to roll up your sleeves and see what you can do. Before you start working on an application, be sure to make a copy of it. Well be making some low-level changes to the code, and its easy to permanently damage something.First lets examine how an application detects that it has expired. Normally theres a piece of code that goes something like this:

move the current time into a memory address

move the expiration date into another address

compare the two addresses
if the time is greater, quit - else continue.
For

registration, it might go something like this:

get the serial number from the user

perform some
arithmetic operations on the serial number

move the result into an address

check if the
result is zero

if it isnt, quit - else record the

registration.
Heres an example for copy protection:

change the floppy disk motor speed

get the

contents of disk sector XXX

perform some hefty
arithmetic on the contents

move the result into an
address

check if the result is 312

if it
isnt, quit - else continue.

You'll probably notice that all three of these examples have something in common - no matter how complicated the initial scheme, they all compare numbers to see if the desired conditions are met. If we concentrate on changing the comparison code, we can break any of these

schemes.Well use MacsBug to discover where the

code actually is. The first thing you need is a

reference point: pick a Macintosh toolbox call that

occurs right after the checking code has failed. For example, if the program beeps if it has expired, you can use Sysbeep. If it puts up an alert box, you can use Alert (or perhaps StopAlert). Use MacsBugs

"ATB" command to drop into the debugger when this toolbox call occurs. For example, "ATB sysbeep" will stop the computer right before it beeps. Then you can dissassemble the code (working backwards) to discover an assembly command that compares the numbers - such as "TST.L" or "CMP.B". The "IP"

command is useful for this; you may have to play

with MacsBug for a few hours to see what each

assembly language instruction does (dont worry if

you dont know much about assembly; I learned

everything I know from watching MacsBug steps

through instructions).When you find the specific set of instructions, there are two possibilities: the first is that the program branches to the bad code and

executes the good code if it doesnt branch.

This

might look something like this (some MacsBug junk removed for clarity):

```
+016C       TST.L        D1                                                   |

4A81+016E             BEQ.S         *+$000E                       |

670C+0170                         *** GOOD CODE HERE***
```

```
+017C    BRA.S         *+$002C                        |
```

```
602A+017E                              *** BAD CODE HERE! ***
```

This is the easiest type of code to fix. All we need to do is replace the "TST.L" and "BEQ.S" instructions with something that will just drop through to the next

instruction at +0170. The perfect thing to replace it with is two "NOP" instructions ("4E71" in machine

language), which does nothing at all except go to the next instruction. See the machine code on the right (4A81, 670C, etc.)? If you open up a code resource in ResEdit, thats what the numbers are. Simply write down a few of them around the instruction you need to change, perhaps: "4A81 670C 4EBA FCB8 2200"

(you get the extra numbers from further on in

MacsBug). Be sure to write down quite a few of them and not just a couple, because theres likely to be a lot of values that are the same in any file, and you might change the wrong one by mistake. Search each and every code resource that you find in ResEdit for the string "4A81670C4EBAFCB82200" (there are no spaces in ResEdit). In this case, when you find it, youll change the characters "4A81670C" to "4E714E71". Provided you changed it in the correct place, the program can now never branch to the bad code.The other possibility is that the branch instruction goes to the good code, so you need to make the program branch every time:

```
+016C

TST.L          D1                                             |
4A81+016E           BEQ.S       *+$000E                |

670C+0170                      *** BAD CODE HERE! ***

+017C   BRA.S       *+$002C                |

602A+017E                      *** GOOD CODE HERE ***
```

In this case, you need to replace the "TST.L" and "BEQ.S"instructions with code that makes the program branch every time. The first instruction should be "6000" (BRA), and the second should be the distance to branch. You can get this from the original instruction in this line: the "BEQ.S *+$000E". The "000E" is what you want. So, using ResEdit, change the "4A81670C" to "6000000E". Once again, be sure to use a few extra characters when you search.So thats the essence of simple code cracking. As you get better at it, youll be able to fix things that are a little tougher than the examples shown here. Just remember that it doesnt have to be complicated - you dont have to break someones complicated encryption scheme, you just have to use a little knowledge of how your computer works to find the weak spot.