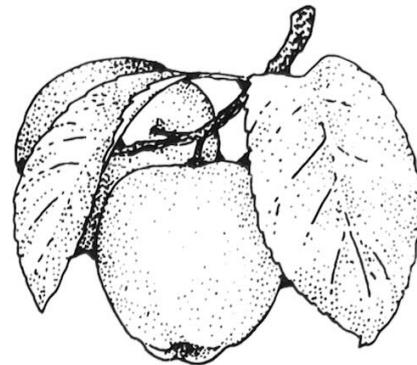


APPLE SOFTWARE PROTECTION DIGEST



Vol. 1, No. 6

1986



BETTER SOFTWARE WARRANTIES

After years of taking the customer for granted and sticking unsuspecting consumers with programs that have not even been properly tested and debugged, it looks like the software industry is about to make some changes. Forced into action by a proposed bill from California Assemblywoman Gloria Molina, the industry has apparently decided that they'd be better off coming up with their own warranty improvements than having even tougher ones imposed on them by law.

For years software publishers have been passing off disclaimers as limited warranties. Typically these insults to the consumers intelligence go something like this:

This program is provided "as is" without warranty of any kind, either express or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the results and performance of the program is assumed by you. Should the program become defective, you (not the publisher or its dealer) assume the entire cost of all necessary servicing, repair and correction.

That's not a warranty, limited or otherwise. It's just a way of squeezing money out of the unsuspecting consumer without having to give him anything of value. To make things even worse, most companies that offer

"warranties" such as the one above, also copy protect their programs so that even if you did want to correct their program, you couldn't.

Anyway, at least one legislator had the guts to come out and tell these guys off. Molinas proposed a bill in 1985 that would force publishers to stand behind their products. The bill is currently tabled and Adapso, a computer and software manufacturers organization and the Software Publishers Association are both encouraging their members to adopt a warranty that says the publisher would stand behind the claims made in advertisements, documentation and specification statements. At least 40 companies have improved their warranties so far and one, Micropro, is even offering a 90-day money-back guarantee. Another company, Hayes Microcomputer Products, offers a warranty that guarantees performance.

Software publishers are finally beginning to realize that it is the customer who keeps them afloat, and if they're going to keep their customers, they're going to have to treat them better. Thanks to you Ms. Molinas, you did a good job getting the ball rolling. Now if you could only convince publishers to remove copy protection.

Jules H. Gilder

Publisher & Editor

Contents

Editorial	1
Cracks Wanted	2
Letters	2
Next Issue	2
Crack Index	2
How to Crack Dazzle Draw	3
Five Byte Disk Analyzer	6
Unprotecting Locksmith 5.0	8
COPY File Parameters	9
REVIEW: Locksmith 6.0	10

Apple Software Protection Digest

Publisher & Editor, Jules H. Gilder; Contributing Editor, J. Scott Barrus. Copyright © 1986 by Redlig Systems, Inc., 2068 - 79th Street, Brooklyn, New York 11214. All rights reserved. No part of this publication may be reproduced, or electronically transmitted or stored without the publisher's written permission. Published at \$24 per year by Redlig Systems, Inc. (718) 232-8429. Reprints of prior issues available at \$3 each. Printed in the U.S.A.

Apple is a registered trademark of Apple Computer Inc.

Cracks Wanted

Listed below are programs that our readers would like to unprotected. Anyone who comes up with a method of removing the protection from any of these programs will get a free three-month subscription, or extension to ASPD, so get those solutions in.

If you have a program that you'd like to see unprotected, please let us know, and we'll add it to our list.

1. Ace Writer II
2. Aztec
3. Bag of Tricks
4. Batter Up!
5. Blazing Paddles
6. Certificate Maker
7. Compress Software
8. Crossword Magic 4.0
9. Crush, Crumble & Chomp
10. Disk-O-Check
11. Magic Memory
12. Newsroom
13. Sargon III
14. Sensible Speller - DOS
15. Sensible Speller - ProDOS
16. Success With Math
17. The Game Show
18. Wizardry
19. Word Handler

Next Issue

The next issue will start to whittle down our list of Cracks Wanted. Among the articles that will be in that issue are:

1. How to Crack Newsroom
2. Unprotecting Sensible Speller (ProDOS)
3. Making a BRUNable Copy of Locksmith's Fast Disk Backup
4. How to Use Two CATALOG Tracks on One Diskette

Letters

Dear Editor:

The issues of your publication that I have received have been very informative, but I'm still learning and I don't have any contributions in the area of copy protection yet. I do have a few questions, however. How does a nibble count protection scheme work? What is bit insertion? How do I form two CATALOG tracks on one diskette? I look forward to hearing from you at your earliest convenience.

Rick A. Crawford
Denver, CO

Don't worry about your not having contributions for us now Rick, but if you come up with something that would be of interest to others, let us know. In the mean time, let's tackle your questions. A nibble count is actually a very simple thing. Generally, one track, usually 34 or 35 but it could be any track, is reserved for the nibble count. This track is specially formatted so that it has a starting header and the rest of the track contains sync bytes. These are special bytes that are 10 bits long and are more difficult to copy than 8-bit bytes. When this track is written, the speed of the drive that is used for writing is generally reduced so that more bytes can fit on the track. This means that drives working at the correct speed can't copy all of the bytes, even if a way is found to read and write the 10-bit bytes. Thus, when the routine that counts the bytes on this track is called, it finds that the wrong number of bytes is present, determines that the disk is a copy and crashes. Bit insertion is when more cycles are used to write a byte to the diskette than usual. This causes more bits to be used to write a byte. That's how we get 10-bit bytes. I'm preparing a tutorial article on both these subjects for a future issue of the digest. As for your last question, in the next Apple Software Protection Digest there will be an article that will show you how to use two CATALOG tracks on one diskette.

Crack Index

In order to make life just a little more convenient for you, each issue of **Apple Software Protection Digest** will contain a list of all the programs cracked so far, and what issues those cracks appeared in. This will save you from going through all past issues of the digest in order to find a particular program.

- Applewriter //e - Vol. 1, No. 3, p. 3
 Bookends - Vol. 1, No. 1, p. 7
 Dazzle Draw - Vol. 1, No. 6, p. 3
 Financial Cookbook - Vol. 1, No. 3, p. 6
 Hayes Terminal Program - Vol. 1, No. 3, p. 11
 Homeword - Vol. 1, No. 2, p. 9
 Homeword Speller - Vol. 1, No. 2, p. 9
 It's the Pits - Vol. 1, No. 5, p. 5
 Locksmith 5.0 - Vol. 1, No. 6, p. 8
 Karate Champ - Vol. 1, No. 6, p. 9
 Microwave - Vol. 1, No. 3, p. 11
 PFS PLAN - Vol. 1, No. 2, p. 9
 PFS Series (ProDOS) - Vol. 1, No. 2, p. 7
 Print Shop - Vol. 1, No. 1, p. 10
 Print Shop Companion - Vol. 1, No. 2, p. 6, Vol. 1, No. 5
 Sensible Grammar - Vol. 1, No. 2, p. 7
 Smart Eyes - Vol. 1, No. 5, p. 4
 Time Is Money - Vol. 1, No. 2, p. 9

**Don't miss a
single issue
SUBSCRIBE
TODAY!**

HOW TO CRACK DAZZLE DRAW

by The Executor & Byte Doc

One of the most popular and difficult programs to crack on the Apple is Broderbund's *Dazzle Draw*. This article will give you step-by-step instructions on how to crack this program. Because the protection techniques used on *Dazzle Draw* are more sophisticated than you find on many other programs, the instructions for cracking it will be a little more complicated as well. However, if you follow the step-by-step procedures slowly and carefully, you should be able to make your own unprotected backup copies of the program. We will try to keep the complicated technical information to a minimum to minimize confusion.

When *Dazzle Draw* is first booted up, at some point during the booting process, you'll hear some unusual sounds from the disk drive. Upon close examination of the disk while it was booting, we found that this occurs when the head is located over track 31 (\$1F). Closer examination, with the aid of a nibble editor, revealed that this sound was probably an indicator of a nibble count routine that is used to protect the program. In addition to the nibble count routine, examination of the diskette revealed that Broderbund also made a change to the standard address marker that is written onto the diskette when it is formatted. While the normal address marker contains the bytes DE AA, Broderbund has changed the marker to DE FF. This change, which is both simple to make and easy to overcome, will prevent ordinary copy programs from duplicating the diskette.

In order to crack *Dazzle Draw* you are going to need to prepare a few things. First of all, you'll need either an Apple //c or //e with 128K of RAM. This is not asking too much, because *Dazzle Draw* will only work on one of these computers anyway. Of course,

you'll also need an original copy of *Dazzle Draw*. Other tools that are required are a sector editor (we're going to modify *COPYP* so that we can use just the sector editor portion of it), one blank diskette, one diskette that has been initialized as a 48K slave (just type INIT HELLO on a 48K or larger system) DOS 3.3 diskette, Apple's ProDOS System (also called User's) diskette and the *COPYP* program (see *ASPD*, Vol. 1, No. 2, p. 2). Once you have assembled all of the things you'll need to crack this program, your ready to begin. *Remember, initialize your DOS 3.3 diskette before you begin.*

First copy it with COPYP

The first step of the cracking process is to make a backup copy of *Dazzle Draw* with *COPYP*. In order to do this however, we're going to have to add one line to *COPYP* and change another. The reason for these modifications is, as mentioned earlier, Broderbund has changed the normal address marker. Specifically, they've changed the last byte in the *address field epilog* (see *ASPD*, Vol. 1, No. 2, p. 20-22 for more details on address and data fields).

To accommodate the modification, we have to change line 910 in *COPYP* so that it reads:

```
910 DATA 222,255
```

This change will make it possible for *COPYP* to read information from Broderbund's specially formatted diskette and write that same information out to a diskette that has been formatted with the normal address field epilog byte.

When you use *COPYP* to backup the original *Dazzle Draw* diskette, use the blank, uninitialized diskette as your target diskette. (From now on, we will refer to the *COPYP* produced

version of *Dazzle Draw* as the backup copy of *Dazzle Draw*. This is not to be confused with the backup copy that *Dazzle Draw* itself permits you to make.) *COPYP* will automatically initialize the blank diskette as a normal diskette. Save the DOS 3.3 diskette that you already initialized for use later on.

Before you run *COPYP* There are a few more things that you are going to have to do. The first is, you're going to have to add a line to the program that will cause it to only copy the first 31 tracks (Track 30 is the last track we're going to copy, but remember, our first track is track 0, which is how we get 31 tracks even though the last one is track 30). As mentioned earlier, track 31 contains nibble counting information. Since we're going to eliminate the need for the nibble counting routine, and this track would normally be very difficult to copy anyway, we won't need it. The tracks beyond 31 are not used. To make track 30 the last one we copy, all we have to do is add line 76 listed below:

```
76 POKE 863,30
```

We've said that tracks 31 to 34 are not used. The way the diskette is currently configured, however, the operating system thinks these tracks are used and thus unavailable. We can change that by doing some sector edits to the diskette's Volume Table Of Contents (VTOC). *Dazzle Draw* is a ProDOS-based diskette, and as such, its VTOC is located on track 0, sector 3. If we change bytes 31 to 34 (\$1F to \$22) in this sector to \$FFs, we can make those tracks available to ProDOS. To do that, the following lines should be added to the *COPYP* program.

```
1000 DATA 0,3,31,0,255
1010 DATA 0,3,32,0,255
1020 DATA 0,3,33,0,255
1030 DATA 0,3,34,0,255
```

With these final additions made to the *COPYP* program, you are now ready to make your backup copy of *Dazzle Draw*.

After the copy is made, label it and set it aside for a moment. Now, take your ProDOS system diskette and re-boot your computer. Exit to BASIC if you're not already there when the boot ends. Now, we're going to create a dummy file with the name PRODOS on our backup copy of *Dazzle Draw*. It is necessary to put this file on the diskette so that we don't get the "UNABLE TO LOAD PRODOS" error message when our copy is booted. To create this dummy file, remove the ProDOS user's diskette from the disk drive and insert your backup copy of *Dazzle Draw*. Now, type the following line:

```
CREATE PRODOS,TSYS
```

This should create a new entry in the ProDOS directory on the diskette, with the name of PRODOS.

Now that we've created the dummy file, we have to make some changes to it. To do this we're going to need a sector editor. You can go out and buy one, or, you can use *COPYP* as a sector editor. To do that, all we have to do is disable the copy portion of the program. This is done by adding line 89 below, which causes a jump to the sector editor. The jump is made at line 89 in order to assure that all of the DATA statements that precede the sector edit data are read and program is ready to read the sector editing data. By typing the number 70 (below) and then pressing the RETURN key, we delete line 70 from the *COPYP* program. It's not absolutely necessary to do this, but since we're not copying anything, there's no need to load in the machine language portion of the copy program. The other lines that are listed below, contain the sector editing information.

```
70
89 GOTO 400
1000 DATA 0,11,238,22,38
1010 DATA 0,11,255,248,9
```

We're going to need to use *COPYP* one more time. Reload the program from your disk so that you start with a copy of the original version and not the one we've just modified. Once again we're going to use only the sector editing capabilities of the program. This time however, we're going to copy an entire sector from the ProDOS system master to our backup copy of *Dazzle Draw*. This gets rid of the strange boot sequence used by the original *Dazzle Draw* diskette and in the process eliminates the check for the copy-protected diskette. Modify the fresh copy of *COPYP* by typing in the following lines:

```
70
89 GOTO 400
460 TK=0: SE=0
470
490 PRINT: PRINT"REMOVE THE
      PRODOS SYSTEM DISK"
500 PRINT"INSERT THE BACKUP
      DAZZLE DRAW DISK"
505 INPUT"PRESS <RETURN>
      WHEN YOU'RE READY";Z$
520 GOTO 550
```

Once these lines have been added to the *COPYP* program, place your ProDOS system diskette in drive 1 and run *COPYP*. When the program tells you to, remove the ProDOS system diskette from the drive and replace it with your backup copy of *Dazzle Draw*. Then press RETURN. The boot sector from your ProDOS system diskette will then be transferred to the diskette containing the unprotected version of *Dazzle Draw* that we are creating.

Capturing the program

We have now finished most of the work that is required to produce an

unprotected copy of *Dazzle Draw*. The only thing left for us to do now is to capture the startup program and save it out to disk. Do not confuse this with the STARTUP program on the ProDOS system diskette.

In order to capture the startup program, and at the same time eliminate the need for the protection checking subroutines, we're going to have to boot the original program several times and interrupt it at specific points. This part of the *crack* may be the most confusing to you, but if you follow each step exactly, you should have no problems.

To cause the boot to stop exactly where we want it to, we're going to move program code that handles the booting process from its place in ROM (on the disk controller card) into RAM so that we can modify it.

STEP 1. Turn on your Apple //e or //c and press RESET (or CTRL-RESET if your keyboard is setup that way) to turn the disk drive off.

STEP 2. Enter the monitor mode by typing: CALL -151

STEP 3. Move the disk controller ROM code to RAM and modify it so that it jumps to the monitor mode by typing the following two line:

```
8600<C600.C6FFM
86F9:59 FF
```

STEP 4. Insert your original *Dazzle Draw* diskette in drive 1 and execute a partial boot by typing:

```
8600G
```

STEP 5. Your Apple will beep and a monitor prompt (*) will appear. Type in the following lines to turn off the disk drive, modify the next stage of the booting process and reboot the program.

C0E8
843:59 FF
86F9:01 08
8659:50
8600G

STEP 6. When you type the 8600G in STEP 5, your drive will reboot and in a moment the Apple will beep and the monitor prompt will appear. Now let's turn the drive off again and make a few more modifications:

C0E8
602B:0F
843:00 60
850:40

STEP 7. Now, type in this short *capture* routine to save everything in memory from \$0000 through \$08FF and store it from \$8000 through \$88FF:

F00:8D FC 0F 8E FD 0F 8C FE
F08:0F BA 8E FF 0F A2 00 BD
F10:00 00 9D 00 80 E8 D0 F7
F18:EE 11 0F EE 14 0F AD 11
F20:0F C9 09 D0 E8 4C 59 FF

STEP 8. Execute this by typing:

8600G

STEP 9. The disk drive will reboot again and in a moment your Apple will beep and the monitor prompt will appear. At this point, turn off the drive and examine what was in the 6502 registers when the program jumped to the monitor by typing in the following lines:

C0E8
FFC.FFF

STEP 10. Make a note of the values returned for locations \$FFC to \$FFF. You'll need them later. The contents of \$FFC = Accumulator, \$FFD = X-register, \$FFE = Y-register and \$FFF = Stack pointer.

STEP 11. Next, disable the configuration save option by typing:

75C5:A7 7B

STEP 12. Now, take the DOS 3.3 diskette that you initialized earlier and insert it into drive 1, after removing the original *Dazzle Draw* diskette. Boot the DOS 3.3 diskette by typing C600G and then save the memory pieces that we've just assembled through this series of partial boots. Saving the file at this point is just a precaution so that you won't have to go through the entire series of partial boots again in case something goes wrong in the following steps. Type in the following lines:

C600G
BSAVE DD,A\$6000,L\$2900

STEP 13. Enter the monitor and move the program code down to \$2100 where it belongs by typing:

CALL -151
2100<6000.88FFM

STEP 14. Type in this routine which reconstructs the memory:

2000:A2 00 BD 00 41 9D 00 00
2008:BD 00 21 9D 00 60 E8 D0
2010:F1 EE 04 20 EE 07 20 EE
2018:0A 20 EE 0D 20 AD 0A 20
2020:C9 41 90 DE 20 9A 61 AD
2028:E9 C0 A2 FF 9A A0 00 A2
2030:60 AD 10 C0 A9 BA 4C 00
2038:64

STEP 15. Now enter the values you wrote down for the four 6502 registers as follows:

2035:xx (value of the Accumulator)
2030:xx (value of the X-register)
202E:xx (value of the Y-register)
202B:xx (value of the Stack pointer)

STEP 16. Save the whole *Dazzle Draw* startup program by typing in the following line:

BSAVE DAZZLE.SYSTEM,A\$2000,
L\$2A00

STEP 17. Put the DOS 3.3 diskette with the DAZZLE.SYSTEM file on it aside now and boot up the ProDOS system diskette. Execute the DOS converter program by typing:

-CONVERT

STEP 18. If you have a two-drive computer system skip down to STEP 24. If you have a one-drive system, go to STEP 19.

STEP 19. Set the prefix by pathname by pressing P twice and type in the pathname /RAM.

STEP 20. Insert the DOS 3.3 diskette that has the startup file on it. Transfer the DAZZLE.SYSTEM file from the DOS 3.3 diskette to the internal RAM disk by pressing T and then typing the name DAZZLE.SYSTEM when the program asks which DOS 3.3 file you want to convert.

STEP 21. Next, insert the ProDOS system disk and type Q to quit. When the program asks you for a file name type in:

/USERS.DISK/BASIC.SYSTEM

STEP 22. Without shutting off the computer, startup the ProDOS file transfer program by typing F.

STEP 23. Move DAZZLE.SYSTEM from your RAM disk to your backup copy of *Dazzle Draw* by pressing F, C and using /RAM/= and /DD/= as pathnames respectively. Go to STEP 26.

STEP 24. Set the prefix by pathname by pressing P twice and type in the pathname /DD.

STEP 25. Insert the DOS 3.3 diskette that has the startup file on it into drive 2 and the backup copy of *Dazzle Draw* into drive 1. Transfer the DAZZLE.SYSTEM file from the DOS 3.3 diskette to the backup diskette by

(continued on page 9)

FIVE BYTE DISK ANALYZER

by David Stoll
Manitoba, CANADA

While this article does not describe a crack for a specific program, it does describe a simple modification to normal DOS 3.3 that renders some protection schemes ineffective. All that is required to implement this modification to DOS is to change just five bytes of code. The changes to the five bytes are implemented via a short BASIC program, which also allows you to scan each track of the diskette and locate the CATALOG track.

I'd like to point out that this modification to DOS does not remove the copy protection from the programs it is used with. It simply makes it possible for the usual DOS commands to access any of the files on the protected disk. In many instances, this is all you need to do to make a backup copy of the protected program.

While this technique will not work with the new sophisticated protection techniques being used, there are still enough programs out that can use this techniques to make it useful. Programs that are particularly susceptible to this technique are those that alter the address field, data field or checksum and those that relocate the diskette's directory to a different track.

Using the program

To use the *Five Byte Disk Analyzer*, you should begin with a normally initialized DOS 3.3 diskette. Once the diskette is initialized, enter the BASIC program listing shown and save it to the diskette.

To check a protected diskette, boot up the *Five Byte Disk Analyzer* program. When the message, "INSERT PROTECTED DISK" is displayed, re-

move the *Five Byte Disk Analyzer* diskette and insert the copy-protected diskette. Then press any key when you're ready, and the program will attempt to CATALOG the diskette. (*Editor's Note: David originally used the CATALOG command in lines 40 and 140. I replaced it with the CALL 42350*). The CATALOG command is implemented by directly calling the machine language routine that handles it because some protection schemes change the names of the DOS commands to make life tougher on the user.

If the diskette's catalog is located on track 17 (\$11) it will be displayed. You can then quit the program and try the conventional DOS commands to see if they work. If they do, fine, if not, you might want to run the *DOS Command Comparer* program that appeared in the last issue of ASPD on page 3, to find out what the new commands are. Even if you get a catalog listing right off the bat, you might want to use the program's scan option anyway because some protection schemes use two catalog tracks and switch back and forth between them.

If the program does not display the diskette's catalog right away, you're presented with a two-choice menu. You can either quit, or scan all of the tracks on the diskette, looking for the catalog track. The program starts scanning from track 1 (track 0 is the boot track and you can't have a catalog there) and goes up to track 34. You could probably safely start scanning from track 3, because tracks 1 and 2 are generally used to store DOS. If you find that most of the diskettes you scan have relocated directories on higher numbered tracks, you can change line 110 so that scanning will start with track 34 and work its way down. To do that, type in the following line:

```
110 FOR X = 34 TO ST STEP -1
```

If you suspect that the diskette you're examining has more than 35 tracks on it and that the catalog is on one of these extra tracks, you'll have to make some more changes to DOS. For more information on adding extra tracks to your diskette or moving the catalog to another track see ASPD Vol. 1, No. 1, p. 8 or Vol. 1, No. 3, p. 9.

```

10 REM FIVE BYTE DISK ANALYZER
20 POKE 47405,24 : POKE 47406,96 : POKE 47497,24 : POKE 47498,96
30 HOME : VTAB 7 : CALL -958 : PRINT TAB(10); "REMOVE DISK" :
  PRINT TAB(10); "INSERT PROTECTED DISK" : VTAB 10 : PRINT
  TAB(10); "PRESS ANY KEY WHEN READY" : GET AN$
40 CALL 42350
50 PRINT : PRINT : PRINT TAB(6); "(S)CAN DISK FOR CATALOG
  TRACK" : PRINT TAB(6); "(Q)UIT" : PRINT TAB(6); "SELECT S OR Q" :
60 GET AN$ : IF AN$ = "Q" OR AN$ = "q" THEN 180
70 IF AN$ = "S" OR AN$ = "s" THEN 90
80 GOTO 60
90 ST = 1
100 ONERR GOTO 170
110 FOR X = ST TO 34
120   HOME : PRINT "TRACK = " : X
130   POKE 44033,X
140   CALL 42350
150   VTAB 24 : CALL -958 : PRINT "PRESS A KEY FOR NEXT TRACK
  (Q = QUIT)" : GET AN$ : IF AN$ = "Q" OR AN$ = "q" THEN 180
160   NEXT X
170   POKE 216,0 : ST = X + 1 : IF B < 35 THEN 100
180   POKE 44033,17 : END

```

Noise and caution

As the program tries to CATALOG each track it is scanning, it will make a lot of noise. Do not be concerned. That's what happens when it tries to read a nonexistent directory on a track. After it tries each track, you will be given the option to quit, or continue the scanning.

When using this program there are two things you should be aware of. First, do not exit the track scanning routine by using the RESET key or turning the power off while the disk drive is running. The possible head bounce or transient currents generated by exiting in this manner may have undesirable results on your original disk. Always use the Q key to quit and avoid any damage. Quitting via the Q key also resets DOS to its normal value for the catalog track.

The second thing you must realize is that this method of searching for the catalog track that is used here, though simple, is crude. The errors and disk drive noises that occur, do so when the file manager routine in DOS checks sector zero of each track for the Volume Table of Contents (VTOC). Bytes 1 and 2 of the VTOC sector point to the starting track and sector of the directory entries. Since these bytes on nondirectory tracks are *dead ends* (but DOS does not know it) the disk drive arm moves, or tries to move, to this destination even if such a track does not exist on the diskette. Assuming you know what a VTOC sector should look like, a track dump program, such as the one found in Worth and Lechner's *Beneath Apple DOS* book, can be used first to see if a particular track is a catalog track.

How the program works

The simplest change made by this program, was the one that allows us to

catalog any track we want. The DOS code from \$ABDC to \$AC05 initializes DOS's File Manager work area and sets it up to CATALOG the diskette. At location \$AC01 (44033) the directory track value is assigned. To catalog a different track, only this one byte has to be changed. By changing this value inside a loop that runs from 1 to 34 and then issuing the CATALOG command (or its equivalent CALL 42350), it is possible to uncover a relocated directory track. In the worst case, this technique should produce 34 errors and one success.

The second part of the *Five Byte Disk Analyzer* defeats some types of protection by changing four bytes of DOS. The routine at \$B944 to \$B99F reads the address field of a track and is called by the File Manager in response to a CATALOG command. The routine checks for the presence of the address field prologue bytes \$D5, \$AA and \$96 so that it can locate the start of a track. It also saves the volume number, track, sector and checksum, verifies the checksum and finally locates the epilogue bytes \$DE and \$AA. In an unmodified DOS, if any of these steps fails, the 6502's Carry Flag is set at \$B942 and control is handed back to the File Manager. If the File Manager finds the Carry Flag set, the CATALOG command is terminated and an error message is displayed.

Fooling the File Manager

If we insert a clear carry (CLC) instruction and a return from subroutine (RTS) instruction before the checksum and the epilogue are checked, we can fool the File Manager into thinking that everything is okay and have it continue reading from the diskette. To make this change we must modify the bytes at locations \$B989 (47497) and \$B98A (47498). This is done by the last two POKES on line 20 of the BASIC program listing.

A similar change is also made in the routine that reads a sector from the diskette. This routine is located from \$B81DC to \$B943. The sector read routine checks for the first two bytes of the address field (\$D5 and \$AA) but not the third byte. It then reads the sector data and stores the 256 bytes in a buffer in high memory. As all the sector data is now in RAM, a similar clear carry and return from subroutine modification can be inserted in this subroutine, at \$B92D (47405) and \$B92E (47406). This is done by the first two POKES of line 20, and the File Manager is once again fooled into thinking that no errors exist.

If the protected diskette can be CATALOGed at this point, then individual files can be accessed and transferred one by one to a previously initialized, DOS 3.3 diskette. If the directory is still located on track 17 (\$11), then at this point you can BRUN FID and transfer the files much more easily than doing it one at a time. If the directory is located on a track other than 17, you'll have to transfer the file to RAM, POKE 44033,17, transfer the file from RAM to an unprotected diskette and then rePOKE the relocated directory track into DOS so that you can access the next file on the protected diskette.

Some diskettes that appear to CATALOG correctly but only display garbage may have their program set up to look and act like DOS. In other words, on booting, the program is loaded and executed as an addition to, or in place of DOS.

If this program has allowed you to access programs on a protected diskette, you may or may not have overcome the protection scheme. It's very possible that there is still some code buried in the program that checks to see if the program is on its original protected diskette. Run the program to find out.

UNPROTECTING LOCKSMITH 5.0

This article was received from one of our readers without a letter or return address. We would like to give the author credit and a subscription extension and if he contacts us we will.

By now, most of you are probably well aware of the value of backing up your software. Now that most software is protected, however, this task is much tougher, hence the popularity of publications such as *Apple Software Protection Digest* and bit copier programs. One of the first commercial bit copiers that was available was *Locksmith 2.0*, which has gone through ten updates to revision 6.0. From version 2.0 up through version 5.0 level F, all were copy protected.

Bit slip protection used

Version 5.0 level F of *Locksmith* was protected with a form of the bit-insertion technique that is called *bit slip* protection. It works this way. The disk drive controller reads a special series of bytes from the diskette once, recording the values as it reads them. Then it reads this series of bits again. Due to hardware limitations (and the added zero bits) the values will *not* match up each time. When a copy is made, the bits will be "cleaned up" and thus each time these special bits are read, they will match up. When that happens, the program decides that the disk it is reading must be an illegal copy, and thus refuses to boot.

Of course, it is possible to make a bit copy of *Locksmith 5.0* with *Copy II Plus* (using the sector copy) and *Essential Data Duplicator* version 3 or 4 (with parameter changes), but the copies you make will still be protected. You can also make a nonworking copy of the program with any copier, including *COPYA* and the fast disk backup routine of *Locksmith*. When a working copy of the program is run, location \$A9 in the zero page gets loaded with

a \$60 (the code for a return from a subroutine). On a bad (nonworking) copy, you will find that location \$A9 gets loaded with the value \$12. During the check to see if the disk is an original or a copy, the program does a jump to location \$A9. If the copy is good, it will encounter the return code and continue execution. But, if the copy is bad, it will encounter a \$12 and crash. Interrupting the program with a RESET will also scramble the code at \$A9, so you won't be able to restart it easily.

How to make a good copy

Since there is no unusual formatting of the diskette with *Locksmith* it is possible to make a "bad" copy of the program with any copy program. We will use *COPYP*, because as you will see in a few minutes, only a one-byte change, accomplished with the aid of the sector editor in *COPYP*, is necessary to convert a bad copy into a good copy.

Once you've made a copy of the *Locksmith* diskette, you could try to break it by scanning the diskette for a series of bytes such as 4C A9 00 for the JMP \$00A9 instruction or you could search for the code that stores a \$12 in location \$A9 when a bad copy is made and change it. The bytes you'd look for in this case would be A9 12 85 A9. Either search would be in vain, however, because the data on the diskette are encoded. The encoding method used is to EOR (exclusively OR) the data by the value of its location on a page of memory. In the case of the value \$12 which is stored when a bad copy is made, the \$12 is EORed with the value of its location, which is \$70. The result is \$62. If you know where to look on the diskette (and I'll tell you in a minute) you can find that value.

You can get a better understanding of what's going on by booting up *Lock-*

smith and once the main menu appears, getting into the monitor by pressing RESET if you have an old F8 ROM installed or using the *Wild Card* (from Central Point Software). Once you're in the monitor, you can try to search for the byte sequences mentioned earlier with a memory searching program, such as the one that appeared in *ASPD* Vol. 1. No. 3, p. 4. Doing this, you should find the 4C A9 00 code at \$1446 and the A9 12 85 A9 code at \$196F. Taking a closer look, we find three segments of code that are of interest:

```
A) 1446- 4C A9 00    JMP$00A9
    1449- 60                RTS

B) 196F- A9 12      LDA #$12
    1971- 85 A9      STA $A9
    1973- 60                RTS

C) 1B19- 4C 46 14    JMP$1446
```

Several cracks possible

A crack for this diskette can be implemented by modifying the code in anyone of the three segments shown above. In case A, we have a very short subroutine whose whole purpose is to jump to location \$A9. If the diskette is a good copy, this location will contain a \$60, and control will return immediately to the subroutine listed in A, where control is returned to the caller and the program continues to execute. A crack that deals with this segment of code would be to replace the three bytes of the jump code with three NOP bytes (\$EAs).

In case B, we have the actual culprit, the code that is executed to inform the program that a bad copy is in fact bad. In this segment of code a \$12 is loaded into the accumulator and then stored in location \$A9, where it waits until it is time to cause the copy to crash. The *Locksmith* protection scheme can be cracked by modifying the code in segment B in any one of three ways.

1. The first byte of code segment B (the \$A9) can be replaced with a \$60, which is the code for a return from subroutine, so that this subroutine is essentially bypassed and ignored.
2. The \$12 can be changed to a \$60 so that the correct value (to indicated a good copy) is always stored in \$A9.
3. The code that stores the \$12 can be replaced with NOP instructions.

Finally, in case C, we can crack the program by making sure that the program never jumps to subroutine that checks the protection in the first place. This would be done by replacing the three bytes of the JMP \$1446 instruction with NOP codes.

Undoubtly, the meticulous cracker could find several additional ways of cracking this program as well, but these are the most obvious. The best way to crack *Locksmith* is to always store the right value, no matter what happens. This prevents crashes caused by other possibly undetected routines that check to see if the copy is good.

A single sector edit is all that is required to convert a bad copy into a good copy. We want to make sure that a \$60 is stored in \$A9. Once the program is loaded into memory, the value of \$60 would be found at location \$70 of the page of memory it's located in. Earlier we said that the byte is encoded by EORing the value with the page location or \$60 EOR \$70, which equals \$10. So if we're going to make the bad copy good, we'll have to store a \$10 at the right spot on the diskette. To find the right spot on the diskette, we have to EOR each byte in case B with its page location (e.g. A9 EOR 6F, 12 EOR 70, 85 EOR 71, A9 EOR 72). If you do this, the byte sequence you wind up with is C6 62 F4 DB. If you search your "bad" copy of *Lock-*

smith for this byte sequence (using *Copy][Plus* or some other diskette scanner), you'll find the sequence on track \$0F, sector \$0E. As we said earlier, the byte we want to change is byte \$70.

Since we now know where the change has to be made, we can use *COPYP* to do it as it makes the copy to begin with. We do that by simply adding the following line to the *COPYP* program:

```
1000 DATA 15,14,112,98,16
```

That's all there is to it. You now have an unprotected copy of *Locksmith*. Next month we'll show you how to strip *Locksmith's* Fast Backup utility of the diskette and convert it into a BRUNable file.

How to Crack Dazzle Draw

(continued from page 5)

pressing T and then typing the name **DAZZLE.SYSTEM** when the program asks which DOS 3.3 file you want to convert.

STEP 26. **DAZZLE.SYSTEM** is now on our ProDOS backup diskette as a binary (BIN) file and must now be converted to a system (SYS) file. We can do this by typing in the following series of lines:

```
BLOAD DAZZLE.SYSTEM,A$2000
DELETE DAZZLE.SYSTEM
CREATE DAZZLE.SYSTEM,TSYS
BSAVE DAZZLE.SYSTEM,A$2000,
L$2A00,TSYS
```

That's all there is to it. You now have an unprotected copy of *Dazzle Draw* that boots up just like the original. You can make additional copies of this program by using any standard copy program such as *COPYA*.

COPYP FILE PARAMETERS

Listed below are several parameter files for use with the *COPYP* program that was presented in *ASPD* Vol. 1, No. 2. You can key these lines in directly or you can do what I do and create text files that contain these lines. Then you can load *COPYP* and *EXEC* in the appropriate file for the program you want to copy.

The following parameters were submitted by Brian A. Troha of Stoughton, WI. Thanks Brian, we're going to add an extra month onto your subscription for your contribution.

Karate Champ

Brian says that the *Karate Champ* crack started out when a friend asked him to make a backup copy of his original. He says he used *Copy][Plus*, but the diskette that resulted didn't work. He then checked the reset vector at \$3F2 and found the address \$BE93 (3F2-93 BE), so he searched the diskette for these values and located the protection routine on the diskette. The following lines when added to *COPYP* will let it copy *Karate Champ*.

```
1000 DATA 0,3,190,32,234
1010 DATA 0,3,191,0,234
1020 DATA 0,3,192,191,234
1030 DATA 0,3,193,144,234
1040 DATA 0,3,194,3,234
1050 DATA 0,3,195,76,234
1060 DATA 0,3,196,147,234
1070 DATA 0,3,197,190,234
```

REVIEW: Locksmith 6.0

by Scott Barrus

As you may recall from my review of *Locksmith 5.0 Level G* from an earlier issue of ASPD (Vol.1, No. 2), I have been anxiously looking forward to receiving the new version of Locksmith. It finally arrived, and I was very excited about some of the new features that it offers. Included in this new version of the program is an Auto Boot Tracer/Debugger, a fast copy utility that recognizes the presence of a Ramworks II card and uses that extra memory to speed up the copying process, a RAM card utilities section and last but certainly not least, easier-to-use parameters. The program still contains many of the good features from its predecessor too.

There are many difference between the new Locksmith and the old one. The first of these is in the manual. Compared to the 140-page manual that accompanied the previous version, the new 75-page manual for Version 6.0 is kind of skimpy. The new manual skims over some of the technical information and operating instructions have been cut to a bare minimum, and sometimes even below that. For example, the section on the Fast Disk Backup (FDB) states that "...FDB will automatically recognize any RAM cards in your Apple, whether they are in slots 0 - 7 or in the auxilliary slot of the Apple //e." The section on the Auto Boot Tracer (ABT) states, "...ABT will prompt you for the slot number of the RAM card. Key in digit from 0 to 7." Upon close examination we see that the memory in auxilliary slot 3 is not mentioned. The words *RAM card* seem to refer only to slot RAM cards and not Ramworks II style cards. When I tried to load the Auto Boot Tracer onto a Ramworks II card located in the auxilliary slot, it would not work, so I called Alpha Logic Business Systems. They said that a software technician would

contact me to help straighten out the situation. So far, no one has returned my call. I guess we can safely say that customer support is not one of the key features of this program.

Among some of the new and interesting features that are included in the new Locksmith program are:

- Disk, nibble, memory and track/sector editor for protected diskettes
- Framing bit analyzer
- RAM card utilities
- Automatic Boot Tracer/Debugger
- DOS 3.3 utilities
- Advanced disk recovery.

The disk, nibble and memory editor (formerly the nibble editor on L.S. 5.) can be used to manually read, search, change and rewrite information either in sector or nibble format. It can also edit ProDOS or DOS 3.3 files, or modify data on any RAM card that is installed in an Apple. Unfortunately, specific information on how to access RAM card data isn't in the manual.

The framing bit analyzer is actually a part of the disk, nibble and memory editor. It performs statistical analysis using precise timing loops and reports the timing relationship of the nibbles.

RAM card utilities are used to test RAM cards in the Apple. The utilities are somewhat limited however in that they cannot test any auxilliary memory found in the Apple //e or //c computers. In addition to testing slot-based RAM cards, the utilities can also dump the contents of any 16K RAM bank into main memory.

Boot tracer is nice, but...

One of the most effective ways of overcoming copy protection schemes is to trace the action of the program as it boots up. The problem with *boot*

code tracing, as it is commonly called, is that it can be complicated. Alpha Logic has attempted to simplify this process by including an automatic boot code tracer in this new version of Locksmith. and it works nicely, as far as it goes. But there are some problems with it, which lead us back once more to the program's ability, or should I say lack of it, to interface with all popular RAM cards. Although the program is supposed to be able to work with *any* RAM card, I could not get it to load onto the Ramworks II card or even into the back 64K of a 128K Apple //e or //c. It would only load into the top 16K. This causes some problems, mostly with ProDOS based software since this top 16K is where ProDOS resides. Thus it is not possible to use the automatic boot tracer to trace any ProDOS-based programs.

The DOS 3.3 utilities that are included with the program can come in very handy. You can use them to alphabetize a catalog, undelete a file, load a DOS file into memory for examination by the disk editor, display a disk free space map, correct catalogs and VTOCs and remove DOS from a diskette so that you'll have more room to store programs and data.

In addition, the program contains some advanced disk recovery utilities that can read partially clobbered diskettes and write out a good copy to another diskette. The disk recovery utilities can even recover data from a diskette that was writtin to while it was inserted into the disk drive off-center.

Parameters are easier to use

The programmers who produced this new version of Locksmith did their homework as far as the copy parameters are concerned. They've made them much easier to use. The same Locksmith Programming Language (LPL) is used in this version of

the program as in previous ones, and the parameters are the same too. But, unlike earlier versions, which loaded parameters in from a file based on the first letter of the parameter and then searching for the exact match, this version presents a list of parameter names. By moving the cursor to the correct name and pressing the RETURN key, you can get the parameters you need.

It makes fast copies

One of the nicest features of this program is its Fast Disk Copy routine.

If you have a 128K Apple //e or //c, the routine will copy an unprotected diskette in only two passes. However, if you have a Ramworks or compatible memory expansion card with more memory (I have 512K), a diskette can be copied in only *one* pass!

This new version of Locksmith seems to work much better than its predecessor and it is definitely worth getting. There are however a few areas where the program can stand improvement. To begin with, it would be nice if the program were setup so that it could be used from a hard disk. Cur-

rently the program is automatically loaded in with the operating system and no separate file exists that can be transferred to a hard disk. The other major objection I have to the program is its poor manual. More time and effort should have been put into producing a complete, informative and useful manual. If you're willing to put up with these drawbacks, however, you'll find this program to be a useful addition to your anti-copy protection arsenal.

Source: Alpha Logic Business Systems, 4119 North Union Rd., Woodstock, IL 60098. Call: (815) 568-5166.

BECOME AN ASSEMBLY LANGUAGE PROGRAMMING WHIZ

"Now That You Know Apple Assembly Language: What Can You Do With It?" will take you step-by-step through the assembly language programming experience. You'll delve into the mysteries of the 6502 stack and learn how to use it to increase the power and versatility of your programs. You'll also learn how to use the Apple's built-in routines to minimize the amount of coding you must do.

Control the output and the input

Frequently it's desirable to gain total control of the computer's output. This book shows you how to *steal control away from the Apple's normal output routines and redirect it to your own pro-*

*NOTE: Shipping and handling fees are not refundable.

**Redlig Systems, Inc., Dept. A 9783
2068—79th St., Brooklyn, NY 11214**

Please rush me _____ copies of **"Now That You Know Apple Assembly Language: What Can You Do With It?"** at \$19.95 each plus \$2 shipping and handling. I understand that if I am not delighted with the book I may return it within 10 days for a prompt and courteous refund. In any case, the Programmer's Number Conversion System and \$5 coupon are mine to keep.

Enclosed is my check for \$ _____

Please charge my credit card:

American Express MasterCard Visa

Card No. _____ Exp. _____

Signature _____

Name _____

Address _____

City _____ State _____ Zip _____

gram. Thus if you wanted, you could see the normally invisible control characters, display text on your screen as black on white instead of the normal white on black, format text sent to a printer into pages and much more.

Expand the power of your Apple by *stealing control away from the normal input routines*. Do things like adding a screen print capability, or *convert part of the normal keyboard into a numeric keypad*. It's even possible to *produce self-modifying programs* by EXECing in commands from RAM instead of from the disk drive. Think about the possibilities that offers for protecting your programs. When you want to go back to Applesoft programming, *you'll be able to do it faster with the aid of Applesoft Shorthand*, an assembly language program that types in one or more Applesoft commands at the press of a key, or use another program in the book to *automatically count the number of lines in your Applesoft program*.

With this book you'll also learn about *generating tones and how to figure out the frequency, producing sound effects, teaching your Apple to send Morse code, restoring accidentally erased Applesoft programs, adding new commands to Applesoft and running two Applesoft programs in memory together*, to name a few.

As an extra bonus for prompt ordering, you'll receive a **FREE coupon worth \$5 off** the price of a disk with all the assembled programs on it or a disk that contains the source code. These disks normally sell for \$15 each. We're offering these FREE gifts for a limited time only, so hurry! **Order today!**

Money-back guarantee *

We're so confident that you'll find this book invaluable and want it in your library, that we're offering a 10-day, no-questions-asked, money-back guarantee. Order the book. Read it and try the programs for ten days. At the end of ten days if you don't think it's worth every penny you paid for it, just send it back in resalable condition and we'll refund your money immediately, no questions asked.

ASPD PROGRAM DISKETTES AVAILABLE FOR ONLY \$15 EACH

Every month we will make a DOS 3.3 diskette available that contains all of the programs for the current issue of *Apple Software Protection Digest*.

Each diskette is available for only \$15 each. Diskette 1 contains the programs from issues 1 and 2. There are currently 4 diskettes available.

To order send a check, money order or your credit card number and expiration date to:

REDLIG SYSTEMS, INC.
2068 - 79th Street
Brooklyn, New York 11214

REDLIG SYSTEMS, INC.
2068 79th Street
Brooklyn, New York 11214

**BULK RATE
U.S. POSTAGE
PAID
BROOKLYN, NY
Permit No. 631**

