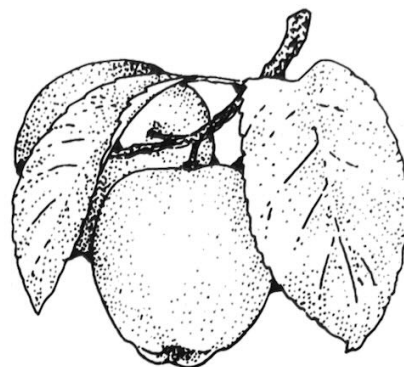


# APPLE SOFTWARE PROTECTION DIGEST

\$3.00



Vol. 1, No. 5

1986



## Contents

Editorial .....	1
Cracks Wanted .....	2
Letters .....	2
Bugs .....	2
Crack Index .....	2
DOS 3.3: In the Back Door, Through the Drive Door. ....	3
How to Crack It's the Pits. ....	5
The Ultimate Line Hider .....	6
Coming Next Issue .....	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 monthly 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.

## THE JUDGE WAS WRONG

In late October, a federal court judge in San Francisco issued a ruling that could have disastrous results for the American software industry. In his ruling, Judge William H. Orrick upheld the right of a software developer to copy-right the overall appearance, structure and sequence of screens used in a program.

The ruling came in a copyright infringement suit filed by both Broderbund Software and Pixellite Software against Unison World, Inc., and the programs involved in the suit were Broderbund's *Print Shop* and Unison's *Printmaster*. Broderbund's was the first of the two on the market and Unison's is functionally identical to it. What Unison apparently did was look at how the program operated and asked programmers to sit down and do the same thing, on their own, without stealing any code from Broderbund. Apparently they succeeded, because the suit did not charge Unison with "plagiarism." Rather, it claims that the use of a similar (although clearly different) screen format and a similar sequence of screens violates the program's copyright. Unfortunately for all of us, the judge upheld their claim.

### The public will suffer

If the decision is not appealed and overturned, all owners of personal computers will suffer for it. No longer will companies have to worry about produc-

ing a bug-free product. If they're the first to announce what potentially could be a good product, all they have to do is foist it upon an unsuspecting public and prove to the world that they were out with it first. No one can compete head on with them, because of their "look and feel" copyright.

Not only will we see the quality of initial offerings drop, but we'll also see the prices of this inferior software skyrocket. With no competition, there would be no incentive for developers to keep prices low. Another unhappy result of this decision will be to stifle the development of better software. If company A comes out with a product and company B figures out a way to make a similar but far superior product, shouldn't they be allowed to do it? In commenting on the judge's ruling, Lindsey C. Kiang, Lotus Corp.'s (the maker of 1-2-3) lawyer said, "I don't buy the idea that progress in the industry depends on copying other people's work." Those are pretty high and mighty words, but where would Lotus be today if it didn't copy the spreadsheet idea from VisiCorp, manufacturers of VisiCalc, the original electronic spreadsheet?

Judge Orrick we feel your decision was dead wrong and hope it gets overturned.

Jules H. Gilder

Publisher & Editor

## 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 so that some of our readers can try their hands at it.

1. Ace Writer II
2. Aztec
3. Bag of Tricks
4. Batter Up!
5. Blazing Paddles
6. Certificate Maker
7. Compress Software
8. Crush, Crumble & Chomp
9. Dazzle Draw
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

## Bugs

Although we spend a lot of time testing and rechecking all the information we present here, every once in a while a problem will crop up. As soon as I find out about it, I'll let you know in this column. It is my sincere hope that this column will be missing from most issues, and be very short in those issues in which it is included.

## Print Shop Companion

In Vol. 1, No. 2 we listed the COPYP

parameters the you could use to make a backup copy of the *Print Shop Companion*. Unfortunately there was a bug in it which was corrected in Vol. 1, No. 4. While this bug allowed you to make backup copies of the program, it wouldn't always work. We have now come up with a sure-fire solution which will consistently produce good backup copies of *Print Shop Companion*. Simply add the following lines to your COPYP program (see Vol. 1, No. 2, p. 2).

```
72 POKE 863,34
1000 DATA 3,6,25,160,96
1010 DATA 3,6,26,0,64
```

## Letters

Dear Editor:

My subscription to the **Apple Software Protection Digest** started with Vol. 1, No. 3 and it's just great. Your explanation of cracking techniques using Applewriter //e as an example suddenly clarified a whole bunch of things I've heard and read elsewhere, but couldn't quite grasp. Alas, I also discovered that Vol. 1, No. 2 apparently has some great stuff in it too, including a crack for the *Print Shop Companion*, for which I have been looking ever since last Christmas. Please send me a back issue of Vol. 1, No. 2, for which I am enclosing a check for \$3.

Donald L. Martin  
Santa Barbara, CA

*I'm glad that you enjoyed our article on "How to Crack Applewriter". We try to put things in simple, nontechnical language so that you don't have to be a hacker to use the information. A new version of the Print Shop Companion crack is in this issue of ASPD. A copy of Vol. 1, No. 2 has already been sent to you.*

Dear Editor:

Your publication has provided me with lots of information and ideas. Keep

up the good work. Enclosed with this letter is a short article on a simple modification to normal DOS 3.3 that renders some protection schemes ineffective. I would like to have this included in one of your future issues. If accepted, please give authoring credit and extend my subscription as you offered.

David Stoll  
Manitoba, CANADA

*I'm glad you enjoy our newsletter David. Thanks for your article, Five Byte Disk Analyzer. We've scheduled it for publication in the next issue of ASPD. Your subscription has been extended. Keep the articles coming.*

## 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  
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  
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  
Time Is Money - Vol. 1, No. 2, p. 9

## DOS 3.3: IN THE BACK DOOR, THROUGH THE DRIVE DOOR

There is a little documented trap or error in DOS 3.3 that will often allow you to get into BASIC or the monitor without any hardware modification. To understand the trap, one needs to examine the way DOS works. In the boot process, normal DOS loads itself and then proceeds to run the HELLO program of the file type set when the diskette was initialized. DOS contains *vectors* or pointers which tell it many things, including where to go in the event of an error. The standard DOS error routine is set up to print error messages when appropriate and then return to BASIC. When a program is running, the ONERR flag traps the condition without interruption.

The weak link in this chain of activity is the small period of time between the loading of DOS and the loading and execution of the HELLO program. Try an experiment with a write-protected backup copy of a normal DOS 3.3 diskette—the System Master will do just fine. Boot the diskette. The moment you see the Applesoft BASIC prompt ], open the door on the disk drive. DOS should be in the process of loading the HELLO program, and because it can no longer read data from the diskette, go immediately to the error handling routine which will print out a message saying that an I/O Error has occurred.

Now, close the door of the disk drive and type CATALOG. After a few moments of grinding sounds, during which time the disk drive is recalibrating the position of the head, you will see the catalog of files on the diskette. And, if you carefully examine the situation, you will find that you have complete access to all the normal DOS commands as well as the monitor. You can then load and examine programs at will. There is one thing you should be very careful about however, even though you have access to DOS and the monitor, you may find that the DOS that is loaded into memory may be a modified *protection* DOS. Part of this modification may

take the form of renamed DOS commands, so you'll have to examine some DOS memory locations to determine if the commands have been changed or not. You can easily do this by entering the short Applesoft program, *DOS COMMAND COMPARER*, that is shown in the listing.

### How the program works

The operation of this program is very simple. It looks in memory where that names of all the DOS commands are supposed to be stored—between locations 43140 and 43271—and takes the data stored there and converts it into a string (lines 50 to 70). In order to determine when the end of a particular command is reached, without having to waste an extra byte as a delimiter between commands (such as a space), the designers of DOS added 128 to the ASCII value of the last letter of each command. We can use this fact to determine where each command ends (line 40) and then set the string we've constructed equal to one of the elements in

the A\$(X) array (line 70). Now that we know what the commands are for the DOS we're examining, we can compare it to the standard set of DOS 3.3 commands by reading them in from the DATA statements in lines 180 to 240. This will produce a comparison listing on the screen. If you want a printed listing just include the following line to turn your printer on:

145 PR#1

and the next line to turn your printer off when the program is finished:

250 PR#0

### Useful for old programs and new

This method of breaking out of a protected program is most useful for older programs where only minor modifications were made to standard DOS to make it difficult to copy the program. This technique is also useful on the newer *hidden bit* protection schemes, where information is hidden between

```

1  REM DOS COMMAND COMPARER PROGRAM
2  REM
10 DIM A$(28)
20 N = 1
30 FOR X = 43140 TO 43271
40   IF PEEK(X) > 128 THEN 60
50   A$ = A$ + CHR$( PEEK(X) ): NEXT X
60   A$ = A$ + CHR$( PEEK(X) - 128 )
70   A$(N) = A$
80   A$ = ""
90   N = N + 1
100 NEXT X
110 PRINT : PRINT "DOS COMMAND COMPARER"
120 PRINT : PRINT "STANDARD DOS","THIS VERSION OF DOS"
130 PRINT "-----","-----"
140 PRINT
150 FOR X = 1 TO 28
160   READ T$: PRINT T$,A$(X)
170 NEXT
180 DATA INIT,LOAD,SAVE,RUN
190 DATA CHAIN,DELETE,LOCK,UNLOCK
200 DATA CLOSE,READ,EXEC,WRITE
210 DATA POSITION,OPEN,APPEND,RENAME
220 DATA CATALOG,MON,NOMON,PR#
230 DATA IN#,MAXFILES,FP,INT
240 DATA BSAVE,BLOAD,BRUN,VERIFY

```



bytes on a specific sector. In general, most software publishers try to maintain a high degree of similarity between their *protected DOS* and standard DOS 3.3 so they won't get caught off guard by hardware modifications by Apple that will make their software unusable.

Two examples of programs that this technique can be used on are *Font Downloader* by RAKWARE and *Smart Eyes* by Addison-Wesley. *Font Downloader* is an example of an older program which can be entered through the back door. After you break out of the boot process by opening the drive door and you fall into the Applesoft mode (where you'll see the ] prompt displayed), you will be able to LOAD and BLOAD all of the program files except two configuration text files (the flags for those are in the BASIC programs and can be manipulated easily). Now, in order to transfer these files to an unprotected diskette, just place a previously initialized DOS 3.3 diskette into drive 2 (you can use drive 1 and swap diskettes if you don't have a second drive) and LOAD or BLOAD each file into RAM one at a time and then SAVE or BSAVE it to the DOS 3.3 diskette.

If you're using the BLOAD and BSAVE commands for binary (B) files, you'll have to determine the starting address (in hexadecimal) of the file by looking at locations \$AA72 and \$AA73. This gives you the address in reverse order, low-byte first. Next, look at locations \$AA60 and \$AA61 for the length of the file, again in hexadecimal and in reverse order. Don't forget to use this information when BSAVEing a file. To BSAVE a binary file called TEST after you've determined the starting address and length of the file you would enter:

BSAVE TEST, A\$aaaa, L\$bbbb

where aaaa is the hexadecimal value of the starting address and bbbb is the hexadecimal value of the length of the file. *Smart Eyes*, from Addison-Wesley, is an example of one of the newer programs that uses the *hidden bit* protection

scheme. In addition to using the hidden bit technique, *Smart Eyes* also uses a nibble count routine for additional protection. This will have to be disabled. Instructions on how to do that follow.

### Disable the nibble count

*Smart Eyes* has a HELLO program that BRUNs SMART.OBJ, which has a starting address of \$4000 and a length of \$5480. SMART.OBJ does some house-keeping on start up and then implements the nibble count protection scheme. Let's look at some of the code from the beginning of the program:

```

1. 4000- AD AD AD LDA $ADAD
2. 4003- A0 00 LDY #$00
3. 4005- 99 63 08 STA $0863,Y
4. 4008- 88 DEY
5. 4009- D0 FA BNE $4005
6. 400B- 20 63 93 JSR $9363
7. 400E- A5 01 LDA $01
8. 4010- D0 F9 BNE $400B
9. 4012- AD 11 11 LDA $1111

```

As you can see from the partial listing above, I've added line numbers to what would normally be a simple disassembly produced by typing 4000L from the monitor mode. The line numbers are only there to make it easy for me to reference specific lines during the discussion. The protection checking routine starts at location \$400B (line 6) with a JSR \$9363. When the program returns from this routine, a flag value will be stored in location \$01 if everything is okay. The program then checks the contents of location \$01 to see if it is any value but zero (lines 7 and 8). If it is, it continues to execute the program (line 9). Otherwise, it goes back and runs the check loop forever (line 8).

The check loop can easily be disabled by replacing the branching code in line 8 with NOP (no operation) instructions as shown below in the new lines 8 and 9. With these NOP instruction in place, it doesn't matter whether the diskette passes the "protection test" or not, the program just continues to go on operating. When unprotecting programs with nibble counts, you may find that the program contains more than one call to

the nibble count routine, but in this case, the program only checks for the protection scheme once.

```

7. 400E- A5 01 LDA $01
8. 4010- EA NOP
9. 4011- EA NOP
10. 4012- AD 11 11 LDA $1111

```

### Cracking it step-by-step

Now I'll give you step-by-step instructions on how to crack *Smart Eyes*.

**STEP 1.** To start off with, you'll need to make a copy of the original diskette. NEVER WORK ON THE ORIGINAL!!

**STEP 2.** After you have a copy of the program, boot it up and get into DOS via the back door as explained earlier.

**STEP 3.** BLOAD SMART.OBJ and change the two bytes at \$4010 and \$4011 to NOPs (\$EAs) and then BSAVE the program back to the diskette.

That's all there is to it. You now have an unprotected copy of *Smart Eyes*. If you want to disable the nibble count on other programs you should look for a call to DOS' RWTS (Read and Write a Track and Sector) routine. Bear in mind that not all calls to RWTS will involve protection checking, especially if the program interacts with the disk a lot, but it's a good place to start. To find this routine scan memory for a JSR \$3D9, which would be represented by the byte sequence: 20 D9 03. I use the scanning program that S-C Software gives away on its *S-C Macro Assembler* diskette. It's short and easy to key in. A copy of it is listed in ASPD Vol. 1, No. 3, p. 4. Once you find the JSR, examine the four locations prior to it. These should contain an LDY instruction and an LDA instruction, which are used to setup the parameters for the RWTS routine. Once you've found that, look for a subroutine jump that goes to the RWTS setup code. In the case of *Smart Eyes*, the subroutine jump we want is located

(continued on page 10)

# HOW TO CRACK IT'S THE PITS

by Phillip Goetz  
Ellicott City, MD

*It's the Pits* is a program produced by Cactus Computer Co. of Moscow, Idaho. It is a total-load program which doesn't do any disk accessing while the program is running. Therefore, it is possible to unprotect the program and store the whole thing as a single disk file. In order to unprotect this program you will need at least a 48K Apple with one disk drive, a DOS 3.3 initialized diskette with at least 105 sectors free the old Integer F8 ROM or some other way to get into the monitor and of course the *It's the Pits* program diskette.

Boot code tracing proved fruitless for me, so I installed my old F8 ROM, booted up, pressed RESET, and looked through memory. The program begins at \$1800 and runs to \$3FFF, with pictures on non-displayable hi-res pages 4 and 5 (\$8000-BFFF). (Oddly enough, this program uses page 2 for display and page 3 as a background preserver. If you want to examine the shape generator which draws on the screen, it begins near \$1900.) On bootup, the program loads these 2 pictures at \$4000-7FFF and moves them to \$8000-BFFF. We can do this too. A little experimenting reveals that the game starts at \$185A. So we need our file to move 2 pictures to hi-res pages 4 and 5, set the reset vector to \$185A, and clear the high score to zero. By setting a high score, pressing reset, and searching zero page, we see that it is kept at \$63-65.

So, here are the steps to unlock *It's the Pits*. If you have an Integer BASIC Apple II, skip steps 2 and 3. Neither I nor Apple Software Protection Digest are responsible for any damage you do to your computer. The main danger is that you could break off pins while pulling the F8 ROM out or putting it in. If you are patient, you shouldn't have any problems.

**STEP 1.** Boot a normal disk with 105 free sectors and rename the HELLO program so it won't be loaded on bootup.

**STEP 2.** Open the Apple lid, touch the power supply to release any static electricity you may have, and carefully pry off the Applesoft F8 ROM. It is on the left-hand side of the motherboard in the middle between front and back with the words ROM-F8 in front of it. Note that there is some kind of mark on one end of the chip, consistent with marks on all the other chips. Whenever inserting a chip, use this mark to make sure that you put it in the right way. If you don't have a chip puller, pry the chip off by gently rocking it back and forth. Don't force it, and make sure you exert very little pressure so it doesn't come flying out and bend its pins.

**STEP 3.** Install an old Integer F8 ROM. You can buy these from most Apple dealers for about \$15. Note that with the old ROM, you have the monitor Step and Trace functions, but cannot use the ESC-I,J,K, or M. You can use ESC-A,B,C, and D, but they are less convenient.

**STEP 4.** Boot *It's the Pits*. When the picture appears, press RESET. You should fall into the monitor and receive a \* prompt.

**STEP 5.** Type 4000<8000.BFFFFM

**STEP 6.** Type the following startup code.

```
1828:A0 00 A9 00 85 42 85 3C
:A9 80 85 43 A9 40 85 3D
```

(continued on page 10)

```
5 D$=CHR$(4)
10 PRINT D$;"BLOOD IT'S THE PITS": HIMEM: 4096
20 TEXT: HOME: PRINT TAB(15)"IT'S THE PITS"
30 G= PEEK(7209): PRINT: PRINT "1. GRIMPI: "G:SL=
   PEEK(7251)+1: PRINT: PRINT "2. M STARTING LEVEL: "SL:
   PRINT: PRINT "3. N SPEEDS: ": FOR C=10976 TO 10981:
   PRINT PEEK(C)" ";: NEXT: PRINT PEEK(10982)
40 PRINT: PRINT "4. A SPEEDS: ": FOR C=10984 TO 10989:
   PRINT PEEK(C)" ";: NEXT: PRINT PEEK(10990)
50 U= PEEK(7048)-128:D= PEEK(7064)-128:L= PEEK
   (7056)-128:R= PEEK(7072)-128: PRINT: PRINT "5. UP "
   CHR$(U): PRINT: PRINT "6. DOWN " CHR$(D): PRINT:
   PRINT "7. LEFT " CHR$(L): PRINT: PRINT "8. RIGHT " CHR$(R)
60 PRINT: PRINT "9. RUN PROGRAM": PRINT: PRINT "WHICH? ":
   GET A$:C= VAL(A$): IF NOT C OR C<0 THEN 20
70 V=C*2+1: VTAB V: ON C GOTO 90,110,130,150,170,180,190,200
80 CALL 6184
90 HTAB 13: GET A$:G= VAL(A$): IF NOT G OR G>9 THEN 20
100 PRINT A$: POKE 7209,G: POKE 7229,G: POKE 7243,G: GOTO 20
110 HTAB 23: GET A$:SL= VAL(A$): IF NOT SL OR SL>6 THEN 20
120 SL=SL-1: PRINT A$: POKE 7251,SL: GOTO 20
130 HTAB 14: INPUT " ";C(0),C(1),C(2),C(3),C(4),C(5),C(6):
   FOR D=0 TO 6: IF C(D)>255 THEN 20
140 POKE 10976+D,C(D): NEXT: GOTO 20
150 HTAB 14: INPUT " ";C(0),C(1),C(2),C(3),C(4),C(5),C(6):
   FOR D=0 TO 6: IF C(D)>255 THEN 20
160 POKE 10984+D,C(D): NEXT: GOTO 20
170 HTAB 7: GET A$:U= ASC(A$): POKE 7048,U+128: GOTO 20
180 HTAB 9: GET A$:D= ASC(A$): POKE 7064,D+128: GOTO 20
190 HTAB 9: GET A$:L= ASC(A$): POKE 7056,L+128: GOTO 20
200 HTAB 10: GET A$:R= ASC(A$): POKE 7072,R+128: GOTO 20
```

# THE ULTIMATE LINE HIDER PROGRAM

by Grant Stevens  
Broad Brook, CT

When I first issued a challenge to ASPD readers to come up with a machine language program that would implement the line hiding technique that used the 5 colons (see ASPD Vol. 1, No.1) I had hoped that someone would come up with a complete program that would not only change the appropriate colon to a zero, but would also insert the colons automatically. But no one did, until recently when I received a program from Grant Stevens. Grant's program, which I call, "*The Ultimate Line Hider*" does that and more.

The program lets you protect a single line or a whole range of lines. In addition, Grant has made provision for allowing you to enter a four-character code which will be imbedded in each coded line. Finally, Grant has included a routine that will unprotect lines that were hidden with this technique. Since Grant implemented all of the features I originally specified in my challenge, I am giving him a free seven month extension to his subscription, even though his entry was late. You did a nice job Grant.

## How it works

*The Ultimate Line Hider* program starts out by setting the ampersand (&) jump vector so that the computer will jump to the start of the program whenever the ampersand key is pressed. The sixteen bytes of code that are used to set this vector will only be run once, after that, they can be eliminated. For this reason, Grant has placed them in the last part of the input buffer, starting at \$2F0, so if it is accidentally wiped out, it has no affect on the program.

The actual program starts on line 1610 where the processor status register is saved on the stack and a check is made to see if a number or a U follows the ampersand. If it is a digit, the pro-

```

1000 *****
1010 ***                                     ***
1020 ***   THE ULTIMATE LINE HIDER   ***
1030 ***                                     ***
1040 ***           by Grant Stevens   ***
1050 ***                                     ***
1060 *****
1070 *
1080 *
1090 *
1100 * SYNTAX: &[U] LNUM1 [-LNUM2] [STREXPR]
1110 *
1120 * RAM USAGE
1130 *
0006- 1140 STRING      .EQ  $6
003C- 1150 A1L        .EQ  $3C
003D- 1160 A1H        .EQ  $3D
003E- 1170 A2L        .EQ  $3E
003F- 1180 A2H        .EQ  $3F
0042- 1190 A4L        .EQ  $42
0043- 1200 A4H        .EQ  $43
0050- 1210 LNUM       .EQ  $50
0069- 1220 LOMEM      .EQ  $69
0094- 1230 MOVEND     .EQ  $94
0096- 1240 MVSTART    .EQ  $96
009B- 1250 LINADRS    .EQ  $9B
009D- 1260 STRLEN     .EQ  $9D
009E- 1270 STRADR     .EQ  $9E
00AF- 1280 PGMEND     .EQ  $AF
00B1- 1290 CHRGET     .EQ  $B1
00B7- 1300 CHRGOT     .EQ  $B7
03F5- 1310 AMPVEC     .EQ  $3F5
1320 *
1330 * ROM USAGE
1340 *
D39A- 1350 MOVEUP     .EQ  $D39A
D4F5- 1360 EXIT       .EQ  $D4F5
D61A- 1370 FINDLIN    .EQ  $D61A
D66C- 1380 CLEAR      .EQ  $D66C
DA0C- 1390 LINGET     .EQ  $DA0C
DD6C- 1400 STRCK      .EQ  $DD6C
DD7B- 1410 EVAL       .EQ  $DD7B
DEC0- 1420 SYNCHEK    .EQ  $DEC0
DEC9- 1430 SYNTAX     .EQ  $DEC9
FE2C- 1440 MOVDOWN    .EQ  $FE2C
1450 *
1460          .OR  $2F0
1470 *
1480 * Set up the ampersand (&) vector to jump to
1490 * the start of the program.
1500 *
02F0-A9 4C 1510      LDA  #$4C
02F2-8D F5 03 1520      STA  AMPVEC
02F5-A9 00 1530      LDA  #START
02F7-8D F6 03 1540      STA  AMPVEC+1
02FA-A9 03 1550      LDA  /START
02FC-8D F7 03 1560      STA  AMPVEC+2
02FF-60 1570      RTS
1580 *
1590 * Execution of the & begins here. Get the line number(s)
1595 * and optional code string.
1600 *
0300-08 1610 START    PHP          Save Hide/Unhide selection.
0301-90 11 1620      BCC  START.1  Branch if digit follows "&".

```

gram branches to line 1700 where the number is retrieved. If it's not a number, then the only other correct character is a capital U. In line 1630, a capital U is loaded into the accumulator and then a jump is made to the syntax checking routine in the Apple ROMs. If the character is a capital U the program returns from the syntax checking routine with the carry bit cleared and then a branch is made to line 1700 where the line number is retrieved. If the character is not the required U, then the carry bit is set and the program jumps to a routine in the Apple's ROMs that prints out the SYNTAX ERROR message (line 1660).

As was mentioned earlier, at line 1700, the number following the ampersand is retrieved here. The Apple's LINGET routine is used to do this. When it gets the number, it converts it to hexadecimal and stores it in zero page locations (LNUM) \$50 and \$51, low byte first. Next the accumulator is temporarily stored on the stack (line 1710) because the next routine that is called destroys the contents of the accumulator. This routine, also in the Apple ROMs, is called FINDLIN. It starts at the beginning of an Applesoft program and searches for the line number that is currently stored in LNUM and LNUM+1. If the line is found, its beginning address is stored in two other zero page locations called LINADRS and LINADRS+1 (\$9B and \$9C).

Upon returning from the FINDLIN routine, the program restores the accumulator by pulling the value it temporarily stored on the stack off (line 1730) and then checks it to see if there is supposed to be a second line number (line 1740). If there is, the program goes to line 1760 where an attempt is made to retrieve the number. If no number is found, a syntax error message is generated (line 1770). If a number is found, it is retrieved and converted to hexadecimal (line 1780). If there is not supposed to be a second line number, or if there is and it is successfully retrieved, control is passed to line 1790 where the buffer area that is used to store the four letter

0303-A9 55	1630	LDA	#\$55	If not a digit, must be a "U".
0305-20 C0 DE	1640	JSR	SYNCHK	
0308-90 0A	1650	BCC	START.1	Error if no line number.
030A-4C C9 DE	1660	JMP	SYNTAX	Print SYNTAX ERROR message.
				Clear Hide/Unhide from stack.
030D-28	1670	PLP		
030E-20 6C D6	1680	JSR	CLEAR	
0311-4C F5 D4	1690	JMP	EXIT	End.
0314-20 0C DA	1700	JSR	LINGET	Get the first line number.
0317-48	1710	PHA		
0318-20 1A D6	1720	JSR	FINDLIN	Point to the selected line.
031B-68	1730	PLA		
031C-C9 C9	1740	CMP	#\$C9	Is there a second line number?
				Branch if no.
031E-D0 08	1750	BNE	NOLNUM	
0320-20 B1 00	1760	JSR	CHRGOT	
0323-B0 E5	1770	BCS	SYNTAX1	Error—no line number found.
0325-20 0C DA	1780	JSR	LINGET	Get second line number.
0328-A2 03	1790	LDX	#\$3	Get ready to clear the
032A-A9 A0	1800	LDA	#\$A0	string storage area.
032C-95 06	1810	STA	STRING,X	Clear it.
032E-CA	1820	DEX		
032F-10 FB	1830	BPL	INITSTR	
0331-20 B7 00	1840	JSR	CHRGOT	Is there a string expression?
0334-F0 18	1850	BEQ	SETLOM	No, set LOMEM.
0336-20 7B DD	1860	JSR	EVAL	Yes, get it.
0339-20 6C DD	1870	JSR	STRCK	Make sure it's a string.
033C-A4 9D	1880	LDY	STRLEN	Find out how long it is.
033E-C0 04	1890	CPY	#\$4	Is it longer than 4 letters?
0340-90 09	1900	BCC	NOTRUNC	No, it's okay, continue.
0342-A0 03	1910	LDY	#\$3	Yes, truncate it.
0344-B1 9E	1920	LDA	(STRADR),Y	Copy string to new area.
0346-09 80	1930	ORA	#\$80	
0348-99 9E 00	1940	STA	STRADR,Y	
034B-88	1950	DEY		
034C-10 F6	1960	BPL	MOVSTR	
	1970	*		
	1980	*		Now, go through the selected range of line numbers
	1990	*		hiding or unhiding lines within the range.
	2000	*		
034E-A5 AF	2010	SETLOM	LDA	PGMEND
0350-85 69	2020	STA	LOMEM	Set LOMEM to default value.
0352-A5 B0	2030	LDA	PGMEND+1	This is required by the
0354-85 6A	2040	STA	LOMEM+1	EXIT routine.
	2050	*		
	2060	*		Process one line.
	2070	*		
0356-A0 01	2080	DOLINE	LDY	#\$1
0358-B1 9B	2090	LDA	(LINADRS),Y	Have we reached end of
				program?
035A-F0 B1	2100	BEQ	DONE	Yes, finish up.
035C-C8	2110	INY		No, do this line.
035D-A5 50	2120	LDA	LNUM	Get current line number
035F-D1 9B	2130	CMP	(LINADRS),Y	and see if it is within the
0361-C8	2140	INY		selected line limits.
0362-A5 51	2150	LDA	LNUM+1	
0364-F1 9B	2160	SBC	(LINADRS),Y	
0366-90 A5	2170	BCC	DONE	It's not, finish up.
0368-28	2180	PLP		Recall Hide/Unhide selection.
0369-08	2190	PHP		
036A-C8	2200	INY		Now Y=4.
036B-B1 9B	2210	LDA	(LINADRS),Y	
036D-90 27	2220	BCC	HIDELIN	Hide the line.
	2230	*		
	2235	*		Unhide one line.
	2240	*		
036F-D0 4E	2250	BNE	NEXTLIN	Don't unhide if not hidden.



code is blanked out. Then in line 1840, a check is made to see if the user entered a string that is to be hidden in each erased line. If there is none, the program jumps to line 2010. If there is, it is retrieved in line 1860.

Once the code data is retrieved, another of the Apple's ROM routines is used to verify that in fact the data is a string and not a number (line 1870). The routine that checks to see if a string was entered, also stores the length of the string in zero page location \$9D. The program next retrieves the length of the string (line 1880) and checks to see if it is more than four characters long (line 1890). If it's not, the Y-register is decremented by one (line 1950) so that it will contain the proper count (3) to handle all four code characters. If the string is longer than four characters, the string is truncated by loading a 3 into the Y-register (line 1910). In either case, the program goes to line 1920 next where each character of the code is retrieved in turn and the high bit is set to zero (line 1930). The string is then stored in a new area (line 1940).

After the string has been moved, LOMEM is set to its default value (the end of the program) and the program then begins to process a single line (line 2080). The first thing that the line processor does is to check if the end of the program has been reached. If it has, control is passed to the routine labelled DONE (line 2100). If not, the Y-register is incremented so that it will be pointing to the low byte of the line number and the number of the current line is loaded into the accumulator (line 2120). It is then checked to see if the current line is within the range of the lines we are processing. If it's not, the program branches to the DONE routine and finishes up. If it is, the processor status byte is pulled off the stack to retrieve the Hide/Unhide selection, and stored back on the stack for use again later (lines 2180-2190).

Next, the Y-register is incremented to 4, so that it points to the fifth byte in the BASIC program line (remember we

0371-A5 69	2260	LDA	LOMEM	
0373-85 3E	2270	STA	A2L	
0375-E9 05	2280	SBC	#\$5	Carry is set.
0377-85 69	2290	STA	LOMEM	Set up parameters for move.
0379-A5 6A	2300	LDA	LOMEM+1	
037B-85 3F	2310	STA	A2H	
037D-E9 00	2320	SBC	#\$0	
037F-85 6A	2330	STA	LOMEM+1	
0381-A5 9B	2340	LDA	LINADRS	
0383-85 42	2350	STA	A4L	
0385-69 04	2360	ADC	#\$4	+1 since carry is set.
0387-85 3C	2370	STA	A1L	
0389-A5 9C	2380	LDA	LINADRS+1	
038B-85 43	2390	STA	A4H	
038D-69 00	2400	ADC	#\$0	
038F-85 3D	2410	STA	A1H	
0391-20 2C FE	2420	JSR	MOVDOWN	Note: Y=4 (forced)
0394-B0 29	2430	BCS	NEXTLIN	
	2440 *			
	2450 * Hide one line.			
	2460 *			
0396-F0 17	2470	HIDELIN	BEQ	PUTSTR
0398-A5 69	2480	LDA	LOMEM	Don't hide if already hidden.
039A-85 96	2490	STA	MVSTART	
039C-69 05	2500	ADC	#\$5	Carry is clear.
039E-85 94	2510	STA	MOVEND	Set up parameters for move.
03A0-85 69	2520	STA	LOMEM	
03A2-A5 6A	2530	LDA	LOMEM+1	
03A4-85 97	2540	STA	MVSTART+1	
03A6-69 00	2550	ADC	#\$0	
03A8-85 95	2560	STA	MOVEND+1	
03AA-85 6A	2570	STA	LOMEM+1	
03AC-20 9A D3	2580	JSR	MOVEUP	Make room for 5 bytes.
03AF-A0 04	2590	PUTSTR	LDY	#\$4
03B1-A9 00	2600	LDA	#\$0	Get ready to hide the line.
03B3-91 9B	2610	STA	(LINADRS),Y	Hide it.
03B5-C8	2620	PSTR.1	INY	Point to next byte.
03B6-B9 01 00	2630	LDA	STRING-5,Y	Get character to hide.
03B9-91 9B	2640	STA	(LINADRS),Y	Hide it in line.
03BB-C0 08	2650	CPY	#\$8	Done yet?
03BD-90 F6	2660	BCC	PSTR.1	No, get another character.
	2670 *			
	2680 * Advance to next program line.			
	2690 *			
03BF-C8	2700	NEXTLIN	INY	
03C0-B1 9B	2710	LDA	(LINADRS),Y	Look for the end of the line.
03C2-D0 FB	2720	BNE	NEXTLIN	Didn't find it, keep looking.
03C4-98	2730	TYA		Found it, set up pointer so that it points to it.
03C5-38	2740	SEC		
03C6-65 9B	2750	ADC	LINADRS	
03C8-85 9B	2760	STA	LINADRS	Point to next line.
03CA-90 8A	2770	BCC	DOLINE	
03CC-E6 9C	2780	INC	LINADRS+1	
03CE-B0 86	2790	BCS	DOLINE	

```

1  REM BASIC PROGRAM TO INSTALL THE ULTIMATE LINE HIDER
2  REM
10 TEXT : HOME
20 PRINT : PRINT : PRINT
30 PRINT "INSTALLING THE ULTIMATE LINE HIDER..."
40 FOR X = 752 TO 975
50   READ Y

```



start with 0 so byte number 4 is actually the fifth byte). In line 2210 the fifth byte of the current BASIC line is loaded into the accumulator and then the carry bit from the processor status register is checked to see if the line is to be hidden or unhidden (line 2220). If it's to be hidden (the carry bit is clear), the program branches to HIDE LN on line 2470.

The first thing the HIDE LN routine does is to test the accumulator (which contains the value of the fifth byte) to see if it is already zero (line 2470). If it is, the line is already hidden and does not have to be re-hidden. If it's not hidden, then the program has to be moved up in memory by five bytes to make room for the data required to hide the line (lines 2480-2580). After the program has been moved in memory, a zero is stored in the fifth byte (lines 2590-2610) and the four character code (or four spaces) is added to the line (lines 2620-2660). Once the line has been hidden, the programs pointers are setup to point to the next line to be processed (lines 2700-2780) and then the program goes back to process another line (line 2790).

If the carry bit was set when it was tested in line 2220, the branch is not taken and processing continues with line 2250. This is where the unhide routine begins. The first thing this routine does is to check if the line was already hidden. If it wasn't, there's no need to unhide it and the next line is retrieved. If it was hidden, we simply have to move the BASIC program down in memory five bytes, thus wiping out the hide-a-line code. This is done in lines 2260-2420. The program then gets the next line (line 2430).

### Using the program

Using the *Ultimate Line Hider* is easy, but you must make sure you use the correct syntax. You may hide or unhide a single line or a range of lines. If you are going to include a four-letter code in you hidden lines, it must be surrounded by quotation marks. Thus to hide lines you could type:

&10 (no code included)

&10"JHG" (3 letter code included)

&10-100

&10-100"ABCD"

To unhide lines you must type in a command line that looks something like one of these:

&U10

&U10-100

When specifying a range of lines, the starting and ending line numbers must be separated by a hyphen and not a comma. To hide or unhide an entire program, use the range 0-63999.

```

60 POKE X,Y
70 NEXT X
80 PRINT : PRINT : PRINT "INSTALLATION COMPLETE."
90 PRINT : PRINT "TYPE 'CALL 752' TO RUN PROGRAM."
100 DATA 169,76,141,245,3,169,0,141
110 DATA 246,3,169,3,141,247,3,96
120 DATA 8,144,17,169,85,32,192,222
130 DATA 144,10,76,201,222,40,32,108
140 DATA 214,76,245,212,32,12,218,72
150 DATA 32,26,214,104,201,201,208,8
160 DATA 32,177,0,176,229,32,12,218
170 DATA 162,3,169,160,149,6,202,16
180 DATA 251,32,183,0,240,24,32,123
190 DATA 221,32,108,221,164,157,192,4
200 DATA 144,9,160,3,177,158,9,128
210 DATA 153,158,0,136,16,246,165,175
220 DATA 133,105,165,176,133,106,160,1
230 DATA 177,155,240,177,200,165,80,209
240 DATA 155,200,165,81,241,155,144,165
250 DATA 40,8,200,177,155,144,39,208
260 DATA 78,165,105,133,62,233,5,133
270 DATA 105,165,106,133,63,233,0,133
280 DATA 106,165,155,133,66,105,4,133
290 DATA 60,165,156,133,67,105,0,133
300 DATA 61,32,44,254,176,41,240,23
310 DATA 165,105,133,150,105,5,133,148
320 DATA 133,105,165,106,133,151,105,0
330 DATA 133,149,133,106,32,154,211,160
340 DATA 4,169,0,145,155,200,185,1
350 DATA 0,145,155,192,8,144,246,200
360 DATA 177,155,208,251,152,56,101,155
370 DATA 133,155,144,138,230,156,176,134

```

DON'T MISS A SINGLE ISSUE OF

Apple Software  
Protection Digest

SUBSCRIBE TODAY!

### DOS 3.3: In the Back Door

(continued from page 4)

at \$9373 where a JSR \$93F9 can be found. The code at \$93F9 looks like this:

```
93F9- A9 94      LDA #$94
93FB- A0 05      LDY #$05
93FD- 20 D9 03   JSR $03D9
```

This is the so-called *smoking gun* and is the clue to search for when you're trying to disable code that checks the status of the diskette.

#### Locking the back door

Now that we've told you how to sneak into many programs through the back door, you're probably wondering why this technique won't work with all programs. The answer is easy, many people simply lock the back door. Locking the back door is not difficult. All you have to do is locate the DOS error handling routine and patch it so that instead of printing out an error message, it jumps to the code that reboots the disk. We'll show you how to do this in a

future issue, but in the meantime think about it. You can use the book *Beneath Apple DOS* to find the necessary locations. If you come up with any interesting solutions, let us know.

### How to Crack It's the Pits

(continued from page 5)

```
:A9 FF 85 3E A9 7F 85 3F
:20 2C FE A9 5A 8D F2 03
:A9 18 8D F3 03 49 A5 8D
:F4 03 A9 00 85 63 85 64
:85 65
```

**STEP 7.** If you press ESC when the game menu is displayed, you will get a flashing number congratulating you on your high score. The number that appears has no apparent function. It is stored at \$1C5D, \$1C61, and \$1C65. To change it to zero, for instance, type:

```
1C5D: 0
1C61: 0
1C65: 0
```

**STEP 8.** Boot the normal disk with 105

free sectors with a 6 ctrl-P.

**STEP 9.**Type: BSAVE IT'S THE PITS, A\$1828, L\$67D8

#### You can customize the game

If you have 5 more free sectors on the disk and want to gain some control over the program, type in the following program that runs the game. I saved it under the name *PITS*. All of the modifications possible with this program are self-explanatory except for the speed changes. The fastest speed is 0, while the slowest is 128.

Do not modify the program to allow higher values than it does (i.e. over 9 Grimpi) or the screen will fill up with garbage during the game. Also, make sure that you never have more than 9 Grimpi. If you are finishing the second, fourth, or sixth level and have 9 Grimpi left, kill one off or else you will gain a tenth Grimpi when you reach the third, fifth, or seventh level. When the game tries to draw the shape for the "digit" 10, it will fill up the screen with garbage.

## COMING NEXT ISSUE

**How to Crack DAZZLE DRAW**

**Unprotecting LOCKSMITH 5.0 Level F and the Fast Disk**

**Backup Routine**

**Review: LOCKSMITH 6.0**

**Five Byte Disk Analyzer**

**Parameters for Backing Up THE HOBBIT, KARATE CHAMP  
and KUNG-FU MASTER**

# BECOM

You've spent a lot of time learning assembly language and finally know the difference between BEQ and IICB. It's time to put your new-found knowledge to work. Time to throw away your Applesoft programming manual and programs that make your Apple work like a super-charged, super-fast computer. Time to graduate from the Applesoft BASIC used by beginners, to the 6502 assembly language used by professionals.

To help make this transition, you need an experienced programmer to guide you. You need to develop a library of subroutines that make programming assembly language as easy as programming in BASIC. You need to take experienced assembly language knowledge and acquire. Most important of all, you need *You Know Apple Assembly Language? Do With It?* because it contains all this information.

## It shows you how, step-by-step

*"Now That You Know Apple Assembly Language, Do With It?"* will take you step-by-step through assembly language programming experience. You'll learn the mysteries of the 6502 stack and learn how to use the power and versatility of your program. You'll learn to use the Apple's built-in routines to make coding you must do.

## Control the output and the input

Frequently it's desirable to gain total control over the output. This book shows you how to *steal* the Apple's normal output routines and redirect them to your program. Thus if you wanted, you could send control characters, display text on your screen instead of the normal white on black, format output into pages and much more.

Expand the power of your Apple by *stealing* the normal input routines. Do things like *expand* the capability, or *convert* part of the normal numeric keypad. It's even possible to *protect* programs by EXECing in commands from the disk drive. Think about the possibilities of protecting your programs. When you want to *aid* of Applesoft Shorthand, an assembly language program that types in one or more Applesoft commands with a key, or use another program in the book to *count* the number of lines in your Applesoft program.

With this book you'll also learn about *getting* how to figure out the frequency, producing a teaching your Apple to send Morse code, *recovering* accidentally erased Applesoft programs, *using* commands to Applesoft and running two programs in memory together, to name a few.

# **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

<b>BULK RATE U.S. POSTAGE PAID BROOKLYN, NY Permit No. 631</b>
--

