ALPHA SYSTEMS

ATARI® SOFTWARE PROTECTION TECHNIQUES

GEORGE MORRISON

SOFTWARE PROTECTION TECHNIQUES
DISK UTILITIES

Atari Software Protections Techinques Disk Utilities is a utility package designed for use by software writers to help protect your software from illegal copying. The theory is described in the book (Atari Software Protection Techniques) that is included in this package, but this disk utility should help even the beginner use some of the methods described.

A menu of options will automatically appear on your screen when the disk is loaded (with the BASIC cartridge in), or just type RUN "D:MENU" from BASIC. Each of the utilities and options contain instructions which appear on the screen when the program is run. For your convenience, most of the programs are listable, and are well documented to help you understand them. I suggest you LIST or RUN each one to see the instructions, but type NO when asked if you wish to execute the program. Also all the program listings from the book are contained on this disk. For example, Figure 4.4 from the book is called "FIG44" on the disk.

The following information will help you better understand some of the programs on the disk

Directory Hider (called HIDER on the disk)

The directory Hider is used to help prevent DOS copies. It is especially useful for menu driven programs, or programs which must open files or run other programs from the disk. The Directory Hider hides your disk directory in a new location on the disk. Your programs will automatically use the hidden directory (because this program changes DOS to point to it). But others trying to copy your programs will see 707 free sectors.

WARNING - Make a back-up of the disk you wish to protect before running this program.

SETSCAN - This program will scan the sectors on the disk for bad or misassigned sectors. It asks for the starting and ending sectors you wish to scan, and then displays a message for each sector.

SECTLOOK - This displays the contents of a sector in character format. Just enter the sector you wish to read, and it will be displayed on the screen.

VTOCER - This program has two parts. Option One shows you which sectors on the disk are used and which are free (according to the VTOC).

Option Two is used to reserve space on a disk for a hidden directory. As indicated in the book, the directory should be hidden in a certain range of sectors. If HIDER can't find the space to hide your directory, a message will be displayed telling you to run this program. Complete instructions are displayed on the screen.

BADWRITE - This program enables you to protect your disks with bad-sectoring. The methods used to have your program check for the bad sectors are explained in the book. This utility lets you create bad sectors on your disk. There are two simple ways to create a bad sector on a disk using only standard hardware.

This utility will do both.

Bad Sector Writer Option One requires that you slow down your disk speed, but is much quicker than Option Two. Some types of disk drives cannot be slowed down enough to write bad sectors, so if yours is one of those, you must use Option Two.

USING OPTION 1. The first step in using Option 1 is to adjust your drive speed down to 220 +-10 RPMS. Before changing your drive speed, go to option 3 - ADJUST DRIVE SPEED. This option will help you get your speed properly adjusted. To write bad sectors adjust your speed to aprox. 220 +-10 RPMs. Try to write the bad sectors at the slowest possible speed without getting I/O errors. To get the best bad sectors, your disk should just barely be able to write.

Adjusting Your Drive Speed

To adjust your disk drive speed, it is necessary to remove the top cover and adjust one screw. To remove the cover, just pry off the four little tabs on the top of the drive with any sharp instrument. Then with a standard phillips head screwdriver, loosen the four screws (under the tabs) that hold the cover on, and gently lift off the cover. There are two basic types of ATARI 810 disk drives around. The newer drives have a circuit board across the top (see diagram "A"). The older drives have no circuit board across the top and have a large white plastic screw in the back left corner of the drive (see diagram "B"). This large white screw can be turned by hand to adjust your speed. It is very sensitive so a quarter turn may be all you need.

The newer drives are a bit trickier to adjust. To find the speed adjustment, look for a small green box with a tiny silver screw on it. It is located in toward the rear and a little left of center in the drive. It is very small but can be adjusted using a micro-screwdriver. The speed adjustment on the newer drive is not very precise. It may take as many as 8 complete revolutions to properly adjust your speed.

The next step is to write the sectors. Use option 1 to do this. Just enter the sectors you wish to create then press return. After all your bad sectors are complete, return to option 3 to adjust your speed back to normal.

USING OPTION 2

This method provides an alternate method of writing bad sectors. I still recommend option 1 as a faster and easier method, however if your disk drive was purchased after Jan 83, or you own a non-Atari disk drive, you may need to use option 2.

To use this method, you must attach two long pieces of tape (folded over onto themselves) firmly to the top of the disk you wish to write bad sectors on (see Diagram 1). Then insert the disk in your drive, so that the tape sticks out when the door is closed (be sure tape is long enough to get a grip on when the disk door is closed). Next, enter the destination disk drive number (the disk drive you wish to write bad sectors on) and the sector number. Then be sure everything is set and type return. The screen will now prompt you to shake the tape. You can gently move the tape back and forth, and alternately push on one piece while pulling on the other. The computer will beep and signal you when the bad sector is written. Then stop pulling while it rechecks the sector. Note that until you get good at it, it can take 10 minutes or more to write a single bad sector. So keep at it, and wait for the computer to signal you that it is done. If you wish to abort the process, hit any key.

DIAGRAM A

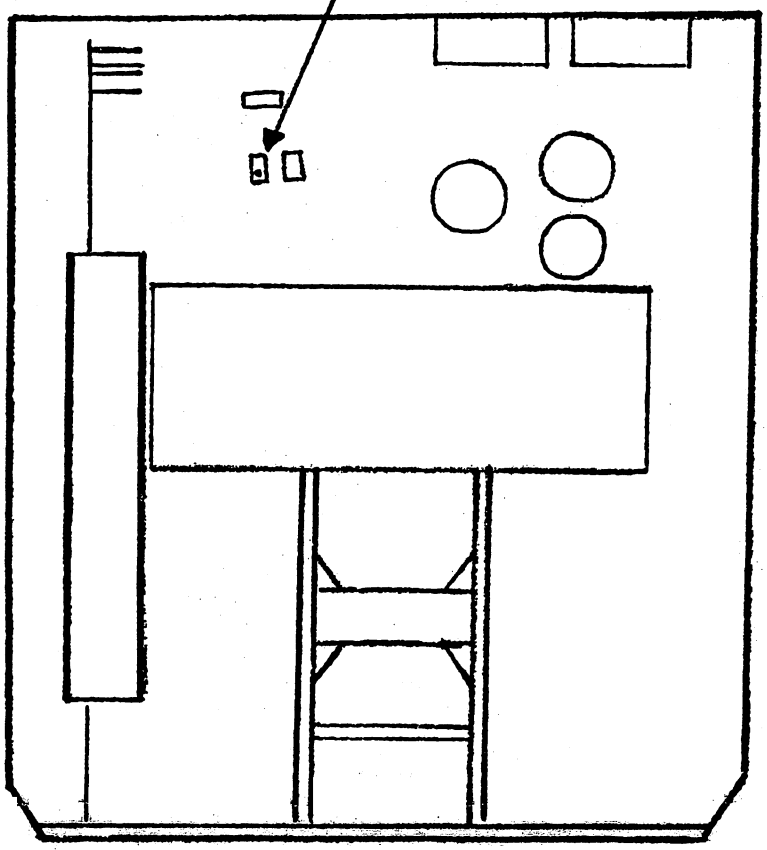


DIAGRAM B

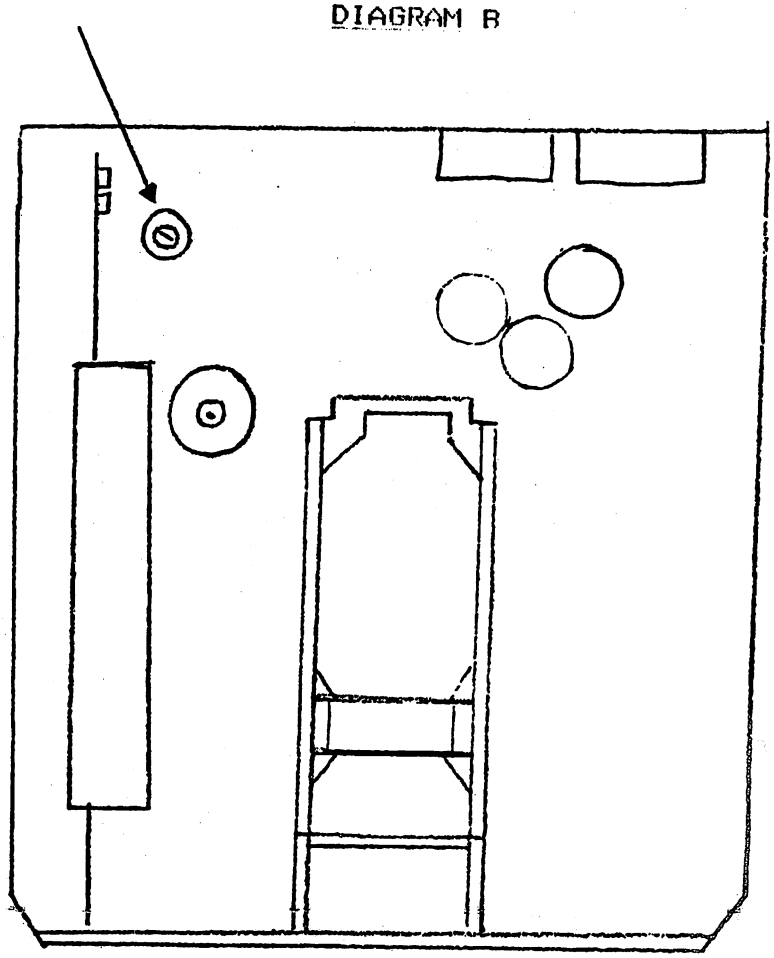
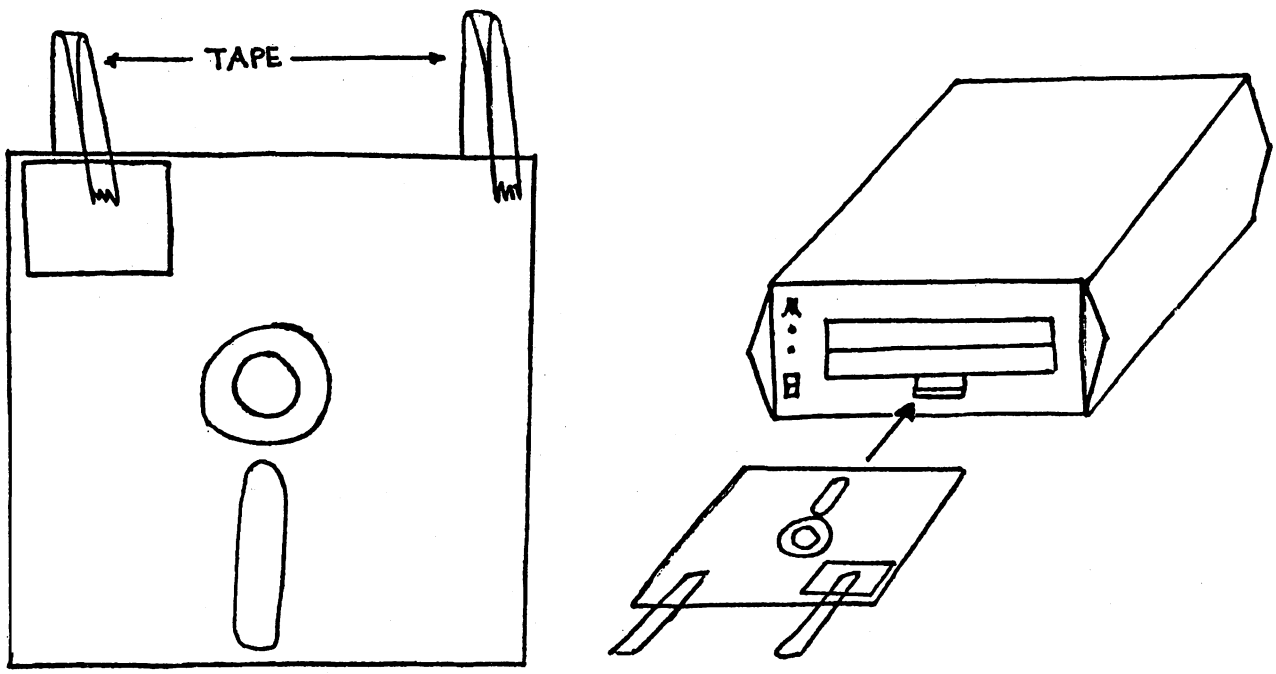


DIAGRAM 1



WARRANTY

ALPHA SYSTEMS warrants to the original consumer/purchaser that this program disk (not including the computer programs) shall be free of any defects in material or workmanship for a period of 60 days from the date of purchase. If a the disk fails to load during this 60 day warranty period, Alpha Systems will repair or replace the disk at Alpha Systems option, provided the disk and proof of purchase is delivered or mailed, postage prepaid, to Alpha Systems.

This warranty shall not apply if the disk (1) has been misused or shows signs of excessive wear, (2) has been damaged by playback equipment, or (3) if the purchaser causes or permits the disk to be serviced or modified by anyone other than Alpha Systems. Any applicable implied warranties, including warranties of merchantability and fitness are hereby limited to 60 days from the date of purchase. Consequential or incidental damages resulting from a breach of any applicable express or implied warranties are hereby excluded.

NOTICE

As with most computer software, all Alpha Systems computer programs are distributed on an "as is" basis without warranty of any kind. The entire risk as to the quality and performance of such programs is with the purchaser. Should the porgrams prove defective following their purchase, the purchaser and not the manufacturer, distributor, or retailer assumes the entire cost of all necessary servicing or repair.

Alpha Systems shall have no liability or responsibility to a purchaser, customer, or any other person or entity with respect to any liability, loss or damage caused or alleged to be caused directly or indirectly by computer programs sold through Alpha Systems. This includes but is not limited to any interruption of service, loss of business or anticipatory profits or consequential damages resulting from the use or operation of such computer programs.

The provisions of the foregoing warranty are subject to the laws of the state in which th disk is purchased. Such laws may broaden the warranty protection available to the purchaser of the disk.

ATARI SOFTWARE PROTECTION TECHNIQUES

by George Morrison

Forward by Ed Stewart
(Author of Letterman)

AN ALPHA SYSTEMS PRODUCT

Printing #6 batch 2, Jan 1985

Printed in the United States of America
10 9 8 7 6 5 4 3

Cover design by Richard M. Morrison

FOREWARD

The need for software authors to protect their property from theft is increasing. The unauthorized duplication of computer programs has become such a widespread activity as to threaten the very existence of the element that has helped to popularize the home computer most, namely the independent software entrepreneur. The phenomenal growth of the computer market has opened up vast new horizons of possibility for both the creator and the thief. It is therefore important for programmers to have at their disposal all of the available techniques to inhibit a potential pirate.

This book provides the fuel necessary to make an informed decision on what protection schemes would be most appropriate to employ and as such fills a void in the literature. It may be argued that disclosure of this information will only encourage piracy. I do not agree with this argument for two reasons. First, an advanced pirate is already aware of the contents of this book and would not benefit in the least from a review of it. Secondly, the would-be pirate does not have the technical acumen to break the protection techniques suggested herein. This book is therefore a valuable asset to you, the software author, in identifying the strong points and shortcomings of the various methods available today.

I found this book to be well written and authoritative in its approach to the

problem. I feel that most readers will find it both informative and helpful in their endeavor to protect their investment.

Ed Stewart, April, 1983
Honeybear Software

ACKNOWLEDGEMENTS

Alpha Systems would like to thank John Liang for the encouragement to market the book. Ed Stewart (the author of letterman) for his technical help. Helen Prozialeck for doing all the little things that helped to get the book written. Richard and Ethel Morrison for their help in making this book a reality.

PREFACE

This book is written with the average software writer in mind. Most of the software protection techniques presented here can be used by anyone with even a small amount of experience with Atari computers. Some topics covered do require a good amount of expertise to really understand, so small programs are included in the book. Some sample programs that make the techniques easy to use are on the optional software disk. Be sure to use the glossary in the back of the book, since some technical terms are needed to describe the protection processes. Also, it is advised that you read the chapters in order so that you can gain a working knowledge before reaching the difficult sections.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION TO SOFTWARE PROTECTION

Talk about bootleg record albums, tapes and movies has been increasing for several years. Copyright infringements by people who tape television programs has also been a growing problem. One fast growing area that hasn't seen much media coverage is software copying. It is estimated that there are two illegal copies of Visicalc (a popular spread sheet program) distributed for each one legally purchased. With the growing problem of software piracy, more people are writing about ways to prevent it. Sources that try to deal with the topic seem to focus only on legal protection techniques. They mention copyrighting or patents, but usually neglect to say that these methods have not been effective in stopping or even slowing down the problem. The reason for the failure of legal protection is the type of people who are pirating software. While record piraters may be big operators taking in millions, most software pirating is done by individuals. Catching them is almost impossible, let alone trying to legally prosecute each one.

WHAT IS SOFTWARE PROTECTION?

Software protection refers to techniques which discourage or prevent people from making copies. The techniques used can

take many different forms. The software producers can threaten legal prosecution of pirates; make a moral plea against copying; give idle threats; make software physically difficult to copy; or attempt to bypass the problem through the use of new marketing or distribution methods. The goal of software protection should be to maximize the return on the investment of the producers, not prevent piracy at any cost. If the protection method prevents the product from being brought to the market at a reasonable price or makes the product too difficult or tedious to use, then the producers have overlooked this goal. There are people so obsessed with protecting their software that it prevents them from selling their programs. Keep the goal of maximizing the return on your investment in mind when considering any software protection method.

THE CONCENTRATION OF THE BOOK

This book is written to help software producers deal with the problems of piracy. Legal protection and ways to discourage copying are dealt with in a good amount of detail, but the primary focus of the book is on methods that make copying physically more difficult. As mentioned above, the majority of copying is done by individuals who make copies for a few of their friends. Such people are not worried about the police coming to their door with search warrants, and the scope of the problem shows that pleading with

them not to copy, has little effect on the vast majority of software pirates. Protecting software should be like a bank protecting its money. To be effective, software protection must try to physically prevent piraters from being able to steal the software. Legal and coersive techniques may discourage some, but I feel the best protection makes copying so difficult that only a few people have enough expertise to copy it, and is so time consuming and tedious that those expert pirates give up before breaking the protection codes. The majority of this book deals with methods to achieve these goals.

THE PROS AND CONS OF SOFTWARE PROTECTION

The problems of piracy. Every bootleg program made deprives the producers of some of their earnings. Some programs can take months or years to write, so it is only natural that the writer wishes to get financial rewards for his efforts. The problem of piracy is so bad that many times the person receives an illegal copy of a program before he has even seen ads for the product. Loosely knit national software trading rings are making the bootleg software available even in the most remote areas. Obviously, something must be done about software copying, but for a moment, consider the other side to the problem.

The need for backups. Unfortunately, software is a fragile and volatile product.

Software can be destroyed by heat, humidity, wear, magnetic fields, faulty reading devices or even dirt. Just touching the exposed surface of a disk can destroy it. Most computer owners have learned that the only reasonably sure way to preserve your software is to make backups. Many people have come to rely on their software for balancing checkbooks and keeping their phone lists, etc., and would have serious problems should their software fail to work. In fact, many businesses have come to rely so heavily on their software that they could not function without it. Unfortunately, well protected software is intentionally very difficult to back up. On one hand we have the need to protect manufacturers from pirates and on the other, the need for the user to be able to back up his software.

The responsibility of the vendor.

Although backing up game software is obviously not as important as home filing or business software, all software should have provisions for backups. Some of the techniques that will be discussed later are well suited to deal with the back up problem, but at the very least, it is the responsibility of the vendor to provide quick and inexpensive back up service by mail. This can be as simple as an offer to replace malfunctioning software by mail for a small handling charge. Keep in mind that providing for backups removes one of the reasons for people to spend their time breaking software protection and may produce good will toward your product and company.

TOTALLY UNCOPYABLE SOFTWARE?

Can any program be made totally uncopyable? This question gets a qualified no. For most practical purposes, any software can be pirated. No matter how complex the protection technique, there are people who can break it. Any protection technique invented by man can be broken by man. Let us say for a moment that truly unduplicatable software is invented at some point in the future. If it is truly uncopyable, then even the manufacturer cannot copy it for distribution purposes. As you can see, truly uncopyable code is not good unless you only wish to sell that one copy. The trick to protecting software is not to make it completely uncopyable, but to make it difficult enough to discourage all but the most advanced and persistent would-be pirates.

I hate to qualify my original statement, but I can think of one case where software duplication is almost impossible. The case that comes to mind is the software available only on an information service like The Source or Compuserve. This is software which you never get possession of because your responses are transmitted to the computer which contains the software. This may be fine for adventure type games, but for the real time arcade graphics type games, it is virtually unusable. Presently the cost is high and the response slow, but maybe the future will show some hope for this method.

CHAPTER 2

GENERAL PROTECTION OF PROGRAMS WRITTEN IN BASIC

Protecting programs written in BASIC requires several protection techniques. In the following chapters, the specifics of dealing with tape and disk software are covered, however, certain protection is needed no matter how BASIC programs are stored. The real problem protecting BASIC code is the SAVE command. Protecting against the SAVE is probably more difficult than at first you would expect. This is because there are many different ways of stopping a programs execution and then saving it via a SAVE"D:program" or SAVE"C:" command. The topics that one must understand to stop this possibility are:

1. Disabling the break key
2. Disabling the system reset key
3. Preventing an error break
4. Preventing LOAD and SAVE combinations
5. Special cases where control must be given to the program user.

All these problems come up with BASIC programs because once loaded into memory, they are very simple to save to disk or tape if the program user is given the opportunity. The trick is not to give them the chance. In other words, your program must not lose control of the computer.

DISABLING THE BREAK KEY

Hitting the break key stops the execution of a program without clearing memory. This allows simple saving of the program. To disable the break key takes only two pokes. They are:

```
POKE 16,112
POKE 53774,112
```

Technically, you are changing the POKEY interrupt vector but suffice it to say that these pokes will disable the break key. It is important to note that these pokes must be repeated after each GRAPHICS command. This is because the GRAPHICS command refreshes those memory locations. Other commands can clear these locations also. To be on the safe side, it is good to repeat these pokes several times throughout the program. Since the break key is sometimes hit accidentally, these pokes are a good idea even in unprotected BASIC programs.

DISABLING THE SYSTEM RESET KEY

The system reset key is similar to the break key in that it stops execution without clearing the program from memory. There are two simple ways to disable this key. The first is:

```
POKE 580, 1
```

This POKE causes what is called a cold start. In other words, if system reset is pressed after this POKE, the system will

restart itself in the same way it would if you turned the computer off and then back on again, and all program memory is cleared. Another way to disable the system reset key is:

```
POKE 9,255
```

This will cause the system to lock up (keys won't work) if system reset is pressed. Either of these two methods is acceptable for disabling the system reset key, however, the first is preferable for disk based software because it will cause the disk to reboot.

PREVENTING AN ERROR BREAK

As you probably know, when an error is encountered in a BASIC program, execution is stopped and an error message is displayed. This provides anyone the opportunity to save your program. Of course, a well written program should not have errors in it, but often unusual input can cause them. For example, if the program asks for a number but the person enters a letter, it may cause an error. Also, unusual circumstances may arise that were not foreseen by the programmer. To help prevent these types of breaks, the BASIC command TRAP can be used. For those not familiar with this, it causes the program to go to a specified line number if an error is encountered. A complete explan-

ation is in your BASIC manual. It should be noted that once a TRAP is used, another TRAP statement must be issued if you want to continue stopping errors.

PREVENTING A LOAD AND SAVE COMBINATION

If your program requires the user to "LOAD" it or to "RUN" it, it is more difficult to protect. This procedure gives the user the opportunity to save it before it is run. One good way to prevent this is by having an automatically booting disk or cassette. The details of this method will be presented in the sections on disk and cassette protection, but essentially an autoboot system causes your program to run automatically when the system is started.

PROTECTING AGAINST LIST

There are several ways to prevent someone from being able to LIST your program. The one I feel is best causes the computer to lock up if a LIST (or any other command) is given with your program loaded in memory. Be sure to have a back up before using this procedure so you will have a listable version to work with should you decide to update your program. To use this procedure, just insert this line as the last line in your program:

```
32500 POKE PEEK(138)+256*PEEK(139)+2,0:
SAVE "D:program":NEW
```

For disk usage, change where it says "program" to the name you wish to save your program as. For cassettes, change the SAVE to a SAVE "C:". Type GOTO 32500 to save the protected version to the disk. This procedure changes the current statement pointer to 0 and will allow your program to run normally even though it cannot be listed.

SPECIAL CASES

At times, it is necessary or desirable to let the user have control of the computer and LIST or MODIFY your program. Even in these cases, it is still possible to protect your programs from copying. The trick here is to make your program need some conditions preset before it will run. One way to do this is to have your autoboot procedure point to a small initialization program that will then load your program. An example would be to have this initialization program POKE a small machine language subroutine into memory and then run your main program. For those not familiar with assembly language, here is a simple example that can be used:

```
POKE 1680,104:POKE 1681,96
```

This assembly language routine will just clear the stack and return to your program when called with a statement like this:

X=USR(1680)

This call can be put in various locations throughout your main program. These are used to make your computer lock up should it try to run without having POKED the subroutine into place. In other words, you can let your program user copy and modify your main program, but it will not work without running your initialization program which is protected. You may point out that the person can just remove these statements and then the program will run correctly. This is true, but hopefully they will think the statement serves some purpose other than just protection and not just remove it. One way to prevent the program user from being able to remove it is to make the subroutine perform some valid function needed by your program. Then, the program will not run (with or without statements) if your initialization program is not run first.

CHAPTER 3

CASSETTE PROTECTION

There are major problems with techniques used to protect cassettes. In fact, many companies have stopped releasing cassette based software because of this. Various methods are available that make cassettes harder to copy, but none offer protection against a copy system called "Audio Duping".

Audio duping is a technique used by large scale software producers to duplicate their tapes. It is also used by software pirates. Audio duping is done by directly recording one cassette from the other using two high quality cassette recorders. A cord is run from the output (or earphone) jack of one recorder to the input (usually aux in) of the other, and the cassette is copied with all filters and noise reduction systems turned off. This yields a working copy of the cassette.

This software protection problem arises from the fact that standard cassettes are used for software. With all the high quality stereo systems around, almost everyone can get access to a cassette recorder good enough to duplicate tapes used for software. Although the use of this method by pirates, stops the software manufacturer from preventing copying altogether, certain techniques are available which at least help protect the program against simple BASIC copies and tape to disk copies. They can also help prevent others from trying to

market your programs with only minor modifications and a different name.

First, the methods discussed in the previous chapter on general BASIC protection should be employed on cassettes using BASIC. One simple way to protect a program against the LOAD and SAVE combination is to make your program unlistable. The technique presented earlier to prevent the list command (see chapter 2) also protects against a simple save command. In this case, you must instruct the purchaser (in the documentation) to type;

```
RUN "C:"
```

A load would not work because once the program is loaded, the protection would prevent a run command from working. This technique also helps prevent a pirate from transferring the tape program to disk or modifying the program for resale.

Machine language programs offer different protection problems. Several companies market programs which transfer machine language programs from tape to disk. Fortunately, these programs are not effective at copying multistage loads. A multi-stage load program is one which loads in several parts. The program uses the standard boot procedure to load a routine, which then loads the rest of the program, or the program can be broken into several segments that load each other in turn. The use of these multi-stage loads is very effective against standard tape to

tape and tape to disk utilities. Only the first segment of a multi-stage program is copied by these utilities because they use the boot info from the beginning of the tape.

As stated earlier, these techniques can make copying or modifying cassette programs harder, but they are ineffective against audio duplicating systems. Some companies deal with this problem by producing two versions of their programs. One scaled down version on cassette, and a higher quality version on disk. This encourages disk owners to buy the disk version rather than just copying the cassette. The manufacturers also count on the fact that cassette owners are less likely to be familiar with copy techniques.

Cassette based programs do offer a wider market than disks and are much easier to produce than cartridges. However, from a software protection standpoint, cassette based programs have serious problems. The final decision on the use of cassettes should be made only after examining your objectives for your program carefully.

CHAPTER 4

GENERAL DISK PROTECTION

This chapter will cover disk protection in general. If the programs you wish to protect are in BASIC, then the techniques presented here should be used in combination with those presented in Chapter 2. The methods presented here are the first step to disk protection. The chapters on bad sectors, misassigned sectors, and hiding protection code, deal with more advanced forms of disk protection.

AUTORUN.SYS File

If your program requires the user to LOAD it before running (in BASIC or assembler), it can usually be copied very easily. A good way around this is with the use of an AUTORUN.SYS file or an autoboot disk. The name AUTORUN.SYS has special meaning to DOS. When a file with that name is on a disk, it will be loaded automatically when the computer is turned on (with the disk drive on, of course).

Using an AUTORUN.SYS file is very easy for assembler programmers. Any machine language program can be set up with a "RUN AT ADDRESS" (DOS option K) and renamed to AUTORUN.SYS. Then it will run automatically when the disk is booted. Another option for assembler programmers is creating an autoboot disk. This is a disk that carries the information to load your program in the boot sectors (sectors 1 and

2). See Atari Technical User's Notes for complete details on this method.

Creating an AUTORUN.SYS file is much more difficult for a BASIC programmer, since the file has to be a machine language program. Figure 4.1 shows a simple BASIC program that will create an AUTORUN.SYS file for you. When the program is run, it will create a file called AUTORUN.SYS that will automatically run your BASIC program for you. The BASIC program must be named "FIRSTPGM" for it to work. Also, the disk must have DOS.SYS on it.

Besides helping to protect your program, an AUTORUN disk has another advantage. It is much easier for beginners to use since it loads and runs your program automatically.

Preventing DOS Copies

The protection methods presented up to now are sufficient to prevent copies from BASIC. This section focuses on preventing DOS copies. From the Atari DOS menu, the user can duplicate a file (option O), copy a file (option C) or duplicate a disk (option J). Preventing these types of copies requires a knowledge of directories and VTOC's.

Disk Directories: The disk directory is probably the most heavily used part of the disk. Whenever a file is accessed (loaded, deleted, read, copied, etc.) DOS uses the directory. The directory contains the names, locations and lengths of all files on

FIGURE 4.1

```
10 REM ** THIS PROGRAM CREATES AN
          AUTORUN.SYS FILE, WHICH WILL
          AUTOMATICALLY
20 REM ** RUN PROGRAM "D:FIRSTPGM"
          WHEN DISK IS LOADED
30 OPEN #4,8,0,"D:AUTORUN.SYS"
40 FOR J=1 TO 153
50 READ A:PUT #4,A
60 NEXT J
70 CLOSE #4
80 DATA 255,255,0,6,138,6,162,0,189,26
,3,201,69,240,5
90 DATA 232,232,232,208,244,232,142,10
5,6,189,26,3,133,205,169
100 DATA 107,157,26,3,232,189,26,3,133
,206,169,6,157,26,3
110 DATA 160,0,162,16,177,205,153,107,
6,200,202,208,247,169,67
120 DATA 141,111,6,169,6,141,112,6,169
,15,141,106,6,96,172
130 DATA 106,6,240,9,185,123,6,206,106
,6,160,1,96,138,72
140 DATA 174,105,6,165,205,157,26,3,23
2,165,206,157,26,3,104
150 DATA 170,169,155,160,1,96,0,0,0,0,
0,0,0,0,0
160 DATA 0,0,0,0,0,76,0,0,0,34,77,71,8
0,84,83
170 DATA 82,73,70,58,68,34,32,78,85,82
,255,255,226,2,227
180 DATA 2,0,6
```

the disk.

The directory is loaded in the approximate center of the disk, in sectors 361 through 368, inclusive. It is created when the disk is formatted with DOS. A machine language program can do away with the directory all together if it is autobooting and doesn't need to access files. In general though, the directory is a required part of the disk. Figure 4.2 shows how the directory is stored on the disk. This diagram is included for advanced users but its understanding is not required.

VTOC's: VTOC stands for "volume table of contents" and is used to keep track of which sectors on a disk are full and which are free. Whenever a file is added or deleted from the directory, the VTOC is updated to show which sectors are now used or free. The VTOC is stored on sector 360 and its layout is shown in diagram 4.3. Understanding this diagram is not required but is included as an aid for advanced users.

Hiding Disk Directories and VTOC's

Hiding disk directories is a very effective technique for stopping novice copiers. It is very widely used and will prevent simple DOS copies. To be most effective in BASIC, this technique should be combined with stopping program breaks, system resets and other methods discussed in Chapter 2. This method is especially good for programs which automatically run

FIGURE 4.2

A DIRECTORY SECTOR LAYOUT
Director Entry

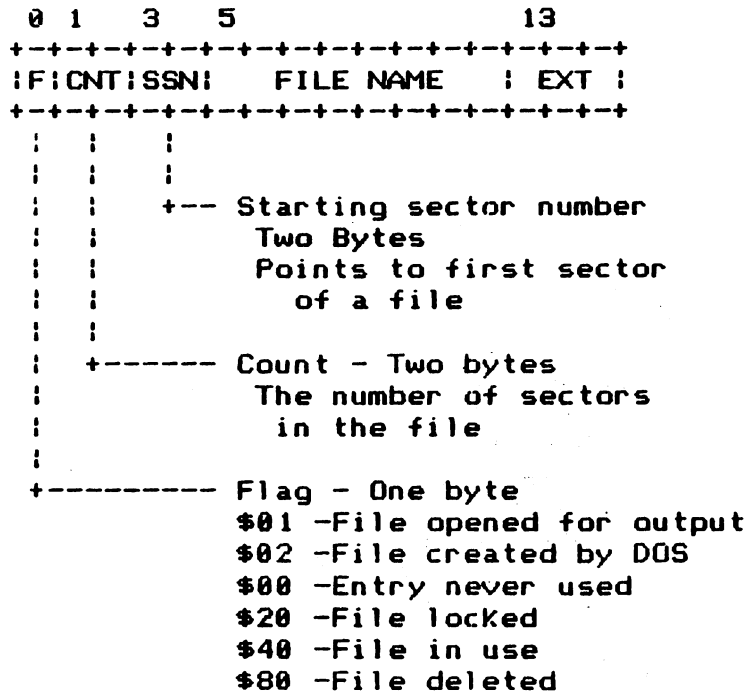


FIGURE 4.3

VTOC SECTOR LAYOUT (Sector 360)
$168 Hex

BYTES	
0	Type Code (0=DOS 2.0)
1-2	Total number of sectors
3-4	Number of unused sectors
5	Reserved
6-9	unused
10-99	Each bit in this area represents a spacific sector (0=used,1=unused)

other programs, and programs where the user may need to access specific listings or files. After it is used, a normal directory listing will show 707 free sectors (or whatever you want it to show), but your programs can still use the hidden directory as they please. Also, some files can be put in the hidden directory and the real directory, letting the user access certain files but not others.

The optional program disk has a program that will hide your directory automatically for you. If you did not purchase this disk, but would like to, see the back of the book for ordering information.

The routines to search for your file in the directory are part of DOS and are loaded when you turn on your computer. A method I developed to hide your directory involves altering part of DOS to point to a new directory in a different location. A warning should be given at this point: BEFORE USING THIS (OR ANY OTHER PROTECTION METHOD), BE SURE YOU HAVE MADE A BACK-UP. The back-up serves two purposes. First, the unprotected back-up gives you the means to change your program in the future (to modify it or just fix bugs, etc.). Secondly, the back-up is needed in case you accidentally damage or destroy your disk during the protection process. Hiding the directory involves five steps. They are:

1. Back up your completed disk.
2. Copy the directory to a new location.
3. Alter DOS to point to your new directory.

4. Write the altered DOS files to your disk.
5. Destroy or change the old directory, VTOC and DUP.SYS file.

Step 1: Back up your completed disk. Before you protect your disk, you should have it finished and complete because once protected, it will be hard to modify. Also, be sure to keep an unprotected back-up for the reasons mentioned above. One requirement of this technique is that you have the DOS files on the disk. If they are not there now, use DOS option H to write them.

Step 2: Copy the directory to a new location. It was stated earlier that the directory rides at sectors 361-368. Normally, DOS looks to these sectors to access files. To hide the directory, we will copy the directory to a new location, then later delete (or just alter) the old directory to trick a normal DOS. In other words, your program can use your hidden directory as usual, even though the real directory shows the disk to be different or empty. To move the directory requires a sector mover. Figure 4.4 contains a basic program that can move a sector of data from one location on the disk to another. In this case, we will move the eight sectors (which make up the directory) to a new location.

To keep this protection method simple, you must stay within certain restrictions. In this case, you can move your directory to anywhere between sector 255 and 510 (the

FIGURE 4.4

```
10 REM ** ROUTINE TO MOVE A SECTOR
          FROM ONE LOCATION TO ANOTHER
20 REM ** SET UP CIO CALL **
30 FOR I=1536 TO 1540:READ X:POKE I,X:
NEXT I
40 DATA 104,32,83,228,96
50 DIM A$(128),B$(1)
60 REM ** SET DRIVE **
70 DRIVE=1:POKE 769,DRIVE
80 REM ** SET COMMANDS **
90 RREAD=82:WWRITE=87:POKE 770,RREAD
100 REM ** GET SECTOR NUMBERS **
110 ? " FROM SECTOR";:INPUT FRMSEC
120 ? :? " TO SECTOR";:INPUT TOSEC
130 POKE 778,FRMSEC-(INT(FRMSEC/256)*2
56):POKE 779,INT(FRMSEC/256)
140 REM ** SET ADDRESS TO STORE READ *
*
150 ADRA=ADR(A$):POKE 772,ADRA-(INT(AD
RA/256)*256):POKE 773,INT(ADRA/256)
160 REM
170 REM ** EXECUTE CALL-CIO ROUTINE **
180 ? "HIT RETURN TO READ SECTOR ";FRM
SEC:INPUT B$
190 Z=USR(1536)
200 REM ** SET WRITE SECTOR **
210 POKE 778,TOSEC-(INT(TOSEC/256)*256
):POKE 779,INT(TOSEC/256)
220 ? "HIT RETURN TO WRITE SECTOR ";TO
SEC:INPUT B$
230 POKE 770,WWRITE
240 Z=USR(1536)
```

reason for this will be explained later). Where you move your directory in this range is not important, but it must be to unused sectors. Figure 4.5 contains a program that will tell you if the sector is free.

For the sake of simplicity, let's say you decide to move your directory to sector 501 through 508. To do this, you would run the sector mover (figure 4.4) and move sector 361 to 501. Then, run again and move 362 to 502, etc. until you reach 368 to 508. You are then ready for step 3.

Step 3: Alter DOS to point to your new directory. Changing DOS is very easy if you know what to do. In this case, we will change DOS to point to our new directory with just one POKE. Since DOS is stored in memory, we can change it by changing these memory locations. To cause DOS to look at our hidden directory, we will POKE location 4226 with a new value. It normally contains 105, and this tells DOS to look to sector 361 for the directory. To compute the new value to POKE into this location, just use this formula:

New POKE value = 105 + (hidden directory sector number - 361)

In our example we moved the directory to start in sector 501, so our new value to POKE would equal 245 (= 105 + (501 - 361)).

FIGURE 4.5

```
10 REM ** ROUTINE TO CHECK IF SECTOR
          IS USED **
20 REM
30 REM * NOTE; TO CLEAR UNUSED SECTORS
40 REM *    START WITH FORMATTED DISK
50 REM *    AND COPY YOUR FILES TO IT
60 FOR I=1536 TO 1540:READ X:POKE I,X:
NEXT I
70 DATA 104,32,83,228,96
80 DIM A$(128),B$(128)
90 REM ** CLEAR STRINGS **
100 A$(1,1)=CHR$(0):A$(128,128)=A$:A$(
2,128)=A$
110 B$(1,1)=" ":B$(128,128)=B$:B$(2,12
8)=B$
120 DRIVE=1:POKE 769,DRIVE
130 RREAD=82:POKE 770,RREAD
140 ? :? " WHAT SECTOR";:INPUT SECN
150 POKE 778,SECN-(INT(SECN/256)*256):
POKE 779,INT(SECN/256)
160 ADRA=ADR(B$):POKE 772,ADRA-(INT(AD
RA/256)*256):POKE 773,INT(ADRA/256)
170 Z=USR(1536)
180 IF A$=B$ THEN ? " SECTOR IS FREE":
GOTO 140
190 ? "SECTOR IS FULL":GOTO 140
```

So you would use this statement to change DOS to point to our hidden directory:

POKE 4226, 245

The reason why we can only move the directory to a certain range of values, is because of this POKE. The minimum value you can POKE is 0 and the maximum is 255, that is why our directory has to be hidden within these 256 sectors.

Step 4: Write altered DOS files to disk. In order to make the modification to DOS permanent, we must rewrite the DOS files. To do this, just type DOS and press return. If your directory was moved properly, the DOS menu should appear. If the menu does not appear, go back to step 2 and try again. For those who made it to DOS, type H (write DOS files). This will write your modified DOS to the disk, so your programs can find the hidden directory.

Step 5: Destroy or change old directory, VTOC and DUP.SYS file. Now comes the time to burn our bridges behind us. First turn your computer off and then back on and thoroughly test your programs. They now use the hidden directory. Next, we will delete or alter the old directory, VTOC and DUP.SYS files.

If you are not familiar with the DUP.SYS file, this file is created when you write DOS files, and must be deleted in order to protect your programs. The DUP.SYS is used to load the DOS menu. The

DOS menu could point a pirate directly to your hidden directory. To delete it, load the DOS menu from another disk, then return your disk to the drive. Use option D to delete the DUP.SYS file.

Next, let's get rid of the old (real) directory so others won't be able to find your programs. The easiest way to protect it is to delete it. Once again, use the sector mover (figure 4.4). This time, copy sector 720 (or any blank sector) to sectors 361-368. This will delete the old directory.

Advanced users may wish to just delete certain files and leave others intact so that your users could list or copy them. To do this, you would need to understand and modify the old directory. Figure 4.2 should be a big help in doing this. Keep in mind that the directory is not used by your programs, your programs use only the hidden directory.

The final step is to change your VTOC. The VTOC is used when a DOS command J (duplicate disk) is issued. This command copies all sectors which the VTOC shows to be full. Fortunately, there is a very simple way to make the VTOC say 707 free sectors (a blank disk). Again, we need our sector mover (figure 4.4). The trick is to copy the VTOC from a blank, formatted disk onto our disk's VTOC. The VTOC is stored in sector 360, so just insert a blank formatted disk in the drive and tell the sector mover to move sector 360 to sector 360. Then read sector 360 from your blank disk, and switch disks to write it to your program disk.

Now your disk is complete. I recommend

that you thoroughly test it again. Now boot up another disk in your drive and go to DOS. Then insert your protected disk into the drive and type A (display disk directory). Surprise! Your disk says there are no files and 707 free sectors, but it still runs your programs perfectly.

CHAPTER 5

BAD SECTORING

WHAT IS A BAD SECTOR?

A bad sector is a term often used by software producers trying to protect their software and by pirates. A bad sector is basically a sector on a disk which cannot be read accurately by the disk drive. This can be an unformatted sector, a sector that was written with a misaligned disk, a sector that was partially overlaid (usually caused by incorrect disk speed) or a sector that was physically or magnetically damaged. I will go into these further, but first, let me deal with the most common misconception about bad sectors and creating bad sectors. The most often asked question about bad sectors is, "Can't you just store bad or random data in a sector to create a bad sector?" The answer is, "No", since the data on a disk is stored as binary 0's and 1's, any pattern of them is valid and can be read by a disk as long as they are properly placed. In other words, it makes no difference to the disk drive what data is stored in a sector. If it can be read accurately, it is considered to be a good sector.

Another common misconception deals with creating bad sectors. People understand that an unformatted sector is bad (this is correct), and say that they can just format the sectors they want to be

good, and leave the others unformatted (to be bad sectors). Unfortunately, this cannot be done with the standard ATARI disk drive. The standard ATARI disk drive accepts only four commands. They are: READ SECTOR, WRITE SECTOR, CHECK STATUS and FORMAT DISK. Because the drive has its own 6507 microprocessor, it controls the actual functions involved with the four commands. The details of how to perform these functions is stored on a ROM chip in the disk drive. When a format disk command is sent to the drive, it takes over and formats the entire disk. Even turning off the computer will not affect the formatting. About the only way not to format the entire disk with an unmodified drive is to turn off the disk drive during the formatting process. This is difficult to control, but is sometimes effective in creating bad sectors.

HOW BAD SECTORS PROTECT SOFTWARE

Most ATARI users are aware that bad sectors are used to prevent the copying of disks, but wonder how they achieve this. A little history of protection techniques would help clear this up. For a long time, hidden directories and other disk protection methods explained earlier were the only methods available (and for that matter, needed) to prevent disk piracy. As sector copiers began to be readily available, it became obvious that some new method of copy protection was needed. It was known that the sector copier would duplicate all

readable data on a disk (at that time), so the problem became how to stop a copy from running. It was reasoned that the program had to have some way of telling if it resided on the original disk or a copy. The original disk had to have some characteristic that could distinguish it from a copy. Bad sectors fit the bill perfectly. The original disk could have a bad sector that could be checked by the program. In other words, the program would be able to tell if it resided on the original disk by checking for the bad sector. The program would run as usual if the bad sector was found, but if it didn't find the bad sector, it would know that it was on a copy disk and take some appropriate action (e.g. lock up the computer, attempt to format the disk, etc.).

CREATING BAD SECTORS

For a medium to large scale software producer, the best way to create bad sectors is to purchase custom hardware or special modifications for the 810 disk drive (See appendix "A" for a list of companies). For small scale software producers, there are several ways to create bad sectors without special hardware. Also, one should know of the techniques presented below because they are sometimes used by software pirates to create bad sectors. Once a bad sector is created on the original disk, most disk duplicating companies can

make batches of them easily (see appendix "A") which saves the small software company from having to recreate the bad sectors on all their disks to be distributed.

As mentioned earlier, there is no way to create bad sectors with the standard 810 disk drive from software alone, however, there are some special techniques that can be used. One very good method is to use other small computers to write bad sectors and tracks. Some computers like the APPLE and IBM-PC allow you to format single sectors and tracks. Their formats are not compatible with ATARI's, thus resulting in bad sectors. However, this technique is not effective if precise control of the bad sectors is needed, and you must have access to and knowledge of the other computers to use this method.

Another method is to physically damage the disk. I have seen this technique used successfully, but it has major drawbacks. Basically, you map out the disk, and using a pin or other sharp object, physically damage the sectors that should be bad. Needless to say, hitting the right sector is very difficult, and permanent damage to the disk must be done. A very similar technique is to magnetically damage sectors on the disk using a powerful magnet or a picoelectric device. This saves the disk from permanent damage, but it is even harder to place the bad sectors precisely where you want them.

Another technique sometimes used is to alter the read/write head alignment on your disk drive. This technique works but I strongly warn against using it because readjusting your alignment properly requires an oscilloscope, and a disk with improper alignment is usually incompatible with a properly aligned disk drive.

Finally, there are two techniques which are effective and relatively easy to use to create bad sectors. The first technique involves attaching a piece of tape to your disk jacket so that when the disk is inserted in the drive, the tape sticks out the door. Essentially, you shake the tape (which is attached to your disk cover) while a program is continually writing and reading the sector you wish to destroy. This technique works but can take as long as 10 minutes to write a single bad sector. The other technique involves adjusting the speed of your disk drive. This method is fast and very precise. It enables you to write as many bad sectors as you wish without doing any permanent damage to your disk. Your drive must be slowed to approximately 220 RPMs (so you can just barely write a sector without an ege to your disk. Your drive must be slowed to approximately 220 RPMs (so you can just barely write a sector without an error). Then, you have the disk write the sector you wish to destroy. When your drive is adjusted back to normal speed, those sectors will be read as bad sectors.

The optional software disk (ordering

information in back of book) that goes with this book contains programs which makes writing bad sectors by these methods easier.

CHECKING FOR BAD SECTORS

Obviously, to use bad sectors as a protection technique, the program must have some way to check for them. Fortunately, this is a simple process. Figure 5.1 shows a simple BASIC program that will help with this. All it does is read a sector, check the status byte and display a message saying if it got an error or not.

Figure 5.2 contains a modification to the program that will cause the computer to lock up if the sector is good, but continue if reading the sector, returned an error code. This routine can be used in your program to verify a bad sector. To use it, just insert this routine at the beginning of the program you wish to protect, then create a bad sector on the disk at the location checked in the program (currently set to sector 710 but can be changed to whatever sector you wish). Now the program will run only on disks that get an error trying to read the specified sector.

Note that this program will register any error in the attempt to read the sector. This means that if the disk drive is turned off or the disk is removed before the read, the program will continue to run as usual.

FIGURE 5.1

```
10 REM ** ROUTINE TO CHECK BAD SECTORS
20 DIM A$(128)
30 REM
40 REM ** SET DRIVE **
50 DRIVE=1:POKE 769,DRIVE
60 REM
70 REM ** SET COMMAND TO READ **
80 RREAD=82:POKE 770,RREAD
90 REM
100 REM ** GET SECTOR NUMBER **
110 ? "WHAT SECTOR ";:INPUT SECN
120 POKE 778,SECN-(INT(SECN/256)*256):
POKE 779,INT(SECN/256)
130 REM
140 REM ** SET ADDRESS TO STORE SECTOR
 AT **
150 ADRA=ADR(A$):POKE 772,ADRA-(INT(AD
RA/256)*256):POKE 773,INT(ADRA/256)
160 REM
170 REM ** SET UP CALL-CIO ROUTINE **
180 FOR I=1536 TO 1540:READ X:POKE I,X
:NEXT I
190 DATA 104,32,83,228,96
200 REM
210 REM ** EXECUTE CALL-CIO ROUTINE **
220 Z=USR(1536)
230 REM
240 REM ** CHECK STATUS CODE **
250 STTUS=PEEK(771):IF STTUS=1 THEN PR
INT "SECTOR WAS GOOD":END
260 PRINT "SECTOR WAS BAD":END
```

FIGURE 5.2

```
10 REM ** ROUTINE TO LOCK-UP COMPUTER
          IF CHECKED SECTOR NOT BAD
20 DIM A$(128)
30 REM
40 REM ** SET DRIVE **
50 DRIVE=1:POKE 769,DRIVE
60 REM
70 REM ** SET COMMAND TO READ **
80 RREAD=82:POKE 770,RREAD
90 REM
100 REM ** SET SECTOR NUMBER TO 710 **
110 SECN=710:POKE 778,SECN-(INT(SECN/2
56)*256):POKE 779,INT(SECN/256)
120 REM
130 REM ** SET ADDRESS TO STORE SECTOR
 AT **
140 ADRA=ADR(A$):POKE 772,ADRA-(INT(AD
RA/256)*256):POKE 773,INT(ADRA/256)
150 REM
160 REM ** SET UP CALL-CIO ROUTINE **
170 FOR I=1536 TO 1540:READ X:POKE I,X
:NEXT I
180 DATA 104,32,83,228,96
190 REM
200 REM ** EXECUTE CALL-CIO ROUTINE **
210 Z=USR(1536)
220 REM
230 REM ** CHECK STATUS CODE **
240 STTUS=PEEK(771):IF STTUS=1 THEN ?
"COPY DISK DETECTED":X=USR(0)
250 PRINT "PROGRAM RUNS NORMALLY":END
```

To prevent a pirate from using this technique to trick your program, it is a good idea to read a good sector after checking for the bad sector and bomb the program if this sector is not good. Essentially then, you first check the bad sector and bomb if it is not an error, then check the good sector and bomb if it is an error.

Now is a good time to mention hiding the protection code. Hiding protection code comprises a set of techniques which disguise your protection functions to help prevent a pirate from finding and removing the protection code. These techniques are discussed in detail in the next chapter. The importance of these methods cannot be overemphasized because if a software pirate can find and disable the protection in the program, this unprotected version can quickly spread through pirate circles.

CONCLUSIONS

Although bad sectoring is the most widely used protection technique, it has a major drawback. The drawback is that bad sectors can be created by anyone who knows how with just a standard 810 disk drive. This means that a pirate can copy the original disk (with a sector copier) and then create bad sectors on the copy wherever they were on the original. This would create a working copy, because all

the checks for bad sectors would yield the same results as the original. Needless to say, this technique is spreading fast through the pirate community and will soon make bad sectoring only effective against novice pirates. However, keep in mind that this protection technique can be applied by even the smallest software producers with the most limited resources, and when combined with some of the other methods discussed, is still effective against many pirates.

CHAPTER 6

HIDING PROTECTION CODE

In order to adequately protect software you must understand the techniques used by software pirates. One of the main techniques is called "hand breaking of the protection code".

Breaking Code by Hand

Hand breaking of software is one of the most powerful software copying techniques. This technique can copy programs using virtually any protection scheme and is considered practically impossible to stop. Breaking the code by hand is also the most difficult and time consuming copy technique used, and requires advanced knowledge of 6502 assembler and software protection techniques. The first step in using this technique involves listing the BASIC code or disassembling the machine language program. A disassembler takes a program (or section of a program) from disk, tape, or memory and converts it into assembler language. Converting the machine language into assembler language makes it much easier to read and understand, but it is still very difficult to find the protection instructions.

Some disassembler and debugger packages have advanced features which can make the process easier. A string search can be a great help in finding things like disk reads or status checks. A string

search, searches memory (or disk, etc.) for a selected number or string of numbers. A sector editor lets you read a sector from a disk and change the contents of it. A tracer lets you follow the program step by step. There are also many other utilities and tools that aid the pirater in finding and eliminating the protection techniques.

Once the code is listed and the protection steps found, the software pirate makes a "fix" to the program to bypass the protection. There are many ways to eliminate the protection once the code is found. Let's say the program checks for a bad sector in sector 700. One way to "fix" this would be to make it read sector 1,000 (a sector that does not exist) instead. This would return an error no matter what disk it reads. Another way would be to jump over the protection instructions, bypassing the check altogether. Still another way is to put a break in the code that would wait for a keystroke before continuing. Then, while running, the program would stop at the break and pause until a key is hit. This would give you time to turn off your disk drive so that it would get a bad sector status no matter what sector the program reads.

Although stopping an experienced and determined software pirate who can break programs by hand is extremely difficult, it must be attempted because this kind of copy is the most costly to sales. Once a program is handbroken to eliminate the protection techniques, anyone with a sector copier can copy it. In other words, once broken, any

number of copies can be made very easily and this unprotected pirated version can spread through circles of pirates extremely fast.

Hiding Protection Codes

Because hand breaking of protection is such a dangerous weapon in the pirate arsenal, it warrants major measures to stop it. The best way a software producer can do this is by making it very difficult to find and "fix" the protection code. This section will cover ways of hiding the protection code. This process disguises portions of the program in order to lead the would-be pirate astray and to make his job much harder.

Protection code is best hidden in an assembly language program, however, it is possible to use some of these methods in BASIC also. This section is geared mainly to those who are familiar with assembly language programming.

The first goal is to stop a simple string search from finding your protection and to prevent the disassembled program's protection methods from being obvious. A good example to demonstrate this is in checking for a bad sector. Normally, when a program checks a bad sector, it would call the CIO function of the operating system. A simple assembly language statement to do this is:

```
JSR $E453
```

This instruction shows up as a series of hexidecimal numbers in memory (which is the machine language equivalent of the instructions). A pirate searching for instructions that check for bad sectors, would see this instruction in the disassembled code and immediately study it to see if it is the protection code. The techniques presented below show how to hide this and other instructions so that they don't show up in a disassembly or string search. In general, these methods create the instructions only after the program begins executing. This process is referred to as "self modifying code".

Self Modifying Code: There are many ways to make programs self modifying. Perhaps the simplest is by overlaying your instructions. Using this method, the program could have an innocent instruction like STA $0000 that would be converted to a disk read (a CIO call) after the program begins executing. To do this, the program could store the numbers representing the disk access call (32,83,228 decimal) into the memory locations where the STA $0000 currently resides. These store instructions could be mixed in with other routines and separated from each other to help hide what they are doing. Only after the program begins running would the innocent STA instruction be transformed by the program into the call CIO instruction. Using this simple technique to disguise a few of the protection instructions makes the pirate's job a lot harder.

Another way to make your program's protection harder to decipher is through indirect addressing. Indirect addressing is where the instructions point to an address which in turn points to another address to complete the instruction. Heavy use of indirect addressing can help make the program much more complex for the pirate.

A tricky method to have your program create its own instructions is called "adding instructions". As you know, all instructions are stored in the computer as binary numbers. There is nothing forcing you to just move them to their locations. The instructions can be created by adding numbers together and storing them in their proper locations. A variation of this would be adding (or subtracting) numbers to other instructions to transform them to new instructions. In this way the code for a particular instruction or address is not even in the program until the instructions which create it are executed.

Probably the trickiest and hardest to decipher method of making self modifying code is combining areas of memory or disk sectors. This method uses two separate sets of meaningless numbers and combines them to create the program instructions. This combination can be done by ORing, ANDing, Exclusive ORing, etc. (these terms refer to assembler language instructions). Let's say for example, that two sectors on a disk contain what appears to be meaningless data. The program could read them into memory and then Exclusive OR (see glossary for definition) them together to create a

whole sector of instructions at one time. To use this method, the programmer must carefully set the sectors up so that they will combine to form the proper instructions. If you wish to get really devious, you would disguise the instructions that do the combining also. Needless to say, these techniques make finding the protection in a program much harder for the pirate.

To the average reader it may seem that the techniques presented above would prevent anyone from deciphering and removing the protection from a program. If you believe this, you are underestimating the skill and determination of advanced software pirates. To discourage the true diehards requires additional measures designed especially to wear down and antagonize your pirate adversary.

Layering Your Protection: This technique is similar to methods used to keep prisoners in jail. The bars on the cell represent only the first layer of protection. The would-be escaper must then get past the guards, get out of the building and finally, past the main wall. Software protection can use a similar form of layering.

After the pirate breathes a sigh of relief, convinced he has finally found and disabled the protection, it can be very discouraging to find that the program still won't run. Instead the pirate must deal with layer 2 of protection, etc. A good way to layer your protection is with "checksums". Checksums refer to adding up

certain areas of memory and comparing them to a stored value. To use this method, you could add up the memory locations which store your protection instructions and store the number in your program. Then, when the program runs, it could add these locations and compare it to the number you stored. If the pirate changes the protection instructions, they would no longer add up to the required number and the program could bomb. In other words, this method protects against someone changing your protection code. Of course, the pirate could then alter the checked value to reflect his changes, but this would add a whole new layer of protection which he would have to disable. You should now be convinced that hand breaking of your protection can be made very difficult, but here is one last technique to harass the pirate.

Wild Goose Chases: Different forms of this technique have been used effectively for centuries. It involves planting extra code in your program to deliberately lead the pirate on a wild goose chase. This could also be used to disguise the program further or just to lead the pirate astray.

Conclusion: All this probably seems like a lot of work to protect your program, it is. But, if it's any consolation, remember that the pirate may have an even harder time deciphering your work than you had creating it. Also, hiding the protection code fights the most dangerous form of

piracy. As stated earlier, if the protection is removed from a program just once, that copy can spread through the pirate community at a very rapid pace. Again, remember that no matter how clever you are in protecting your program, there will be someone who can break it.

CHAPTER 7

MISASSIGNED SECTORS

WHAT ARE MISASSIGNED SECTORS?

Misassigned sectoring is one of the most powerful disk copy protection techniques available today. There are very few people and almost no utilities (short of major disk drive modifications) that can successfully copy software protected by this method. Misassigned sectors (also known as custom formatting or duplicate sectors) are sectors with incorrect sector ID data assigned to them. This is a difficult concept, so I will start by explaining the normal disk format.

Each normally formatted disk has 40 tracks which are concentric circles or bands of data (see diagram 7.1). Each track has 18 sectors on it arranged in various orders, depending on the disk drive. A track also contains a 19th slice that serves as an index to define the start of each of the 40 tracks. (see diagram 7.2) Each sector contains 128 bytes of user data that can be read and written through the normal programming methods.

In addition, each sector contains 44 bytes of ID data that can only be used internally by the disk drive. This ID data is written and maintained by the drive's Floppy Disk Controller or FDC (the disk drive's internal controller) and is the key to misassigned sectors. The 44 bytes contain the following information:

1. a sector number

FIGURE 7.1

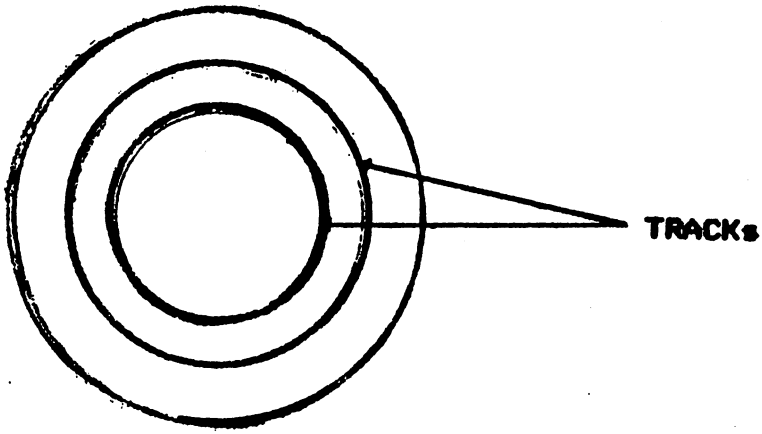
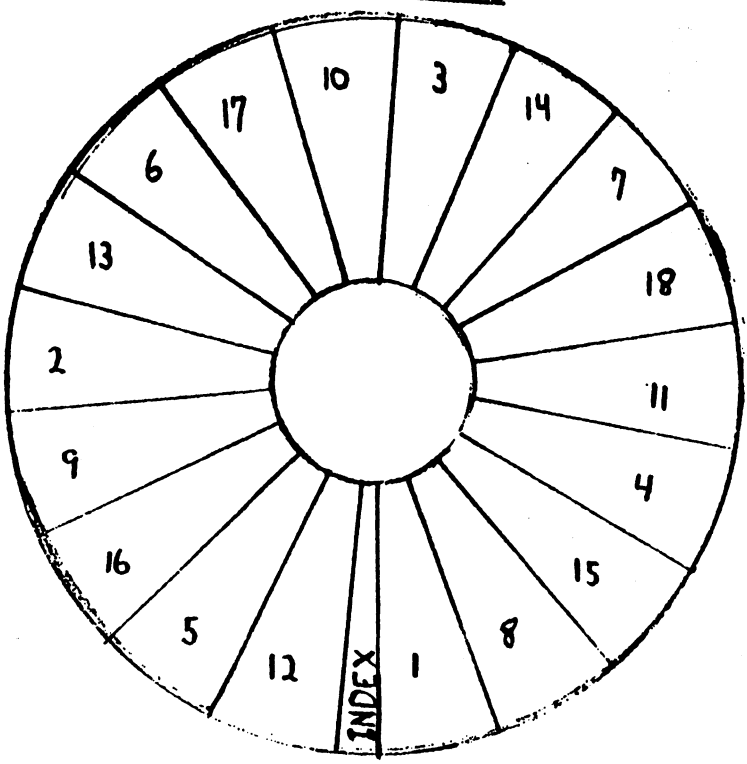


FIGURE 7.2



2. a track number
3. CRC's (cyclic redundancy checks)
4. a data mark
5. filler data

Misassigned sectors work by altering the sector ID data in ways that cannot be done on a standard 810 disk drive. Normally, the sector and track numbers are written when the disk is formatted and cannot be changed. The CRCs are automatically generated by the controller during every write operation and are used to verify the data when the sector is read (it works much like a checksum). Similarly, the data mark (normally a hex $FB) which is used to mark the start of the user data (the 128 bytes you are familiar with) is created automatically by the controller.

There are three basic types of misassigned sectors and all can be created by changing selected parts of the sector ID data. The types of misassigned sectors are: forced CRC errors, bad data marks and duplicate sectors. As their name suggests, forced CRC errors are sectors in which the CRC bytes do not match the data. Normally, this would signal what is called a soft error. When a soft error occurs, the system reads the data again to see if it can get a matching CRC. The disk drive does not grind (as with bad sectors) but will usually read the sector four times and seem to slow down. A bad data mark is a data mark other than the standard $FB and also causes a soft error. Duplicate sectors are the trickiest to use and detect.Normally, the sectors on a track are numbered 1

through 18, however, a disk using duplicate sectors might have 2 sector 17's for example. When a track has more than one sector with the same sector number, it is referred to as a duplicate sector.

HOW MISASSIGNED SECTORS PROTECT SOFTWARE

As with bad sectors, misassigned sectors allow the program to identify the original disk vs. a normally formatted copy disk. But the misassigned sector can also go further by causing copies to be missing whole sectors of data.

The program can check for the presence of a CRC or bad data mark error in a specific sector. If the error is not found, the program would bomb because it knows it resides on a copied disk (just like with bad sectors). However, with both CRC and data mark errors the data of the sector can remain intact. This means that the program can check the error and check (or make use of) the data on the sector as well. Even the most creative software pirate can't create a sector with a bad data mark or CRC which also has good data using a standard 810 disk drive. Figure 7.3 shows one way a program can check the data on a sector as well as its status.

Duplicate sectors are a bit harder to understand, so here is a simple example of how they might be used to protect a disk. Say the original disk is custom formatted to contain two sector 17's. One of them has data and one is all zeros. A simple way for

FIGURE 7.3

```
10 REM ** ROUTINE TO CHECK DATA AND
          STATUS OF A SECTOR **
20 REM
30 REM ** SET UP CALL-CIO ROUTINE **
40 FOR I=1536 TO 1540:READ X:POKE I,X:
NEXT I
50 DATA 104,32,83,228,96
60 DIM A$(128),B$(128)
70 A$=" ":A$(128)=" ":A$(2)=A$
80 REM
90 REM ** SET DRIVE **
100 DRIVE=1:POKE 769,DRIVE
110 REM
120 REM ** SET COMMAND TO READ **
130 RREAD=82:POKE 770,RREAD
140 REM
150 REM ** SET SECTOR NUMBER **
160 ? " WHAT SECTOR";:INPUT SECN
170 POKE 778,SECN-(INT(SECN/256)*256):
POKE 779,INT(SECN/256)
180 REM
190 REM ** SET ADDRESS TO READ **
200 ADRA=ADR(A$):POKE 772,ADRA-(INT(AD
RA/256)*256):POKE 773,INT(ADRA/256)
210 REM
220 REM ** EXECUTE CALL-CIO ROUTINE **
230 Z=USR(1536)
240 IF PEEK(771)=1 THEN 270
250 IF A$(1,27)<>"SECTOR MUST MATCH TH
IS DATA" THEN 270
260 ? "PROGRAM RUNS BECAUSE SECTOR
        IS BAD BUT DATA IS GOOD":END
270 ? "PROGRAM COULD BOMB BECAUSE
        SECTOR IS NOT RIGHT":END
```

the program to be sure it resides on the orignal disk is to read sector 17 twice in a row and compare the results. On the original custom formatted disk, the first read would get one sector 17 and the second read would continue on the disk and get the other. Comparing them would show they are not the same, and the program would proceed as normal. On a normally formatted disk (with the program copied on it), however, there is only one sector 17, so reading it twice would get the same sector both times. If the program finds the two reads are the same, it could end or lock up the keyboard, because it would know that it resides on a copied disk. Figure 7.4 contains a simple BASIC program that can check for a duplicate sector.

CREATING MISASSIGNED SECTORS

Misassigned sectors cannot be created using a standard, unmodified 810 disk drive. Although this makes the technique harder for a small software writer to use, it also means that this technique is very difficult to break. Special hardware or major modifications to an 810 disk drive are needed to create misassigned sectors. There are several advanced programming systems costing anywhere from $225.00 to $5,000.00 for the ATARI that have the capability to create, or at least copy the misassigned sectors. There are a few companies that make inexpensive 810 modifications that allow this. See appendix "A" for a partial list of companies that sell

FIGURE 7.4

```
10 REM ** ROUTINE TO CHECK MISSASIGNED
        SECTORS **
20 DIM A$(128),B$(128)
30 A$(1,1)="X":A$(128,128)=A$
40 B$(1,1)="X":B$(128,128)=B$
50 DRIVE=1:POKE 769,DRIVE
60 RREAD=82:POKE 770,RREAD
70 ? " WHAT SECTOR";:INPUT SECN
80 POKE 778,SECN-(INT(SECN/256)*256):P
OKE 779,INT(SECN/256)
90 REM
100 REM ** SET ADDRESS TO STORE FIRST
         READ **
110 ADRA=ADR(A$):ADRAL=ADRA-(INT(ADRA/
256)*256):ADRAH=INT(ADRA/256)
120 POKE 772,ADRAL:POKE 773,ADRAH
130 REM
140 REM ** CALC POKES FOR 2ED READ **
150 ADRB=ADR(B$):ADRBL=ADRB-(INT(ADRB/
256)*256):ADRBH=INT(ADRB/256)
160 POKE 772,ADRAL:POKE 773,ADRAH
170 REM ** SET UP CALL-CIO ROUTINE **
180 FOR I=1536 TO 1540:READ X:POKE I,X
:NEXT I
190 DATA 104,32,83,228,96
200 REM ** EXECUTE CALL-CIO ROUTINE **
210 Z=USR(1536)
220 POKE 772,ADRBL:POKE 773,ADRBH
230 Z=USR(1536)
240 IF A$<>B$ THEN 280
250 Z=USR(1536)
260 REM ** CHECK IF READS ARE EQUAL **
270 IF A$=B$ THEN PRINT "SECTOR WAS GO
OD":END
280 PRINT "SECTOR WAS MISSASIGNED":END
```

hardware, services, or modifications that can be used for misassigned or bad sectors. One note to large scale software producers, although misassigned sectors are tricky to create initially, most disk duplicating companies can copy them for you with no problems. So this technique doesn't slow down large scale production.

HOW PIRATES COPY MISASSIGNED SECTORS

Since special hardware is needed to create misassigned sectors, software protected by this method is usually broken by hand. In other words, manually breaking the protection codes (see breaking codes by hand in Chapter 6). There are several special techniques that help in hand breaking misassigned sectors. First, the would-be pirate determines the location of the misassigned and bad sectors. The duplicate sectors can be found by writing a program that reads all the sectors on a track in different orders, and compares the results. In other words, first they may read sector 1 and store it. Then read sector 1 again and compare it to what was just stored. If they are different, this sector is flagged as a duplicate sector. Sometimes the duplicate sectors are right next to each other and can be missed, so the sectors must be read in different orders to be sure all the misassigned sectors are found. Also, the program does a status check on bad sectors to see if they have data mark or CRC errors. Once all the misassigned sectors are found, the program is disas-

sembled and debugged with the basic hand-breaking methods discussed in Chapter 6. Once again, note that breaking by hand yields a program that is unprotected and can be copied by anyone as many times as they wish.

Another technique used to copy programs protected with misassigned sectors is by using special hardware or major disk drive modifications. In this case, the hardware enables the user to make an exact duplicate of the original disk (misassigned and bad sectors included). This technique requires no special expertise by the user, but yields an uncopyable copy. This technique makes an exact copy of the original, so the copy itself is protected from being copied. As mentioned earlier, the special hardware can cost anywhere from $225.00 to $5,000.00.

There is also a technique which is not currently used but may be available in the future. A company whose name I won't mention, advertised that their utility will copy all ATARI software available before a specified date. They have not yet delivered on their promise, but basically, their technique is this. After someone laboriously breaks each program by hand, the program fixes are saved on disk. Next, a program is written which identifies the program to be copied by the location of bad and misassigned sectors. So, when a copy is made, the fixes are put in to make the copy work. Essentially, they are selling the way to break the protection techniques on specific programs. This is close to selling

the program itself, but is a difficult thing to prove in court. As stated earlier, the company has not delivered on their promise (and has essentially ripped off many purchasers) but it will be interesting to see what the legal consequences are if they do.

PROTECTING MISASSIGNED SECTORS

Protecting misassigned sectors is much like protecting bad sector code. The major threat here is that someone will break your protection scheme by hand and distribute unprotected copies which will spread quickly through the pirate community. The best way to prevent hand breaking of codes is by using the techniques presented in the previous chapter, Hiding Your Protection Code. Hopefully, this will make it as difficult as possible to break the code by hand so that you will discourage all but the most talented and diehard pirates.

CHAPTER 8

ROM AND EPROM CARTRIDGES

ROMs are Read Only Memories. As you know, the ATARI operating system is on a 10K ROM board, and cartridge games are on ROM chips varying from 4K to 16K. There are two cartridge slots on the ATARI 800 and one on the ATARI 400 and 1200XL. The right hand cartridge slot on the 800 uses memory locations $8000 through $9FFF (HEX). The left cartridge slot uses memory locations $A000 through $BFFF for 4K and 8K cartridges and $8000 through $BFFF for 16K cartridges. When a cartridge is present, it will disable the RAM (on a 48K system) that uses the same addresses as the cartridge. At first glance, ROM cartridge software is a natural at being difficult to copy since no simple duplicate tape or sector copier would work here. However, reading ATARI's technical user notes will tell you most of what you need to know to save the cartridge data to disk or tape. There are also several programs floating around that do most of the work for you. Another way is by using an EPROM burner (more details on this later).

ROM COPY TECHNIQUE I

Essentially, to save a cartridge to disk or tape, a program dumps the memory locations (where the cartridge is stored) to the disk or tape. This data is then usually converted into a binary load file (some

cartridge copy programs create these automatically) and loaded using the binary load option of DOS. Some programs also need a special routine to clear out screen memory before running. Although this sounds complicated, keep in mind that once properly saved to disk or tape, an unlimited number of copies can be made, just by using DOS, and these spread around very fast.

PROTECTING AGAINST TECHNIQUE I

The copy technique mentioned above was effective on all cartridges up until about the time ATARI came out with Asteroids and Missile Command. Fortunately for cartridge makers, a good technique to thwart this copy method was found. This technique works by storing numbers into the memory locations where the program resides, which make the program bomb (or cause some error like a background with no player, etc.). As you know, you cannot store numbers into a ROM (Read Only Memory) so this process has no effect on the original cartridge. However, since the copy is loaded in from tape or disk and stored in RAM, this technique will stop it from running. Here is an example of how this technique might be used. Note that this example requires knowledge of assembly language and so is geared toward the more technically advanced user. Suppose that in memory location $A005 a JSR $CD (assembly language statement meaning jump to subroutine) is stored. Remember that $

means the number is in hexadecimal. This command would be represented by a $20 in location $A005, a $00 in location $A006 and a $CD in location $A007. With this instruction, the program would run correctly. To protect this program, one of the instructions in the program might store zeros in location $A007, so that when it reached this instruction, the program would jump there and bomb. So if the program is on a ROM cartridge, the instruction to store zeros in location $A007 would be ignored because you can't store the numbers in ROM, and the program would execute normally. If, however, the program was loaded into RAM from a disk or tape, this store instruction would alter the program in RAM and when it hit the JSR (jump to subroutine) it would bomb because there would be no subroutine there.

Using this protection technique makes copying the ROM cartridge much more difficult because you would need a good assembly programmer to find the instruction causing the error. Of course, this method can be made even more effective by hiding the protection instructions. The techniques to hide the instructions are very similar to those used to hide disk protection instructions, and as always, the more complicated and convoluted they become, the better your chances are of them not being broken.

ROM COPY TECHNIQUE II

This copy method has some drawbacks

but, all in all, is the simplest and most effective ROM copy technique being used by software pirates today. To explain this technique, I will introduce a new term - EPROM chip. An EPROM chip is an Erasable, Programmable, Read Only Memory (also PROM, Programmable Read Only Memory chips can be used but cannot later be erased and revised). EPROMs are just like ROMs except that the instructions can be altered using a device called an EPROM burner. This, figuratively speaking, burns instructions into the chip and makes it operate just like a ROM. To erase an EPROM, just put it under an ultraviolet light and it is ready for reprogramming. An EPROM burner with the proper software can also read the contents of a ROM (or EPROM, etc.) and store those instructions to disk or tape. This permits one to strip off the contents of a ROM cartridge and then reproduce a duplicate EPROM or PROM cartridge. This EPROM will operate exactly like the original. Even the protection technique mentioned above is totally ineffective against the EPROM copy. EPROM burners range in price from $20.00 (in kit form without software) to several thousand dollars, but one that would adequately do the job for ATARI cartridges would cost about $120.00 to $200.00. That puts an EPROM burner within the grasp of most serious computer owners.

PREVENTING ROM COPY TECHNIQUE II

This is a good news/bad news situation. The good news is that to use an EPROM burner takes a certain amount of

expertise not everyone has, and the cost per copy can be quite high. For example, for a two chip, 8K cartridge, chips themselves can cost between $3.00 and $9.00 and the board to mount them on can cost $10.00 to $20.00, not to mention the cost of the EPROM burner itself. The bad news, however, is that for those with the time, money, and expertise to make EPROM cartridges, there is no effective technique to stop them currently available. Note though that the cartridges made by this method are exact copies, and so offer the same level of protection as the original. In conclusion, ROM cartridges remain one of the best ways to distribute your program from a protection viewpoint.

CHAPTER 9

HARDWARE DATA-KEYS

As its name implies, a hardware data-key is a hardware device. It usually plugs into the joystick port and can be "read" by the computer like a joystick or paddle. A hardware data-key accompanies a program and must be plugged in for the program to operate properly. Its sole purpose is to protect the program from being copied by software pirates. An added function of the data-key could be to allow the purchaser protection of his files from others. For example, if a data base program is protected by such a key, the user can use the key to help prevent unauthorized access to his files. Hardware data-keys have the potential to be one of the safest and best protection techniques used. The purchaser can be allowed to back up the program as many times as is needed, but without the data key, copies are worthless. This means that data-keys potentially solve one of the biggest problems with software protection because they prevent copies from functioning for pirates, but they allow the purchaser to have functional backups.

How Data-Keys Protect Programs

The simplest way for hardware data-keys to work is by having the program check the value passed from the key and compare it to a value stored in the

program. If the values passed from the key are incorrect (meaning no key or a counterfeit key is present) the program bombs or self-destructs. A more complex system might have the key pass several values or even use a few separate keys plugged into joystick ports or the serial interface port.

To have the program checked for the presence of the key is very simple. Here is an example of a basic statement that will check for a simple data-key:

```
IF PADDLE (1) NOT = 100 THEN NEW
```

If paddle (1) or a data-key passing in equal value is not set to 100, then the program will erase itself from memory. A more complex key might require statements like this:

```
10 IF PADDLE (0) = 210 AND PADDLE (1) =
80 AND STRIG (0) = 1 THEN XX = 1

20 IF XX<> 1 THEN NEW
```

These statements would check a data key for three separate values before allowing the program to proceed. As you see, checking a data-key (in a joystick port) is just like reading values from paddles and joysticks, but a data-key can pass several values at once that would be impossible to duplicate with the standard controllers.

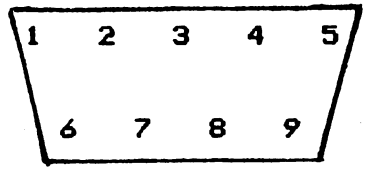
Building Data-Keys

Construction of a data-key varies

depending on the desired functions. To understand data-key construction, you must understand how values are read from the controller jack. Diagram 9.1 shows the pin arrangement of a controller jack. As you can see, the different pins can be used to read separate values from whatever is plugged into the jack. The joystick and trigger pins can only be used for a simple on or off input, but potentiometer A or B (the paddle inputs) can read a resistance value between 0 and 228. Using all joystick and paddle inputs together give a total of 1,611,504 possible combinations from each controller jack. A typical data-key might be two resistors encased in plastic and attached to a standard joystick or paddle plug (see diagram 9.2). These resistors would act like a pair of paddles permanently set to certain values. Wires could be run to the joystick pins if you wish to check for additional values from the data-key. As diagram 9.1 shows, a single joystick port can check many separate values at the same time. For added protection, the software could require two or more data-keys plugged into separate ports simultaneously. The key should be constructed in such a way that it is difficult to take apart and study. Encasing the parts in plastic or a permanently sealed casing is very good for this. If mass produced, these keys should cost under 50¢ to make. Keeping the cost low is important, since these costs wind up being passed on to the consumer.

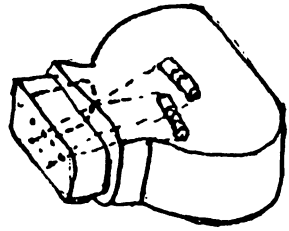
FIGURE 9.1

Controller Jack Pin Functions



1) Joystick Forward
2) Joystick Back
3) Joystick Left
4) Joystick Right
5) Potentiometer B Input
6) Trigger Input
7) +5 Volts
8) Ground
9) Pot A Input

FIGURE 9.2



Copying Data-Key Protected Software

A single data-key is like a lock with 1,611,504 possible combinations. For a pirate to determine the correct one by trial and error would take years. This fact has lead many to believe that the data-key is an ideal solution to prevent piracy. However, this logic has a serious flaw. It is true that trial and error methods would be futile, but a software pirate has an easier way to break the code.

All the pirate has to do to determine the proper combination of values on a data-key is run a simple BASIC program with the data-key plugged into a joystick port. The program could easily read the joystick and paddle values and display them on the screen. This technique would immediately give away the key's combination. Fortunately, knowing the combination alone is not sufficient to produce working copies of the program because something is needed to pass these values to the computer for each copy. The pirate would either have to build his own data-key or modify his paddle controllers to also pass joystick values. Keep in mind that a pirate could build one key with switches on it that could be used on any program requiring a data-key key of this type.

Another possible way to copy a program protected by a data key is by breaking the code by hand (see chapter 6). Once again, this menacing technique used by pirates could yield a completely unprotected

program that could be copied with no need for duplicate data-keys. The pirate would find and remove the portions of the program which check for the presence of the data-key, then the program would run as usual.

Preventing the Data-Key Copy Techniques

As mentioned earlier, most data-keys can be simply decoded by a software pirate, but he still must somehow reproduce that key or change the software to bypass the protection. Once again, hiding the protection code is the best way to discourage pirates from removing the protection. In this case, well hidden protection code is made more valuable by the lack of easy alternatives available to the pirate. No special hardware or software is available which makes it easy to copy software protected by data-keys. So, anyone attempting to copy the software, must have a good amount of technical knowledge.

Hardware data-keys also have good potential for improvement. If a simple and inexpensive key is built that can accept a signal from the computer and respond only after a certain time interval, it would make the key much more difficult to copy. Attempts by pirates to read the key's combination with a program would not work because the key would not respond until it got the proper input. An even more promising technique would be the use of a microprocessor in the key. This would

enable the key to perform a complex "handshake" type of communication with the computer, and this could stop all but a very few advanced software pirates.

Conclusions

The use of hardware data-keys for software protection involves some trade-offs. It does offer a relatively high level of copy protection, but it adds cost to the program and is disliked and inconvenient for the purchaser. Your decision on these trade-offs depends on the particular product and market being considered, but data-keys should not be overlooked as a possible protection method.

CHAPTER 10

LEGAL PROTECTION TECHNIQUES

There are three methods available to legally protect your program. Of course, they do not stop someone from copying your programs, but they do give you legal recourse should you find a company copying your ideas or bootleggers selling your programs. The three methods are patents, copyrights and trade secrets. Each has various requirements and gives different amounts of protection.

Patents. As of March, 1981, in the U.S. Supreme Court decision of Diamond v. Diehr, the U.S. Patent Office began issuing patents for software program inventions. Before that time, the Office said that software inventions were unpatentable, but since then, several software patents have been issued. These are the first patents issued in the United States for software and offer the opportunity for software writers to license their software for income and get the tax benefits of long term capital gains.

Basically, a patent is a contract between the government and the inventor. A patent gives the inventor the right to exclude other members of the public from making, using or selling the invention. In general, this right lasts for 17 years. After this period the inventor is powerless to exclude the public from using or selling the invention.

The patentor must make public (in the

patent) enough information to enable one with "ordinary skill in the art of invention" to make and use the patented invention. This is called the "enabling disclosure", and its purpose is to enhance the public's awareness of new inventions.

The patent for a software program invention has three parts. The specification, a set of drawing figures, and one or many claims.

The specification is the main body of the text and explains what field the invention is in and what problems it solves. The specification usually emphasizes the advantage of using the invention such as reduced costs, greater accuracy, increased speed, or enhanced productivity. It also describes how the invention achieves these things and should teach the readers about its use.

The drawing figure section has drawings and charts which help the specifications explain the invention's importance. For software patents, this section usually includes a listing of the program and flow charts explaining it. It also contains other charts and diagrams which help explain the originality or use of the invention.

The claims section defines exactly what it is the public is excluded from making, using or selling. The claims should be a clear and concise explanation that defines the invention. The claims are what allow the inventor to license others to make, use or sell his invention. Also, they permit the inventor to obtain licensing fees (royalties)

that are recorded as long term capital gains and taxed at a lower rate than ordinary income.

Almost anyone who writes software, either independently or for a corporation, could benefit from the new patent rules. Keep in mind that only features of programs and not the programs themselves are patentable. Also, the feature must be new and achieve advantages over currently existing systems. The U.S. Patent Code contains the exact criteria that determine newness. Besides being new, the Patent Code states that it must be "non-obvious". If your idea has significant advantages over current systems, it can be argued that it is non-obvious.

Records showing when the idea was conceived and when it was put into practice (if it was put into practice) must be kept. All records should be properly witnessed and kept safely. For more details on patents, see the book, How to Protect and Benefit from your Ideas which can be ordered from the American Patent Law Association, 2001 Jefferson Davis Highway, Arlington, Virginia, 22002.

Copyrights. Software copyrighting is another area that has become much more effective in the past few years. The boundaries and interpretation of the law have been changing very fast. Basically, software copywriting is very similar to copywriting books or songs. If you can prove that someone is copying your

copywrited program, you can stop them from copying and recover damages. You are eligible to receive statutory damages and attorneys fees, even if you cannot prove actual damages. If your program is properly registered and your copyright is in order, you can collect up to $50,000.00 in statutory damages. A copyright covers anyone who sees the work and receives proper notice. To give notice, just clearly display the copyright in the program and documentation. Under old copyright laws, the work had to be published before it can receive copyright protection, but now copyright protection begins when the work is fixed or completed. To apply for a copyright, the work must be registered with the copyright office. You must send them a copy of the source code and documentation. It is also a good idea to send a copy to yourself and a lawyer or friend by registered mail. Leave these sealed so that the date you did the work can be verified.

Copyrights have proven themselves effective recently when MicroPro International won a $250,000.00 suit against Data Equipment Corporation. They claimed that Data Equipment violated their copyright by distributing unauthorized copies of their programs to customers. In another case, ATARI was able to show a copyright violation, even though the program had been completely rewritten. The court decided that K.C. Munchkin (North American Phillips Company) was close enough to Pac-Man to be a copyright violation, even though it was by no means

an exact copy.

Trade Secrets. A trade secret is defined as something of value that gives the owner an edge over the competition. It is generally something not known to the public. A trade secret can be ideas, know-how, software or even just information that the owners can benefit from. The formula for Coca Cola or McDonald's recipe for french fries are examples of trade secrets. Trade secrets are usually state protected as opposed to patents and copyrights which are protected Federally. This means that laws pertaining to trade secrets can vary from state to state. Fortunately, all states afford some protection to software under the laws that govern unfair competition or breach of a confidential relationship.

To claim someone violated your trade secret, you must show that they disclosed or used the information which they agreed to keep confidential. If a third party gains knowledge of your information and the information is not part of a confidential agreement, you cannot stop or seek damages from the third party. In other words, trade secret protection can be used against another party provided they agree to keep your information confidential.

Conclusions. The legal protection techniques can offer you compensation should you find your program being pirated or copied. One great problem, however, is finding the bootleg copies. Someone

advertising your software in a national magazine may be an easy target, but try finding the 14 year old kid who makes a copy for a friend. Even if you could find him, would you want to prosecute a 14 year old? If so, how much do you think you can collect? In conclusion, legal protection techniques are good protection against another company attempting to profit from your programs. For this, I highly recommend their use, but keep in mind that they do little to stop the vast majority of small-time software pirates.

CHAPTER 11

COERCIVE PROTECTION TECHNIQUES

In this day and age, most would agree that the best protection methods are ones that physically prevent copying. The majority of this book deals with these methods, however, another major area of software protection exists. This miscellaneous group of techniques hope to prevent illegal copying in a number of ways. I refer to these as coercive protection methods since most try to prevent piracy through psychological means. Some have had a good deal of success and are certainly worth using in addition to the "real" protection methods.

Serial Numbered Software

Serial numbered software is software that has serial numbers in the code. The serial numbers are registered to the purchaser either at the time of sale or when the purchaser sends in his warranty or registration card. The numbers are used to identify the source of bootleg copies of the program. If a pirated copy of the program is discovered, the serial numbers should lead you back to the original purchaser. By telling the purchaser (in the documentation or program) that he will be responsible for pirated copies found with his serial numbers, he is hopefully discouraged from allowing the program to be copied. The use of this technique to

discourage software piracy has met with some success. I know of cases where even a hard-core pirate would not let some of his serial numbered software be copied.

The serial numbers should be displayed and also hidden in the software to be effective. It is relatively easy for a pirate to delete the displayed numbers, but the hidden and/or encoded serial numbers would be difficult to find. Probably the biggest problem with this method is registering the purchasers to the software. You can be sure that a pirate intent on spreading copies of the software won't send in a registration card. The manufacturer can keep track of those copies shipped directly to the purchasers, but it is nearly impossible to keep track of all copies sold retail or through large distributors. You may try to induce the purchaser into registering by offering future updates or enhancements, but even honest purchasers frequently neglect warranty and registration cards.

Protection Through Intimidation

In this technique, the program and documentation contain warnings to the user usually saying that this program is copywritten and that unauthorized copiers will be prosecuted. The documentation can remind the purchaser that pirating is punishable by up to $50,000.00 in fines and 5 years in prison. Sometimes software manufacturers go as far as to say that attempts to copy their programs may cause

damage to the program or the copier's computer. This can backfire though because if the purchaser's computer does break, he may believe that your software is to blame and cause problems for your company. In general, pirates draw the line at selling bootleg programs, but don't fear the consequences of making free copies for their friends.

Self-Destructing Code

Self-destructing code is used mainly for business software when the seller wants to allow the potential purchaser to try the program before buying it. The program is set up to run only a given number of times and then it automatically self-destructs (erases or formats itself). The idea is that the user will try the program and like it enough to buy it, but if he tries to keep the sample without paying, the program will self-destruct. Another use of self-destructing code is to make copies that appear to work, but after several uses, the copy self-destructs.

This method is best suited for disk software and is relatively simple to implement. Essentially, the program updates a counter on the disk each time it is used, and formats itself when the limit is reached. Of course, the program must check to be sure that the disk is not write protected, as this would stop it from being able to format or update itself. Writing a sector then checking the status is all that is needed to check for write protection.

Then, if the status is bad, the program should end and display a message telling the user to remove the write protect tab before running.

Unfortunately, there are loopholes in this method. If the user can copy the program, he could save one with several uses left (before self destruction). Then only run copies of this disk, and when they near self destruction, he could just recopy his saved disk again, ad infinitum. I don't recommend this method because even a legitimate purchaser can inadvertently destroy his non-write protected disk.

Freeware

Freeware is a unique marketing concept invented by Andrew Fluegelman of Tiburn, California. Essentially, he gives his products away free and actively encourages you to make copies for all your friends. The catch is that the first thing you see when you run the programs is a notice asking for a $25.00 contribution if you like the program. Since you are under no obligation to make a contribution, he relies on the good faith he has created by giving away the program and on the guilt feelings he can inspire with the notice at the start of the program.

Fluegelman has three basic principles of freeware, they are:

1. The value and utility of software is best assessed by the user on his/her own system. Only after using a program can one really determine

whether it serves personal applications, needs and techniques.

2. The creation of independent personal computer software can and should be supported by the computing community.

3. Copying and networking programs should be encouraged, rather than restricted. The ease with which software can be distributed outside traditional commercial channels reflects the strength, rather than the weakness of electronic information.

If the freeware concept is to be used, certain legal precautions should still be taken. The program should still be copyrighted to prevent others from selling it, but you should probably issue a limited license in it that allows the recipient to use and copy the program for others, provided that they do not change the notice asking for contributions. The legal consequences are not certain, so caution should be taken in this form of marketing.

The real question regarding the viability of the freeware concept pertains to its profitability. Fluegelman claims that about 2/3 of the people sending him a blank disk and requesting his program, end up sending the contribution and he estimates about 15% of the people who receive the program second hand do the same. Depending on the size of the market,

this could be a significant income and would easily rival that of protected programs after pirated versions have spread. However, many experienced software producers are skeptical of Fluegelman's claims and believe the concept has no future. Undeterred, Fluegelman plans on continuing and expanding his line of freeware products. Anyone wishing more information on freeware can communicate with Andrew Fluegelman and can reach him c/o The Headlands Press, Inc., P.O. 862, Tiburon, California, 94920. His compuserve ID is #71435,1235.

Selling Unprotected Software

Several companies (mainly supplying Apple software) advertise their software as being unprotected, and/or modifiable by the user. Some take the view that pirates cannot be stopped, so why waste time protecting your programs. Others use this as a marketing technique to encourage sales. Some just wish to allow users to make back-ups of their programs. There are also several variations of selling unprotected software. Infocomm sells minimally protected software but sells some of their programs with extensive and well done documentation packages that some people buy even if they can get a copy of the software free. Other companies just plead with the purchasers not to copy their software by explaining the amount of work that went into making it, etc.

The success of selling unprotected

software is difficult to gauge, but some companies claim that they increased sales by advertising that their software is unprotected. This also offers a marketing opportunity for those who are not willing or capable of protecting their software.

One sure way to make your profit even writing completely unprotected software is to write for magazines. You get paid for the article and program and do not have to worry about copying. This is also good for programs without the market potential to warrant spending money on a sales promotion, and it can help build a programmer's reputation. Computer magazines currently pay about $50.00 - $120.00 per page for articles and programs, and offer a software writer a good way to get started without the worries of production and marketing.

CHAPTER 12

RECOMMENDED METHODS OF PROTECTION

The protection methods you choose are dependent on many factors. These factors include your intended market, the price of your product, expected sales volume, your methods of marketing your product, and your personal tastes and preferences. However, certain techniques stand out as being more secure and have advantages if they meet your particular needs. Most often a combination of techniques is best. Legal methods such as copyrights and cohersive methods like serial numbered software can be combined with physical protection methods like bad sectoring and hidden directories. This section will discuss the best of these physical protection methods.

Currently, one of the most secure methods to sell your software is on ROM cartridges. Although they can be copied by EPROM burners, this copy technique is expensive and creates copies that are as protected as the originals. Also, if the ROM cartridge protection techniques (explained earlier) are employed, then about the only way to make copies (without an EPROM burner) is to break the protection code by hand. Using this method also reduces the problem of having to provide back-ups because of the high reliability of the ROM's, and don't forget that the market for the cartridges is potentially larger than that for disks. Keep in mind, however, that ROM cartridges pose certain restrictions.

The program must fit in 16K of memory and your expected sales volume must be quite large, to offset the high production costs. If you can handle these restrictions without significantly downgrading your product, I believe that ROMs are an excellent distribution method from the standpoint of protection.

If you do not wish to fit your programs on 16K or do not like ROMs for other reasons, I believe misassigned sectors is the next best alternative. Although the cost of special hardware to pirate software protected with misassigned sectors is coming down, and its availability is going up, it is still better than most other alternatives. Keep in mind that people who pirate programs using special hardware, create copies that are still protected.

Next down the line I would place bad sectoring. There are many people who can copy programs protected by bad sectoring, but it is better than nothing and relatively easy for the software producer to use. If this method is employed, the bad sectors should be scattered around the disk to help discourage people from copying it. Also, be sure the program only checks for one or two bad sectors, because any more, and the disk will load very slow and can really bother the purchasers.

Probably the easiest to use protection method is to hide the disk directory and wipe out the VTOC. Although this only stops the very novice copiers, it is the easiest way to create and reproduce disks for a small software maker since no special

hardware or hardware adjustments are needed. Also, this technique at least helps prevent others from selling your program with just minor modifications.

The hardware data-key can be somewhat effective if used with a combination of other methods, but they add cost to your program and create an inconvenience for the purchaser. If your program is so good that you feel your purchasers will not mind the extra cost and inconvenience, then hardware data-keys offer a relatively good degree of protection for your software.

As stated earlier, a combination of legal, cohersive and physical protection techniques is probably your best solution, but there are certainly cases where it is very desirable to allow the purchasers to copy, list and even modify your code. Don't forget about the possibility that by modifying your program, others may market it under their own name with minor modifications.

There is a major warning that should be given at this point. Some of the most well protected programs created are in widespread circulation among the pirate community because of internal company leaks and unprotected copies given to dealers as demos, etc. By giving a dealer an unprotected copy as a demo, software producers are defeating their own protection schemes. These copies inevitably wind up in the hands of a software pirate who distributes them. Many of the best protected programs become available to pirates from this source.

Another way for a company to defeat its own protection methods is by software leaks. Many companies (most notably Atari) have inadvertently allowed unprotected programs (some still under development) to leak out through their employees or visitors. One well known case was Atari's Centipede which was easily available to many pirates on unprotected disks almost a year before Atari's official release of the cartridge. The point is - don't make it easy on pirates by releasing unprotected versions to anyone, and be very careful about internal leaks.

CHAPTER 13

THE FUTURE OF SOFTWARE PROTECTION AND PIRACY

Predicting the future is always hard, but given the size and scope of the piracy problem, you can be sure that many new protection schemes will be developed and used.

One area which has future potential is hardware data-keys. Although the current ones are not very effective, with the cost of chips falling, you will eventually see cheap mass-produced data-keys which can give a high amount of protection. The data-keys of the future may be microprocessors that have a complex handshake signal with the computer that can be very hard to break. Data-keys or a new generation of ROM cartridges could contain memory and even their own microprocessors that would perform some of the work needed by the program. These steps would make the programs extremely difficult to copy.

Some of the new micro and mini computers (like Apple's LISA) contain serial numbers built into the hardware, which can be checked by the software. This means that programs could be set to run only on a specific machine. Serial numbers are widely used in large main frame computer systems and will probably become available in smaller machines. Although this technique seems to have great promise, there are problems with it. First, it is not

necessarily in the interest of micro manufacturers to use this method since they benefit from having a lot of cheap software available for their computers. Secondly, the software still can be hand broken to ignore the serial number, and lastly, this method is not very good for mass marketing items like games because each piece of software must be coded with the purchaser's serial number at the time of the sale.

The future will also see new directions in dealing with the piracy problem. We may see such a large market for software that it could be distributed so cheaply (much like paperback books) that it pays to buy originals just for the convenience. The software could be broadcast over radio or cable TV channels for a low enough price that it would be bought rather than copied (most people purchase cable TV service even though they can have friends record the shows for them). Another area that may grow is data base software. This is software that is available only on an info service (like Source or Compuserve). The future may see fiber optic links giving such fast response that even certain arcade type games are possible on these systems.

Other changes may come in the legal area. There may be a significant strengthening of the laws dealing with software, and possibly a crack-down on pirates. The whole field of software production on micros will see significant changes along with the changing computer environment. The best a software producer can do is stay up with the most current

copy and protection techniques and hope to stay ahead of the game.

Although the information above points to a brighter future for software producers, keep in mind that pirates are not standing still either. Falling hardware prices have put sophisticated equipment to duplicate tapes, disks or cartridges within the reach of the slightly well-to-do pirate. In fact, a few hundred dollars can buy pirates the equipment to make exact copies of any tape, disk or cartridge currently available. Also, the ranks of the pirate groups are growing fast. Many loosly knit pirate clubs have contacts all over the country, meaning that bootleg software can spread even faster than most producers can make it. Even people with no connections can pick up any computer magazine and find ads for utilities that can copy most currently used tape and disk protection schemes (except misassigned sectoring and a few others).

Other specters will emerge in the near future also. The ATARI 1200XL allows users to change the operating system, meaning that advanced programmers may come up with a way to dump memory to tape or disk, after a program has been loaded, and the protection checks are done. This also offers the opportunity to copy ROM cartridges and alter the operating system to make them run properly from RAM. Although the 1200XL makes these .methods easier, they can be done on an ATARI 800 with hardware modifications.

And so the battle goes on. Software

producers and pirates will continue to advance their arts with neither being the clear victor. Only by actively using state of the art protection methods can software producers hope to stay ahead. I am confident this book can help you achieve that goal.

APPENDIX A

Companies selling hardware and/or services to create or duplicate protected disks, cassette, EPROM and ROM

ALF COPY SERVICE
1448 Estes
Denver, Colorado 80215
(303) 234-0871
Disk duplication or service for Atari, Apple and TRS 80

ALPHA SOFTWARE PROTECTION CONSULTING
4435 Maplepark Road
Stow, Ohio 44224
Service specializes in protection techniques including bad sectoring and misassigned sectoring. Also disk duplication service.

CAMELON COMPUTING
Department of Physics & Astronomy
Box 119A
Dickenson College
Carlisle, Pennsylvania 17013
(717) 245-1717
ROM and EPROM cartridge boards

EASTERN HOUSE
3239 Linda Drive
Winston-Salem, North Carolina 27106
(919) 924-2889
EPROM burners, software, cartridge boards and cases.

ELCOMP PUBLISHING, INC.
53 Redrock Lane
Pamona, California 91766
(714) 623-8314
EPROM burners and boards for cartrige production.

EXPANSION PRODUCTS CO.
P.O. Box 4217
Mountain View, California 94040
Tape duplication, disk duplication service.

HAPPY COMPUTING
P.O. Box 32331
San Jose, California 95152
(408) 251-6603
Sell inexpensive disk drive modification packages and software capable of creation and duplication of disks with bad and/or misassigned sectors.

HONEYBEAR SOFTWARE
Ed Stewart, Programmer/Consultant
1840 Orchard Lane
Akron, Ohio 44312
(216) 877-4166
Consultant for creation and distribution of software.

L.E. SYSTEMS, INC.
8642A Spicewood Springs Road #532
Austin, Texas 78759
(512) 258-3828 or 258-0867
Hardware supplies for professional Atari custom disk production, including misassigned and bad sectoring, etc. Also hardware for large scale duplication of custom disks in quanity.

MICROSETTE COMPANY
475 Ellis Street
Mt. View, California 94043
(415) 962-0220
Cassette duplication service

MPC PERIPHERALS CORPORATION
9424 Chesapeake Drive
San Diego, California 92123
(714) 278-0630
Cartridge EPROM burners/software

RECORDED PUBLICATION LABORATORIES
1100 State Street
Camden, New Jersey 08105
(609) 963-3000
Disk copy service

GLOSSARY

Back-Ups: A copy of a program kept for safekeeping in case the original is accidentally damaged, lost or destroyed.

Bad Sector: A sector on a disk that cannot be read without errors.

Bomb: When a program stops functioning (this can result in a locked keyboard, etc.). Other similar terms include: crashed, blow-up, abended, died, blew.

Breaking by Hand: Refers to copying protected software by manually determining the protection scheme and changing the program to remove it or bypass it. This usually involves LISTING or disassembling the program.

Breaking Software: The act of duplicating protected software. You figuratively break the protection code.

Code: Refers to the program, commands or instructions (in any programming language). Writing a program is often referred to as coding.

Coercive Protection Techniques: Protecting programs by trying to convince people not to copy them.

Controller Jack: The socket where the joystick or paddle is plugged in.

Copy Protection Method or Technique: A method used to help prevent people from duplicating software.

Data-Key: See Hardware data-key

Disassembler: A program that reads machine language data and converts it to assembler code that can be more easily understood.

EPROM: Erasable programable read-only memory. Can be used to make or duplicate cartridges.

Exclusive OR's: Comparing two binary numbers and putting out a 1 only if one of the numbers compared is a 1 and the other is a 0.

Freeware: A unique marketing concept where programs are given away but a contribution is asked for.

Hand Breaking Software: See breaking by hand.

Hardware-Data-Key: A device used to prevent a copied program from running. Usually small and fits into a joystick port. Usually must be plugged in for the program to run properly.

Joystick Port: See controller jack.

Legal Protection Techniques: Using the law to protect your programs.

Pirate: Someone who tries to illegally copy software.

Pirating: The act of duplicating software for illegal use or distribution.

Producers: See software producers.

RAM: Random access memory.

ROM: Read only memory.

RPM: Revolutions per minute. An Atari disk's normal speed is 288 RPM.

Sector: A 128 byte area of a disk. There are 18 sectors on a track and 720 sectors on a disk.

Software Pirate: See pirate.

Software Producers: People who write, manufacture or finance software.

Track: A complete circle on a disk. There are 40 tracks on a disk and 18 sectors in a track.

User: Anyone who uses the program or computer service.

Write-Protect: A method to prevent accidentally writing on a disk or tape. If the notch on the right side of a disk is covered, the disk drive will not write to the disk.

NEW RELEASE

ATARI SOFTWARE PROTECTION TECHNIQUES

BOOK II + DISK II - Advanced Software Protection

Also written by George Morrison

For those who have this book and wish to continue and learn more about Software protection and back-up methods, this ALL NEW sequel brings you the latest innovations in this fast moving field. It explains the new protection methods used by such companies as Synapse* and Electronic Arts*. It also includes complete reviews and explanations of products such as;

The Happy Enhancement*	The Chip*
The Impossible*	The Pill + Super Pill*
The Scanalyzer	many others

Explaining specifically what they copy, what they won't, how they are used, and the details of how they work. Book II also includes such topics as;

transmitting protected programs	logic bombs
copying disks with more than 19 sectors/track	bank-select cartridges
	Random Access codes
data encryption	new trends in software law
phreaking methods	sample BASIC + Assembler programs
Program worms	on-line security

The Advanced software protection disk (included with Book II) contains more do-it-yourself protection and analysis programs including;

automatic program protector	forced password appender
custom format detector	data encrypter
newest protection demos	and more

Check your local computer store or order directly from Alpha Systems

* denotes products and companies not related to Alpha Systems.

ALPHA SYSTEMS
4435 Maplepark Road
Stow, Ohio 44224

ATARI SOFTWARE PROTECTION TECHNIQUES

The software piracy problem is reaching staggering proportions. It is estimated that for each piece of business software sold, there are two illegal copies. For game software, the ratio approaches ten illegal copies for each one sold. The law seems to be no deterrent to the growing ranks of software pirates.

Large scale software producers have been enlisting many complex software protection techniques to combat the problem. Now, this book offers everyone the opportunity to learn the state-of-the-art in software copy protection for Atari computers. By using examples and sample programs, this easy to understand book is ideal for advanced programmers down to beginners just interested in learning more about their computers.

by George Morrison

Foreward by Ed Stewart
(author of Letterman)