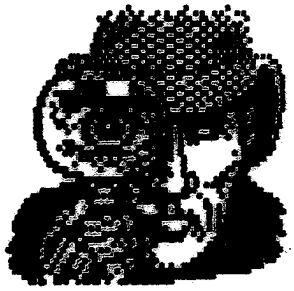
KRACKER JAX



REVEALED

< < < INTRODUCTION: PROTECTION SCHEME TYPE A > > >

Owners of the 1571 disk drive may not realize it, but every time they boot their favorite program and it bangs the disk drive head, that program is using this form of protection. It is common knowledge among experienced users that this form of copy protection is hazardous to the health of the 1541 drive. Let's face it: would YOU write a program that purposely banged YOUR disk drive's read/write head against it's end stop?

This protection is still being used by many software publishers, knowing full well that the drive knock is probably the major source of alignment problems with the 1541/1571 disk drives. We at Kracker Jax can't see any purpose in the continuation of this form of protection.

Sure, you can back up your software with almost ANY nibble utility on the market. The problem is that the backup is ALSO protected and will bang the drive as well. It is this protection type that we especially urge you to learn to break, just so you can preserve the alignment of your disk drive.

The operation of this scheme is simple. The programmer writes a routine in the program (generaly in the boot) to seek out a non-standard sector on the disk. If that non-standard sector is found, the drive will usually bang, and the program will continue operations. If not, the program will cease to operate or "crash". These non-stanadard sectors are generally write errors, and are documented in your 1541/1571 drive manual. The most commonly used are the following:

20: Block header not found / drive banger.

21: Sync character not found / sector not formated properly / drive banger.

22: Data block not present / drive banger.

23: Checksum error in data / very common / drive banger.

26: Attempt to write with write protect on / some programs check for the write protect / no drive bang.

27: Checksum error in header / drive banger.

29: Disk ID mismatch / whole track formated with wrong ID characters / no drive bang.

Many of the programs using this scheme are checking the protection with simple drive commands and the kernal routines in the computer ROM. Keep in mind that this check can be done with Basic programming as well as machine language. Once understood, most are fairly easy to unprotect.

Most of the time the programmer will check for the bad sector with a block read. It will look something like this: U1: aa bb cc dd, or B-R: aa bb cc dd . The aa denotes channel, bb denotes drive number, cc denote track, and dd denotes sector. A character or two is then returned from the drive, and a comparison is made. If the comparison is satisfactory, the program continues operation. If

not, the program flow is ended or set in an endless loop. Our task will be to either give the program the proper characters, or to short circuit the program flow around the protection check.

Before starting to work on any of the following programs, please do a disk log, an error scan, noting all write errors, and make a Three Minute Backup which will remove all errors. Place a write protect on the original disk.

< < < PROGRAM: TAPPER <> PUBLISHER: BALLY MIDWAY > > >

Procedure:

Loading the original produces a drive rattle twice. An error scan shows write errors on the original. A backup made with Three Minute Backup produces a non-working copy. Before starting to work on this program, do a disk log and an error scan to determine error type and location. Also make two Three Minute Backup copies.

Working with your backup:

1/ In order to look at the boot with our monitor, we must change its location in memory. The reason for this is because this boot cannot be stopped once it has been started. This is a simple procedure. From your utility disk, load DISK DR <> LOAD"DISK DR",8,1 <>. When the cursor reappears type RUN and hit RETURN. Remove the utility disk and insert one of your Tapper backups in the drive. Hit RETURN again and you will be shown Track 18, Sector 1. Cursor over to position 3 and hit the J key. This will take you to the first sector of the Boot file. The first four bytes in this sector are the pointer bytes. Bytes 0 and 1 are the pointers denoting this as the only sector and the number of bytes used in this sector. Bytes 2 and 3 are the program address bytes in reverse order. Place the cursor over the byte in position 2 and hit the @ key. Now, type a 1 and hit RETURN. The cursor should now be on position 4. Again hit the @ key and type an 8 and hit RETURN. To make the changes on the backup, hit the C key and hit RETURN. The sector is now changed on the disk. We have just changed the boot address to $0801, which places it in Basic memory. The boot will not run properly , but we may load and examine it.

2/ Turn your computer off and insert the reset button assembly into the cartridge port. Turn on the computer and load the $8000 monitor from the utility disk <> LOAD "32768",8,1 <>. Type SYS 32768 and hit RETURN. With the monitor active, place the altered backup in your drive and load the boot file <> L "BOOT",08 <>. disassemble code at $0801 (D 0801) and scroll down through the code. The code from $0838 to $0853 is a loader routine that loads in the file LOADER and then jumps to $C000. This gives us the information we need to trace the program flow. Take this backup out of the drive and put the other backup in its place. (Remember, we altered the boot file on this backup.)

3/ Load the LOADER file <> L "LOADER" ,08 <>. When the load is complete, interperet memory at $C000 (I C000). Scroll down through the code watching the left hand screen. At address $C1A3 you'll find the block read command : U1 2,0,32,8. This is the command to read Track 32 Sector 8. Our error scan has shown a 21 error in this location. Now let's disassemble and locate the protection code.

4/ This protection scheme is written as a series of JSR (GOSUB in Basic). Remember each JSR ends with a RTS (RETURN). Code will be explained in segments. Try to follow the program flow.

A- Starting at $C000 in the DISASSEMBLE mode, scroll down to $C017 :
JSR C116.

B- Disassemble $C116 : JSR C14B.

C- Disassemble $C14B : JSR C172.

D- Disassemble $C172 : Opens a channel to the drive then RTS.

E- Disassemble $C14E : JSR C188.

F- Disassemble $C188 : Sends Block Read command to the drive then RTS.

G- Disassemble $C151 : JSR C1AF.

H- Disassemble $C1AF : Inputs two characters from the error channel and stores them at $C1CA and $C1CB then RTS.

I- Disassemble $C154 : The accumulator is loaded with the error character placed in $C1CA and compared with a $32. The accumulator is then loaded with the error character placed in $C1CB and compared to a $31. This is the hexadecimal equivalent of a 21 error ($32=2, $31=1). Notice that if both comparisons ARE equal, the accumulator is loaded with a 0, if not, it's loaded with a 1, then a RTS.

J- Disassemble $C119 : The accummulator is compared with 0, and if equal a branch to $C127 occurs. To see what happens, type G C127 and hit RUN/STOP -RESTORE. Code was transfered to the $8000 area of memory and was activated by the RUN/STOP-RESTORE. You'll have to turn off the computer and reload the monitor and the LOADER file again.

K- Disassemble $C11B : Increment $C1AB (increments the track of the Block Read to 33).

L- Disassemble $C11E : Increment $C1AB (increment the sector of the Block Read to 09).

M- Disassemble $C120 : JSR C14B : Goes back through the error check routine once again but now the 21 error at Track 33, Sector 9 is checked (the second drive rattle). This time if the code is not branched to the message screen as before, it will return back to $C01A to resume normal loading.

N- This program can be broken in many different ways. Three will be given.

1- Place three NOPS at $C017 (EA EA EA). This will erase the code that sends the program to the protection check in the first place (our choice). The program will never do an error check.

2- Place a BNE at $C119 and $C124 (D0). This will instruct the program to operate in an opposite fashion in regards to the protection, in other words, crash if an error is found.

3- Place a $30 at $C162 and $C169. This will instruct the program to expect NO error at the Block Read locations. Again, if an error is found, the program will crash.

0- Choose one of the above methods and make your changes using the MEMORY command. After the change is made the LOADER file may be scratched and saved. Checking the disk log shows us the start address of $C000 and the end address of $C2BC. Remember to add one byte to the end address <> S "@0:LOADER",08,C000,C2BD <>.

5/ Your backup is now broken and will never rattle the drive again. Another benefit of this particular break is the fact that now you may file copy this program.

< < < PROGRAM: BUCKAROO BANZAI <> PUBLISHER: ADVENTURE INTERNATIONAL > > >

Procedure:

Booting the original produces a drive rattle early in the load. An error scan shows write errors on the original. A backup made with Three Minute Backup produces a non-working copy. Before starting to work on this program, do a disk log and an error scan to determine error type and location. Also, make a Three Minute Backup copy.

Working with your backup:

1/ The disk log shows us that the boot file SAGA resides in Basic memory, so let's begin by loading the boot and examining it <> LOAD "SAGA",8: <>. List it out and notice it loads the file SAGA.OBJ and does a SYS to 4863 ($1300).

2/ Turn your computer off and insert the reset button assembly. Turn the computer on again and, from your utility disk, load the $C000 monitor <> LOAD "49152",8,1 <>. When the load is complete, sys the monitor in with SYS 49152. Now load the SAGA.OBJ file from your backup, and follow the program flow <> L "SAGA.OBJ",08 <>. Start your dissasembly at $1300 (D 1300). We will break the code down into sections for you. Try to follow along and inspect the code as we go through it.

A- $1300-$1323 : Loads the SAGA.C64 file

B- $1324 : Does a JSR to $137A which IS the protection check routine

C- $137A-$13BE : Opens the error channel to the drive and sends the Block Read command to check Track 34, Sector 4. Interpret memory at $13BF to see the U1 (I 13BF). Then a jump to $13CE is taken.

D- $13CE-$13FE : Two bytes are received from the error channel and stored at $1556 and $1557. Then a check of these two addresses for the proper error bytes is done. The bytes are compared to $32 (2 in decimal) and a $31 (1 in decimal). These bytes correspond to a 21 error in decimal. If the comparison is incorrect, the program branches to $13FF. Do a GO 13FF (G 13FF) to see what happens. (You'll have to reload your monitor and SAGA.OBJ file again). If the comparison is correct, the program continues along until it encouters the RTS at $13FE. This will branch the code back to $1327, and the program load will continue.

3/ This protection scheme is fairly simple, and extremly easy to defeat. Four different methods will be given to break this title. Choose one and make your changes with the MEMORY command.

A- Place three NOPs at $1324. This will erase the JSR to the protection routine. The program will never even look for protection now (our choice).

B- Place an $F0 at $13E9 and $13F0. This will tell the program to fail if an error IS found.

C- Replace the code at $13E3 with A9 32 EA (LDA 32 EA) and the code at $13EA with A9 31 EA (LDA 31 EA). This loads the accumulator with the correct bytes the protection check is looking for.

D- Change $13E6 from a $32 to a $30 and $13ED from a $31 to a $30. This tells the program to look for NO error ($30=0 in decimal). The program will crash if an error is found.

4/ After your changes are made, all that is left is to save the code back to your backup. The disk log tells us the file resides from $1300 to $1575. Be sure to add one byte to the end address <> S "@0:SAGA.OBJ",08,1300,1576 <>.

5/ Your backup is now free from the restrictions of copy protection. It will no longer bang your drive head and can even be file copied. This scheme can be found in aproximatly this form in many different programs. Don't be surprised if you see it again.

< < < PROGRAM: SARGON III CHESS <> PUBLISHER: HAYDEN SOFTWARE > > >

Procedure:

Booting the original produces a drive rattle twice at the end of the load. An error scan shows write errors on the original. A backup made with Three Minute Backup produces a non-working copy. Before starting to work on this program, do a disk log and an error scan to determine error type and location. Also, make a Three Minute Backup copy.

Working with your backup:

1/ Turn the computer off and insert the reset button assembly into the cartridge port. Turn the computer back on and from your utility disk, load the $8000 monitor <> LOAD "32768",8,1 <>. Sys the monitor in with SYS 32768. Place your backup in the drive and load the boot file <> L "SARGON III",08 <>. Start your dissasembly of code at $02A7 (D 02A7). The code from $02A7 to $02F2 loads the COPYRIGHT 1984 file in and jumps to $C000.

2/ Load the file COPYRIGHT 1984 <> L "COPY*",08 <>. We will explain the code a section at a time, so try to follow as we go through it. Using the DISASSEMBLE command, disassemble memory beginning at $C000 (D C000).

A- Disassemble $C000 : $C000-$C091 sets up a loader routine that loads HAYDEN SOFTWARE and JUMPS to $C311.

B- Disassemble $C311 : $C311-C336 opens an error channel to the drive and sets the Y register to 0.

C- Disassemble $C337 : JSR $C376

D- Disassemble $C376 : $C376-$C389 sends Block Read command to Drive to check Track 2, Sector 15. The address $C2F7,Y is accessed. Since Y has been set to 0, the true address IS $C2F7. Interpret memory at $C2F7 to see the B-R (I C2F7). This subroutine returns when an RTS is encountered.

E- Disassemble $C33A : JSR $C38A

F- Disassemble $C38A : $C38A-$C3A0 inputs two bytes from the error channel and compares it to a $30 (0 or no error in decimal). If NO error is found, a branch to $C373 is taken. This in turn jumps to a reset vector and the program crashes. If errors are found, the program flows until the RTS is encoutered.

G- Disassemble $C33A : Loads the Y register with 0D (13 in decimal).

H- Disassemble $C33F : JSR $C376 : Same as step d, except this time the address $C2F7,0D ($C2F7+0D) is sent to the drive. This address is the same as $C304 and is the B-R command for Track 3, Sector 16 (I C304).

I- Disassemble $C342 : JSR $C38A : Same as step f. Checks for error and RTS if found.

J- Disassemble $C345 : Close all channels and files; continue setup and jump to start of program.

3/ This protection scheme is fairly simple and can be defeated in many ways. Four will be given. Choose one, and make your changes with the MEMORY command. When the change has been made, all that is left is to save the file back to the disk. The disk log tells us the file resides in memory from $C000 to $C3A2. Remember to add one byte to the end address when you save it <> S "@0:COPYRIGHT 1984",08,C000,C3A3 <>.

A- Change the address $C08F from 4C 11 C3 (JMP C311) to 4C 45 C3 (JMP C345). This will jump the program flow completely around the protection check (our choice).

B- Change $C33A and $C342 from 20 8A C3 (JSR C38A) to EA EA EA. This will erase the JSR to the error check.

C- Change $C397 from F0 0A (BEQ reset address) to EA EA. This will erase the branch to the crash and the program flow will be forced to continue on.

D- Change $C395 from C9 30 (CMP 30) to C9 32. This will force the program to crash if an error IS found.

4/ After your changes are made, you will have a completely broken copy that can be fast copied and even file copied.

< < < PROGRAM: THE SLUGGER <> PUBLISHER: MASTERTRONICS > > >

Procedure:

Loading the original produces a drive rattle. An error scan shows write errors on the original. A backup made with Three Minute Backup produces a non-working copy. Before starting to work on this program, do a disk log and an error scan to determine error type and location. Also, make a Three Minute Backup.

Working with your backup:

1/ Checking the disk log shows us the boot file is in Basic memory so let's start by loading it <> LOAD "THE SLUGGER",8: <>. List it and examine the loader. It loads various files and then does a SYS 514 ($0202). The disk log again tells us the address $0202 is the start of the GOFILE file.

2/ Turn the computer off and install the reset assembly into the cartridge port. Turn the computer back on, and from your utility disk, load the $2000 monitor <> LOAD "8192",8,1 <>. Sys it in with SYS 8192. Now from your backup, load the GOFILE file <> L "GOFILE",08 <>. Start disassembly at $0202 (D 0202). Scroll down through the code and notice that this file loads the CODE file and Jumps to $0340.

3/ From the backup, load the CODE file <> L "CODE",08 <>. Start disassembly at $0340 (D 0340). The disassembly is given in the sections below. Try to follow along as we go through it.

A- $0340-$036A : Opens the error channel to the drive.

B- $036B-$037D : Sends U1 (Block Read) command to the drive to read Track 6, Sector 7. Use the INTERPRET command to see the U1 (I 03E3).

C- $037E-$0385 : Set up to read two bytes from the error channel.

D- $0389-$0396 : Inputs a byte from the error channel and compares it to a $32 (2 in decimal). Another byte is retrieved and compared to a $33 (3 in decimal). Each compare results in a branch to a crash address if not satisfied. Otherwise the program flow continues on to a Jump to $03A1. These compares are the 2 and the 3 of a number 23 error. The error scan confirms a 23 error at Track 6, Sector 7.

E- $03A1-$03A8 : Close error channel and normal program flow continues.

4/ The break in this program is fairly simple. Four different methods will be given. Choose one and make your changes with the MEMORY command.

A- Change $0340 to 4C AB 03 (JUMP $03AB). This will cause the program to jump completly around the protection check (our choice).

B- Change $038D and $0394 to $30. This will instruct the protection check to look for NO error ($30=0 in decimal).

C- Change $038E and $0395 to $F0. This will cause the protection to branch to the crash if an error IS found.

D- Change $0389 to A9 32 EA (LDA 32) and $0390 to A9 33 EA (LDA 32). This will load the accumulator with the bytes it wants in the compares. The bytes will not be inputed from the error channel.

5/ When your changes are made, all that's left is to save them to the backup. The disk log supplies the start and end addresses. Be sure to add one byte to the end address <> S "@0:CODE",08,0340,0401 <>. Your backup is completely broken and may be file copied to another disk.

< < < INTRODUCTION: PROTECTION SCHEME TYPE B > > >

This protection scheme has allowed software publishers a means of protecting their programs from the finest nibblers on today's market. It employs a loader that resides in RAM at $C000. This loader does the protection check and then procedes to gather a Basic boot from the program disk. This boot is placed in RAM at the beginning of Basic ($0801-). Our task in each of these schemes will be to let the original disk pass protection and then place the boot in memory. At this point we can retrieve the boot and from then on use it to load our back-up, leaving the protection check completely behind.

Before starting, you must understand the way a Basic program is placed in memory and how the pointers affect it. The reason for this is that most of the time upon reset, the beginning pointers will be destroyed and we will have to repair them ourselves.

The pointers used by Basic are very specific, and if not correct, the Basic program will fail to operate properly. To show you how a Basic program looks in memory, let's inspect the example on your work disk.

First, load the $C000 monitor from your Utility disk <> Load "49152",8,1 <> and sys it in by typing SYS49152 and hitting RETURN. You should be in the monitor now so load again from your work disk the file called BASIC EXAMPLE <> L "BASIC EXAMPLE",08 <>. After the load, examine memory from $0801-$0890 <> M 0801 <>. Scroll up and down through the code. You should be looking at the same code as shown below. Please note that the example below has all pointer bytes underlined for ease of viewing.

```
:0801 0E 08 05 00 99 22 93 11
:0809 11 11 05 22 00 35 08 0A
:0811 00 99 22 20 20 20 54 48
:0819 49 53 20 49 53 20 41 4E
:0821 20 45 58 41 4D 50 4C 45
:0829 20 4F 46 20 48 4F 57 20
:0831 41 20 22 00 5A 08 14 00
:0839 99 22 20 20 20 42 41 53
:0841 49 43 20 50 52 4F 47 52
:0849 41 4D 20 49 53 20 46 4F
:0851 52 4D 41 54 45 44 20 22
:0859 00 84 08 1E 00 99 22 20
:0861 20 20 49 4E 20 54 48 45
:0869 20 4D 45 4D 4F 52 59 20
:0871 4F 46 20 54 48 45 20 43
:0879 4F 4D 4D 4F 44 4F 52 45
:0881 2E 22 00 00 00 FD BD FF
:0889 D0 FF FF E6 FF FE 00 00
```

The format of Basic is as folows. Starting at $0801, the bytes 0E 08 denote the placement of the next line number in memory in reverse order ($080E). The next two bytes, 05 00 denote the current line number in reverse order ($0005=5). Follow the bytes from here until you get to the next 00. This byte (residing at

address $080D) denotes the end of the first line in this program. The next four bytes are again the pointers for the second line in our Basic program. The address $080E and $080F contain the bytes 35 08. These are the address of the next line number in our program, again in reverse order ($0835). The next two bytes starting at $0810 are 0A 00 which is the current line number of our program, again in reverse order (000A=0A in hex or line 10 in decimal). This format is followed all through any normal Basic program and ends only when three hex zeros are encountered (00 00 00). This tells Basic that the programs end has been found. You'll find these bytes in our example starting at $0883.

This means that this program could be saved with your monitor using the addresses from $0801-$0885. The $0801 being the beginning of Basic and the $0885 the last of the three zero bytes. The actual save command would be <> S "FILENAME",08,0801,0886 <>. We used the end address $0886 because all monitor saves need one extra byte added to the actual ending address ($0885+1=$0886).

By understanding the structure of Basic, we can now repair any damage done to our pointers when we reset out of our program loads. Now let's move on to our example programs.

< < < PROGRAM: TITLE BOUT <> PUBLISHER: AVALON HILL > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with Three Minute Backup produces a non-working copy. A backup made with a nibbler also produces a non-working copy. Before starting to work on this program, please make a non-working backup of the original.

Working with your backup:

1/ Start by validating the BAM <> OPEN15,8,15,"V":CLOSE15 <> to make room for a new file we will be adding later. Scratch The first file from your backup <> OPEN15,8,15,"S0:AH":CLOSE15 <>.

Working with your original:

2/ Place a write protect tab on the original to ensure its safety during the breaking process.

3/ Turn off your computer and insert your reset assembly into the cartridge port. Turn the computer on again and load the boot file and start the load process <> LOAD"AH",8,1 <> . Allow the program to load until the screen turns black and the words LOADING DATA appear in the middle of the screen. At this point, reset the computer.

4/ Remove the original disk from your drive and insert the utility disk. Load the $C000 monitor <> LOAD"49152",8,1 <>. When the load is complete, sys the monitor in with SYS 49152. The monitor should be active now. Remove the utility disk from the drive and replace it with the backup work disk.

5/ Interpret memory starting at $0801 (I 0801). Scroll through memory and notice the Basic program. Our task is to repair the pointers and save the program to your backup (see scheme B intro). Using the memory command (M 0801) inspect code at 0801. Notice that the first two bytes are 00 00. These two bytes represent the start of the next line in this Basic program. Obviously, these bytes have been destroyed by the reset because the next line couldn't be zero. To find the correct bytes to replace the two zeros, follow this procedure. We know the first four bytes are pointer bytes ($0801-$0804). We also know that the next time a zero byte appears in memory ($0811), it signals a new line. The next address is the address that the pointer will point to ($0812). Therefore, the first two bytes in this program should be 12 08 because all addresses are read in reverse order. Now we can scroll to the two zeros at $0801 and type over them 12 08 and hit RETURN. The first four bytes starting at $0801 should now be 12 08 01 00 (the 01 00 bytes represent the current line number in reverse 01 00=00 01). Our Basic program is now repaired and all that's left is to locate the end of the program and save it to your backup disk. To find the program end, use the HUNT command in your monitor. We'll hunt for the three zero bytes that signal the end of Basic. <>H 0801 8000 00 00 00 <>. As the first bytes begin to be reported, hit the number 1 key to stop the hunt. We are only interested in the first address reported. In this case, it should be $1C15. Using the MEMORY command, inspect memory around the address $1C15.

You will notice that the third zero is at the location $1C15. We now have all the information needed to save the new boot to your backup. The start address is $0801 (beginning of Basic) and the end address is $1C16 (all monitors require us to save the actual address plus one:$1C15+1=$1C16). Make sure your backup is in the drive and save the memory from $0801-$1C15 <> S"AH",08,0801,1C16 <>.

6/ When the save is complete, you will have a broken copy that will no longer do a protection check. We have essentially replaced the auto boot and the protection check with the result, a Basic boot.

< < < PROGRAM: SUPERBOWL SUNDAY <> PUBLISHER: AVALON HILL > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with the Kracker Jax copier produces a non-working copy. A backup made with a nibbler produces the same non-working copy. Before starting to work on this program, please make a non-working backup of the original.

Working with your backup:

1/ Start by validating the BAM <> OPEN15,8,15,"V":CLOSE15 <> to make room for a new file we will be adding later. Scratch The first file from your backup <> OPEN15,8,15,"S0:START":CLOSE15 <>.

Working with your original:

2/ Place a write protect tab on the original to ensure its safety during the breaking process.

3/ Turn your computer off and insert the reset assembly into the cartridge port. Turn the computer on again and load the boot file and start the load process <> LOAD"START",8,1 <>. Allow the program to load until the game menu is on the screen. At this point, reset the computer.

4/ Remove the original disk from your drive and insert the utility disk. Load the $C000 monitor <> LOAD"49152",8,1 <> . When the load is complete, sys the monitor in with SYS49152. The monitor should be active now. Remove the utility disk from the drive and replace it with the backup work disk.

5/ Interpret memory starting at $0801 (I 0801). Scroll through memory and notice the Basic program. Our task is to repair the pointers and save the program to your backup (see scheme B intro). Using the memory command (M 0801) inspect code at 0801. Notice that the first two bytes are 00 00. These two bytes represent the start of the next line in this Basic program. Obviously, these bytes have been destroyed by the reset because the next line couldn't be zero. To find the correct bytes to replace the two zeros, follow this procedure. We know that the first four bytes are pointer bytes ($0801-$0804). We also know that the next time a zero byte appears in memory ($0811), it signals a new line. The next address is the address that the pointer will point to ($0812). Therefore, the first two bytes in this program should be 12 08 because all addresses are read in reverse order. Now we can scroll to the two zeros at $0801 and type over them 12 08 and hit RETURN. The first four bytes starting at $0801 should now be 12 08 00 00 (the 00 00 bytes represent the current line number in reverse 00 00=00 00; yes, we CAN have a line number 0!). Our BASIC program is now repaired and all that is left is to locate the end of the program and save it to our backup disk. To find the program end, use the HUNT command in your monitor. We'll hunt for the three zero bytes that signal the end of Basic. <>H 0801 8000 00 00 00 <>. As the first bytes begin to be reported, hit the number 1 key to stop the hunt. We are only interested in the first address reported. In this case it should be $0A6E. Using the memory command, inspect memory around the address $0A6E. You will notice that the

third zero is at the location $0A70. We now have all the information needed to save the new boot to our backup. The start address is $0801 (beginning of Basic) and the end address is $0A71 (all monitors require us to save the actual address plus one: $0A70+1=$0A71). Make sure your backup is in the drive and save the memory from $0801-$0A70 <> S"START",08,0801,0A71 <>.

6/ When the save is complete, you will have a broken copy that will no longer do a protection check. We have essentially replaced the auto boot and the protection check with the result, a Basic boot.

< < < PROGRAM: GULF STRIKE <> PUBLISHER: AVALON HILL > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with the Kracker Jax copier produces a non-working copy. A backup made with a nibbler produces the same non-working copy. Before starting to work on this program, please make a non-working backup of the original.

Working with your backup:

1/ Start by validating the BAM <> OPEN15,8,15,"V":CLOSE15 <> to make room for a new file we will be adding later. Scratch the first file from your backup <> OPEN15,8,15,"S0:BOOT":CLOSE15 <>.

2/ Turn off your computer and insert the reset assembly into the cartridge port. Turn the computer on again and remove your backup from the drive. Insert the utility disk and load the $C000 monitor <> LOAD"49152",8,1 <>. After the load, sys in your monitor with SYS 49152 and hit RETURN. We want to fill memory from $0801-$2000 with EA so, use the FILL command <> F 0801 2000 EA <>. We now have a marked workspace to load our program into. Use the reset button to reset the computer and clear the screen.

Working with your original:

3/ Place a write protect on the original to ensure its safety during the breaking process.

4/ Load the boot file and start the load process <> LOAD"BOOT",8,1 <>. Allow the program to load until the screen clears and then turns blue. At this point, reset the computer.

5/ Remove the original disk from your drive and insert the utility disk again. Load The $C000 monitor <> LOAD"49152",8,1 <>. When the load is complete, sys the monitor in with SYS49152. The monitor should be active now. Remove the utility disk from the drive and replace it with the backup work disk.

6/ Interpret memory starting at $0801 (I 0801). Scroll through memory and notice the Basic program. Our task is to repair the pointers and save the program to our backup (see scheme B intro). Using the MEMORY command (M 0801), inspect code at $0801. Notice that the first two bytes are 00 00. These two bytes represent the start of the next line in this Basic program. Obviously, these bytes have been destroyed by the reset because the next line couldn't be zero. To find the correct bytes to replace the two zeros, follow this procedure. We know the first four bytes are pointer bytes ($0801-$0804). We also know that the next time a zero byte appears in memory ($080D), it signals a new line. The next address is the address that the pointer will point to ($080E). Therefore, the first two bytes in this program should be 0E 08 because all addresses are read in reverse order. Now we can scroll to the two zeros at $0801 and type over them 0E 08 and hit RETURN. The first four bytes starting at $0801 should now be 0E 08 0A 00 (the 0A 00 bytes represent the current line number in reverse 0A 00=000A =10 in decimal). Our Basic program is now repaired

and all that is left is to locate the end of the program and save it to our backup disk. To find the program end, use the HUNT command in your monitor. We'll hunt for the first three EA bytes that signal the end of the program that we loaded in <> H 0801 2000 EA EA EA <>. This search will bring back the address $08A5. Dissasembly of code around this address reveals a small machine language program placed under a Basic program. To properly capture all the neccessary code, we must save the code from the beginning of Basic ($0801) to the beginning of our EA bytes ($08A5). Because all monitors require us to add one extra byte to the end address, use this command: <> S"BOOT",08,0801,08A6 <>.

7/ When the save is complete, you will have a broken copy that will no longer do a protection check. We have essentially replaced the auto boot and the protection check with the result, a small program consisting of a Basic loader with a machine language routine placed under it.

< < < PROGRAM: CREATIVE CONTRAPTIONS <> PUBLISHER: BANTAM SOFTWARE > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with the Kracker Jax copier produces a non-working copy. A backup made with a nibbler produces the same non-working copy. Before starting to work on this program, please make a non-working backup of the original.

Working with your backup:

1/ Start by scratching the first file from your backup <> OPEN15,8,15,"S0:CREATIVE":CLOSE15 <>.

Working with your original:

2/ Place a write protect on the original to ensure its safety during the breaking process.

3/ Turn your computer off and insert the reset assembly into the cartridge port. Turn the computer on again and load the boot file and start the load process <> LOAD"CREATIVE",8 <>. When the cursor appears, type RUN and hit RETURN. Let the program load for about 15 seconds and reset the computer.

4/ Remove the original disk from your drive and insert the utility disk. Load the $C000 monitor <> LOAD"49152",8,1 <>. When the load is complete, sys the monitor in with SYS49152. The monitor should be active now. Remove the utility disk from the drive and replace it with the backup work disk.

5/ Interpret memory starting at $0801 (I 0801). Scroll through memory and notice the Basic program. Our task is to repair the pointers and save the program to our backup (see scheme B intro). Using the MEMORY command (M 0801), inspect code at 0801. Notice that the first two bytes are 00 00. These two bytes represent the start of the next line in this Basic program. Obviously, these bytes have been destroyed by the reset because the next line couldn't be zero. To find the correct bytes to replace the two zeros, follow this procedure. We know the first four bytes are pointer bytes ($0801-$0804). We also know that the next time a zero byte appears in memory ($0818), it signals a new line. The next address is the address that the pointer will point to ($0819). Therefore, the first two bytes in this program should be 19 08 because all addresses are read in reverse order. Now we can scroll to the two zeros at $0801 and type over them 19 08 and hit RETURN. The first four bytes starting at $0801 should now be 19 08 0A 00 (the 0A 00 bytes represent the current line number in reverse 0A 00=000A =10 in decimal). Our Basic program is now repaired and all that is left is to locate the end of the program and save it to your backup disk. To find the program end, use the HUNT command in your monitor. We'll hunt for the three zero bytes that signal the end of Basic. <> H 0801 8000 00 00 00 <>. As the first bytes begin to be reported, hit the number 1 key to stop the hunt. We are only interested in the first address reported. In this case it should be $0879. Using the MEMORY command, inspect memory around the address $0879. You will notice that the third zero is at the location $087B. We now have all the information needed to save the new boot to

our backup. The start address is $0801 (beginning of Basic) and the end address is $087C (all monitors require us to save the actual address plus one: $087B+1=$087C). Make sure your backup is in the drive and save the memory from $0801-$087C <> S "CREATIVE",08,0801,087C <>.

6/ When the save is complete, you will have a broken copy that will no longer do a protection check, and will even load faster than the original. We have essentially replaced the auto boot and the protection check with the result, a Basic boot.

< < < INTRODUCTION: PROTECTION SCHEME TYPE C > > >

This protection scheme employs the use of a "fat track" to prevent the user from making his backup. To make matters worse, the fat track is placed on the outer (36-40) tracks.

Most of the examples covered in this manual work aproximatly the same. The following general loading procedure is taken with each.

1/ The boot is loaded and autostarts the program.

2/ A fast loader is set up and activated.

3/ The logo screen is loaded in and activated.

4/ The protection routine is decrypted.

5/ The files pertaining to the program are loaded in. These are generally encrypted.

6/ The protection is checked, which places a numeric value ($FF) in the disk drive's memory.

7/ The value is checked using a memory read.

8/ The value is used as a part of a decryption routine to decrypt the main program. Proper decryption takes place ONLY if the correct value is returned.

9/ The code then jumps to the start of the program.

The Activision examples in this manual represent this protection scheme in it's most difficult form to un-protect. You'll find this same scheme being used by other software publishers, but generally not encrypted. They usually check for the value in the same way and start the program if found. One example of this will be given, and will be unprotected by a different method. Understanding this routine is imperative, because this scheme has been improved, and will be covered in it's expanded form in updates to this manual.

< < < PROGRAM: COUNTDOWN TO SHUTDOWN <> PUBLISHER: ACTIVISION > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with Three Minute Backup produces a non-working copy. A nibbled backup produces the same non-working copy. Before starting to work on this program, please make a (non-working) backup of the original, and a disk log to log the file addresses.

1/ Turn off your computer and insert your reset button assembly into the cartridge port. Turn the computer on again. Load the $C000 monitor from your utility disk <> LOAD"49152",8,1 <>. At the completion of the load, type SYS 49152 and hit RETURN. The monitor should be active now.

Working with your backup:

2/ With your backup in the drive and the monitor active, load the boot file <> L "COP*" ,08 <>. When the load is complete, dissasemble memory at $02A0. You'll find a loader routine that loads in the 1985 file and jumps to $0B79.

3/ Load the 1985 file into memory <> L " 19*",08 <>. After the load, start disassembly of code at $0B79 (D 0B79). The code is as follows: $0B79-$0BCB sets up a fast loader and loads in the logo screen. $0BCC is a JSR (GOSUB in BASIC) to the logo screen. $0BCF is the start of the main program load. It is this code that is of interest to us.

4/ The code at $0C40-$0C61 is a decryption routine. Examine it because it is the key to the de-protection. This routine allows decryption and examination of the protection code. At the end of this decryption routine is a RTS ($0C61). Using the Memory command (M 0C61), change the 60 to a 00. This will allow a normal operation of code until the 00 (Break or Stop) is encountered. The program, once started, will stop right after the decryption, allowing us to examine the protection routine.

5/ For our purposes, we will skip over the fast loader and logo screens. Let's start the program after the logo screen is run ($0BCF). Type G 0BCF and hit RETURN. The screen should turn black. Wait for about five seconds and reset the computer. Return to the monitor with SYS 49152. Using the INTERPERET command, examine code from $0A00 on (I 0A00). Code at $0AEF reveals a Block Execute (executes a protection check routine placed in drive memory) and code at $0B72 reveals a Memory Read that reads the value placed in the drive by the protection check. This value, in this scheme, is always an $FF. Examine code at $0B42. The value is being returned to the computer by a Memory Read with a kernal routine. The $FFCF routine brings back the value $FF. It is then EORed with $AA which turns it into a $55 and then stores it at location $0B65. Our job is to place the correct value in $0B65 and disable the routine overwriting it. This can be accomplished by placing three NOPs at $0B47 which will allow the routine to Memory Read the value but not place it in computer RAM. All that is left is to place the correct value of $55 at $0B65.

6/ Now we have the correct values to plug into the code to disable and give the protection check what it wants. The last step is to place the changes on

the disk. This is best done with a sector editor because to scratch and replace the 1985 file will destroy neccessary code placed on the disk. This code is not accessed in the normal fashion, so it will be overwritten if we do a scratch and save of the 1985 file. Finish the job by following these steps:

A/ We know the code was originally encrypted, so we must place our values on the disk in encrypted form. The three bytes at $0B47 and the single byte at $0B65 are the only changes needed. Reload the 1985 file <> L 19*" .08 <>. Again go to location $0B61 and place a 00 in memory. Inspect the three bytes a $0B47. They should be 29 7A 91. The byte at $0B65 should be a A2. These are the bytes we will look for on our backup with the sector editor.

B/ The code can now be decrypted by typing G 0BCF. Again the screen will turn black. After a few seconds, reset the computer and reactivate the monitor with SYS 49152. Using the MEMORY command (M 0B47), change the code at $0B47 from 8D 65 0B to EA EA EA. Change the code at $0B65 from A2 to 55.

C/ Now that our changes are in memory, we may re-encrypt the file (and our byte changes) by again typing G 0BCF. Again, reset out and SYS the monitor back in with SYS 49152 and hit RETURN. Examine memory at $0B47 and find the encrypted byte changes. They should be 4E F5 70. The byte at $0B65 has changed to 5B. Now we know the changes, and the location, so we may now do the actual changes to the backup.

D/ Reset the computer and load the sector editor from the utility disk <> LOAD"DISK DR",8,1 <>. When the cursor appears, type run and hit RETURN. Remove the utility disk and place your backup in the drive. Hit RETURN. You will be shown track 18, sector 1. By placing the cursor at position 35, you will be on the file pointers of the 1985 file. Press the J key to jump to the beginning of the 1985 file. When the sector comes on the screen, examine the first four bytes. The first two are links to the next sector of the file. The next two are the address bytes in reverse order ($0700). We know our changes are in memory block $0B00 so we can use the N key to page through memory. Press N to go to apx 0800, press again to go to 0900, press again to go to 0A00, and once more to $0B00. This block turned out to be track 17, sector 3 on our version. Yours could be in a different location on the disk but the idea will be the same.

E/ Using the cursor key to move through the code, we find the original three bytes 29 7A 91 at location 83. The change to 4E F5 70 can be accomplished with the @ key. The changes must be the decimal equivilant. These are 78 245 112. Change each byte by placing the cursor over the byte to be changed, and type @ and then the decimal number change. Hit RETURN when the change is made to lock it in. When all three bytes are changed, continue searching with the cursor for the A2 byte. This can be found at 113. Using the same change procedure, change it to a decimal 91 ($5B). When all changes have been made and locked in, press C to copy the sector back to the disk.

7/ You now have a copy that can be fast copied. The placement of data on the disk in methods other than directory files will not allow you to file copy. One other point of interest is the fast loader installed in many pieces of this publisher's software. This fast loader is NOT compatible with the 1571 disk

drive. In many (but not all) of the programs, you may disable the fast loader and allow the program to load on the 1571 by changing the jump to the main program in the autoboot. Countdown does not work by doing this but, just as an example, you would change the 79 0B (JMP 0B79) to CF 0B (use DISK Doctor and the decimal eqivilents). This would bypass the fastloader and the logo screen. A small price to pay for the 1571 owners.

< < < PROGRAM: WEB DIMENSION <> PUBLISHER: ACTIVISION > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with Three Minute Backup produces a non-working copy. A backup made with a nibbler produces the same non-working copy. Before starting to work on this program, please make a (non-working) backup of the original, and a disk log to log the file addresses.

1/ Turn off your computer and insert your reset button assembly into the cartridge port. Turn the computer on again. Load the $C000 monitor from your utility disk <> LOAD"49152",8,1 <>. At the completion of the load, type SYS 49152 and hit RETURN. The monitor should be active now.

Working with your backup:

2/ With your backup in the drive and the monitor active, load the boot file <> L "COP*" ,08 <>. When the load is complete, disassemble memory at $02E0. You'll find a loader routine that loads in the 1985 file and jumps to $0C3D.

3/ Load the 1985 file into memory <> L " 19*",08 <>. After the load, start dissasembly of code at $0C3D (D 0C3D). The code is as follows: $0C3D-$0C5B sets up a fast loader and loads in the logo screen. $0C5C is a JSR (GOSUB in BASIC) to the logo screen. $0C5F is the start of the main program load. It is this code that is of interest to us.

4/ The code at $0CE5-$0D06 is a decryption routine. Examine it, because it is the key to the de-protection. This routine allows decryption and examination of the protection code. At the end of this decryption routine is a RTS ($0D06). Using the MEMORY command (M 0D06), change the 60 to a 00. This will allow a normal operation of code until the 00 (Break or Stop) is encountered. The program, once started, will stop right after the decryption, allowing us to examine the protection routine.

5/ For our purposes, we will skip over the fast loader and logo screens. Let's start the program after the logo screen is run ($0C5F). Type G 0C5F and hit RETURN. The screen should turn black. Wait for about five seconds and reset the computer. Return to the monitor with SYS 49152. Using the INTERPERET command, examine code from $0A00 on (I 0A00). Code at $0AB6 reveals a Block Execute (executes the protection check placed in drive memory) and code at $0AC2 reveals a Memory Read that reads the value placed in the drive by the protection check. This value is, in this scheme, always an $FF. Examine code at $0A92. The value is being returned to the computer by a Memory Read with a kernal routine. The $FFCF routine brings back the value $FF. It is then EORed with $FF which turns it into a $00 and then stores it at location $0AB5. Our job is to place the correct value in $0AB5 and disable the routine overwriting it. This can be accomplished by placing three NOPs at $0A97 which will allow the routine to Memory Read the value but not place it in computer RAM. All that is left is to place the value of $00 at $0AB5.

6/ Now we have the correct values to plug into the code to disable and give the protection check what it wants. The last step is to place the changes on the disk. This is best done with a sector editor because to scratch and replace the 1985 file will destroy neccessary code placed on the disk. This code is not

accessed in the normal fashion, so it may be overwritten if we do a scratch and save of the 1985 file. Finish the job by following these steps:

A/ We know the code was originally encrypted, so we must place our values on the disk in encrypted form. The three bytes at $0A97 and the single byte at $0AB5 are the only changes needed. Reload the 1985 file <> L 19*" ,08 <>. Again, go to location $0D06 and place a 00 in memory. Inspect the three bytes at $0A97. They should be 19 8E E8. The byte at $0AB5 should be a 8A. These are the bytes we will look for on our backup with the sector editor.

B/ The code can now be decrypted by typing G 0C5F. Again the screen will turn black. After a few seconds, reset the computer and reactivate the monitor with SYS 49152. Using the MEMORY command (M 0A97), change the code at $0A97 from 8D B5 0A to EA EA EA. Change the code at $0AB5 from AC to 00.

C/ Now that our changes are in memory, we may re-encrypt the file (and our byte changes) by again typing G 0C5F. Again, reset out and SYS the monitor back in with SYS 49152 and hit RETURN. Examine memory at $0A97 and find the encrypted byte changes. They should be 7E D1 08. The byte at $0AB5 has changed to 26. Now we know the changes, and the location so we may now do the actual changes to the backup.

D/ Reset the computer and load the sector editor from the utility disk <> LOAD"DISK DR",8,1 <>. When the cursor appears, type RUN and hit RETURN. Remove the utility disk and place your backup in the drive. Hit RETURN. You will be shown track 18, sector 1. By placing the cursor at position 35, you will be on the file pointers of the 1985 file. Press the J key to jump to the beginning of the 1985 file. When the sector comes on the screen, examine the first four bytes. The first two are links to the next sector of the file. The next two are the address bytes in reverse order ($0A00). We know our changes are in memory block $0A00 so we are in the proper block to make our changes. This block turned out to be track 17, sector 2 on our version. Yours could be in a different location on the disk, but the idea will be the same.

E/ Using the cursor key to move through the code, we find the original three bytes 19 8E E8 at location 155. The change to 7E D1 08 can be accomplished with the @ key. The changes must be the decimal equivilant. These are 126 209 08. Change each byte by placing the cursor over the byte to be changed, and type @ and the the decimal number change. Hit RETURN when the change is made to lock it in. When all three bytes are changed, continue searching with the cursor for the 8A byte. This can be found at position 185. Using the same change procedure, change it to a decimal 38 ($26). When all changes have been made and locked in, press C to copy the sector back to the disk.

7/ You now have a copy that can be fast copied. The placement of data on the disk in methods other than directory files will not allow you to file copy. One other point of interest is the fast loader installed in many pieces of this publisher's software. This fast loader is NOT compatible with the 1571 disk drive. In many of the (but not all) of the programs, you may disable the fast loader and allow the program to load on the 1571 by changing the jump to the main program in the autoboot. Web Dimension will work by doing this. Just change the 3D 0C (JMP 0C3D) to 5F 0C (use DISK Doctor and the decimal equivilents). This will bypass the fastloader and the logo screen. A small price to pay for the 1571 owners.

< < < PROGRAM: FIREWORKS CELEBRATION KIT <> PUBLISHER: ACTIVISION > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with Three Minute Backup produces a non-working copy. A backup made with a nibbler produce the same non-working copy. Before starting to work on this program, please make a (non-working) backup of the original, and a disk log to log the file addresses.

1/ Turn off your computer and insert your reset button assembly into the cartridge port. Turn the computer on again. Load the $C000 monitor from your utility disk <> LOAD"49152",8,1 <>. At the completion of the load, type SYS 49152 and hit RETURN. The monitor should be active now.

Working with your backup:

2/ With your backup in the drive and the monitor active, load the boot file <> L "COP*" ,08 <>. When the load is complete, disassemble memory at $02E0. You'll find a loader routine that loads in the 1985 file and jumps to $0C3D.

3/ Load the 1985 file into memory <> L " 19*",08 <>. After the load, start disassembly of code at $0C3D (D 0C3D). The code is as follows: $0C3D-$0C5B sets up a fast loader and loads in the logo screen. $0C5C is a JSR (GOSUB in BASIC) to the logo screen. $0C5F is the start of the main program load. It is this code that is of interest to us.

4/ The code at $0CE2-$0D03 is a decryption routine. Examine it, because it is the key to the de-protection. This routine allows decryption and examination of the protection code. At the end of this decryption routine is a RTS ($0D03). Using the MEMORY command (M 0D03), change the 60 to a 00. This will allow a normal operation of code until the 00 (Break or Stop) is encountered. The program, once started, will stop right after the decryption, allowing us to examine the protection routine.

5/ For our purposes, we will skip over the fast loader and logo screens. Let's start the program after the logo screen is run ($0C5F). Type G 0C5F and hit RETURN. The screen should turn black. Wait for about five seconds and reset the computer. Return to the monitor with SYS 49152. Using the INTERPERET command, examine code from $0A00 on (I 0A00). Code at $0AB6 reveals a Block Execute (executes the protection check placed in drive memory) and code at $0AC2 reveals a Memory Read that reads the value placed in the drive by the protection check. This value, in this scheme, is always an $FF. Examine code at $0A92. The value is being returned to the computer by a Memory Read with a kernal routine. The $FFCF routine brings back the value $FF. It is then EORed with $FF which turns it into a $00 and then stores it at location $0AB5. Our job is to place the correct value in $0AB5 and disable the routine overwriting it. This can be accomplished by placing three NOPs at $0A97 which will allow the routine to Memory Read the value but not place it in computer RAM. All that is left is to place the value of $00 at $0AB5.

6/ Now we have the correct values to plug into the code to disable and give the protection check what it wants. The last step is to place the changes on the disk. This is best done with a sector editor because to scratch and replace the 1985 file will destroy neccessary code placed on the disk. This code is not

accessed in the normal fashion, so it may be overwritten if we do a scratch and save of the 1985 file. Finish the job by following these steps:

A/ We know the code was originally encrypted, so we must place our values on the disk in encrypted form. The three bytes at $0A97 and the single byte a $0AB5 are the only changes needed. Reload the 1985 file <> L 19*" ,08 <>. Again go to location $0D06 and place a 00 in memory. Inspect the three bytes at $0A97. They should be 19 8E E8. The byte at $0AB5 should be an 8A. These are the bytes we will look for on our backup with the sector editor.

B/ The code can now be decrypted by typing G 0C5F. Again, the screen will turn black. After a few seconds, reset the computer and reactivate the monitor with SYS 49152. Using the MEMORY command (M 0A97), change the code at $0A97 from 8D B5 0A to EA EA EA. Change the code at $0AB5 from AC to 00.

C/ Now that our changes are in memory, we may re-encrypt the file (and our byte changes) by again typing G 0C5F. Again reset out and SYS the monitor back in with SYS 49152 and hit RETURN. Examine memory at $0A97 and find the encrypted byte changes. They should be 7E D1 08. The byte at $0AB5 has changed to 26. Now we know the changes, and the location so we may now do the actual changes to the backup.

D/ Reset the computer and load the sector editor from the utility disk <> LOAD"DISK DR",8,1 <>. When the cursor appears, type RUN and hit RETURN. Remove the utility disk and place your backup in the drive. Hit RETURN. You will be shown track 18, sector 1. By placing the cursor at position 35, you will be on the file pointers of the 1985 file. Press the J key to jump to the beginning of the 1985 file. When the sector comes on the screen, examine the first four bytes. The first two are links to the next sector of the file. The next two are the address bytes in reverse order ($0A00). We know our changes are in memory block $0A00 so we are in the proper block to make our changes. This block turned out to be track 17, sector 4 on our version. Yours could be in a different location on the disk, but the idea will be the same.

E/ Using the cursor key to move through the code, we find the original three bytes 19 8E E8 at location 155. The change to 7E D1 08 can be accomplished with the @ key. The changes must be the decimal equivilant. These are 126 209 08. Change each byte by placing the cursor over the byte to be changed, and type @ and the the decimal number change. Hit RETURN when the change is made to lock it in. When all three bytes are changed, continue searching with the cursor for the 8A byte. This can be found at position 185. Using the same change procedure, change it to a decimal 38 ($26). When all changes have been made and locked in, press C to copy the sector back to the disk.

7/ You now have a copy that can be fast copied. The placement of data on the disk in methods other than directory files will not allow you to file copy. One other point of interest is the fast loader installed in many pieces of this publisher's software. This fast loader is NOT compatible with the 1571 disk drive. In many (but not all) of the programs, you may disable the fast loader and allow the program to load on the 1571 by changing the jump to the main program in the autoboot. Fireworks Kit will work by doing this. Just change the 3D 0C (JMP 0C3D) to 5F 0C (use DISK Doctor and the decimal equivilents). This will bypass the fastloader and the logo screen. A small price to pay for the 1571 owners.

< < < PROGRAM: RINGS OF ZILFIN <> PUBLISHER: S.S.I. > > >

Procedure:

Loading the original produces a rattle free load, and an error scanner shows no standard errors. A backup made with Three Minute Backup produces a non-working copy. A nibbled backup produces the same non-working copy. Before starting to work on this program, please make a (non-working) backup of the original, and a disk log to log the file addresses.

Working with your backup:

1/ Turn off your computer and insert your reset button assembly into the cartridge port. Turn the computer on again. Load the backup disk <> LOAD "*",8,1 <>. Hit RETURN and the program will autoboot. Let the load continue untill the screen turns black and the drive comes to a stop. The program has failed protection and has "crashed".

2/ Hit the reset button to return the system back to normal. Remove the backup, and insert your utility disk in the drive. Load the $C000 monitior <> LOAD "49152",8,1 <> and sys it in by typing SYS 49152 and hit RETURN. When the monitor comes up, use the INTERPERET command to search memory for any drive commands. Start your search at the beginning of BASIC memory (1 0801). Scrolling down through memory, keep your attention on the left side of your screen. When you come to the memory at $6FDE, you'll find a B-E (Block Execute) and a M-R (Memory Read). This is the area of memory that contains the protection code.

3/ Disassemble memory at $6F77 (D 6F77), and scroll slowly down through the code. The code from $6F7A to $6FDD represents a subroutine that is called from the main program. This code does a Block Execute from track 35 sector 10. This means it loads that block from the program disk into the disk drive memory and executes that routine. At the completion of the routine, the code returns to computer RAM and resumes operation. Upon it's return, a Memory Read of the drive memory is done, looking for a single byte placed in drive memory by the protection check. This byte is transfered from the drive to computer RAM location $6FDD and is then compared to an $FF. If the byte is not an $FF, the code is directed to an endless loop. If it is an $FF, the code continues until a JUMP FFC3 is encoutered. Because the kernal routine FFC3 has been acessed by a JUMP and not a JSR, it forces an RTS in the code flow. This RTS returns the protection check to the main program.

4/ Defeating this protection scheme is simple. We can place a RTS at the beginning of the routine. This will short circuit the protection check completely by sending the program flow back to the code that called for it originally. Before changing code, let's find out which file contains the protection check. Looking over the disk log, we find that the file P99 is the only likely candidate. The starting address is $6000 and the ending address is $6FF4. Remove your utility disk from the drive and again insert the backup in it's place. Double check the file by loading P99 directly from the backup <> L "P99",08 <> . Again disassemble code around $6F7A (D 6F7A) and make sure this file is the correct one that has the protection check. When satisfied, use the MEMORY command to change the byte at the address $6F7A (M 6F7A) to a 60.

Now scratch and save this file to your backup. Remember to add one byte to the ending address. <> S "@0:P99",08,6000,6FF5 <>.

5/ Your backup is now completely broken. It can be fast copied and, because we have forced the program to not use the protection check, it can even be file copied. Remember, the Block Execute (which now is not used) accesses a specific spot on the disk, and is not picked up by directory files. Finally, note the name placed on the diskette directory. You'll find it on many programs. Now you know the secret of XEMAG 2.0 protection.

< < < INTRODUCTION: PROTECTION SCHEME TYPE D > > >

When this protection scheme was first introduced, the copy programs available were unable to backup any software that used it. Most of the nibble utilities on the market today have the capability of producing a backup. This scheme is usually referred to as the "long sector". The following similarities are characteristic of this protection. A nibble utility can back up the title, while a fast copier can't. The load is rattle free and smooth. An error scan produces a number twenty read error on the last sector of any particular track.

This protection is based on placing an extra sector on any chosen track (sometimes more than one track) on the original disk. This sector contains one block of valid program data. A non-nibbler or file copy utility will not pick up this sector, because it is not standard disk format. This will prevent the program from operating properly. Our job in each of the following programs will be to gather the block of data and place it in the program at the proper location.

The protection itself is nothing more than a special Block Read set up to read the non-standard block of data. The routine almost always starts out as an encrypted block. This block begins as a decryption routine that decrypts one block of data. This, in turn, reveals a protection check that does nothing more than read in the long sector and place that long sector data directly over itself. By doing this, the valid code completly hides the protection check itself.

Recognizing the decryption routine is the best way to locate the protection check. Once located, we will start the routine up and let it gather the data we need to break the title. Then a simple memory save is all that's needed to complete the job.

The benefit of breaking the programs using this protection scheme is the fact that almost all of them are file copiable afterwards. This means they can be placed on a disk with other programs.

Please note that this protection scheme is very important to understand. The reason for this is the fact that there is a new scheme now on the market that very closely resembles it (at least in the break method). This new scheme is NOT copiable by any nibble utility and must be hand broken. You'll find this new scheme discussed in the next chapter.

< < < PROGRAM: IMPOSSIBLE MISSION <> PUBLISHER: EPYX > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows a number twenty error on track 16, sector 20. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities are capable of providing a backup. Loading the backup results in a load that stalls rather quickly. We can assume the protection is in the loader file. Before starting to work on this title, please make a Three Minute backup and do a disk log (print-out is best).

Working with your original:

1/ Turn off your computer and insert your reset button assembly into the cartridge port. Turn the computer on again and, from the utility disk, load the $8000 monitor <> LOAD "32768",8,1 <>. Sys the monitor in with SYS 32768 and hit RETURN. Let's begin by loading and inspecting the boot file <> L "RUN ME",08 <>. At the end of the load, start disassembly at $02A7 (D 02A7). Scroll down through the code and notice that the boot loads the file LOADER (LO*) and jumps to $B000.

2/ Load the LOADER file <> L "LO*",08 <>. Because this file resides in the BASIC interpreter loacation, we must turn BASIC off before we can examine any code. Change address location $0001 from 37 (77 on C-128) to 36 (76 on C-128). Use the MEMORY command (M 0001) to make your change. When the change has been made, we can inspect the code beginning at $B000.

3/ Disassemble starting at $B000 (D B000) and inspect the code from $B000 to B00F. This is a decryption routine and is the heart of this protection scheme, as discussed in the introduction. Our job will be to trade the protection code for the valid program code. Believe it or not, this is the easy part.

4/ Make sure you have a write protect on your ORIGINAL and that the original is in the disk drive. Start the program working by typing GO B000 and hit RETURN. The drive should start up and, a few moments later, the screen should change colors. At this point, reset your computer and turn the disk drive off and on again. Resys the monitor back in (SYS 32768) and again turn off BASIC as described above. Disassemble code at $B000 (D B000) again and note that the code has, indeed, changed. The encrypted code has been replaced with loader code. All that's left now is to save the file back to the backup.

Working with your backup:

5/ Checking the disk log provides the start and ending addresses ($B000-$B1A2) ot the LOADER file. When saving it, be sure to add one byte to the end address <> S "@0:LOADER",08,B000,B1A3 <>.

6/ Your backup is now protection free and may be file copied. One small problem remains. That is the directory. The repair for this is simple. Using the Name/Id Changer on the utility disk, change the disk name AND the ID number. You must use five figures when changing the ID number. For example, you could name the disk IMPOSSIBLE and renumber it IM 2A. When this is completed, your break will be complete and even the directory can be viewed.

< < < PROGRAM: BREAK DANCE <> PUBLISHER: EPYX > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows a number twenty error on track 16, sector 20 and track 15, sector 20. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities are capable of providing a backup. Before starting to work on this title, please make a Three Minute backup and do a disk log (print-out is best).

Working with your original:

1/ Turn off your computer and insert the reset assembly into the cartridge port. Turn your computer on again. From your utility disk, load the $8000 monitor <> LOAD "32768" ,8,1 <>. Now, type NEW, and hit RETURN. When loading the boot file on this disk, it will autoboot and continue running. In order to inspect it, here's a trick to use. We're going to load the autoboot into BASIC memory for the purposes of inspection. Load the boot file this way: <> LOAD "BOOT",8 <>. When the load is complete (you may have to hit RUNSTOP/RESTORE), sys the monitor in with SYS 32768 and hit RETURN. You can now find the boot file in BASIC memory at $0801. Interpret memory and scroll down from $0801 (I 0801). Notice the INTRO. Disassembly of memory at $0801 (D 0801) and scrolling down reveals a loader file that loads the INTRO file and jumps to $2015.

2/ Load the INTRO file <> L "INTRO",08 <>. Start by disassembling memory at $2015 (D 2015). Scroll down through memory, and at $201A note the JSR $26B9. Disassemble $26B9 (D 26B9). Here we find the decryption routine that is the heart of this protection scheme. Refer to the Introduction for general information on this. Our task is to replace the encrypted data with valid program data. This is relatively easy. Be sure you have a write protect on your original and that the ORIGINAL is in the drive. Type GO 26B9 to start the program up. The drive will run for a short time, and then stall. When the drive stops, reset the computer and resys the monitor back in (SYS 32768). Disassemble memory at $26B9 again and notice that the code has indeed changed. This is the valid program code we needed for the break.

Working with your backup:

3/ Now, all that's left is to save the retrieved data back to the backup. Checking the disk log provides the start and end addresses of $2000-2A00. Be sure to add one byte to the end address and save it to the backup <> S "@0:INTRO",08,2000,2A01 <>.

4/ Turn the computer off and on, and boot up your backup. It should load past the point that it loaded before our break. Unfortunatly, It still refuses to load fully. Remember, we did find two seperate number twenty errors on the original. We have disabled half of the protection, now let's do the rest.

5/ Reload the $8000 monitor <> LOAD "32768",8,1 <>. Sys it in with SYS 32768. From the half broken BACKUP, reload the INTRO file <> L "INTRO" 08 <> . Again, start your disassembly at $2015 (D 2015). Scroll down, and try to follow the program flow. At $2140 you'll find a JUMP $C000. Using the MEMORY command

change the 4C at $2140 (M 2140) to 00 and hit RETURN. This will stop or BREAK the program flow just before it jumps to $C000, allowing us to inspect memory in the LOADER file. Activate the INTRO file by typing GO 2015.

6/ When the drive stops, reset the computer and reload your $8000 monitor <> LOAD "32768",8,1 <> . Sys it in with SYS 32768. Start by disassembling the code at $C000 (D C000). You'll find a jump to $C024. Disassembly of $C024 reveals another decryption scheme. This is the second protection routine.

Working with your original:

7/ Place the original disk in the drive and Type GO C024 to start the program up. The drive should start up and in a short time the game menu will come on the screen. At this point, reset the computer and resys the monitor back in with SYS 32768.

Working wiyh your backup:

8/ Checking the disk log provides us with the start and end address of the LOADER file. Again, remember to add an exrta byte to the end address. Save it back to your BACKUP <> S "@0:LOADER",08,C000,CF81 <>. When the save is complete your backup will be completely broken. One small problem remains. The directory cannot be read properly. To fix it easily, just use the NAME/ID CHANGER on your utility disk. Be sure to use five figures when you give it a new ID number. For example, you could name it BREAK DANCE and renumber it BD 2A.

< < < PROGRAM: PITSTOP II <> PUBLISHER: EPYX > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows a number twenty error on track 16, sector 20. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities are capable of providing a backup. Before starting to work on this title, please make a Three Minute backup, format a blank work disk, and do a disk log (print-out is best).

Working with your original:

1/ Turn off your computer and insert your reset button assembly into the cartridge port. Turn the computer on again and, from the utility disk, load the $C000 monitor <> LOAD "49152",8,1 <>. Sys the monitor in with SYS 49152 and hit RETURN. Let's begin by loading and inspecting the boot file <> L "PITSTOP",08 <>. At the end of the load, start disassembly at $02A7 (D 02A7). Scroll down through the code and notice that the boot loads the file RUN ME and jumps to $0820.

2/ Load the RUN ME file <> L "RUN ME",08 <> . Disassemble memory starting at $0820 (D 0820) and scroll down through the code. This file loads all game files and then at $08E4 does a jump to $9403. The disk log tells us this address is located in the PITS file. Load the PITS file <> L "PITS",08 <>. When the load is complete, we can start our inspection at $9403.

3/ Because this file occupies memory in the BASIC interpreter ($A000-$BFFF), we have to turn BASIC off. This can be accomplished by changing $0001 from a 37 (77 on the C-128) to a 36 (76 on the C-128). Use the MEMORY command to make your changes (M 0001). When done, start disassembly at $9403 (D 9403). You'll find a decrytion scheme at this location ($9403-$9412) that is the heart of this protection scheme (see the Introduction). Make sure your ORIGINAL has a write protect tab on it and is in the drive. Start the program working by typing G 9403 and hit RETURN. The drive should start up and a few moments later, the game menu should come on the screen. At this point, reset your computer and remove the original disk from the drive. From the utility disk, reboot the $C000 monitor and sys it in again (SYS 49152). Again turn off BASIC. Now place your formated work disk in the drive and save the changed code from $9403-$9512 <> S "SECTOR",08,9403,9512 " <>. When the save is complete, remove the work disk from the drive.

Working with your backup:

4. Complete the break by following the steps below.

A- Reset the computer. Place your backup in the drive and scratch the PITS file <> OPEN15,8,15,"S0:PITS" <>. Resys the monitor back in (SYS 49152).

B- From the original disk, load the PITS file <> L "PITS",08 <>.

C- From the work disk, load the saved SECTOR file <> L "SECTOR",08 <>. This will lay the code retrieved from the break process over the encrypted

protection check code.

D- Turn off BASIC again as described above.

E- Place your backup in the drive and save the PITS file now in memory <> S "PITS",08,1000,C000 <>.

5/ Your Backup is now broken. All that's left is to repair the directory. This can be accomplished easily with the NAME/ID CHANGER on the utility disk. Be sure to use five figures in the new ID number. For example, you could name the disk PITSTOP and renumber it PS-II. When this is complete, you can view the directory and file copy this title.

< < < THE BODY TRANSPARENT <> PUBLISHER: DESIGNWARE > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows a number twenty error on track 32, sector 16. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities are capable of providing a backup. Before starting to work on this title, please make a Three Minute Backup, and do a disk log (print-out is best).

Working with your backup:

1/ Let's begin our break by preparing the backup to receive the changes we will be making. From your utility disk, load the NAME/ID Changer and rename and re-ID your backup. Be sure to use five figures in the new ID. For example, you could name the backup BODY TRANS and number it BT-2A. This will make the directory listable.

2/ Because this program does not use directory files to store information, we run the risk of overwriting program code when we save our changes to the backup. There is a sure way to avoid this. That is to allocate or use all availabe blocks in the BAM. What we are going to do is fool the drive into believing that there are no blocks free on the disk. When we scratch a file, the blocks used by THAT file will become free for use. Then when we save that file back to the disk, they will be placed on the exact same blocks that they came from.

3/ From the utility disk, load and run DISK DR. Place the backup in the drive and press RETURN to get to track 18, sector 1. Press - to go to track 18, sector 0. This is the BAM Sector and here is where we will allocate all blocks. Use the cursor key to cursor to position 4 (all references will be in decimal). With the cursor on position 4 press the @ key and then press 0. Repeat the @ key and 0 key untill all values from position 4 thru 71 are changed to zero. This takes care of tracks 1 thru 17. Now cursor over to position 76 and do the same changes from position 76 thru 143. This will take care of tracks 19 thru 35. Now, to make the changes to the disk, press R and then Y and hit RETURN. The new BAM is now on the backup. Your backup is now ready to receive new information. Load the directory and check it. You should have a listable directory with zero blocks free.

Working with your original:

4/ Turn off the computer and insert the reset assembly into the cartridge port. Turn the computer on again and load the boot file from the original <> LOAD "DWARF",8: <>. You can list this file and inspect it. You'll find it loads the file called BOOT2 and then a SYS 49152 ($C000).

5/ From your utility disk, load the $2000 monitor <> LOAD "8192",8,1 <>. Sys it in with SYS 8192. Now load the BOOT2 file <> L "BOOT2",08 <> and start disassembly at $C000 (D C000). The first instruction at $C000 is a JSR to $C028. Disassemble $C028 (D C028) and here you'll find the decryption routine that is the heart of this protection scheme. It resides from $C028 to $C037. The break itself is very simple. Make sure you have a write protect tab on the ORIGINAL and that it is in the drive. Start the program by typing G C028 and

press RETURN. The drive will spin for a short time and then stop. At this point, reset the computer and resys the monitor back in with SYS 8192. Again disassemble code at $C028. You should find new code in the place of the encrypted code. All that's left is to save this broken loader back to the backup.

Working with your backup:

6/ Reset the computer and place your prepared backup in the drive. Scratch the BOOT2 file <> OPEN15,8,15,"S0:BOOT2" <> . Resys the monitor in with SYS 8192. The disk log provides the start and end addresses of the BOOT2 file. Be sure to add one byte to the end address. With your backup in the drive, save the BOOT2 file back to the backup <> S "BOOT2",08,C000,C151 <>.

7/ Your backup is completely broken and can now be copied with any whole disk copier. Unfortunatly, it remains non-file copiable because of the way the programers set up the disk files.

< < < INTRODUCTION: PROTECTION SCHEME TYPE E > > >

This protection scheme is, at this writing, one of the most effective and prevalent methods of defeating today's nibble copiers. When you know what to look for, you'll find this scheme is being employed by many different software houses. I like to think of this protection as the "big brother" of the long sectors discussed in the previous section.

This scheme can be recognized by the following similarities. When a disk error check is done, no write errors will be found on the original. When booted, no drive rattle will be encountered. The program cannot be backed up with either a fast copier or a nibbler. Usually, you will find data in the directory other than normal directory data. Most important: when tracing the program through it's loading process, you will generally run into a decryption routine and a sector or two of encrypted code. When this encryption is located, you can be sure it is hiding the protection check code.

Remember, I stated that a sector or two in memory will be encrypted, and that this area in memory surely contained the protection check. Well, one other thing needs to be mentioned. This is the fact that this encrypted memory starts out as garbled code, then decrypts into a protection check routine and finally after the protection check has been satisfied, is REPLACED with valid program code. This code, as previously stated, is one or two sectors in length and can be found anywhere on the program disk. You'll find that the directory track (track 18) is the most likely spot. In most cases, we can let the program insert the hidden code in it's proper place. Then a memory save and replacement over the encrypted code in the proper file will not only defeat protection but will totally remove the check for it.

Most of the programs protected with this scheme can be defeated with a simple memory save, but a few have had to have some of the code re-written by hand. This is relatively uncommon and cannot be explained in the scope of this manual. Experience will prove to be the best teacher.

Before starting to work on the following programs, please do a disk file log (print out is best), format a blank work disk, and have a (non-working) backup available. Please make sure you have a write protect tab on your original program disk as you will be using it in the breaking process. Now let's get on to the specifics.

< < < PROGRAM: INFILTRATOR <> PUBLISHER: MINDSCAPE > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows no standard errors. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities also provide a non-working backup. Loading the backup results in a load that stalls rather quickly. We can assume the protection is in the loader file. Before starting to work on this title, please make a backup (non-working) and a disk log (printout is best).

Working with your original:

1/ Place a write protect tab on your original to protect it during the breaking process.

2/ Turn off your computer and insert the reset assembly into the cartridge port. Turn your computer on again. From your utility disk, load the $C000 monitor <> LOAD "49152" ,8,1 <>. When the load is complete, sys the monitor in with SYS 49152. When loading the boot file on this disk, it will autoboot and continue running. In order to inspect it, here's a trick to use. We're going to load the autoboot in BASIC memory for the purposes of inspection. With the monitor active, type X and hit RETURN. You are now back to BASIC. Type NEW and hit RETURN. Now load the boot file this way: <> LOAD "INFILT",8 <>. When the load is complete, return to the monitor by hitting RUNSTOP/RESTORE. Then resys the monitor back in with SYS 49152. You can now find the boot file in BASIC memory at $0801. Interperet memory and scroll down from $0801 (I 0801). Notice the INTRO. Disassembly of memory at $0801 (D 0801) and scrolling down reveals a loader file at $082D-$0854. This loader loads the INTRO file and jumps to $0880.

3/ Load the INTRO file <> L "INTRO",08 <>. When the load is complete, disassemble memory at $0880 (D 0880). Scroll down through memory to $089A. You'll find a JSR 0A25. Disassemble $0A25 (D 0A25) and scroll down to $0A25. Here you'll find a JSR 0C18. Disassemble $0C18 (D 0C18) and notice that we have just run into a decryption routine. Inspect this routine because this is the heart of this protection scheme. Scroll down through the code and notice that it is garbled for about one sector ($0C18-$0D18). As mentioned in the introduction, this code is an encrypted protection scheme that will decrypt into a protection checker and then load valid program code over itself. This will not only allow the program to operate properly, but will also hide the protection code from the curious.

4/ The break is fairly simple now that we know where the protection is. Start the program code up by typing G 0C18 and hit RETURN. The drive should start up and run for a short time. When the drive stops, turn the drive OFF and ON again and reset the computer with your reset button. Restart the monitor by again typing SYS 49152 an hit RETURN. Now go back and disassemble code at $0C18 again (D 0C18). Surprise; the code has changed into good code. To get an idea what is there, interperet memory at $0C18 (I 0C18) and scroll down through memory. You'll see that this is the completion of the loader file. All the data needed to run the loader file properly is now in memory. All that is left to do is replace the INTRO file on the disk with the INTRO file NOW in memory. This can

be accomplished with a small memory save. From the disk log, we know that the INTRO file starts at $0880 and ends at $16C3. Remove the original disk from the drive and insert your backup in it's place. Replace the INTRO file now in memory with the one now on your disk. Remember to add one byte to the ending address <> S"@0:INTRO",08,0880,16C4 <>.

5/ Your backup is now broken and will not even check for protection. For those wishing to look at the protection check code, redo the steps above but when you type G 0C18, reset the computer in about one second. If the drive is allowed to run more than a moment or two, the protection code will be hidden.

< < < PROGRAM: BOP 'N WRESTLE <> PUBLISHER: MINDSCAPE > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows no standard errors. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities also provide a non-working backup. Before starting to work on this title, please make a backup (non-working), format a blank disk, and do a disk log (printout is best).

Working with your original:

1/ Make sure a write protect tab is on your original to protect it during the breaking process.

2/ Turn off your computer and insert the reset assembly into the cartridge port. Turn your computer on again. From your utility disk, load the $2000 monitor <> LOAD "8192" ,8,1 <>. When the load is complete, sys the monitor in with SYS 8192. When loading the boot file on this disk, it will autoboot and continue running. In order to inspect it, here's a trick to use. We're going to load the autoboot in BASIC memory for the purposes of inspection. With the monitor active, type X and hit RETURN. You are now back to BASIC. Type NEW and hit RETURN. Now load the boot file this way: <> LOAD "B",8 <>. When the load is complete, return to the monitor by hitting RUNSTOP/RESTORE then resys the monitor back in with SYS 8192. You can now find the boot file in BASIC memory at $0801. Interperet memory and scroll down from $0801 (I 0801). Notice the BOP1. Disassembly of memory at $0801 (D 0801) and scrolling down reveals a loader file at $082D-$0854. This loader loads the BOP1 file and jumps to $0816.

3/ Load the BOP1 file <> L "BOP1",08 <>. When the load is complete, disassemble memory at $0816 (D 0816). Scroll down through memory to $0889. You'll find a JMP C000. Using the MEMORY command (M 0889), place a 00 (BRK) at $0889. If we start the code running from $0816 it will execute and stop just before it would have jumped to $C000. We can then disassemble memory at $C000 and trace the program flow. Use the GO command to execute this code (G 0816).

4/ The load will resume and the LOGO file and LOADALL file will be loaded. When the program stalls, reset out and reboot your monitor from the utility disk <> LOAD"8192",8,1 <>. When the load is complete, sys the monitor in with SYS 8192. Disassemble code at $C000 (D C000) now and scroll down through memory. You'll find a very long loader file. When you reach the code at $C27A you'll find a JMP C3FD. Disassembly of C3FD shows no valid code so this is a likely spot to place another break in the program flow. Using the MEMORY command (M C27A), place a 00 (BRK) at $C27A. Now restart the program with another GO command (G C000). When the program stalls out, reset the computer again and reload and activate your 2000 monitor <> LOAD"8192",8,1 <>. Now we can disassemble memory at $C3FD and again follow the program flow (D C3FD). This returns a JMP to 0B40. Disassembly of memory at $0B40 reveals the decryption code that we discussed in the introduction. This is the heart of this protection scheme.

5/ Let's execute the code at $0B40. Make sure your original is in the drive. Start up the code with G 0B40. The drive should start up and soon stall again.

Reset out, resys your monitor in (SYS 49152), and disassemble code again starting at $0B40. You'll now find different code. Remove the original copy and place your formated work disk in the drive. We can now save this new code to our work disk <> S "CODE",08,0B40,0C52 <>.

Working with your backup:

6/ We now have the code necessary to break this title. Now we have to place it on the disk in the proper spot. Checking the disk log, we find the files LOGO, BNK12A, TITLE, and BOP1 all have the correct addressing to be likely places for this file. We must load and check in each one with our monitor the address $0B40. The file BNK12A turns out to be the correct file. Now all that is left is to place our changed code over the original code. Because BNK128 begins in screen memory, we will have to pull a few tricks out of the bag to replace our revised code. Remember, this file starts in screen memory, and we can't save screen memory properly. Follow these steps and try to reason them out as we go through them.

A- Load DISK DR from your utility disk. When the cursor reappears, type RUN and hit RETURN. Place your backup in the drve and hit RETURN. You'll be shown track 18, sector 1. The jump link to the BNH12A file is at position 195. Cursor over to position 195 and hit the J key. You will be taken to the first sector in the file. The first four bytes in the file are the pointer bytes. We want to change the program address from $0400 to $0900, so cursor over to position 3 and hit the @ key. Now, hit the 9 and press RETURN. Hit the R key to make the change to the backup.

B- Reset the computer and load the $C000 moniter from your utility disk <>LOAD"49152",8,1 <>. Sys it in with SYS 49152. Now, from your formated disk, load the CODE file <> L"CODE",08 <>. We now will transfer it to a holding spot in memory, for later use <> T 0B40 0C50 7B40. This will send the code to $7B40.

C- Now from the BACKUP load the altered file BND128 <> L "BND128",08 <>. Remember, it will now load five sectors ahead of it's normal spot (from $0400 to $0900). When the load is complete, disassemble the code at $1040. Again here is our decryption routine.

D- Transfer the code we placed at $7B40 to its proper place in the altered file <> T 7B40 7C50 1040 <>. When the cursor reappears, check the code at $1040. It should now contain the new code we saved from the break.

E- Save the altered file back to the backup <> S "@0:BNK12A",08,0900,6101 <>. Note we are adding five sectors to every address, plus one byte to the end address.

F- Now all that's left is to change the file address back to $0400. Follow the same procedure as in step 6a, except change the address pointer from an 09 to an 04.

7/ You now have a completely broken copy. The protection scheme has been totaly wiped out.

< < < PROGRAM: PRINT SHOP COMPANION <> PUBLISHER: BRODERBUND > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows no standard errors. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities also provide a non-working backup. Before starting to work on this title, please make a backup (non-working) of both sides, and do a disk log (printout is best).

I must admit that this program was fairly difficult to trace through the loading sequence. After several tries, it was time to reason the situation out. Watching the backup load a few times lit up the old mental light bulb. The load seemed complete; the only problem were the ICONS on the first menu screen. They were there, but non-operative. Checking the directory provided the file I felt deserved immediate attention.

Working with the original:

1/ Make sure to place write protect tabs on the original to protect it during the breaking process.

2/ Turn the computer off and insert your reset assembly into the cartridge port. Turn the computer on again and from your utility disk, load the $C000 monitor <> LOAD"C000",8,1 <>. Sys the moniter in with SYS 49152. Remove the utility disk from the drive and replace it with your original (Side A). Load the file ICONS <> L "ICONS",08 <>. The disk log tells us this file resides at $6000 in memory, so let's start our disassembly at $6000 (D 6000). Cursor down through memory and notice the decryption scheme at $6005-$6012. Remember from the introduction, this is the heart of this protection scheme.

3/ Let's execute the code at the beginning of the decryption scheme. Start it working with G 6005. The drive should start up, and in a short time, stall again. Reset the computer and resys the monitor back in with SYS 49152. Disassemble memory at $6000 again. Cursor down through memory and notice the code HAS changed. We now have all the data neccesary in memory to break this program. Let's save our altered ICONS file back to the backup.

Working with your backup:

4/ Checking the disk log shows that the ICONS file starts at $6000 and ends at $69AD in memory. With the backup in the drive, save the ICONS file, remembering to add one byte to the end address <> S "@0:ICONS",08,6000,69AE <>. Now turn the disk over and save the file to Side B as well.

5/ You may now load and check your backup. You'll find it to be completely broken, and now it can even be fast copied. For those who want to see the actual protection check, you can go back through the same steps as before. When you do the G 6005, just reset out after about ONE second. If the drive is allowed to run, it will pick up the break data from the original, and hide the protection check in memory.

< < < PROGRAM: BANK STREET SPELLER <> PUBLISHER: BRODERBUND > > >

Procedure:

Loading the original produces a rattle free load, and an error scan shows no standard errors. A backup made with Three Minute Backup provides a non-working backup. Nibble utilities also provide a non-working backup. Before starting to work on this title, please make a backup (non-working), and do a disk log (printout is best).

Working with the original:

1/ Make sure to place a write protect tab on the original to protect it during the breaking process.

2/ Turn the computer off and insert your reset assembly into the cartridge port. Turn the computer on again and from your utility disk, load the $C000 monitor <> LOAD"C000",8,1 <>. Sys the monitor in with SYS 49152. Remove the utility disk from the drive and replace it with your original. Load the boot file BSS <> L "BSS",08 <>. Using the disk log to guide us, let's disassemble memory at $02C4 (D 02C4). Cursor down through memory and notice the loader loads the file BSSL and does a jump to $7000. Let's load the BSSL file ourselves and follow the load sequence <> L "BSSL",08 <>.

3/ When the drive stops, disassemble memory at $7000 (D 7000). Cursor down through memory, and inspect the long loader file that loads in the entire program and the jumps to the start address. At the address $20C7 you'll find a JMP 0803. Using the MEMORY command (M 70C7) type a 00 over the 4C and hit RETURN. This will allow the loader to operate, and, when done, will BREAK just before the jump to $0803. We can then follow the program flow, beginning at $0803. Start the loader execution by doing a GO 7000 (G 7000). The drive will start up and the files will appear on the screen as they are being loaded. When the drive finally stops, reset the computer and re-sys the monitor back in (SYS 49152).

4/ Now let's disassemble memory at $0803 (D 0803). The first instruction we find is a JSR 09E1, so disassemble $09E1 (D 09E1). This disassembly reveals the decryption scheme that is hiding the protection check. You'll find it resides at $09E1 - $09F2. Study it closely, for it is the heart of this protection scheme.

5/ Be sure your original disk is in the drive and start the code up by doing a GO 09E1 (G 09E1). The drive will start up and in a few seconds will stall again. Again, reset the computer and resys the monitor in with SYS 49152. Disassemble memory at $09E1 (D 09E1) and inspect the code again. It has changed into valid program code. Now all that's left is to save the changed code back to the disk.

Working with the backup:

6/ Inspection of the disk log tells us that the file BSS0 is the likely candidate to contain the protection code. You may , as we did, load BSS0 and inspect the proper addresses to ensure our save is to the proper file. Then,

when satisfied, just redo step five and, when the code has been replaced again, save the file back to your BACKUP disk. The disk log tells us the file resides for $0800-$1600. Be sure to add one byte to the end address <> S"@0:BSS0",08,0800,1601 <>.

7/ When this save is complete, your backup will be completely broken, and may be copied with any fast copier. For those who want to inspect the protection code, just load in the BSS0 file and do a GO to 09E1. After about one second, reset out and resys the monitor in and inspect that memory area. You'll find the protection code intact. If you allow the drive to run for long, the protection code will be replaced by valid program code.

< < < MACHINE LANGUAGE MONITOR COMMANDS > > >

```
Assemble----: A aaaa ooo xxx
Compare-----: C ssss eeee ssss
Disassemble-: D ssss (eeee)
Fill--------: F ssss eeee xx
Go----------: G aaaa
Hunt--------: H ssss eeee xx
Interpret---: I ssss (eeee)
Registers---: R
Save--------: S "file name",08,ssss,eeee+1
Load--------: L "file name",08
Transfer----: T ssss eeee nnnn
Exit/Basic--: X
```

```
a  = address
o  = opcode
x  = bytes
() = optional
s  = start address
e  = end address
n  = new address
```

<===>

< < < DISK DOCTOR COMMANDS > > >

```
@ = Change Byte          t = Text Mode
+ = Scan Forward         - = Scan Back
n = Next Block           N = Previous Block
j = Jump to Link         J = Previous Link
b = New Block            B = Last Block
r = Rewrite Block        c = Copy Block
s = Swap Disks           p = Print Block
Q =Quit

  Clear = Renew the current sector display.
 Home   = Position the cursor over position 0.
 Cursor Keys = Position the cursor R/L or U/D.
Return = Position the cursor over the first byte of the next line.
```

<===>

< < < BOOKS FOR FURTHER READING > > >

Commodore 1541 Disk Drive Owners Manual .....(c)Commodore Business Machines

Commodore 64 Programmer's Reference Guide .....(c)Commodore Business Machines

Inside Commodore Dos .....(c)Reston Publishing Co.

Machine Language For Beginners .....(c)Compute! Books Publication

Mapping The Commodore 64 .....(c)Compute! Books Publication

Program Protection Manual (Vol 1 & 2) For The C-64 .....(c)CSM Software Inc.

< < < MACHINE LANGUAGE MONITOR COMMANDS > > >

```
Assemble----: A aaaa ooo xxx
Compare-----: C ssss eeee ssss
Disassemble-: D ssss (eeee)
Fill--------: F ssss eeee xx
Go----------: G aaaa
Hunt--------: H ssss eeee xx
Interpret---: I ssss (eeee)
Registers---: R
Save--------: S "file name",08,ssss,eeee+1
Load--------: L "file name",08
Transfer----: T ssss eeee nnnn
Exit/Basic--: X
```

```
a  = address
o  = opcode
x  = bytes
() = optional
s  = start address
e  = end address
n  = new address
```

<==>

< < < DISK DOCTOR COMMANDS > > >

```
@ = Change Byte          t = Text Mode
+ = Scan Forward         - = Scan Back
n = Next Block           N = Previous Block
j = Jump to Link         J = Previous Link
b = New Block            B = Last Block
r = Rewrite Block        c = Copy Block
s = Swap Disks           p = Print Block
Q =Quit

  Clear = Renew the current sector display.
 Home  = Position the cursor over position 0.
 Cursor Keys = Position the cursor R/L or U/D.
Return = Position the cursor over the first byte of the next line.
```

<==>

< < < BOOKS FOR FURTHER READING > > >

Commodore 1541 Disk Drive Owners Manual .....(c)Commodore Business Machines

Commodore 64 Programmer's Reference Guide .....(c)Commodore Business Machines

Inside Commodore Dos .....(c)Reston Publishing Co.

Machine Language For Beginners .....(c)Compute! Books Publication

Mapping The Commodore 64 .....(c)Compute! Books Publication

Program Protection Manual (Vol 1 & 2) For The C-64 .....(c)CSM Software Inc.

< < < KRACKER JAX REVEALED > > >

Thank you for your purchase of Kracker Jax Revealed. This manual is the product of countless hours of investigation, writing, and rewriting. I'd like to thank my family - especially Hope, my computer widow - for their patience while staring at my back.

I hope this manual provides you with the spark to fire your own curiosity. Remember, as you go through the tutorials provided, that they are the culmination of hours of hunt and search. What appears easy now was hidden and difficult the first time through. Each protection scheme is a puzzle waiting to be unraveled. Only the most inquisitive and curious will succeed.

Keep this in mind as you try new schemes. No rules apply, anything goes. What worked before may or may not work now. It's you against the protection programmers, and they play to win.

< < < LIMITED WARRANTY > > >

We have attemped to assure that this manual, along with the software and hardware, works as specified. We would appreciate receiving notice of any errors you may find.

Neither the author nor any distributor of this product will be liable for any damages which may be a result of errors or omisions, use or misuse of this product. Should there be any defects in the software provided in this package, we will replace the defective diskette within 90 days from the date of purchase. You will find our software unprotected, and are encouraged to make a back up for your own use.

<==========>

Important Notice: This instructional material has been written for educational and archival purposes only. You are advised that the Federal Copyright Law allows you the right to back up, for archival purposes, any computer program you have purchased. Any other use could be unlawful and is not advised nor encouraged. By using this product, you agree to be bound by the terms of this notice.

THE FOLLOWING TITLES ARE REGISTERED TRADEMARKS OF THE FIRMS LISTED BELOW.

Title Bout, Superbowl Sunday, Gulf Strike ..... Avalon Hill.
Pitstop II, Break Dance, Impossible Mission ..... Epyx.
Print Shop Companion, Bankstreet Speller ..... Broderbund.
Countdown To Shutdown, Web Dimension, Fireworks Kit ..... Activision.
Infiltrator, Bop 'N Wrestle ..... Mindscape.
Creative Contraptions ..... Bantam Software.
The Body Transparent .....Designware.
The Rings of Zilfin ..... S.S.I.
Tapper ..... Bally Midway.
Buckaroo Banzai ..... Adventure International.
Sargon III ..... Hayden Software.
The Slugger ..... Mastertronics.