# Unveiling The Pirate Part 2: Programming Sleight Of Hand

One of the most exciting things a programmer can master is the art of making your computer do weird things. Our Bits and Pieces section is one example of where grown men and women search out and publish new and improved ways to make your computer beg for mercy. The same is true with most programmers who get in the mood for some really hot protection. It is all just an extension of the same idea, to make your computer perform tricks that are not normally part of its act. This article has been written to show you some new and improved ways to add confusion and pain to the lives of pirates everywhere. Though not a glossary on every method that can be achieved, it should suffice to whet your appetite and start you on the trek to find some more. Protection can be addicting.

### A Quick Word On Compiling

Though not a subject that excites many in the programming community, I felt it best to discuss compiling for a few minutes, just to let you know that it is still around and is prone to breakage if attacked by the serious pirate.

Compiling a BASIC program is usually a reliable way to discourage most hackers. Once the infamous words of COMPILED BY . . .. are discovered, most hackers turn away. Again, there are exceptions to this rule. Available through the bootleg community right now are a few decompilers for the most widely used compilers. There is one that I am quite familiar with that can decompile a compiled program back into its original state as it was before compiling ever took place. If your only method of hiding protection was compiling, then forget it. This one will bare to the world whatever tricks you were performing. A pretty low life trick, but it is extremely effective.

If the chances are that your program will not be too badly exposed if decompiled, then compile to your hearts content. Just remember that there are a number of hackers out there right now with their decompilers, revving up their disk drives in anticipation. Make sure that your code is a little more tricky in the method used to check for protection. In this way they may have to work a little harder to achieve the same end result, if it can be done.

### POKES, Line Tricks And DOS Tricks

As Mary Poppins once stated so elegantly, "these are a few of my favourite things." In this I support her convictions all the way. Give me a memory map and a computer, then stand back. It can be quite a challenge to set my devious little mind to work on new and improved methods to supercede the evil wishes of that master of deceit, the pirate. Aar de aar Billy, have you broken that package yet?

Below will be found all sorts of tricks that have either been well concealed in the past, or never thought of before. Combining several techniques is your best defense. However your protection must share space with your program. Too much protection and your program may run out of memory. It's up to you.

### Auto Check For Protection: CHRGET

For anyone wishing to have their program automatically check for protection, CHRGET is often found to be rather handy. Located at $70-$87 with the PET/CBM and $73-$8A on the C64/VIC, a slight change within will allow you to reroute the CHRGET flow to where ever your protection desires. For the PET/CBM, locations $79-$7B can be modified quite effectively to include a JUMP followed by a 16 bit address. The same applies for locations $7C-$7E on the C64 and VIC. If you reproduce everything from these locations down to the end of CHRGET, and master the basic concepts of how CHRGET works, then you are sure to be able to CHRGETize your protection with few snags.

There is one very large disadvantage in using CHRGET for executing vast amounts of code though. BASIC execution will be slowed down relative to the amount of extra code CHRGET is expected to execute in its quest for protection. Keep your protection code compact and your program won't suffer too badly in performance.

### Locations To POKE About With

The table on the following page is not a chart of yet undiscovered orifices to prod about in, but a compendium of locations in RAM that can really turn a computer on. Proceed below and open up a vast new world of tricks you can play on your yet unsuspecting computer.

### #01 USR Function Jump

As has been explained by so many people in the past, the USR function in BASIC can be used quite effectively to access machine code from BASIC. By altering the jump address to where ever you like, indirect methods of accessing protection or plain and simple code can be found. A SYS to this address will also produce some fine results, if you are so inclined. Whatever your requirements, this vector can be handy at times.

| 40 Col PET | 80 Col CBM | C64 | VIC 20 | Description |
|---|---|---|---|---|
| $0000-$0002 | $0000-$0002 | $0310-$0312 | $0000-$0002 | (01) USR Function Jump |
| $0090-$0091 | $0090-$0091 | $0314-$0315 | $0314-$0315 | (02) Hardware Interrupt Vector |
| $0092-$0093 | $0092-$0093 | $0316-$0317 | $0316-$0317 | (03) Break Interrupt Vector |
| $0094-$0095 | $0094-$0095 | $0318-$0319 | $0318-$0318 | (04) NMI Interrupt Vector |
| N/A | N/A | $031A-$031B | $031A-$031B | (05) OPEN Vector |
| N/A | N/A | $031C-$031D | $031C-$031D | (06) CLOSE Vector |
| N/A | $00E9-$00EA | $0324-$0325 | $0324-$0325 | (07) INPUT Vector |
| N/A | $00EB-$00EC | $0326-$0327 | $0326-$0327 | (08) OUTPUT Vector |
| N/A | N/A | $0328-$0329 | $0328-$0329 | (09) Test STOP Vector |
| N/A | N/A | $032A-$032B | $032A-$032B | (10) GET Vector |
| N/A | N/A | $0330-$0331 | $0330-$0331 | (11) LOAD Link |
| N/A | N/A | $0332-$0333 | $0332-$0333 | (12) SAVE Link |
| $0028-$0029 | $0028-$0029 | $002B-$002C | $002B-$002C | (13) Pointer : Start Of BASIC |
| $0034-$0035 | $0034-$0035 | $0055-$0056 | $0055-$0056 | (14) Pointer : Top Of Memory |
| $009E | $009E | $00C6 | $00C6 | (15) # Chars in Keyboard Buff |
| N/A | $00E3 | $0289 | $0289 | (16) Max Size of Keyboard Buffer |
| N/A | N/A | $0300-$0301 | $0300-$0301 | (17) Error Message Link |
| $03FA-$03FB | $03FA-$03FB | N/A | N/A | (18) Monitor Extension Vector |

### #02 Hardware Interrupt Vector

Another name for this vector is the IRQ Vector, one that I am sure you have seen mentioned in our Bits and Pieces section. This vector can be changed to point to whatever code you want executed repeatedly, like a check for protection. One point to remember before performing any major operations with this vector, though. The more pre-interrupt code you have, the slower will be the execution of normal code. An important trade off to consider.

Another point to remember is to save the processor status and all registers that are corrupted on the stack before any operations. Then restore the values before transferring control back to the system.

For 4.0 BASIC people, you have probably noticed that you cannot use the disk drive for very much if the IRQ procedure has been changed. Try the following trick and I am sure you will be pleased. Point the BRK interrupt vector at your code, then point the IRQ vector at $E454. This location is a zero byte, or a BRK instruction. With every IRQ, the machine will break, then shoot over to your code. Clean, simple and a pretty neat trick to know. And all SAVEs and LOADs from disk will go off without a flaw.

One final word on the IRQ. If all else doesn't interest you then point the normal IRQ vector 3 bytes forward to disable the STOP key. Pretty boring, but what the heck, it works.

### #03 Break Interrupt Vector

As mentioned above, the BRK vector can help you out with your IRQ driven wonder. But it also can be used for a few other things. Often, when trying to break programs, break points are desired at specific spots to check how certain activities are going. For this, a BRK instruction will be placed in the machine code for the program to BRK into the machine language monitor when encountered, or back into BASIC mode with the C64 and VIC unless a monitor is in operation. At other times, even when the program has been protected to the hilt, it can be made to crash. The first thing that a 4.0 hacker will usually do is break into the machine language monitor to look at the code, or save everything for later viewing. Point the BRK interrupt vector at reset:

$FD16 (L/H = 22/253) for 4.0 BASIC
$FCE2 (L/H = 226/252) for the C64 & VIC

Every time a BRK is encountered, the computer will reset to a cold start. Dirty pool, but why make life for a hacker easy.

### #04 NMI Interrupt Vector

Hooked up to my 8032 is a little device called the break box. Jim Butterfield and a few others have been talking of this little wonder for years. With the break box you can stop a program in its path and go to READY mode. Or you have one more option, at least with my particular model. You can break directly into the 4.0 monitor, without disturbing the program in the least. Well, for the first action of breaking into READY mode, there is a cure. Point the NMI vector at reset (see above). When an attempt is made to break to READY mode, you will be met with a cold start of the computer. A pretty simple way to deter a few hackers.

### #05 OPEN Vector

This vector is only accessible by those with either the C64 or Vic 20, which is a real shame for all the 4.0 users reading. Commodore really had their heads screwed on straight the day they introduced neat vectors like this. To confuse the heck out of pirates everywhere, change the meaning of OPEN within your program. Point it at anything that you feel like, and watch the confusion grow. Imagine LOADing in a program with a simple:

OPEN 5,8,5,"program name"

Flip the LOAD/VERIFY flag and try it. It seems to work. If all else does not appeal to you, point the OPEN vector at reset when not in use. Might never be used, but why not. Some hacker may use a technique involving the OPEN statement, of which a cold start would come as a surprise.

### #06 CLOSE Vector

As with the OPEN vector, the CLOSE vector is also limited to those with either the C64 or VIC. The same techniques apply with this vector as was with the OPEN vector, so cloud the issue a little when using this one and point it to everything but what it really is. As I have said before, if all else fails, point at reset while not in use.

### #07 INPUT Vector

This is a favourite of mine. Place the following auto run code somewhere in memory, and point the input vector at it once in program mode. As long as you do not use an INPUT statement anywhere in your program, this bit of code will not be executed until you have gone back to READY mode. Once that happens, the code is executed and your program will be re-run all over again. Terrific if someone is trying to crash your program, or you don't want anyone to see your program crash. Keep this code in mind when thinking about re-routing some of the other vectors too. Just imagine the confusion level if every move the pirate makes causes the program to re-run.

Another use for this vector is to point it at reset, as I have belabored with every vector so far. The moment your computer goes back into READY mode, the machine will execute a cold start. It's mean, but it works.

*** Auto Re-Run Code For 4.0 BASIC, C64 and VIC 20 ***

```
lobas   =  $28    ;4.0 basic - low  byte start of basic
loval   =  $01    ;4.0 basic - low  byte value of start of basic
lobas   =  $2b    ;c64 & vic
loval   =  $01    ;c64 & vic (default)
hibas   =  $29    ;4.0 basic - high byte start of basic
hival   =  $04    ;4.0 basic - high byte value of start of basic
hibas   =  $2c    ;c64 & vic
hival   =  $08    ;c64 (default)
hival   =  $10    ;vic (default)
lovar   =  $2a    ;4.0 basic - low  byte start of variable
lovar   =  $2d    ;c64 & vic
hivar   =  $2b    ;4.0 basic - high byte start of variable
hivar   =  $2e    ;c64 & vic
loend   =  $c9    ;4.0 basic - low  byte end of program
loend   =  $ae    ;c64 & vic
hiend   =  $ca    ;4.0 basic - high byte end of program
hiend   =  $af    ;c64 & vic
clr     =  $b5e9  ;4.0 basic - reset basic and perform 'clr'
clr     =  $a659  ;c64
clr     =  $c659  ;vic
fix     =  $b4b6  ;4.0 basic - fix chaining
fix     =  $a533  ;c64
fix     =  $c533  ;vic
cont    =  $b74a  ;4.0 basic - perform 'cont'
cont    =  $a857  ;c64
cont    =  $c857  ;vic

;*** auto re-run with reset of start of basic***

lda  *loval  ;reset start of basic to normal
sta  lobas
lda  *hival
sta  hibas
lda  loend   ;reset end of program/start of variables
sta  lovar
lda  hiend
sta  hivar
jsr  clr     ;reset basic and do 'clr'
jsr  fix     ;fix chaining
jmpcont      ;perform 'cont'
```

### #08 OUTPUT Vector

The OUTPUT vector is another versatile vector that can be used for all sorts of functions. For every key that is pressed on the keyboard, the OUTPUT vector is used. For many of your disk operations, the OUTPUT vector is also used. Point this vector at the code of your choice, make sure that the code is not too verbose, save and retrieve all corrupted registers on the stack, and an alternative to CHRGET protection has been found. Those of you with PETs are excluded from this one though. It seems Commodore wasn't that bright when the PET evolved, but adapted later for all the others. For all around use, this one is a pleasure to use. Few people know of its existence, so why not use this factor to your advantage.

### #09 Test STOP Vector

Again, another point scored for C64 & VIC users. One more vector that the PET/CBM owners drool to have. This vector allows you to disable the STOP key:

```
POKE 808,239
```

. . .disable the RUN/STOP-RESTORE combo:

```
POKE 808,225
```

. . .or whatever you can find to disable when rummaging about through the code. Again, as I have mentioned earlier, develop some really interesting code for your protection, then point a vector at it. This one is as good as you will probably find, so consider it when looking for a new and improved vector to take advantage of.

### #10 GET Vector

Another vector you C64 and VIC owners have over the rest of the world, and a pretty good little beast to have on your side. As you probably know, GET is a BASIC keyword, associated with a command to GET characters from some device or another, usually the keyboard. Point this vector at some ingenious protection code, and watch as the code is executed every time you attempt to GET anything within your program. A sneaky and underhanded way to bring a pirate to his knees, but who really cares. Its your program,

so why not try to keep it that way.

**#11 LOAD Link**

Add this one to the C64/VIC resume, for the PET/CBM owners have been left in the cold once again. Without flogging a dead horse any more than necessary, change this link address to point elsewhere and you may be able to confuse many who peek into the lower recesses of your code. Use your imagination and you will be surprised as to the number of tricks that can be performed with just a few vectors and a couple of knowledgable protection routines.

**#12 SAVE Link**

Take a peek up above and read again, this time remembering that we are discussing a different vector. PET/CBM owners have once again been given the dirty end of the stick, with the C64 & VIC people coming out with another winner on their side. Use imagination and plenty of raw, brute spunk, and watch as the hackers fall aside to the power of your code. Science fiction writing has always been a secret ambition of mine.

**#13 POINTER: Start Of BASIC**

Executing BASIC at a different location other than normal is an interesting way to confuse a moronic pirate. LOAD in your normal program, change the Start of BASIC, then chain in the next section of code. Execution will be immediate, and confusion will suddenly run rampant. In this way, you could have numerous programs in memory at the same time, all stacked above each other with each performing a specific function. If broken, only one section would easily be viewed. A technique that is rather unorthodox, but logical considering the circumstances.

**#14 POINTER: Top Of Memory**

As most already know, lowering the top of BASIC memory will give you a place to store additional code where it can't be disturbed. Alter these pointers to anywhere you like, and you can mix a good proportion of BASIC and machine code together in harmony.

**#15 Number Of Characters In Keyboard Buffer**

For some auto boot routines, this one is terrific. A few issues back I wrote an article about using the keyboard buffer and this location for an easy boot technique. The technique is very fast and simple and a pleasure to see in action. Once the program knows what is to be done, print the actions required on the screen in calculated positions. Poke carriage return characters (CHR$(13)) into the keyboard buffer to coincide with the screen contents, then poke the number of characters presently in the keyboard buffer into this location. The next step is to END the program. The keyboard buffer will take over from there to do what ever you have requested. A nice sleight of hand.

**#16 Maximum Size Of Keyboard Buffer**

This location is one which can come in very handy. Normally set to allow 10 characters into the buffer at a maximum, you can alter this for any value up to 255. Alter this location to any value you care before executing routines that require an input from the keyboard, then down to 0 once the routine has received all that it wants. If the program ever crashes for whatever reason, what good would it do anyone? The keyboard has been effectively turned off, thus stopping the code snoop in his tracks. Sneaky and very reassuring to use.

**#17 Error Message Link**

A programmer that I know, Brian Munshaw, developed an ingenious method of using this link to his advantage on the Commodore 64. He has designed a fabulous graphics package, but did not want to use CHRGET to check for his special keywords. Using CHRGET wedges quite often slows down BASIC execution a little bit too much when heavy checking is required. Brian designed a little bit of code that is pointed to by the error message link. Normal keywords will not generate an error, but pretty much everything else will. This code checks the error generated to see if it was in the range of one of the new keywords, or just an ordinary error by some spastic programmer. If the error was the new keywords fault, then the code will branch to the routine to handle it. For normal errors it will jump to the normal error processing routine. Pretty terrific, and also a great way to design packages that will confuse many people. Design your own language for your program, and incorporate some protection techniques into the language. The work required to fix your program up would be more than most hackers would want to allocate.

**#18 Monitor Extension Vector**

This vector is used only on 4.0 BASIC machines by utilities that extend the machine language monitor with extra commands. Change this one to reset, $FD16, and even if you have missed a few of the pirates spies, the moment they try to disassemble your code, or what ever else they have planned with their extended monitor, the computer will jump to a cold start. For two pokes, not bad.

There are many more POKEs that are as yet still in hiding, and will remain so for the balance of this issue. By stretching this article on too long in this area, many other points would have to be skipped. In future issues we will be printing new and improved POKEs to add to this collection.

And now, DOS tricks and other shenanigans to further inflame the boils of irate pirates worldwide.

**Playing Tricks With DOS**

The disk drive should be classified both as a tool and a toy, at least for me. These versatile little containers of brilliance can liven up even the dullest of days, and make the average program sing with delight. Not only can you tell the drive to make some pretty obtuse maneuvers, but you can also tell the disk that it enjoys what is happening. In this way, you can let fly with a few smoke bombs even the most astute hacker can't see through. This is war, so let them have it with all you got!

Keep in mind that multiple stacking of disk protection tricks can be of great benefit in the final outcome of this war.

### Change Block Count Of Files

You can physically change the block count of files on disk, and confuse the heck out of people who copy your files. They will never be completely sure if they got the right file, or the complete file. There will also be the chance that they would never look at a small file when they are trying to locate the main program. To change the block count, look in the directory track of the disk. Held in low/high byte fashion, they are the last two bytes in the files directory entry. Anything from 0 to 65535 is acceptable.

If an easier route is desired to change the block count, take a peek through this issue and you are bound to trip over a program I wrote to change the block count of files, scratch protect files, and back-up protect your diskettes. Another utility, DiskMod, will help you do the same thing, but it's as little less automatic.

### Re-Direct Track & Sector Pointers Of Files Back Unto Thine Self

For a bit of fun with your disk, foul up a couple of files in this way. Files are held on disk with the first two bytes of each block as the pointer to the next block of data. By changing this pointer to point back to one of the blocks prior, copying files could be somewhat tedious at best. A COPY command would go forever.

From program mode you could do a few things to compensate. Read a specific section of the file in one byte at a time, then close up the file. Or you can write to the disk to reset the pointers correctly, LOAD in the program, then reset the pointers back in your strange fashion. A hackers nightmare is born.

### LOAD And SAVE Programs In A Novel Way

When preparing this article I was indecisive whether to let you know this little gem of protection. The number of people who know of it is a mystery to me, but I do know it to be a carefully guarded among those who hold the secret. The purpose of our magazine is to educate, so lets peek into this one and become educated.

Program files can be saved to disk as SEQ or USR files, with the right technique. These same files can then be LOADed back into memory and re-executed as PRG files.

To SAVE a program file as a sequential file try this :

```
save "0:filename,s",8
```

To LOAD the same program back in as a program file:

```
load "0:filename,s",8
```

The file will appear on the directory as SEQ, but it will really be PRG. To SAVE and LOAD USR files, substitute a 'u' instead of the 's' in the above examples.

### Make Disk Non BACKUPable

As stated earlier, there is a program to be found in this issue that will perform this trick with your diskettes. It will allow you to protect your disk from regular duplication. Not only this, but you cannot SCRATCH files, SAVE to the disk or RENAME any files for all time thereafter. The only trouble is that COPYing is still allowed, but we're working on it.

On the disk surface, there are markers called DOS Version Identifiers. These little beasts tell the drive what version of DOS the diskette was formatted on. By altering these bytes, the drive will be fooled into believing that the disk is wrong for it. For a 1541/2021/4040 diskette, the identifier is a 65 decimal, or $41. It is located on track 18, sector 0 as the third byte from the start. Change this to what ever you like, and your disk will become an alien.

On the 8050/8250 drives, the DOS Version Identifier is located on track 38, with both drives using sectors 0 and 3, and the 8250 also using sectors 6 and 9. The normal byte value is 67 decimal, or $43, and is also the third byte from the start. Change this animal to whatever you like, and watch the hackers sweat just a little more.

Now I am going to pop the bubble that was just created. While pouring over some maps that I have been preparing on the disk drives, I came up with a technique to backup a backup protected disk. All that you do is tell the drive that it really is as strange as the disk says it is. Then you can write to the disk, SCRATCH, SAVE and BACKUP. Performed from within program mode, you could control just what activity is taken with your disks. The code is as follows.

```
10 id = ascii value of new DOS identifier
20 open 15,8,15
30 print#15,"m-w"chr$(159)chr$(16)chr$(1)chr$(id);
40 close 15
```

This code works for 4040, 8050 and 8250 drives. For the others, excuse me but I haven't come up with the code yet. Maybe in a future issue.

### Change File Type On Disk

This technique, though not one that should be encouraged by anyone but a person bent on the destruction of all hackers, is pretty inventive to say the least. Confusing the heck out of a hacker is exactly what the following will do. Below I have prepared a chart with disk massaging tricks.

| HEX | Description |
|---|---|
| $00 | Unclosed DEL File (•) |
| $01 | Unclosed SEQ File (•) |
| $02 | Unclosed PRG File (•) |
| $03 | Unclosed USR File (•) |
| $04 | Unclosed REL File (•) |
| $80 | Closed DEL File |
| $81 | Closed SEQ File |
| $82 | Closed PRG File |
| $83 | Closed USR File |
| $84 | Closed REL File |
| $88 | Closed DEL File – fouled up file type description – copys as DEL |
| $89 | Closed SEQ File – fouled up file type description – copys as SEQ |

$8a Closed PRG File – fouled up file type description – copys as PRG
$8b Closed USR File – fouled up file type description – copys as USR
$8c Closed REL File – fouled up file type description – won't copy
$c0 Scratch Protected & Closed DEL File
$c1 Scratch Protected & Closed SEQ File
$c2 Scratch Protected & Closed PRG File
$c3 Scratch Protected & Closed USR File
$c4 Scratch Protected & Closed REL File

Bit Representations :

| | |
|---|---|
| bit 0 | off = DEL file, on = SEQ file |
| bits 0 & 1 | bit 0 off and bit 1 on = PRG file |
| bits 0 & 1 | bit 0 on and bit 1 on = USR file |
| bits 0, 1 & 2 | bit 0 off, bit 1 off and bit 2 on = REL file |
| bits 3 – 5 | not normally used |
| bit 6 | off = normal file / on = scratch protected file |
| bit 7 | off = open file / on = closed file |

As the charts above show, there is quite a bit to be learned about the diskette itself. For years people have been wanting to scratch protect their files, with the best example being teachers in schools with hundreds of students bent on file destruction. Commodore, with their typical streak of brilliance, have never bothered to tell anyone much of anything at all. Well, files can be scratch protected on every version of DOS so far. The trick is to set bit six of the file type indicator on the directory entry of the file. Once set, a '<' will appear to the right of the file type, and the file cannot be scratched. Everything else is still allowed though. Don't you think that the time has come for Commodore to start letting us in on the workings of their drive units? They have never used this feature for anything whatsoever, but they knew it was there all along. Why couldn't they let a few people in on their secret, and maybe benefit a whole lot of people in the process? I leave you to fill in the answer to this one.

The file type values of $89 to $8b are of special interest to me. You could save a PRG file to disk, then change the file type description on disk to anything from $89 to $8b. These files will not LOAD in normally, but they will COPY correctly. From program mode you could COPY the file on disk first to something like a USR file, LOAD the USR file in as PRG, then SCRATCH the recently copied file. It is a lot of work, but it would prove to be good armor against the ever illustrious hacker.

### Change Header And Filename To Foul Up Directory

Filenames and headers are often not thought of to foul up the hacker. Most users of software never have to do a directory of the disk anyway, so why not throw a few curve balls in the directory department.

By curve ball I mean a few pseudo control characters. Try the following header manipulation and I am sure you will quickly understand what I am getting at.

### Format A Disk, With A Twist

```
10 fu$ = chr$(141) + chr$(19) + chr$(19) + chr$(147)
   + chr$(15) + chr$(143)
20 open 15,8,15, "n0: " + "" + fu$ + ""
30 close 15
```

For those of you with the CBM machines, you will benefit most from this demonstration. Not only will this header, once viewed through a catalog, clear the screen, it will also set two windows at the top left hand corner of the screen. In this way, once a catalog is performed, the keyboard will appear useless. To remedy the situation, touch the HOME key twice in a row. This will clear the windows and bring your computer back to life.

The reason for the fouled up header is the first character in string FU$, CHR$(141). On the keyboard you can reproduce this one as a capital reversed M. It is the illustrious carriage return, and allows you into a whole new world of program foul ups. If you were to look at Jim Butterfields Super Chart, numerous interesting character combinations can be thought up that will allow you to foul thine hacker. Use your own special combinations of these characters, preceded always by CHR$(141) to royally mess up filenames on disk, REM statements in BASIC programs, and variables within BASIC programs. You can also change the colour of program listings with the right combo of characters. For more information on this marvelous technique, read Jeff Goebels column this issue. Jeff discusses methods to further protect your programs with control characters.

### Ye Olde Standbye

One final trick to perform with file names is the old stand by. Look below for a directory entry that can be yours with the right combination of key presses.

67 ": filename prg

To get this result, you have to SAVE the program in a little bit different manner. Type in:

SAVE"0: : filename",8

. . .but press the (shift) key with the space bar between the two colons. This shifted space will fool the disk into belleving that the filename is finished, and that it should write the balance outside of quotes. This is due to the fact that filenames are stored on disk padded with shifted spaces, chr$(160)'s. In case you do not already know, 67 is an imaginary block count. Your block count will be what ever size your program takes up on disk.

### And A Way And A Way And A Fife And Drum

A rather hasty departure from the DOS you might feel, but quite a bit has just been covered for your protection needs. It is now your job to figure out how best to use the information I have supplied to create your own disk protected land mines.

**Auto RUN Programs**

Most programs that you find that are auto run are just too simple to break. My favourite method to do this is to reset the start of BASIC to the stack before LOADing in the program. For 4.0 BASIC people try poke 41,1. Once you have LOADed most auto runners, the program will crash with SYNTAX ERROR? displayed prominently on the screen. Now poke 41,4 and you can list the program. At this point some of these programs are rather touchy to play with. Type in 63999 then press (return). You have effectively deleted line 63999, which most likely doesn't exist anyway. This will also allow you to do whatever you like with the program from there on in.

This method is just too easy for what otherwise would be a pretty good technique. Auto boot programs should automatically reset the start of BASIC before executing the auto run code. Jim Butterfields 'Lock Disk', Richard Mansfields 'Bootfixer', and a host of others allow you to make your programs auto run, but they do not reset the start of BASIC. Quite a while ago I modified Richard Mansfields Bootfixer to do this, and today I bring these mods to you. You can find this program in Compute! Magazine of October 1982, Issue 29 on pages 170 & 172. The program is good, but not perfect. First, this program works only with 1541, 2031 and 4040 drives. The reason is simple. On line 100 the variable T=18. To convert to 8050/8250 change this variable to T=38. Next, to produce a better auto run code that is more difficult to break, enter the following lines.

```
475 data 169, 1, 133, 40, 169, 4, 133, 41
510 for pb = 105 to 129 : read by : print#15, "b-p:2" ;pb
```

Once your programs have been modified with the new Bootfixer, they will be impervious to most attacks. But do not be mislead. There are plenty of other techniques that, while they are not as simple, will break auto run code. Reset all your vectors to computer reset on the execution of your program, and lay a few more traps to confuse the hacker, and you may produce your own version of a hacker cracker. Try tucking valuable code below BASIC, use this code often within program mode, and the hacker will be more prone to leave your program auto run. One final point to remember before releasing your bullet proof wonder. Fire up your favourite wordprocessor and load in your program as a text file. Now look closely at the garbage on the screen. Can you gain any knowledge from this display that would help you break your own code? If your answer is no, pat yourself on the back. If not, foul up the program a little more and try again. Who knows, maybe some hacker will respect your efforts so much that your code will be left alone.

**Use Some Of The Other Programs In This Issue**

Even if 100,000 people read this article, rest assured that this figure represents only a small fraction of the total number of Commodore users world wide. Not only that, but you also now know that it is easier to protect than de-protect a program, given the right attitude. Protection can be fun, and with plenty of imagination and raw courage, you might tame the savage hacker. Take the extra measure of legal contracts, serial numbers and fine quality manuals, and you might win in this technological war. And take one final step, as explained below.

We have published quite a few articles and programs this issue to protect your creations, so read them with a notebook at hand. Jot down everything that interests you, and cross reference your notes to the pages of our magazine. Take some time with the protection of your program and flowchart it out carefully. Remember that hackers often deserve their title. With careful thought and imagination you might be able to outsmart even the craftiest pirate. The hacker will try every trick that can be thought of to destroy your work. Figure out how you would break your own program, then further strengthen your defenses. But try to keep in mind that the extra time is worth the extra revenue. If your program is poor to begin with, all the protection in the world will not make it any better. What ever you decide to do, be imaginative about it and change your methods of attack from update to update. Just have to keep those hackers on their toes.

Many of you will notice that quite a few tricks remain unsaid in this article. The reason for this is simple. Not everybody is as interested as you are in protection, so why fill an entire issue with one specific train of thought. In future issues I intend to cover this subject further, so write to us and give us your thoughts on the matter. We may have gained a few enemies, but hopefully we have also found many new friends. The purpose of this article was not to harvest a new crop of hackers world wide, but to educate and warn programmers about the extent of piracy today, and how to combat it. I hope that you feel this has been accomplished.

**A Final Few Swings At The Insidious Pirate**

No matter how much the pirates try to justify their actions, one fact remains. They are thieves, and are stealing from the pockets of their fellow programmers. A theft in any other form would constitute a crime, one punishable by law. This method of theft is over the heads of our best legal minds, so how much legal protection can we expect. The best legal protection in the world will not help in the case of a crime that cannot be proven. Our only true hope is to somehow unite the programmers and hackers into one, thus eliminating the destructive element.

It would not be a surprise if one day we find few new programs coming onto the Commodore scene. Why bother spending mega hours of time and energy in the creation of a truly fine piece of work, only to have your just rewards stolen by an over zealous thrill seeker. Hackers, please take my advice. Write a program that you know will sell, protect the heck out of it and market it. If your hacker friends do not get to it first you may be able to walk away with enough profit to buy yourself a much greater thrill. Buy a Porche and drive yourself into a frenzy. Buy an island and become a recluse. Buy a distillery and become permanently intoxicated. Just buy something that will give you a greater rush than what is experienced by staying up all night staring at your computer screen, bashing away at little tiny keys, and making zero profit for yourself and the company that you ripped off. The acceptance of yourself by your other high tech pirates is nice, but imagine how nice it could be skiing in Switzerland, surfing in California, mountain climbing in British Columbia, racing in Monte Carlo or doing whatever else you find to tickle your fancy. Drop the underground software network and stop stealing from your friends.